**THESE DE DOCTORAT DE**
**SORBONNE UNIVERSITE**
préparée à EURECOM

École doctorale EDITE de Paris n° ED130
Spécialité: «Informatique, Télécommunications et Électronique»

Sujet de la thèse:

# Protocol-level Attacks and Defenses to Advance IoT Security

Thèse présentée et soutenue à Biot, le 05/12/2024, par

Marco Casagrande

| | | |
|---|---|---|
| Président | Prof. Aurélien Francillon | EURECOM |
| Rapporteurs | Prof. Thomas Eisenbarth | University of Lübeck |
| | Dr. Clémentine Maurice | CNRS |
| Examinateurs | Prof. Kasper Rasmussen | University of Oxford |
| | Prof. Nils Ole Tippenhauer | CISPA |
| Directeur de thèse | Prof. Refik Molva | EURECOM |
| Co-directeur de thèse | Prof. Daniele Antonioli | EURECOM |

# Abstract

The Internet of Things (IoT) has become pervasive in modern life, with an ever-expanding market. These devices serve a variety of purposes, aiding and accompanying users in daily activities. However, as they frequently communicate with other devices, such as smartphones running companion apps, they present a significant attack surface. Furthermore, their constrained resources—limited computational power, battery capacity, and cryptographic support—make them attractive targets for malicious actors. For instance, adversaries could compromise the privacy of fitness trackers by leaking sensitive health data, undermine the security of FIDO2 authenticators, or jeopardize the availability and safety of electric scooters, potentially resulting in hazardous outcomes such as fire risks.

Conventional security solutions used in the web domain, such as TLS and certificate-based authentication, are often unsuitable for IoT connectivity. Instead, IoT devices typically rely on vendor-specific, application-layer protocols operating over standard transport layers such as Bluetooth, Bluetooth Low Energy (BLE), NFC, or USB. These protocols reinvent standard security mechanisms—such as key agreement, authentication, session management, and encryption—with varying degrees of success, often introducing significant vulnerabilities. Gaining a comprehensive understanding of these protocols typically requires extensive reverse engineering and manual analysis.

In this thesis, we examine a diverse range of IoT devices, spanning various levels of complexity and security robustness. We introduce smart tools designed to facilitate security assessments of these devices. These tools not only enable the detection of these vulnerabilities but also provide advanced features that support future research. These include capabilities such as fitness tracker impersonation, firmware binary analysis for electric scooters, and the development of virtual environments for FIDO2 devices.

Our evaluation focuses on the ecosystems of Xiaomi and Fitbit fitness trackers, Xiaomi electric scooters, and FIDO2 authenticators. Regardless of whether their underlying protocols are open-source (e.g., FIDO2) or proprietary (e.g., Xiaomi), or whether their hardware capabilities vary significantly (e.g., fitness trackers vs. electric scooters), we uncovered multiple protocol-level vulnerabilities. These weaknesses allowed us to compromise the security, privacy, availability, and safety of the devices, with tangible real-world consequences for both users and the broader vendor ecosystems.

# Résumé

L'Internet des objets (IoT) est devenu omniprésent dans la vie moderne, avec un marché en constante expansion. Ces dispositifs remplissent une variété de fonctions, aidant et accompagnant les utilisateurs dans leurs activités quotidiennes. Cependant, comme ils communiquent fréquemment avec d'autres appareils, tels que les smartphones exécutant des applications compagnons, ils présentent une surface d'attaque significative. De plus, leurs ressources limitées—faible puissance de calcul, capacité de batterie restreinte et support cryptographique limité—en font des cibles attrayantes pour des acteurs malveillants. Par exemple, des adversaires pourraient compromettre la confidentialité des trackers de fitness en divulguant des données de santé sensibles, saper la sécurité des authentificateurs FIDO2, ou encore compromettre la disponibilité et la sécurité des trottinettes électriques, entraînant potentiellement des risques d'incendie.

Les solutions de sécurité conventionnelles utilisées dans le domaine du web, telles que TLS et l'authentification par certificats, sont souvent inadaptées à la connectivité IoT. À la place, les dispositifs IoT reposent généralement sur des protocoles spécifiques au fabricant, opérant au niveau de la couche application, par-dessus des couches de transport standards comme le Bluetooth, le Bluetooth Low Energy (BLE), le NFC ou l'USB. Ces protocoles sur mesure réinventent fréquemment des mécanismes de sécurité essentiels—comme l'accord de clés, l'authentification, la gestion des sessions et le chiffrement—avec un succès variable, introduisant souvent des vulnérabilités significatives. Une compréhension approfondie de ces protocoles nécessite généralement une ingénierie inverse poussée et une analyse manuelle.

Dans cette thèse, nous examinons une gamme variée d'appareils IoT, couvrant divers niveaux de complexité et de robustesse en matière de sécurité. Nous introduisons des outils intelligents conçus pour faciliter les évaluations de sécurité de ces dispositifs. Ces outils permettent non seulement la détection de ces vulnérabilités, mais offrent également des fonctionnalités avancées qui soutiennent la recherche future. Parmi ces fonctionnalités, on trouve la possibilité d'imitation de traqueurs de fitness, l'analyse de binaires de firmware pour les trottinettes électriques, et le développement d'environnements virtuels pour les dispositifs FIDO2.

Notre évaluation se concentre sur les écosystèmes des traqueurs de fitness Xiaomi et Fitbit, des trottinettes électriques Xiaomi et des authentificateurs FIDO2. Qu'il s'agisse de protocoles open-source (par exemple, FIDO2) ou propriétaires (par exemple, Xiaomi), ou que les capacités matérielles varient considérablement (par exemple, traqueurs de fitness par rapport aux trottinettes électriques), nous avons découvert plusieurs vulnérabilités au niveau des protocoles. Ces faiblesses ont permis de compromettre la sécurité, la confidentialité, la disponibilité et la sûreté des dispositifs, avec des conséquences tangibles dans le monde réel tant pour les utilisateurs que pour les écosystèmes de fournisseurs plus larges.

# Acknowledgements

I would like to express my deepest gratitude to my supervisor, professor Daniele Antonioli, for his unwavering dedication and support throughout the past four years. His guidance and the vast knowledge he has shared with me have been invaluable, and I am immensely thankful for the opportunity he gave me to pursue my PhD at EURECOM.

A heartfelt thanks to professor Mauro Conti and Eleonora Losiouk, who supervised me during my Master's Degree Thesis and continue to collaborate with me in ongoing research. I am grateful for their continued mentorship and support, as well as to all the other people in academia who collaborated with me during this journey.

I am sincerely grateful to the reviewers of my thesis for their valuable feedback, and to the members of the jury, who dedicated their time and expertise to evaluating my work.

To my friends at EURECOM, I owe a special thanks for the many occasions when their help and support were invaluable. To Feras, who is still trying to get me to move to Switzerland after my defense. To Dario Nisi, my best office mate, Luigi Russo and Andrea Olivieri for the insightful conversations, and to professor Aurelien Francillon and Aurelien Hernandez for their support in the lab. I'm also grateful to my padel friends and rivals for the fun on and off the field.

A big thank you to my family, who have supported me unconditionally throughout the years, even accepting, without hesitation, the sudden news that I would be moving to France for three years. I hope it was worth it and that I've made you proud.

To my friends in Veneto, I appreciate you more than you know. You've been there for me through countless Wednesday nights, despite my antics and the silly events I organize. It makes me happy to see how our bond did not fade, and never will, regardless of the distance.

Lastly, to my love Erica, who more than anyone else, has shared in my victories and my losses. You've been my source of motivation during tough times, and your support has meant the world to me. I will never be able to tell how I appreciate your patience and help. I will always be there for you, just as you've been for me.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Internet-of-Things (IoT) devices are embedded systems designed to connect and exchange data with other systems over the internet. These devices span a wide range, from small, everyday gadgets like fitness trackers to sophisticated systems like electric scooters. They are typically built to perform specific, limited tasks while prioritizing energy efficiency and cost-effectiveness. For instance, they collect data from users and their environments through sensors. However, their design constraints mean that IoT devices often operate with minimal computing power, memory, and storage in order to reduce costs and extend battery life. To achieve connectivity while conserving energy, they rely on low-power wireless protocols such as Bluetooth Low Energy (BLE) and Near Field Communication (NFC). Their compact form factor limits their ability to integrate high-end hardware like displays, GPS, or advanced cryptographic modules.

Given these constraints, IoT devices typically rely on a supporting *ecosystem* to extend their functionality. A cloud-based backend, for instance, can handle resource-intensive tasks such as data storage, complex analysis, and machine learning on the data collected by the device's sensors. Meanwhile, a mobile companion app often serves as a proxy, enabling user interaction and providing the device with internet connectivity. This ecosystem allows the device to provide advanced features through external infrastructure.

One of the biggest challenges in IoT security is balancing efficiency and affordability with security, which is often deprioritized. IoT devices must rely on lightweight cryptography, due to the significant processing power and memory required by strong encryption and the cost and lack of space for cryptographic modules. Most devices do not support certificate-based authentication or public key cryptography. This is especially problematic when attempting to protect security-critical operations such as pairing, session establishment, and firmware updates. The widespread deployment of IoT devices across critical functions—ranging from personal health to online security and mobility—means that successful attacks can have far-reaching consequences, impacting not only individual users but potentially *millions* of people and the entire vendor's ecosystem. Below are examples highlighting the significant effects of breaches on security, privacy, availability, and safety.

Fitness trackers store and manage highly personal data, such as heart rate, physical activity, and even incoming SMS or phone call notifications. An attacker could leak or hijack this sensitive health data, compromising the user's privacy and potentially exposing them to identity theft or unwanted surveillance. FIDO2 authenticators manage cryptographic keys and credentials that secure access to online accounts. If an attacker successfully breaches these devices, it could lead to the loss of access to critical online services, such as email, banking, or

work accounts. This not only disrupts the user's daily life but also opens the door to further security risks, including account takeovers or financial fraud. Electric scooters continuously monitor their lithium-ion battery status to ensure rider safety. If compromised, a scooter's safety mechanisms could be disabled, or false data about the battery's condition could be fed to the user, leading to malfunctions or even battery fires. This not only endangers the rider but also poses risks to those nearby. Moreover, IoT devices are designed for on-the-go use, and being unable to access them when needed can cause significant inconvenience. For instance, a locked e-scooter or authenticator could prevent users from accessing essential services or transportation at inconvenient moments.

## 1.1 Problem Statement

IoT devices rely on standard communication transports, such as BLE, NFC, and USB. However, these standards often fail to address the unique requirements of specific manufacturers, prompting them to build *ad-hoc* protocols over these transports. For instance, the standard BLE GATT services lack adequate authentication mechanisms to meet the requirements of ecosystems like Xiaomi's, which employ server-side authentication. Similarly, in the FIDO2 standard, NFC communication between an authenticator and a client uses the Client-To-Authenticator Protocol (CTAP) over ISO14443 (a contactless communication standard) and ISO7816-4 (a specification for smart cards).

Manufacturers lack the resources to design secure IoT systems while also reimplementing standard security mechanisms across multiple *layered* protocols. In comparison, the Constrained Restful Environments (CoRE [1]) Working Group took four years to release their application-layer protocol, i.e., the Constrained Application Protocol (CoAP [2]), and four more years to extend [3] their IoT network stack to TCP and TLS. Moreover, vendors contribute to the proliferation of insecure IoT protocols by refusing to develop a single and robust IoT framework, instead *fragmenting* their ecosystem into smaller ones, each with their own poorly designed protocol. As a consequence to the complexity and fragmentation of layered protocols, manufacturers frequently introduce critical vulnerabilities on security-critical operations such as pairing, session establishment, and firmware updates, putting the entire ecosystem at risk. These issues can be found in different layers and components of the ecosystem. For example, in the protocol's design, the device's code or hardware, and the user interaction.

We focus on *protocol-level* issues, found in the design and logic of IoT communication protocols. A protocol-level vulnerability stems from flaws in how the protocol is defined, such as missing authentication or weak encryption schemes. These vulnerabilities, the attacks exploiting them, have a broad scope that affects all devices running the protocol, irrespective of their software and the hardware. This means that even future versions of affected devices remain vulnerable. For instance, our attacks on the Mi Band 5 were equally effective on the Mi Band 6, even though they were developed before we were aware of the Mi Band 6's existence.

Fixing protocol-level issues is particularly challenging, especially when *backward compatibility* is a concern. Addressing these vulnerabilities often requires redesigning or updating the protocol specification, which, in the case of widely adopted standards like BLE and FIDO2, demands collaboration across industries and standards bodies. For instance, we advocated for improvements to the pairing and session establishment protocols used by Xiaomi fitness trackers and engaged in discussions with the FIDO Alliance to update their Reference Threat Model.

## 1.2 Research Questions

This thesis explores four fundamental research questions confronting huge and unsolved challenges in the evolving landscape of IoT security. In selecting our research questions, we aimed to address critical gaps in the current understanding and security of IoT ecosystems, focusing on areas that are both underdeveloped and impactful.

**RQ1 - How can we improve IoT protocol security testing?** We believe it is crucial to hold manufacturers accountable for inadequate security, especially when they misrepresent the guarantees their products offer. To do so effectively, a strong foundation in protocol analysis techniques and available tools is essential. In evaluating an IoT ecosystem, we identify two core activities: analyzing the attack surface and understanding the device's underlying logic, including reverse-engineer if necessary.

The attack surface of IoT devices varies considerably, shaped by factors such as communication channels (e.g., BLE radio and USB connectors), hardware components (e.g., sensors and displays), and system architecture (e.g., presence of backend servers and internal subsystems). For example, fitness trackers and e-scooters are vulnerable to long-range attacks over BLE, while FIDO2 authenticators are exposed to close-proximity threats via NFC or local threats through USB connections. Some attack vectors exploit specific weaknesses, such as FIDO2 client impersonation attacks, which take advantage of the lack of visual feedback on authenticators. Similarly, Xiaomi's server-side pairing protocol can be manipulated to remotely unpair devices from their owners' accounts. The choice of transport mechanism also impacts how traffic is captured, decrypted, and analyzed. For instance, Android's HCI snoop log allows monitoring of BLE traffic without link-layer encryption, whereas decrypting a TLS session requires obtaining the pre-master secret key. Scalable security testing must consider all these different scenarios and variables.

Many vendors rely on security through obscurity, keeping their protocols closed-source and undocumented. This practice makes protocol analysis highly dependent on the effectiveness of reverse engineering (RE). RE is necessary for decrypting application-layer messages (e.g., those exchanged after key agreement) and understanding the binary format and semantics of the protocol's communication. Additionally, RE is a continuous effort, as proprietary protocols may evolve over time with the release of new patches or device generations. For instance, the Mi Band 4 employs a different pairing protocol than the Mi Band 3 and underwent another major update in 2021. Despite the existance of third-party tools developed by security experts, those are too device-specific, offer limited testing capabilities, and become obsolete as soon as the protocols evolves. There is a need for better tooling and a stable testing environment not requiring to buy or run the hardware.

**RQ2 - How can we find IoT protocol-level vulnerabilities and attacks?** IoT ecosystems present a range of security challenges that, despite their apparent simplicity, have proven difficult to address and have persisted for over a decade of research. These challenges stem from inherent issues such as limited device resources, fragmented protocols, and the unfair delegation of security responsibilities to end-users. Advancing IoT security requires a deep understanding of these persistent issues and addressing their root causes. Below, we highlight a selection of the critical security risks we have evaluated.

Weak or non-mutual authentication opens the door to impersonation and spoofing at-

tacks. Inadequate encryption and integrity protection expose communications to eavesdropping, forged messages, and replay attacks. Insecure firmware updates result in devices being compromised by malware or unauthorized modifications. Faulty protocol or firmware version negotiation makes them vulnerable to downgrade attacks, reducing their security guarantees to the ones on weaker iteration of the protocol or firmware. Insufficient user awareness and misleading user confirmation due to poor UI and non-transparent warnings during security-critical decisions. Manufacturer over-reliance on security through obscurity rather than strong security mechanisms leads to protocols with no security guarantees as soon as they are reverse-engineered.

**RQ3 - How can we design secure IoT protocols?** Countless academic efforts have attempted to address the security risks we previously discussed. One common strategy involves porting lightweight versions of established security mechanisms. For example, the introduction of CoAP, a web protocol similar to HTTP but optimized for constrained environments. As well as the development of DTLS in Constrained Environments (DICE [4]), which implements cryptographic primitives like Elliptic Curve Diffie-Hellman Ephemeral (ECDHE), also used in TLS. A second approach is the creation of ad-hoc solutions tailored to specific IoT ecosystems. An example is the development of security-enhancing FIDO2 extensions or the redesign of protocols to more efficiently use existing cryptographic primitives. For instance, we leveraged Xiaomi's implementation of the Tiny Encryption Algorithm (TEA) to encrypt firmware.

The primary limitation of these countermeasures is their lack of backward-compatibility and interoperability. Researchers often design entirely new protocols that are incompatible with existing ones or propose adding new hardware components to devices. For example, multiple FIDO2-related works suggest to fix client impersonation by integrating a secure display. However, many of these solutions are not (easily) portable across devices due to differences in software, architecture, or transport. For instance, while we were able to patch a session downgrade vulnerability in e-scooter firmware, extending that fix to other firmware versions required significant additional coding and reverse engineering efforts. We find it unrealistic to expect users to discard their devices and upgrade to the newer models just to address security flaws they didn't cause. Researchers should help developing countermeasures that can be applied across a wide range of devices, ideally without requiring manufacturer intervention. Improving user interaction mechanisms and releasing tools to patch devices, would empower users to take control of their own security, rather than relying solely on official updates. We do not address the current fragmentation of IoT protocols but we focus on solving high-level security concepts adaptable to various protocols, leaving implementation details to developers.

**RQ4 - How can we design privacy-aware IoT protocols?** Achieving privacy in IoT is a critical challenge as the proliferation of interconnected devices generates vast amounts of personal data, often collected and transmitted without explicit user consent. The importance of addressing privacy concerns lies in the potential consequences of data breaches, which can lead to user tracking and fingerprinting, GDPR breach, and the manipulation of user behavior. Currently, privacy in IoT is unregulated, with manufacturers simply posting an arbitrary document (e.g., Xiaomi's fitness tracker Privacy Notice [5]) that still does not make them accountable in case of privacy infringments (i.e., Xiaomi never acknowledged our several attacks leaking user health and private data). The Mozilla Foundation engaged in the Privacy Not Included initiative [6], releasing their own privacy evaluation of IoT products.

We consider unacceptable that, in the current state, the devices that collect most personal data are the ones protecting it the least and leaking it to malicious actors, dishonest manufacturers, and advertisers. Several attempts to systematize IoT security have been made, such as the Fitbit Golden Gate framework [7] which promotes reliable and secure communications, but no explicit desire to guarantee privacy. We consider privacy attacks as important as security ones, and fix them, for example by proposing rotating and dynamic BLE advertising on fitness trackers and credential identifiers in FIDO2. We highlight how there are no mechanisms in place for the user to manage which data is collected by their devices and which data is sent to the backend for processing. This would be a crucial step toward enabling users to take control of their own privacy.

## 1.3 Contributions

This thesis makes four key contributions to the study of analysis techniques, vulnerabilities, attacks, and countermeasures in IoT systems. We place a particular emphasis on protocol-level vulnerabilities due to their significant academic and real-world impact, driving fundamental improvements in security and privacy. Our research exposes critical flaws in the underlying logic and structure of protocols used by devices such as fitness trackers, e-scooters, and FIDO2 authenticators, advocating for a transition towards secure-by-design principles by vendors. We now summarize the key contributions of this thesis and discuss how each work addresses the research questions in its respective context.

### BreakMi

The contribution "BreakMi: Reversing, Exploiting and Fixing Xiaomi Fitness Tracking Ecosystem" [8] has been published in the IACR Transactions on Cryptographic Hardware and Embedded Systems (CHES 2022).

In BreakMi, we analyze fitness trackers, demonstrating how even the smallest and most inexpensive devices can significantly impact both users and the manufacturer's ecosystem. We address *RQ1* by releasing a virtual fitness tracker and companion app that communicate using the Xiaomi protocol. We publish the Xiaomi GATT services, characteristics, and their functions, along with dissectors for Xiaomi packets and their binary data format. We address *RQ2* by identifying six remote and proximity attacks stemming from ten vulnerabilities. Our attacks break security mechanisms and allow the injection of fake data into both the tracker and the backend system. We address *RQ3* by improving Xiaomi's pairing and authentication protocols, enhancing security guarantees without compromising backward compatibility or performance. Finally, we address *RQ4* by demonstrating how fitness trackers leak confidential data, such as health data and sleep schedule, and can employed in user tracking.

### E-Spoofer

The contribution "E-Spoofer: Attacking and Defending Xiaomi Electric Scooter Ecosystem" [9] has been published in the Proceedings of the 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec 2023).

In E-Spoofer, we reverse-engineer and exploit the pairing and session establishment protocols of Xiaomi personal e-scooters. We address *RQ1* by releasing a virtual e-scooter and companion app that replicate BLE communications exactly as the original, enabling testing without the need to purchase hardware. We systematize Xiaomi's four protocols and provide tools to verify the cryptographic primitives in use (e.g., none, AES, or ECDH). We address *RQ2* by presenting two proximity and remote attacks that exploit six vulnerabilities. A malicious pairing attack allows an attacker to impersonate the app and unlock an e-scooter from a distance, facilitating theft. A session downgrade attack leverages a forgotten protocol version downgrade by the developers. We address *RQ3* by redesigning Xiaomi's protocols with an intentional pairing gesture and mutual authentication. We also release a tool to detect firmware vulnerable to our attacks, as well as a patched firmware for a model no longer updated by Xiaomi. Lastly, we address *RQ4* by achieving unauthorized access to the e-scooter and all private information it contains, including mileage and the time spent in the current ride.

## E-Trojans

The contribution "E-Trojans: Ransomware, Tracking, DoS, and Data Leaks on Battery-powered Embedded Systems" is currently under submission at the IEEE International Symposium on Hardware Oriented Security and Trust (HOST 2025).

In E-Trojans, we demonstrate how an internal attacker can compromise the safety of an e-scooter driver, leading to hazards such as short circuits and fire risks. We address *RQ1* by releasing an e-scooter analysis and patching tool that can introduce malicious capabilities into legitimate battery management system firmware. We address *RQ2* by presenting four attacks that can be executed from within the e-scooter's battery management system firmware. Our findings include the first instance of undervoltage battery ransomware as well as a series of Denial of Service (DoS) attacks. We identify four vulnerabilities that facilitate these attacks, including the absence of firmware signature verification. We address *RQ3* by securing firmware through encryption and signature verification, and by protecting the internal UART bus with a secure protocol such as SCP03, along with implementing rate limiting measures. Finally, we address *RQ4* by proposing a user tracking attack that fingerprints battery details and leaks personal data over BLE advertising.

## CTRAPS

The contribution "CTRAPS: CTAP Impersonation and API Confusion Attacks and Defenses on FIDO2" will be submitted to the 10th IEEE European Symposium on Security and Privacy (EURO S&P 2025).

In CTRAPS, we propose two attack strategies to break the security and privacy of FIDO2, an open-source and robust authentication standard. We address *RQ1* by developing a virtual FIDO2 testbed, which includes a virtual client for arbitrary invocation of the Authenticator API and a virtual relying party capable of registering any type of credential on an authenticator. We also enhance existing FIDO2 dissectors to capture more packets and better visualize the information. We address *RQ2* by presenting eleven proximity and remote attacks and discussing their seven root causes, along with the limitations of the unrealistic FIDO2 reference threat

model. Our attack strategies include client impersonation (e.g., NFC reader) capable of zero-click attacks and a novel API confusion technique, where authorization permits granted by the user for certain API calls are hijacked and used for other, potentially destructive, API calls. Our attacks demonstrate that introducing new features (e.g., discoverable credentials meant to protect against third-party data breaches) can inadvertently create new attack vectors (e.g., deleting discoverable credentials via the Credential Management API). We address *RQ3* by proposing seven countermeasures, including authenticating the CTAP client and improving authorization mechanisms. We analyze how our solutions impact usability while maintaining backward compatibility and low costs, in contrast to hardware-based solutions (e.g., adding a screen). At last, we address *RQ4* by demonstrating a zero-click unauthorized extraction of all relying parties and credential identifiers stored on the authenticator.

## 1.4 Thesis Outline

The rest of the thesis is divided into six chapters, organized as follows. Chapter 2 provides some background related to protocol-level analysis, attacks, and countermeasures in the landscape of IoT security. We focus on how prior works addressed our four research questions. Chapter 3 is based on the paper "BreakMi: Reversing, Exploiting and Fixing Xiaomi Fitness Tracking Ecosystem" [8]. We analyze the proprietary and closed-source protocols of Xiaomi fitness trackers. We study their pairing, authentication, and communication phases. Chapter 4 is based on the paper "E-Spoofer: Attacking and Defending Xiaomi Electric Scooter Ecosystem" [9]. We analyze the proprietary and closed-source protocols of Xiaomi electric scooters. We study their four versions of pairing and session establishment. Chapter 5 is based on the unpublished paper "E-Trojans: Ransomware, Tracking, DoS, and Data Leaks on Battery-powered Embedded Systems". We examine the protocols for the firmware update and internal bus communications of Xiaomi e-scooters. Chapter 6 is based on the unpublished paper "CTRAPS: CTAP Impersonation and API Confusion Attacks and Defenses on FIDO2". We analyze the open-source and peer-reviewed Client-To-Authenticator-Protocol in the FIDO2 standard. We focus on the CTAP Authenticator API exposed by the authenticator. Chapter 7 concludes the thesis. In this chapter, we outline future works, research directions, and limitations of our study.

# Chapter 2

# Background

We provide relevant background information on the IoT ecosystem to establish a foundational understanding for the contents of this thesis.

Figure 7.1 presents a high-level overview of a typical IoT ecosystem. In this model, an IoT device collects data from its surroundings via sensors (e.g., accelerometer and heart rate monitor) and interacts with the user through actuators (e.g., touch buttons). The device owner interacts with the system primarily through a companion app installed on their smartphone. Communication between the IoT device and smartphone occurs over wireless (e.g., BLE and NFC) or physical (e.g., USB) transports, utilizing either proprietary protocols (e.g., Xiaomi) or open standards (e.g., FIDO2). The companion app connects to the manufacturer's backend over Wi-Fi, typically secured by a TLS channel. The backend is responsible for handling resource-intensive operations that are beyond the computational capabilities of the device or smartphone, while also managing the storage and processing of user data. This system model provides a clear framework for identifying the key points where protocol analysis, vulnerability assessment, and security enhancements can be applied.

In the following sections, we compare existing related work to the contents of our thesis. We review how other researchers tried to address our research questions, discussing their findings, and comparing them with our contribution. For more information regarding the related work specific to each contribution presented by this thesis, please refer to Section 3.10 for BreakMi, Section 4.9 for E-Spoofer, Section 5.8 for E-Trojans, and Section 6.10 for CTRAPS.

## 2.1 RQ1

*How can we improve IoT protocol security testing?* Researchers have analyzed and tested protocols using various approaches we categorize on the targeted surface, as illustrated in Figure 7.1. We review the entry points to perform analysis on IoT ecosystems.

**App-to-backend traffic and API analysis.** Traffic between the app and backend often contains security-critical operations, such as pairing, and sensitive data like the user's health metrics and daily routines. In [10], the authors verified the integrity and encryption of fitness tracker communications with the manufacturer's backend using a man-in-the-middle (MitM) attack via an HTTPS proxy. Similarly, in [11], researchers reverse-engineered the firmware update and backend traffic of Xiaomi smart home devices, including vacuum robots, allowing them to disconnect the devices from Xiaomi's cloud and add them to an isolated cloud. Burn-

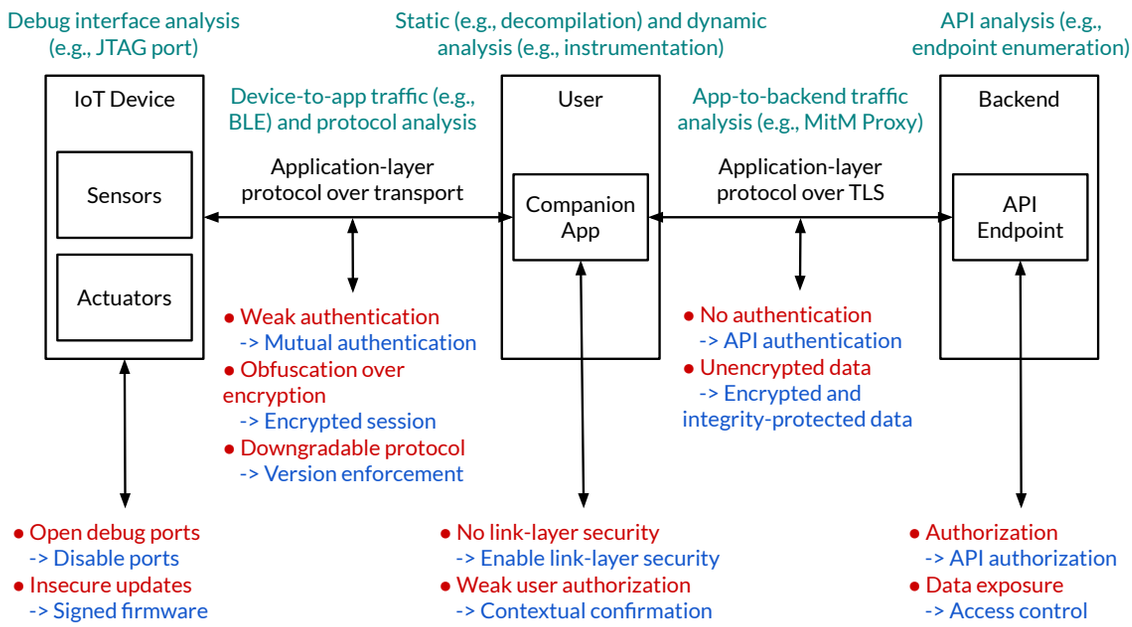



Figure 2.1: High-level system model for an IoT ecosystem. The IoT device (left) collects data via sensors and actuators. Users interact with the device through a companion app (middle), connected to the vendor's backend (right). We indicate available analysis techniques in teal. We report common vulnerabilities in red and a possible defense against them in blue.

Fit [12] tested the resistance of three fitness tracking backends to DNS spoofing, discovering vulnerabilities in all three. API analysis has also been a focus of traffic analysis. For instance, a hacker exploited insecure API endpoints of the Bird e-scooter rental service backend to remotely book scooters or trigger alarms [13]. In [14], researchers exploited a vulnerability in Fitbit's API to perform unauthorized server-side pairing, effectively unpairing the original user from their Fitbit Charge 2.

**Device-to-app traffic analysis.** Analyzing traffic between the device and app varies in difficulty depending on the transport protocol. Bluetooth Low Energy (BLE) is a common transport for IoT devices, and capturing BLE traffic is relatively straightforward if the user has control over the communicating smartphone. On Android, users can enable the HCI snoop log to collect BLE traffic. Using this technique, one researcher learned how to spoof notifications on the Mi Band 3 and reverse-engineered its GATT server, including both standard (e.g., heart rate, steps) and custom Xiaomi BLE transfer services [15]. Apple's Magic Pairing protocol for seamless BLE pairing was reverse-engineered in [16], revealing BLE advertising details and communication with iCloud. Periscope [17] examined Android apps that configure smartphones as BLE peripherals, using static analysis techniques like code decompilation to study BLE traffic. When physical access to the device is unavailable, BLE traffic can be intercepted over the air using hardware sniffers like Ubertooth, as done in [18] to discover correlation attacks.

**Automated protocol analysis.** Reverse-engineering closed-source and undocumented binary protocols is a challenging task. Automated methods like fuzzing and forensics scale well as data input increases, enabling researchers to quickly detect vulnerabilities and explore diverse execution paths. In [19], the authors applied digital forensics to Xiaomi smart speakers,

using NLP and text mining to associate speech patterns with user intentions. By analyzing proprietary Xiaomi protocols, they identified specific events like playing music. The authors of [20] developed AFLIoT, a greybox fuzzer for Linux-based IoT binaries, enabling them to fuzz real-world devices such as the Xiaomi R1D router and discover unique crashes. The Avatar [21] framework was capable of on-the-fly static analysis and device emulation regardless of the presence of peripherals. Frankenstein [22] emulated BLE devices in a virtual environment, executing large portions of code and allowing the injection of wireless frames. Incision [23] expanded on those frameworks by using a feedback loop gradually improving the reverse-engineering of the protocol, in an automated way.

**Manual protocol analysis.** Manual methods, such as code decompilation and dynamic instrumentation, involve more human effort but often uncover more complex vulnerabilities. In [24], researchers performed static code analysis on fitness tracking apps by decompiling app bytecode into Java or Smali code when necessary. Similarly, the authors of [25] analyzed privacy concerns in e-scooter apps using static analysis tools like MobSF to examine app permissions and third-party libraries. Firmware debugging was employed by [26] to find a backdoor in the Fitbit Charge HR firmware, and by [27], where JTAG ports in a Fitbit Charge 2 were accessed to dump its firmware and encryption keys. Formal verification methods, though partially automated, require protocol modeling to prove security guarantees. For instance, in [28], the authors demonstrated that FIDO U2F authentication is compromised when the server app identifier is not validated. Additionally, formal verification has been used in other IoT protocols like EDHOC and MQTT to uncover security issues [29, 30].

**Our improvements.** In our research, we applied a variety of these techniques to reverse-engineer and analyze protocols in fitness trackers, e-scooters, and FIDO2 authenticators. We developed custom security analysis and testing tools with unique capabilities not previously available, and made them public. Our improvements include: (i) *virtual environments* that enable security testing without requiring physical IoT devices or their companion apps; (ii) *attack toolkits* implemented as command-line scripts and Android apps, facilitating reproducibility of our attacks; (iii) *Wireshark dissectors* to improve visualization and understanding of custom protocol packets; and (iv) *Frida hooks* that extract real-time information about protocol execution at runtime.

## 2.2 RQ2

*How can we find IoT protocol-level vulnerabilities and attacks?* The academic community has extensively studied and reported vulnerabilities and attacks in IoT security. However, the diversity and continuous evolution of IoT ecosystems consistently drive new discoveries. Below, we present a selection of the attacks most relevant to this thesis.

**Attacks on IoT ecosystems.** Security researchers have exploited vulnerabilities across various layers of IoT ecosystems, including devices, companion apps, and backends. Numerous instances of fitness trackers, a popular target, have been victim of eavesdropping, impersonation, man-in-the-middle attacks, and data leakage. The root causes of these attacks typically stem from a lack of authentication and encryption in the communications between the fitness tracker, its companion app, and the backend. These issues were found either at the protocol level or were caused by implementation oversights by manufacturers. For example, the authentication

protocols of the Fitbit Charge HR were reverse-engineered in [26], while researchers in [14] were able to leak private data from the Fitbit Charge 2 and perform a malicious firmware update on the device. Electric scooters, though less frequently studied, have also been targeted. The Xiaomi M365, for instance, was shown to suffer from mutual authentication flaws between the device and the companion app [31]. Further exploits included locking mechanisms that abruptly brake the scooter [32] and a man-in-the-middle (MitM) attack to insert custom audio files into Lime e-scooters [33]. Vulnerable APIs in the Bird app allowed attackers to bypass QR code verification [13]. Xiaomi's diverse IoT ecosystem has also drawn attention. Researchers reverse-engineered Xiaomi vacuum cleaners in [34], bypassing secure boot, decrypting communication with the backend, leaking data, and injecting fake records. Other Xiaomi targets include smart speakers [19], security cameras [35, 36], and routers [20]. Side-channel attacks have proven effective on authenticators as well, as demonstrated in [37, 38, 39].

**Attacks on BLE security.** As the most prevalent transport layer in IoT devices, BLE has been subject to numerous attacks. Key negotiation in BLE has been compromised by downgrade attacks that force low-entropy encryption [40] or exploit BLE features [41]. All iterations of BLE pairing mechanisms, including legacy pairing [42], Simple Pairing [43, 44], and Secure Connections Only Mode [45], have been successfully attacked. Method Confusion [46] enables man-in-the-middle attacks by pairing devices using two different modes, which is hard for users to detect. Privacy attacks, such as tracking users via MAC addresses, have been demonstrated through botnets [47] or by exploiting BLE advertising across apps [48]. A large-scale study in [49] examined Android apps that share BLE channels, revealing the potential for abuse by malicious apps. BLECryptracer [50] leveraged this flaw to perform rogue firmware updates on BLE devices. Application-layer protocols built on top of BLE have also been scrutinized. For example, researchers in [51] compromised the ZeroConf protocol in Apple's ecosystem, allowing them to intercept Bluetooth channels used by Tencent QQ and Scribe apps.

**Cryptographic misuse in IoT.** Despite its critical role in securing IoT, cryptography is often misapplied. Cryptographic key extraction has been a major area of focus, particularly in automotive systems. Vulnerabilities were found in keyless entry systems, including Tesla Model S [52] and X [53], and KeeLoq-based entry systems [54]. Transponder weaknesses that immobilize cars, such as Hitag2 [55] and Megamos Crypto [56], were also exploited. Xiaomi devices exhibited cryptographic weaknesses, from excessive reliance on obfuscation [57, 58] to firmware decryption key leaks [59]. FIDO2's cryptographic protocols, including privacy, revocation, attestation, and post-quantum security, were formally verified in [60]. However, a public key substitution attack was identified in [61] for FIDO2, exploiting write access to the server database. In [62], Samsung's TrustZone Keymaster, responsible for managing FIDO2 credentials, was found vulnerable to an AES-GCM IV reuse attack due to a downgrade that allowed a fixed IV to be used instead of a random one.

**Our improvements.** In our research, we found new, impactful, and low-cost attacks and vulnerabilities in fitness trackers, e-scooters, and FIDO2 authenticators. Our improvements include: (i) *twenty-seven IoT vulnerabilities* such as unauthenticated pairing, replayable session, unsigned firmware, and trackable devices; (ii) *twenty-three IoT attacks* such as the first battery ransomware and a zero-click authenticator factory reset from remote and proximity; and (iii) *the API confusion attack strategy* misdirecting the user into calling an unintended CTAP API.

## 2.3 RQ3

*How can we design secure IoT protocols?* The IoT field is in constant need for robust countermeasures to the increasing threats it is facing. The research direction is not fixing the protocols, but detecting signs of tampering or the presence of a malicious actor instead. This is a sensible way for the user to mitigate damage, but does not solve the core issues at the protocol-level, leaving the system forever vulnerable. We discuss two common ways to defend IoT devices.

**Intrusion and anomaly detection.** Intrusion detection systems (IDS) mitigate the impact of an attacker already infiltrated in a system, accepting that sometimes it is impossible to fix the protocol's design. The authors of [63] proposed a supervised IDS for smart home, using machine learning on traffic captures to classify legitimate and malicious packets. Passban [64] tried to solve the issue that edge IoT devices cannot efficiently deploy signature-based IDS due to resources and infrequent updates, by relying on anomaly detection. Anomaly detection was also used by [65], an IDS for IoT environments that detect classic network attacks, such as ARP poisoning, DNS spoofing, and flooding via COAP/HTTP. Oasis [66] is a BLE framework capable of injecting intrusion detection algorithms into the firmware of BLE controllers. Its detection module relies on instrumentation and uses heuristics for each specific attack.

**Remote attestation.** The goal of remote attestation is ensuring that the software running on the IoT device is legitimate and was not tampered. This practice is often not supported by devices or other components in the ecosystem. Researchers have analyzed remote attestation in Yubico authenticators [67], finding that the proposed Asynchronous Remote Key Generation produces unforgeable challenge-response signatures. Other attestation mechanisms modes have been evaluated, such as [68], providing global key revocation and anonymity to the encryption scheme underlying key wrapping, and [69], featuring different attestation modes. Alternatives to remote attestation have been discussed in literature. For example, in [70] the researchers present an efficient trust mechanisms that prevents Sybil attacks in IoT networks with constant access to the backend, such as smart grids in agricolture. Trusted Execution Environments have been employed to guarantee authorized access to resources, such as in [71], where access control policies are put in place and policy-infringing applications within TEE are scrutinized. The Internet Engineering Task Force (IETF) proposed Software Updates for Internet of Things (SUIT) for secure software updates, that has been successfully tested on low-resource devices by [72]. The authors of [73] designed an online data integrity monitor for sensor systems managing data failure and recovery.

**Our improvements.** In our research, we proposed practical and backward-compatible defenses to fix our attacks and solves their root causes at the protocol-level. We responsibly disclosed our finding to the vendors, that ignored (i.e., Xiaomi), fixed (i.e., Google), or are currently evaluating (i.e., FIDO2 and Microsoft) our feedback. Our improvements include: (i) *strong protocol enhancements* that raise their security and privacy with robust countermeasures, such as mutual authentication, contextual user confirmation, firmware verification, and trusted clients; (ii) *vulnerability assessment tools* that check whether a device is affected by our attacks, such as our Yara signatures for the Xiaomi e-scooter firmware and our Android app testing a Yubico implementation vulnerability; and (iii) *user patches* that protect their devices without relying on security updates from the vendor, such as our patch to the Xiaomi M365 firmware.

## 2.4 RQ4

*How can we design privacy-aware IoT protocols?* The IoT field is in constant need for robust countermeasures to the increasing threats it is facing. The research direction is not fixing the protocols, but detecting signs of tampering or the presence of a malicious actor instead. This is a sensible way for the user to mitigate damage, but does not solve the core issues at the protocol-level, leaving the system forever vulnerable. Privacy can be broken by the manufacturer, who is not inherently malicious, or third-party attackers with malicious intentions.

**Privacy violations by manufacturers.** Devices frequently upload data to the manufacturer's backend due to storage limitations and for server-side data processing. However, there is often no transparent method to verify what information is being transmitted, raising concerns about potential oversharing of user private data. For instance, the authors of [18] discovered that Fitbit Flex trackers collect more data than what is officially disclosed. They found evidence of per-minute activity records being sent to the backend, information that is never shared with the user. Furthermore, manufacturers do not always adhere to the privacy standards they advertise. In a study evaluating eleven vendors, researchers in [74] identified two vendors that were non-compliant with their own privacy policies. One potential solution to mitigate vendor interference is to decouple devices from their backends. For example, in [75] a security expert developed custom firmware for a Xiaomi vacuum cleaner that connects to alternative, more secure personal cloud endpoints. However, implementing such modifications typically requires root access, which is not feasible for most devices. Additionally, IoT companion apps often share sensitive information with advertising companies, who use this data for targeted marketing campaigns. The authors of [76] highlight the alarming volume of confidential data that can be retrieved from sensor networks.

**Privacy violations by third-parties.** Malicious third-parties exploit the communication channels of IoT devices to leak confidential data and track users. The authors of [47] demonstrated how BLE advertising can be used to identify users based on the traffic patterns of their fitness trackers. The BLEScope [77] analysis tool extracts service and characteristic UUIDs from BLE apps and fingerprints them based on static information. It also assesses whether apps encrypt and authenticate application-layer traffic by analyzing cryptographic API calls. In another study [10], the researchers evaluated the security of Wi-Fi traffic between fitness trackers and backend. The find a lack of encryption and integrity protection, which allows attackers to eavesdrop on user web login information and sensitive data such as health records. Similarly, the authors of [78] revealed that attackers can read, modify, and delete records stored in Fitbit memory banks, exposing all contained data. As demonstrated in [79], a malicious ISP or network observer could infer private home activities by analyzing encrypted Internet traffic from smart appliances. In [80], the authors discovered that exploiting illegal states and unauthorized logins could even grant attackers access to private home camera footage.

**Our improvements.** In our research, we find and discuss vulnerabilities, attacks, and countermeasures involving privacy and GDPR-protected health data. Our improvements include: (i) *four privacy-related vulnerabilities*, such as unprotected sensitive memory in e-scooter firmware and trackable credential identifiers in FIDO2; and (ii) *nine privacy-breaking attacks*, such as fitness tracker impersonation (i.e., reading SMS messages from phone) and user tracking via e-scooters and FIDO2 authenticators.

# Chapter 3

# BreakMi

***This contribution, titled "BreakMi: Reversing, Exploiting and Fixing Xiaomi Fitness Tracking Ecosystem", has been published in the IACR Transactions on Cryptographic Hardware and Embedded Systems (CHES 2022) [8].***

## 3.1 Introduction

*Fitness tracking systems* are complex and pervasive technologies used to monitor sensitive (health) data. They are composed of wearable devices connected to a mobile application acting as a gateway to cloud services. We have seen impactful security and privacy breaches affecting those systems. For example, researchers found backdoors in trackers' firmware [26], managed to flash malicious firmware wirelessly [14], leaked private data, including health records and login credentials [81], and disabled communication encryption [27]. It is not straightforward to fix these issues as vendors might not patch them at all, and fitness trackers might not support (secure) remote patching. Furthermore, vendors must distribute the software fix securely and on a large scale.

*Xiaomi* is the worldwide fitness tracking leader. In 2020, Xiaomi sold 13.5 million devices and had 24.5% of the market share [82]. Nevertheless, the Xiaomi fitness tracking ecosystem received little attention from security researchers, despite being pervasive and promising security and privacy guarantees to its users [5]. Currently, there is only incomplete and outdated information about Xiaomi security mechanisms [24, 10]. On the other side, *Fitbit* (its main competitor) has received more consideration from security researchers. For example, recent work demonstrated that popular Fitbit devices are susceptible to attacks such as packet sniffing, data exfiltration, and code injection through firmware updates [78, 18, 83, 26, 14].

In this work, we perform an extensive and up-to-date security evaluation of the Xiaomi fitness tracking ecosystem that is currently lacking. We analyze all Xiaomi trackers since 2016 (i.e., Mi Band 2/3/4/5/6 and Amazfit Cor 2) and up-to-date Xiaomi companion apps (Mi Fit and Zepp). Via extensive static and dynamic reverse-engineering experiments, we reconstruct Xiaomi's *proprietary* Pairing, Authentication, and Communication protocols used to connect trackers and apps via BLE. We find that these protocols are implemented at the application-layer over a BLE link-layer. Moreover, we discover that Xiaomi ignores standard BLE security mechanisms (e.g., BLE pairing and secure sessions), although its devices support them.

Then, we uncover *severe specification-level vulnerabilities* affecting the self-baked Xiaomi protocols. For example, keys are sent in cleartext, authentication is unilateral and replayable,

and the BLE traffic is neither encrypted nor integrity protected. As the vulnerabilities target the protocols' design, they can be exploited *regardless of* the hardware and software details of the target (e.g., firmware, operating system, app, and BLE versions). Additionally, those issues might even be exploitable on other Xiaomi products sharing the same application-layer security mechanisms.

To demonstrate the impact of the presented vulnerabilities, we develop and evaluate *six practical attacks* on actual devices. With our over-the-air (OTA) attacks, an attacker in Bluetooth range with a victim can impersonate a tracker to an app, an app to a tracker, man-in-the-middle (MitM) them, and eavesdrop on their communication. Alternatively, with our remote attacks, indicated in the paper as software-based (SB), an attacker can remotely eavesdrop on data from a tracker or impersonate an app by abusing Android BLE API within a malicious app. Our attacks are high impact because they affect the whole Xiaomi ecosystem and enable the attacker to achieve valuable goals such as eavesdropping on sensitive data exchanged by a tracker and a smartphone (e.g., health data, SMS, and notifications) or sending arbitrary commands to the tracker and the smartphone.

We developed `breakmi`, a security evaluation toolkit for fitness tracker ecosystems to automate our RE efforts and attacks. `breakmi` has three modules: protocol dissector, security mechanisms, and attacks. The protocol dissector module understands Xiaomi's proprietary application-layer protocols, allowing for fast and automated analysis of its application-layer packets. The security mechanisms module reimplements Xiaomi's custom security mechanisms, such as Pairing and Authentication. The attacks module deploys our attacks automatically, including over-the-air or remote impersonation and MitM on arbitrary trackers and apps. We will release `breakmi` in the open after responsible disclosure.

To show the effectiveness of our attacks, we present an *extensive evaluation* of Xiaomi trackers and apps. In particular, we successfully attacked all Xiaomi trackers released since 2016 (i.e., Mi Band 2/3/4/5/6 and Amazfit Cor 2) and the latest versions of the Xiaomi fitness mobile apps (i.e., Mi Fit version 4.8.1 and Zepp version 5.9.2). Two of the presented attacks are remotely targeting the Android platform. We successfully conduct them on six popular Android versions (i.e., Android 6/8/9/10/11/12) to confirm their widespread impact. According to [84], these versions represent 90% of the Android ecosystem.

To effectively address the presented vulnerabilities and attacks, we propose *five* countermeasures. All of them incur minimal overhead because they rely on a few additional messages and on lightweight security features. We redesign Xiaomi proprietary protocols to implement four out of the *five* proposed countermeasures at the application-layer. The fifth countermeasure applies to the link-layer, where we encourage the activation of the standard BLE link-layer security (i.e., BLE legacy pairing, LE Secure Connections), already supported by all Xiaomi devices.

To check the effectiveness of our attacks and the extensibility of `breakmi` to other vendors, we also analyzed the *Fitbit ecosystem* (the second biggest ecosystem after Xiaomi). We looked at two popular Fitbit trackers (i.e., Charge 2 and Charge 4) and the Fitbit mobile app (v 3.54.1). Our results show that the Fitbit ecosystem provides better (still proprietary) security mechanisms than Xiaomi. However, it is *still vulnerable* to five out of the six presented attacks. While testing our attacks on Fitbit, we extended `breakmi` by adding Fitbit's custom protocols and security mechanisms.

To encourage further research on the topic, we describe our reverse-engineer *methodology* in detail. Specifically, we used a mix of static and dynamic techniques for reconnaissance, traffic

analysis, and app analysis. Additionally, we developed custom scripts and tools to automate our analyses that are now part of `breakmi`. Overall we spent a considerable time reversing the Xiaomi ecosystem (i.e., one year RE effort).

We summarize our contributions as follows:

- We reverse-engineer the proprietary security protocols used by Xiaomi to protect the BLE link between its trackers and companion apps. Those protocols include Pairing, Authentication, and Communication at the application-layer, and do not take advantage of BLE link-layer security mechanisms already supported by its devices.

- We uncover novel and severe vulnerabilities in the specification of those protocols enabling an attacker to target the ecosystem as a whole. The list of vulnerabilities includes unilateral and replayable authentication, improper key agreement, and lack of encryption and integrity protection of sensitive data.

- We show how to exploit these vulnerabilities, and we perform high-impact Xiaomi-compliant attacks either over-the-air or remotely (via a malicious app). We design and release `breakmi`, a toolkit to automatically analyze and attack the Xiaomi ecosystem. We address the presented vulnerabilities and attacks by proposing five practical and low overhead countermeasures that fix Xiaomi's vulnerable protocols.

- We compare Xiaomi with Fitbit, and we find that four of the identified vulnerabilities and five of the proposed attacks are portable to Fitbit. We extend `breakmi` to the Fitbit ecosystem, and we successfully conduct the attacks.

**Responsible disclosure** We responsibly disclosed our findings to Xiaomi in March 2021 via the HackerOne platform. Xiaomi considered our report as a single and known vulnerability, namely "lack of encryption," scheduled to be fixed on an undisclosed timeline. We disagree with this response as we reported multiple classes of vulnerabilities leading to several attacks (and not a single vulnerability)[1]. We also *responsibly disclosed* our findings to Fitbit in January 2022 through Google Vulnerability Reward Program, and Fitbit acknowledged our attacks and will deploy a fix in April 2022.

## 3.2 Background

In this section, we introduce Bluetooth Low Energy and what is known about the Xiaomi fitness tracking ecosystem.

### 3.2.1 Bluetooth Low Energy (BLE)

Bluetooth Low Energy is the de-facto standard wireless technology for low-power wireless services, including fitness tracking. It is defined in the Bluetooth standard [85] and provides a client-server architecture to exchange data using a specific format. Security-wise, BLE includes

[1]Our experiments did *not* involve or expose third-party users, but we analyzed our own devices in a controlled environment.

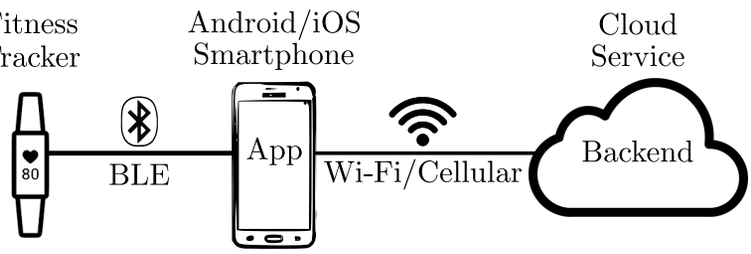



Figure 3.1: Xiaomi fitness tracking system architecture. The tracker is a battery-powered embedded device supporting BLE. The smartphone runs a fitness tracking application and is capable of communicating with the tracker via BLE and with a backend server via the Internet.

*pairing* and *session establishment* mechanisms that should provide confidentiality, integrity, and authenticity guarantees at the link-layer.

The BLE client-server architecture is specified by the Generic ATTribute Profile (GATT) and uses the ATTribute Protocol (ATT) protocol [85, p. 1531]. In Bluetooth terminology, the client is defined as the *central*, and the server as the *peripheral*. The client sends read, write, and notification requests to the server. The server answers accordingly to the request type and the availability of data.

In a fitness tracking use case, the GATT server is the tracker device (e.g., wristband) and the GATT client is a smartphone application. The server exposes fitness data, such as heart rate and step count, while the client can periodically query such data. BLE data is exchanged using a hierarchical and object-oriented format defined in [85, p. 284].

The top-level of the hierarchy is a profile, and it contains a set of services. Each service provides characteristics or other services. A characteristic provides a value field with optional fields, such as descriptors and properties. Each characteristic can be configured with access-control flags (e.g., read-only, write-only, or read-write).

BLE provides pairing and secure session establishment protocols to secure the link-layer. Before exchanging data over GATT, the client and the server can pair to agree upon a long-term pairing key and use it to establish a secure session (e.g., by using ECDH).

On the contrary, session establishment is implemented using AES-CCM authenticated-encryption, keyed with a fresh session key (derived from a pairing key). The server can protect a GATT characteristic by requiring a client to pair before accessing it by setting its encryption and authentication security permissions.

A BLE device supporting a Bluetooth version greater than or equal to 4.2 can support a security mode known as *Secure Connections (SC)* that enhances pairing and session establishment by only using FIPS-compliant algorithms.

### 3.2.2 Xiaomi Fitness Tracking Ecosystem

The Xiaomi fitness tracking ecosystem includes wearable tracking devices, smartphones running a tracker companion app, and backend infrastructure. As we see from Figure 3.1, the Xiaomi components communicate *wirelessly* using a combination of short-range and long-range technologies. The tracker and the smartphone use *BLE* (introduced in Section 3.2.1), while the smartphone and the backend require Internet connectivity through Wi-Fi or a cellular network. We note that other fitness tracker vendors, including Fitbit, employ the same general

architecture.

Xiaomi trackers are wearable and battery-powered devices composed of sensors to collect health data, such as step count and heart rate, and actuators, such as buttons and a touch screen. They can also control the associated smartphone (e.g., lock/unlock the screen) and receive notifications (e.g., SMS, WhatsApp).

Xiaomi ships two families of trackers called *Mi Band (MB)* [86] and *Amazfit* [87]. Mi Band is the most popular family and so far includes *six* generations: MB 1 (2014), MB 2 (2016), MB 3 (2018), MB 4 (2019), MB 5 (2020), and MB 6 (2021). Amazfit has two generations: Cor 1 (2018) and Cor 2 (2019). The two families are manufactured by the same company (Huami [88]) and have similar hardware and software capabilities. For example, the Cor 1 is a MB 2 clone, and the Cor 2 clones the MB 3.

Xiaomi's official companion app is *Mi Fit* and is freely available for Android [89] and iOS [90]. The app provides a user interface to configure and manage a tracker and is compatible with all Mi Band and Amazfit Cor generations. Another official app that supports Xiaomi trackers is *Zepp*, available on Android [91] and iOS [92]. There are also several third-party apps compatible with Xiaomi.

The Xiaomi backend is an Internet-accessible infrastructure that manages several aspects of the ecosystem. It stores the list of registered users and their associated trackers. It also backups the configurations of the trackers and the apps. Additionally, it distributes firmware and resource file (e.g., fonts, images) updates to the trackers. The backend is managed by Huami, which is also the developer of the Mi Fit and Zepp applications (and the tracker's manufacturer).

On one hand, Xiaomi does not provide any information about its security architecture and mechanisms. However, on the other hand, it claims to guarantee its users' confidentiality, security, and privacy (see the Privacy Policy [5] dated May 2020). Hence our work investigates how these claims are actually implemented in practice.

## 3.3 Analysis of Xiaomi Fitness Tracking

From our reverse-engineering experiments, we found that Xiaomi uses three *closed-source* and *proprietary* application-layer protocols in three different operations: *Pairing*, *Authentication*, and *Communication*.

These three protocols define the format, and the purpose of any BLE packet exchanged between Xiaomi companion apps and trackers. A single vulnerability in the protocols can be used to exploit *any* supported Xiaomi tracker and app. Hence they must be well designed and implemented, but this work (experimentally) shows the contrary. We note that Xiaomi protocols are orthogonal to the link-layer security mechanisms provided by BLE introduced in Section 3.2.1.

In our experiments, we also uncover that Xiaomi *disables* BLE link-layer security and privacy features despite being supported by its trackers and apps. BLE security mechanisms were designed to protect the emerging IoT market, including the fitness tracking industry, and it is not evident why Xiaomi simply ignores them. This choice considerably increases the risk of a security breach as Xiaomi only trusts its vendor-specific protocols.

Now we describe these protocols in detail. The presented information required extensive reverse-engineering (RE) efforts described in Section 3.9. Then, we pinpoint vulnerabilities in

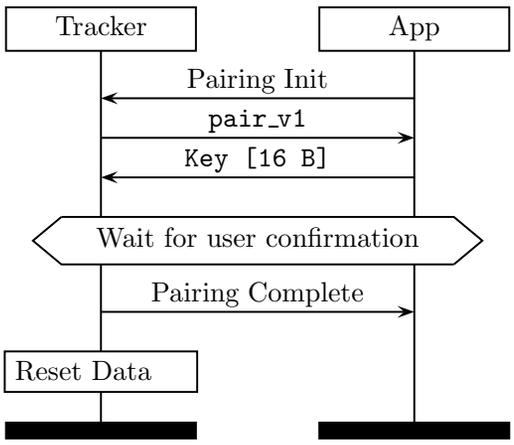



Figure 3.2: Xiaomi Pairing v1. The app generates and sends to the tracker a 16-byte pairing key (`Key`) in the clear. The tracker shows a pairing confirmation message to the user. Once the user confirms, the tracker resets its data, and pairing is completed.

their specification, including lack of mutual authentication and replay protection. The vulnerabilities are critical as they affect the whole Xiaomi ecosystem regardless of the hardware and software details of the trackers and the apps.

### 3.3.1 Reverse-Engineered Protocols

We isolate three Xiaomi proprietary application-layer protocols, and we name them Pairing, Authentication, and Communication. Pairing is used to establish a long-term secret (i.e., pairing key) between a tracker and an app. Authentication is employed to prove ownership of a pairing key. Communication runs only after a successful Authentication and enables interaction between trackers and an app.

Those interactions include sending health data from the tracker and sending commands or notifications from the app to the tracker or vice versa. Now we describe their technical details. We use the terms *tracker* and *app* to refer to any Xiaomi-compliant device, and, when needed, we indicate the specific tracker model or app name.

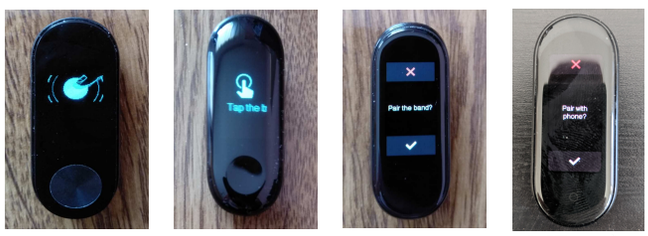
#### Pairing

Pairing is used to establish a 16-byte pairing key between the tracker and the app. The pairing key is the *root of trust* between the devices and must be kept secret and stored securely. We observed two versions of Pairing.

The first version, which we call Pairing *v1*, is used by Mi Band 2/3 and Amazfit Cor 1/2 trackers. Instead, Pairing *v2* is employed by Mi Band 4/5/6 and is server-based as it involves Xiaomi backend. Mi Fit and Zepp apps support both pairing versions. Now we describe the technical details of Pairing v1/v2.

Pairing v1 is supported by MB 2/3 and works shown in Figure 3.2. The app sends a pairing initialization message (`Pairing Init`). The tracker responds with a Pairing version message (`pair_v1`). The app generates and sends a 16-byte pairing key (`Key`) to the tracker *in the clear*. Then, the tracker shows the user a pairing confirmation message (see Mi Band 2/3 in Figure 3.3) and waits for user confirmation (together with the app). Once the user confirms

pairing, the tracker sends a success message (`Pairing Complete`) to the app and resets its stored data, completing Pairing v1.




MB 2 MB 3 MB 4 MB 5/6

Figure 3.3: Mi Bands pairing confirmation messages. To accept pairing, a user must either press a hardware button (Mi Band 2/3) or touch a software button (Mi Band 4/5/6). Note that Amazfit Cor 1/2 use similar pairing confirmation messages.

Xiaomi introduced Pairing v2 in 2019, and it is supported by MB 4/5/6. The protocol involves interactions with the Xiaomi *backend* using HTTPS. The protocol works as depicted in Figure 3.4. The app sends a Pairing Init message, and the tracker replies with a `pair_v2` and a truncated digest of its public key (SHA1(`pub_k`)). As we observed the same digest on all trackers that we tested (i.e., `1863c2cce5d159413bed92c4b163c279`), we are confident that all trackers are using the *same* public key.

Then, the app sends a random number request, and the tracker answers with `R`, a 16-byte random number. `R` and the tracker public Bluetooth address (`TR_A`) are inputs to a *custom* key derivation function (`kdf`) that generates the pairing key (`Key`). As such, `R` is the *pairing key seed* and is sent in the clear. We reverse-engineered `kdf` and discovered that it computes a SHA256 of the concatenation of `TR_A` and `R` and outputs `Key`, the leading 16 bytes of the digest. The function is expressed as:

$$\texttt{Key} = \texttt{kdf}(\texttt{TR\_A}, \texttt{R}) = \text{SHA256}(\texttt{TR\_A} \| \texttt{R})[0:16].$$

Next, the app and the backend establish a TLS session, and the app sends SHA1(`pub_k`) and the `Key` encoded as base64 to the backend. The backend computes a signature (`Sig`) of `Key` using its private key (`pri_k`) and sends the base64-encoded signature to the app (B64(`Sig`)). The app provides the signature to the tracker that verifies it and sends back an acceptance message (Valid `Sig`). Then, pairing completes identically to Pairing v1, with the user having to accept a pairing confirmation message (see Mi Band 4/5/6 in Figure 3.3).

### Authentication

Authentication has only one version and works as depicted in Figure 3.5. The app sends an authentication request message (`Auth Req`). The tracker answers with a 16-byte challenge (`Chal`). The app computes a 16-byte response (`Resp`) by encrypting `Chal` with `Key` using AES in ECB mode and sends it to the tracker. The tracker computes its own response, checks it against `Resp`, and sends a positive authentication message (`Auth OK`) if `Resp` is verified. As a result, the tracker authenticates that the app owns the correct pairing key and unlocks access to its private data (e.g., step count). However, the tracker *never* authenticates to the app, and the authentication messages are sent in the clear.

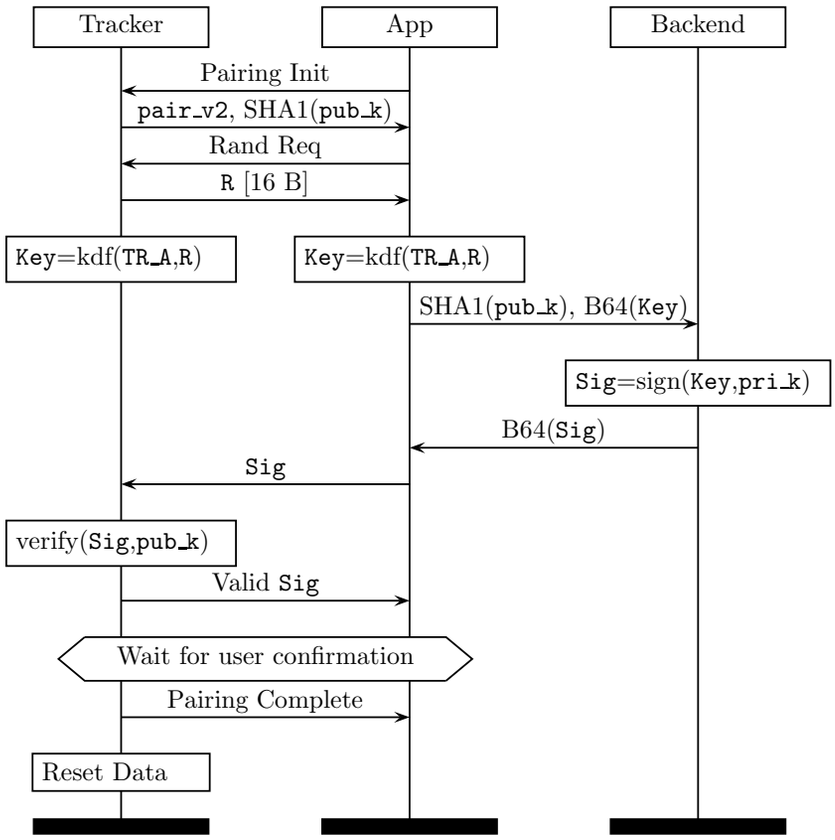



Figure 3.4: Xiaomi Pairing v2. The app starts the protocol by sending a Pairing Init message. The tracker sends back `pair_v2` and a public key digest (SHA1(`pub_k`)). Then the app requests a random number, and the tracker replies with `R`, which is 16 bytes long. Both devices run a custom key derivation function (`kdf`) to compute `Key` from `R` (key seed) and the Bluetooth address of the tracker (`TR_A`). Then, the app sends SHA1(`pub_k`) and `Key` base64-encoded to the backend. The backend computes a signature (`Sig`) of `Key` with its private key and sends `Sig` base64-encoded to the app. The app presents `Sig` to the tracker, which verifies it and sends back a confirmation message. Then, the tracker shows the user a pairing confirmation message, and if the user accepts, pairing is completed, and the tracker resets its data.

### Communication

Once a tracker and an app complete Pairing and Authentication, they run the Communication protocol to exchange data, commands, and notifications. We discovered that Communication *is neither encrypted nor integrity protected* despite the tracker and the app sharing a pairing key. This finding is surprising, as Xiaomi trackers and apps do support encryption primitives and crypto hardware acceleration. Moreover, prior authoritative reports, such as the ones from Mozilla [93, 94, 95], state that Xiaomi uses encryption (and meets Mozilla's minimum security standards) when this is not the case.

Communication is implemented on top of BLE GATT (introduced in Section 3.2.1). The tracker is the GATT server, and the app is the GATT client. The server has a set of public services (e.g., Generic Access) and characteristics (e.g., Device Name). Each characteristic has access control bits to set reading and writing permissions.

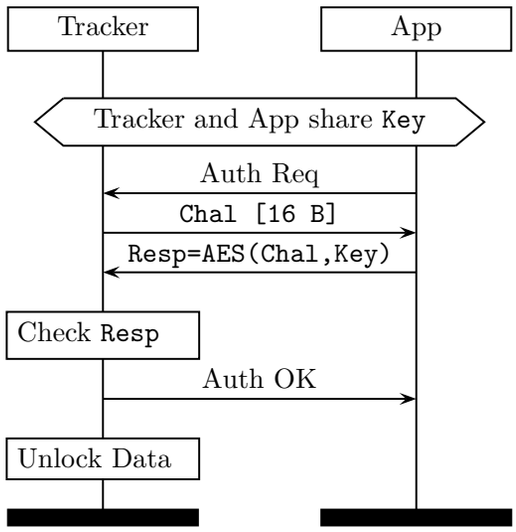



Figure 3.5: Xiaomi Authentication. Using a challenge-response procedure, the tracker *unilaterally* authenticates that the app owns the shared pairing key (`Key`).

On top of GATT, Xiaomi uses a custom data locking mechanism where a tracker GATT characteristic cannot be accessed until the app has authenticated to the tracker (via a successful run of Authentication). Two examples of locked characteristics are heart rate and step count.

### 3.3.2 Protocol-level Vulnerabilities

We analyzed the Pairing, Authentication, and Communication protocols (described in Section 3.3.1), and we identified thirteen severe vulnerabilities in their *specifications*. Most of them, such as unilateral/replayable authentication and lack of encryption and integrity protection, were *publicly unknown*. The issues affect *all* trackers released since 2016, even the newest releases (i.e., MB 6) and the most recent app versions. We now describe the vulnerabilities grouped by protocol.

#### Pairing v1 (MB 2/3, AC 1/2)

- *Pairing key sent in the clear.* The app sends the pairing key to the tracker in the clear and with no integrity protection (see `Key` in Figure 3.2).

- *Pairing not authenticated.* The devices do not authenticate each other during Pairing. Hence the app and the tracker cannot determine if they are pairing with a legitimate device.

- *Pairing key generated by the app.* The pairing key is generated and distributed by the app, despite the tracker being able to run a key agreement protocol (e.g., ECDH).

- *Weak user confirmation.* As shown in Figure 3.3, pairing confirmation is weak as the user only has to interact with the tracker, and the tracker does not show any contextual information.

### Pairing v2 (MB 4/5/6)

- *Pairing key seed sent in the clear.* The tracker sends the pairing key seed to the app in the clear and with no integrity protection (see `R` in Figure 3.4). This issue is as bad as in Pairing v1, as the pairing key is deterministically computed from its seed and its publicly available information (i.e., the tracker's Bluetooth address).

- *Pairing only (weakly) authenticates the app.* Pairing v2 does not authenticate the tracker to the app but authenticates the app to the tracker. In particular, the tracker must receive a correct signature from the app. The signature algorithm is deterministic, and both inputs are public, so an attacker could reverse the algorithm and obtain the signature.

- *Pairing version can be downgraded/upgraded.* The tracker decides the version of Pairing by sending either `pair_v1` or `pair_v2`. As this message is not integrity protected, it can be manipulated to force a specific pairing version (e.g., downgrade from v2 to v1 or upgrade from v1 to v2).

- *Pairing key generated by the tracker.* The pairing key only depends on the tracker, despite the app being able to run a key agreement protocol (e.g., ECDH). This issue is worse than the one highlighted for Pairing v1 key generation. The tracker is a computationally-constrained device and is more likely to generate a low-entropy key than an app running on a smartphone.

- *Default keypair.* The hash of the public key is the same for all MB 4/5/6 that we tested, and this entails that all our trackers share the same public key. Our device sample is limited, but we found the same key even across devices bought in different countries. This implementation is risky because an attacker can compromise the whole ecosystem by leaking the default private key.

- *Weak user confirmation.* As shown in Figure 3.3, pairing confirmation is weak for the same reasons as Pairing v1.

### Authentication

- *Unilateral app authentication.* The protocol unilaterally authenticates the app (see `Resp` in Figure 3.5), but it does not require to authenticate the tracker. Indeed, there is no way for an app to check if it is connected with a legitimate tracker.

- *Replayable authentication.* The protocol is vulnerable to replay attacks as, given a fixed challenge, there is no way to generate different responses (i.e., there is no nonce). Hence a fitness tracker cannot be certain that a valid response comes from a trusted app.

### Communication

- *No encryption.* Despite sharing a pairing key and supporting encryption algorithms, the tracker and the app do not encrypt their sessions. Hence sensitive data exchanged over BLE can be effortlessly obtained.

- *No integrity protection.* Despite sharing a pairing key and supporting Message Authentication Codes (MAC), the tracker and the app do not integrity-protect their communication. As a result, sensitive data can be manipulated at will.

## 3.4 Proposed Attacks

We now describe six attacks to demonstrate the severity of the issues presented in Section 3.3.2. As the attacks exploit *architectural* vulnerabilities in the Xiaomi protocols, they are effective on all devices employing those protocols. Developing the attacks required extensive RE efforts as we target proprietary and unknown protocols (unlike BLE pairing and session establishment). We discuss four over-the-air attacks and two software-based attacks.

Our OTA tracker impersonation, OTA app impersonation, and OTA MitM require proximity with the target, as they exploit BLE traffic and minimal equipment.

Our SB app impersonation and SB eavesdropping are *remote* and require the installation of a malicious app on the victim's phone. Despite only asking for Internet and Bluetooth normal permissions and requiring *no root access*, this app can abuse the Android BLE API to interfere with BLE traffic. Next, we present our threat model, describe the attacks, discuss how each attack maps to the vulnerabilities presented earlier and their impact.

### 3.4.1 System Model

Our system model has the same architecture presented in Section 3.2.2 and depicted in Figure 3.1. There are three entities: a tracker, a companion app, and a backend. The tracker and the app communicate over BLE, and the app communicates with the backend via Wi-Fi or a cellular network. The entities use the strongest security mechanisms at their disposal.

For example, the communication between the tracker and the app is protected using Xiaomi Pairing, Authentication, and Communication protocols that we reverse-engineered and described in Section 3.3.1. Similarly, the app and the backend use TLS. Such mechanisms should protect against passive and active attacks, including eavesdropping, device impersonation, and man-in-the-middle (MitM) attacks.

Our victim is a user of the Xiaomi ecosystem. She might use any supported trackers, such as Mi Band (MB) 2/3/4/5/6 and Amazfit Cor (AC) 1/2, and any version of the Mi Fit and Zepp companion apps. We assume that the victim has installed the app, registered an account with the Xiaomi backend, and paired her tracker with the smartphone app. Hence the user can establish authenticated sessions between the tracker and the app.

### 3.4.2 Attacker Model

Our attacker targets Xiaomi Pairing, Authentication, and Communication protocols as the vulnerabilities in these protocols can be exploited regardless of the hardware and software details of the target tracker and app. In other words, she is looking for *Xiaomi-compliant* vulnerabilities.

The attacker only knows public information advertised by the tracker over BLE (e.g., the public BLE address of the tracker), and she has no physical access to the target devices. Hence

the attacker does not know any pre-shared secret between the victims (e.g., pairing keys) and cannot tamper with the devices' operating system and firmware.

The attacker has *four* goals: (i) she aims at impersonating the tracker to the app and (ii) the app to the tracker; (iii) she wants to establish a MitM position between the tracker and the app; (iv) she desires to eavesdrop the data exchanged between the tracker and the app.

The attacker can use *over-the-air (OTA)* or *software-based (SB)* attacks. Our attacker model is based on the Android threat model proposed by Mayrhofer et al. [96]. This threat model labels our OTA attacks as both *Proximal Access* and *Network-level* threats. In particular, our OTA attacks involve *T.P1 - Devices in physical proximity, but not under direct control, of an attacker who can control radio communication channels, including BLE)*, *T.N1 - Passive eavesdropping and traffic analysis*, and *T.N2 - Active manipulation of network traffic.* The attacker can sniff BLE traffic, jam the BLE spectrum, craft, and send custom BLE packets to the app and the tracker.

The same threat model labels our SB attacks as *Application Code* threats. In particular, our SB attacks involve *T.A1 - Abusing APIs supported by the OS.* The attacker can remotely attack the victim through a malicious app already installed on the victim's smartphone, a common requirement for most Android malware [97, 98, 99]. This requirement is reasonable as users often install unwanted apps on their smartphones fairly often. Kotzias et al. [100] estimated that 67% of unwanted apps are directly installed from the Google Play Store or alternative markets (10.4%). When launched, our malicious app stealthily abuses Android BLE API. It only requires normal permissions related to the Internet and Bluetooth and does not need root privileges.

### 3.4.3 OTA Tracker Impersonation Attack

The attacker can wirelessly impersonate any tracker by presenting itself to a victim app as a spoofed tracker, running the unilateral Authentication protocol without having to authenticate, and then starting a Communication session that is neither encrypted nor integrity protected. The attack does not require knowledge of the pairing key, does not trigger Pairing (which requires user interaction), and can be launched anytime a target app is in BLE range with the attacker. The attack leverages Xiaomi's unilateral authentication and the lack of encryption and integrity protection of Communication.

The technical details of the attacks are presented in Figure 3.6. The attacker advertises her presence as the impersonated tracker by copying its features, including its Bluetooth address and GATT server. The victim (App) recognizes the attacker as trusted and sends her an authentication request (`Auth Req`). The attacker answers with a random challenge (`Chal`), and the app computes and sends back a response (`Resp`) derived from a pairing key unknown to the attacker. The attacker ignores `Resp`, answers with a positive authentication message (`Auth OK`), and unlocks her own GATT server. Afterwards, during Communication, the app considers the impersonated tracker as trusted.

### 3.4.4 OTA App Impersonation and MitM Attacks

The attacker can impersonate any app over-the-air or MitM an app and a tracker using a replay attack on the non replay-protected Authentication protocol and can start an insecure Communication session.

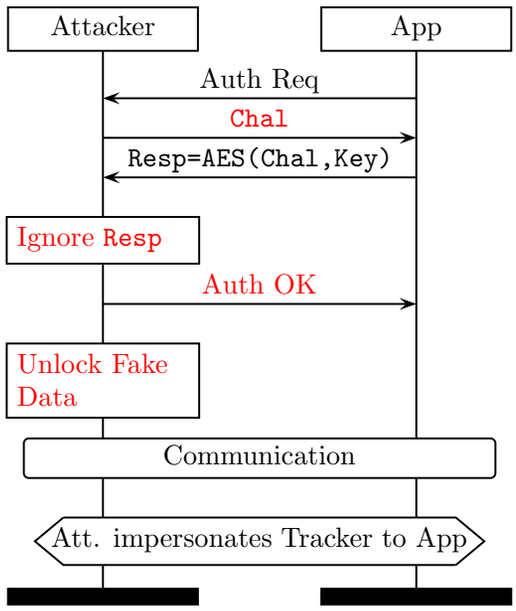



Figure 3.6: OTA tracker impersonation attack. The attacker impersonates a tracker by spoofing the tracker's BLE address and BLE advertisement packets. The victim (Mi Fit or Zepp app), which is already paired with the impersonated tracker, recognizes the attacker as trusted and sends her an authentication request (Auth Req). The attacker's device answers with a random challenge (`Chal`). The app solves it and sends back a response (`Resp`) computed from the pairing key unknown to the attacker. The attacker ignores `Resp`, answers with a positive authentication message (`Auth OK`), and unlocks its fake GATT server. As a result, the app starts the Communication protocol with the impersonated tracker, believing it is trusted.

In particular, the attacker can start Authentication in parallel with the app and the tracker. When the tracker sends a challenge, the attacker replays it to the app, relays the app response to the tracker, and successfully authenticates to the tracker without knowing the pairing key. Then, the attacker can either impersonate the app by dropping her connection with the legitimate app and starting a Communication session with the tracker or MitM the Communication session between the app and the tracker.

The attacks do not require knowledge of the pairing key and do not trigger Pairing. Unlike the OTA tracker impersonation attack, these ones require both the app and the tracker to be in BLE range with the attacker. This issue is not that significant as the tracker and the app are typically carried and used by the same person.

The technical details of the OTA app impersonation and MitM attacks are presented in Figure 3.7. The attacker advertises her presence as a trusted tracker, and the app sends an `Auth Req` message to the attacker to start Authentication. The attacker sends a parallel `Auth Req` message to the tracker to initiate Authentication with the tracker. The tracker sends a `Chal` to the attacker, who relays it to the app. Then, the app computes `Resp` and sends it to the attacker, who replays it to the tracker to prove ownership of a pairing key that she does not know. The tracker sends an `Auth OK` message to the attacker, and the attacker sends an `Auth FAIL` message to the app if she wants to impersonate it or sends an `Auth OK` message to preserve her MitM position.

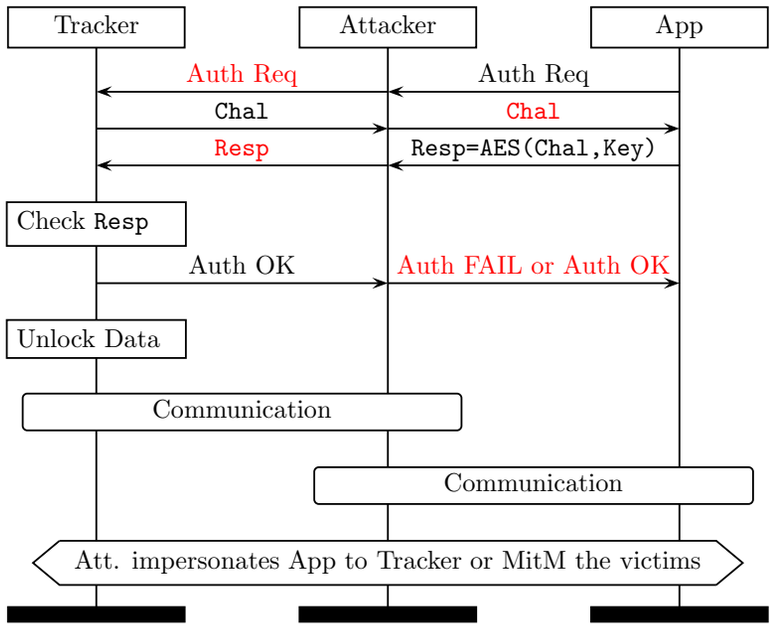



Figure 3.7: OTA app impersonation and MitM attacks. The attacker impersonates the app and sends an authentication request (`Auth Req`) to the tracker (MB 2/3/4/5/6 and AC 1/2) as the app. The tracker sends an authentication challenge (`Chal`), and the attacker relays it to the legitimate app. Then, the app computes a response (`Resp`) and sends it to the attacker, believing that it is talking to the victim tracker. The response is computed from a pairing key (`Key`) only known to the victims. The attacker relays `Resp` to the tracker, which checks it and sends back a positive authentication message (`Auth OK`). Then, the attacker has two options. She can impersonate the app by sending the app a negative authentication message (`Auth FAIL`) and taking over the communication session. Otherwise, she can relay the positive authentication message (`Auth OK`) to the app and establish a man-in-the-middle position between the victim app and tracker.

### 3.4.5 SB App Impersonation Attack

The attacker can impersonate any Xiaomi Android app and remotely pair with a victim tracker. The attack works *regardless of* the pairing's protocol version. Similarl to other Android malware threat models [96], the malicious app is already installed on the victim's smartphone. This app acts stealthily, runs with normal `BLUETOOTH` and `INTERNET` permissions, and does not require root access. The main advantage of this attack over the OTA counterpart is that it can be conducted *remotely* (i.e., over the Internet). Its main drawback is that it requires *user interaction* to trigger a new pairing session. However, since pairing confirmation involves interactions only with the trackers, the user has no way to tell if the tracker is pairing with a malicious or legitimate app.

The SB app impersonation on Android is depicted in the right part of Figure 3.8 and works as follows. The attacker abuses the Android BLE API to discover a tracker paired with the Xiaomi app. This task can be done by finding Xiaomi proprietary commands in the smartphone GATT traffic or by looking for the BLE address of the tracker in the list of connected devices.

If the target tracker supports Pairing v1, the attacker starts a pairing protocol with the tracker. The tracker cannot tell if the pairing request is authentic and completes pairing

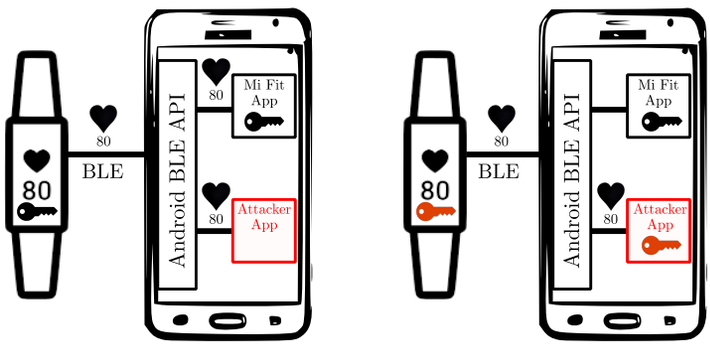



Figure 3.8: SB eavesdropping (left) and app impersonation (right) attacks. On the left, the attacker eavesdrops on all data exchanged by the tracker and the app by querying the tracker's BLE GATT server without knowing the pairing key (black key). On the right, the attacker impersonates the app by re-pairing with the tracker and establishing a new pairing key (red key) unknown to the impersonated app. SB eavesdropping and app impersonation on Pairing *v1* require only `BLUETOOTH` and `INTERNET` Android permissions. SB app impersonation on Pairing *v2* also requires `BLUETOOTH_ADMIN` and `ACCESS_FINE_LOCATION`.

with the attacker's malicious app. Otherwise, if the tracker supports Pairing v2 (i.e., server-based), the attacker must use a different strategy. In particular, the attacker sends a *factory reset* proprietary command (that we reverse-engineered). The command does not require prior authentication. It deletes the bond between the tracker and the legitimate app, changes the tracker's BLE address, and puts it in pairing mode. Then, the attacker can find the tracker and complete the pairing. This attack strategy completely defeats server-based pairing, which still lacks strong device authentication. Finally, regardless of the attacked pairing version, the legitimate app *cannot* connect back to the tracker (e.g., in Figure 3.8, the tracker accepts the red key, but the app has the black one).

### 3.4.6 OTA and SB Eavesdropping Attacks

Due to the lack of encryption during Pairing, Authentication, and Communication, the attacker can effortlessly launch OTA and SB eavesdropping attacks.

In the OTA case, the attacker can sniff sensitive data exchanged between tracker and app. For example, during Pairing v1 she can sniff the pairing key, during Pairing v2 the pairing key seed, during Authentication the challenge-response pairs (to be used in an offline brute-force attack), and during Communication all the data sent including health parameters from the tracker and notifications from the app.

In the SB case, the attacker can sniff sensitive data from remote trackers via a malicious app, taking advantage of the lack of encryption and a known issue with the Android BLE API. Android allows applications with `BLUETOOTH` permissions to sniff all GATT data received by a smartphone. Thus, any application co-located with the Mi Fit app can sniff all the traffic coming from a tracker without pairing or authenticating with it. For example, in Figure 3.8, we depict a malicious app (Attacker App in red) sniffing the heart rate value sent by the tracker by abusing Android BLE API.

### 3.4.7 Discussion

**Mapping between attacks and vulnerabilities** In Table 3.1, we present a mapping between our attacks and the vulnerabilities presented in Section 3.3.2. The Pairing v1/v2 protocols can be targeted by OTA eavesdropping, SB eavesdropping, and SB app impersonation. OTA and SB eavesdropping exploit the data sent in the clear during the Pairing protocol (pairing key in Pairing v1 and pairing key seed in Pairing v2). SB app impersonation exploits the missing/weak authentication and weak user confirmation to connect a malicious device to a legitimate tracker. The Authentication protocol can be targeted by OTA tracker impersonation, OTA app impersonation, and OTA man-in-the-middle. OTA tracker impersonation exploits unilateral app authentication (the tracker does not need to authenticate with the app). The OTA app impersonation exploits the replayability of BLE traffic between app and tracker, allowing any device to mimic a legitimate tracker. The OTA man-in-the-middle is a combination of the OTA app and tracker impersonations. The Communication protocol can be targeted by OTA tracker impersonation, OTA app impersonation, OTA man-in-the-middle, OTA eavesdropping, and SB eavesdropping. The lack of encryption and integrity protection in any BLE packet exchanged between app and tracker allows an attacker to eavesdrop and manipulate BLE traffic freely.

**Attacks' Impact** The attacks' impact is high for several reasons. The proposed attacks, and their root causes (that we RE), were not known by the community. Also, our attack techniques include novel aspects. For example, the SB remote attacks combine unknown Xiaomi vulnerabilities with known Android issues.

The proposed attacks disprove the security and privacy claims made by Xiaomi in their Privacy Policy [5], as we highlight in Section 6.7. We demonstrate that all Xiaomi trackers released since 2016 are vulnerable to the proposed attacks, and future releases will be vulnerable as well, as our attacks are Xiaomi-compliant. Our attacks affect millions of users, as Xiaomi is the world's leading fitness tracker manufacturer.

The proposed attacks are cheap and low-effort (e.g., only require commercial-off-the-shelf products and minimal equipment) and are easy to deploy. An attacker can violate users' privacy by leaking and manipulating sensitive data (e.g., health records and 2FA SMS) and enforcing malicious factory reset and firmware update requests.

## 3.5 Implementation

In this section, we present the implementation of `breakmi`, a toolkit that we developed to reverse-engineer and attack Xiaomi's proprietary Pairing, Authentication, and Communication protocols. `breakmi` reimplements these protocols and automates our experiments and attacks. The toolkit contains a protocol dissector module, a security mechanisms module, and an attacks module. We will release `breakmi` as open-source, and now we describe how we implemented each module.

### 3.5.1 Protocol Dissector Module

Our toolkit, `breakmi`, includes a protocol dissection module capable of speaking Pairing v1/v2, Authentication, and Communication protocols (presented in Section 3.3). The dissectors mod-

| Vulnerabilities | OTA Attacks | | | SB Attacks | | |
|---|---|---|---|---|---|---|
| | Tracker Imp. | App Imp. | MitM | Eaves. | Eaves. | App Imp. |
| **Pairing v1** | | | | | | |
| Pairing key sent in the clear | - | - | - | ✓ | - | - |
| Pairing not authenticated | - | - | - | - | - | ✓ |
| Weak user confirmation | - | - | - | - | - | ✓ |
| **Pairing v2** | | | | | | |
| Pairing key seed sent in the clear | - | - | - | ✓ | ✓ | - |
| Pairing only (weakly) authenticates app | - | - | - | - | - | ✓ |
| Weak user confirmation | - | - | - | - | - | ✓ |
| **Authentication** | | | | | | |
| Unilateral app authentication | ✓ | - | ✓ | - | - | - |
| Challenges and responses replayable | - | ✓ | ✓ | - | - | - |
| **Communication** | | | | | | |
| No encryption | ✓ | ✓ | ✓ | ✓ | ✓ | - |
| No integrity protection | ✓ | ✓ | ✓ | - | - | - |

Table 3.1: Mapping between the exploited vulnerabilities identified in Section 3.3.2 and the OTA and SB attacks presented in Section 5.4. A checkmark (✓) indicates that a vulnerability is exploited to conduct an attack.

ule can detect and craft any Xiaomi proprietary message given a capture. We develop the dissectors as an aid for RE. We implement the module defining fifteen custom dissection classes for scapy [101], an interactive packet manipulation program. Each class encodes a message type using a specific binary layout. Table 3.2 lists all messages that we can dissect and customize. The table's first column indicates the message type, the second column the message sender, and the third column the packet layout. For example, the Pairing Key message is used by an app during Pairing v1 to send the pairing key and the packet contains a leading `0100` and then `Key`.

Pairing v1/v2 messages are managed by Xiaomi's custom Auth GATT characteristic ( 00:00:00:09:00:00:35:12:21:18:00:09:AF:10:07:00), which can be found under Xiaomi's custom GATT service `0xFEE1`. The dissectors extract capture packets with Pyshark [102], a Python API for Wireshark [103], and label them as v1 or v2. For Pairing v2, they also look for the `Signature` transmitted on Xiaomi's custom Chunked Transfer characteristic (00:00:00:20:00:00: 35:12:21:18:00:09:AF:10:07:00), under Xiaomi's `0xfee1` service.

| Message | Sender | Opcode/Value |
|---|---|---|
| Pairing Init | App | 0100 |
| `pair_v1` | Tracker | 100104 |
| Pairing Key | App | 0100, `Key` |
| Pairing Complete | Tracker | 100101 |
| Pairing Fail | Tracker | 100204 |
| `pair_v2` | Tracker | 10018101 |
| SHA1(`pub_k`) | Tracker | `Const` |
| Random Req | App | 820002 |
| Random Resp | Tracker | 108201, `R` |
| User Confirmation | Tracker | 108301 |
| Server Check | Tracker | 10008401010000 |
| Auth Req | App | 0200 or 820002 |
| Auth Chal | Tracker | 100201, `Chal` or 108201, `Chal` |
| Auth Resp | App | 0300, `Resp` or 8300, `Resp` |
| Auth Complete | Tracker | 100301 or 108301 |
| Auth Fail | Tracker | 100304 or 108307 |

Table 3.2: Reversed Xiaomi application-layer opcodes and relevant values. `Key`, `Const`, `R`, `Chal` and `Resp` are 16-byte values shown in hex. `Const` equals to `1863c2cce5d159413bed92c4b163c279`.

The Auth characteristic also serves Authentication messages. The dissectors monitor the status of Authentication and the challenge-response. In our experiments, we crafted Authentication messages with different opcodes and noticed that MB 4/5/6 accept opcodes used by MB 2/3, as shown in Table 3.2.

The Communication protocol involves several characteristics, but we focused on the standard Heart Rate Measurement (00:00:2A:37:00:00:10:00:80:00:00:80:5F:9B:34:FB) and the custom Steps (00:00:00:07:00:00:35:12:21:18:00:09:AF:10:07:00). The dissectors decode the custom data format and display the effective value transmitted from the tracker to the app.

### 3.5.2 Security Mechanisms Module

Our toolkit also implements the custom key derivation function for Pairing v2 and challenge-response Authentication procedures. By using these functions, `breakmi` is capable of deriving a valid pairing key from its seed (`R`) and a valid authentication response (`Resp`) from a challenge (`Chal`), and a pairing key (`Key`). We invested much time in understanding how the key derivation and challenge-response procedures work, and we reimplemented them with Python. In particular, we used Python's cryptography module from PyCA [104] to implement SHA256 for the key derivation and AES-ECB for the challenge-response part.

To reverse the key derivation and the challenge-response, we used a mix of static and dynamic techniques. For the static analysis, we decompiled the Mi Fit APK with JADX [105] and looked at the recovered source code. Unfortunately, the Mi Fit app is obfuscated, and we could not recover the key derivation and authentication logic. At this point, we switched to dynamic binary instrumentation with Frida [106]. Our dynamic approach was successful, as we were able to reverse the key derivation and authentication logics by hooking their entry point at runtime and observing their inputs and outputs.

### 3.5.3 Attacks Module

The OTA tracker impersonation attack, presented in Figure 3.6, was implemented using Bleno [107], an open-source BLE peripheral written in Node.js. Our Bleno script imitates any Xiaomi tracker by exposing the same BLE features (e.g., BLE advertisements and GATT server) collected by an extensive study of legitimate Mi Band 2/3/4/5/6. It was challenging to collect this information from all trackers and make sense of the (proprietary) characteristics and services exposed by the trackers. Once a victim app finds our Bleno tracker, we can complete Pairing v1 and Authentication as a trusted device, and we expose a malicious GATT server to the app during Communication.

The OTA app impersonation attack, presented in Figure 3.7, was implemented using Noble [108], an open-source BLE central, and by re-using the tracker impersonation described above. Our tool connects to nearby Xiaomi apps and trackers, performs a replay attack on the Authentication protocol, disconnects from the app, and establishes a communication session with the tracker as a trusted app. Once connected, our fake app retrieves data from the tracker, such as the step count and the user's heart rate. Furthermore, it can send fake SMS and phone call notifications and activate alarms. The OTA MitM attack implementation uses the same logic of the app impersonation. However, instead of disconnecting from the app after the replay attack is completed, the tool keeps two parallel connections and establishes a MitM position between the app and the tracker.

The toolkit also includes a malicious app that can be used to perform SB app impersonation and eavesdropping attacks. Our app requires only Android's `BLUETOOTH` permission to interact with the tracker over BLE and `INTERNET` permission to exfiltrate data remotely. Android classifies these permissions as normal, so they are granted during installation without triggering any user prompt.

SB eavesdropping and SB app impersonation utilize the same setup that periodically checks active BLE connections using Android `getConnectedDevices` API and waits for a Xiaomi tracker to appear. As soon as a tracker connects, the malicious app launches the attack.

During SB eavesdropping, our app subscribes to relevant characteristics and can eavesdrop on all BLE data coming from the tracker, including sensor readings, commands, pairing key seeds, and authentication challenges. During app impersonation, our app starts a new pairing session with the tracker by sending a `Pairing Init` without disrupting the communication between the tracker and the legitimate app. The malicious app eventually negotiates a new pairing key as in the right part of Figure 3.8 and gains access to protected data. The SB app impersonation attack on Pairing *v2* entails the additional challenge of interacting with the Xiaomi backend to retrieve a signature. The malicious app sends a factory reset command to the tracker, which causes the change of its BLE address. A scan (requiring Android `BLUETOOTH_ADMIN` and `ACCESS_FINE_LOCATION` permissions) allows our app to perform a new

| Device | Year | BTv | Pv | BS | SC | SoC | FW |
|---|---|---|---|---|---|---|---|
| Mi Band 2 | 2016 | 4.2 | 1 | ✓ | ✗ | DA14681 | 1.0.1.81 |
| Mi Band 3 | 2018 | 4.2 | 1 | ✓ | ✗ | DA14681 | 2.4.0.32 |
| Cor 2 | 2019 | 4.2 | 1 | ✓ | ✗ | DA14681 | 0.3.0.44 |
| Mi Band 4 | 2019 | 5.0 | 2 | ✓ | ✓ | DA14697 | 1.0.9.66 |
| Mi Band 5 | 2020 | 5.0 | 2 | ✓ | ✓ | DA14697 | 1.0.2.64 |
| Mi Band 6 | 2021 | 5.0 | 2 | ✓ | ✓ | DA14699 | 1.0.1.36 |

Table 3.3: Fitness trackers' technical specifications. The columns contain the device name, release year, supported Bluetooth version (BTv), Pairing protocol version (Pv), BLE link-layer security support (BS), BLE Secure Connections (SC) support, used system-on-chip (SoC), and Mi Fit firmware version (FW).

pairing process and associate the tracker to our malicious Xiaomi account stealing its ownership from the legitimate user.

Since all attacks performed by `breakmi` are fully automated, we propose it as a continuous evaluation tool for the Xiaomi ecosystem. Even if Xiaomi were to enable link-layer security, the vulnerabilities we found at the application-layer would continue to exist. By using `breakmi`, anyone would be able to test for those vulnerabilities in future Xiaomi fitness trackers.

## 3.6 Evaluation

This section describes the evaluation of the attacks (presented in Section 5.4) using `breakmi` (described in Section 6.6). Our evaluation confirms that all Xiaomi trackers since 2016 and the most recent version of Xiaomi companion apps are *vulnerable* to our protocol-level attacks. In addition, it proves that `breakmi` works in practice and is cheap to deploy. As a result, millions of Xiaomi users are potential targets, and their sensitive health and personal data can be leaked and manipulated by bad actors.

### 3.6.1 Setup

In our evaluation, we tested all trackers shipped by Xiaomi since 2016. The sample includes Mi Band 2/3/4/5/6 and Amazfit Cor 2. The MB 1 is out of scope because it is known to ship with no security at all [109]. We did not test the Amazfit Cor 1 because of its limited availability and market share. However, we expect that it is vulnerable to our attacks as it is a MB 2 clone. Moreover, we tested the latest versions of Mi Fit (v 4.8.1) and Zepp (v 5.9.2) as the official Xiaomi mobile applications. The apps are compatible with all Xiaomi trackers and are available for Android and iOS. Despite our attacks *not* requiring root permissions, we use rooted devices to facilitate our experiments.

Table 3.3 presents the trackers' technical specifications. The Mi Band 2, Mi Band 3, and Cor 2 support Bluetooth version (BTv) 4.2 and Pairing version (Pv) v1. The others support Bluetooth 5.0, Pairing v2, and BLE Secure Connections (SC). All trackers support BLE security (BS) at the link-layer, but Xiaomi is not taking advantage of that. The tracker system-on-chip

| | Mi Fit | Zepp | MB2 | MB3 | AC2 | MB4 | MB5 | MB6 |
|---|---|---|---|---|---|---|---|---|
| OTA Tracker Impersonation | - | - | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| OTA App Impersonation | ✓ | ✓ | - | - | - | - | - | - |
| OTA Man-in-the-Middle | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| OTA Eavesdropping | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| SB Eavesdropping | - | - | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| SB App Impersonation | ✓ | ✓ | - | - | - | - | - | - |

Table 3.4: Evaluation results for OTA and SB attacks. The first column shows the attack name, the following eight columns contain the targets (two companion apps and six fitness trackers). A checkmark (✓) means that the attack was successful, and a hyphen (-) means that the attack does not apply to that target. MB and AC abbreviate Mi Band and Amazfit Cor. The SB attacks on the Mi Fit and Zepp apps were successfully tested on six Android versions (see Table 3.5).

(SoC) is either a DA14681, a DA14697, or a DA14699, all manufactured by Dialog Semiconductor [110]. We also report the firmware version (FW) of the tracker at the evaluation time.

Since our SB attacks depend on an issue with the Android BLE API (in addition to Xiaomi ones), we tested six popular Android versions using different smartphones: Android 12 (Google Pixel 4A), Android 11 (Google Pixel 2 XL), Android 10 (Google Pixel XL), Android 9 (Samsung Galaxy J5), Android 8.1 (Xiaomi Redmi 5 Plus) and Android 6 (Samsung Galaxy S5). We note that we cannot test our attacks on the Android emulator as it does not support Bluetooth emulation.

Our OTA attacking device is an Acer Aspire 3 laptop connected with a BLE sniffer and a BLE dongle. The laptop runs Ubuntu version 18.04 and supports Bluetooth 4.2. The sniffer uses three BBC Micro Bit boards and btlejack [111]. The BLE dongle is a CSR8510 A-10 Controller with Bluetooth 4.0 support. The dongle is needed as we have to change the attacking device's BLE address, and the laptop (BLE controller) does not allow this operation.

The OTA impersonation and MitM attacks were performed by the attacking device running `breakmi` and acting as both a spoofed tracker and a spoofed app. When impersonating the tracker, the attacking device advertises as a spoofed tracker, while during an app impersonation, it scans for trackers as a spoofed app. OTA eavesdropping was performed using the BLE sniffer. The SB app impersonation and eavesdropping attacks were performed by running the malicious app in the background and letting the legitimate app and the tracker communicate as usual.

### 3.6.2 Results

The attacks' evaluation results are shown in Table 3.4, where we demonstrate that the attacks are effective across *all* evaluated devices (if the attack applies to that device). We can wirelessly impersonate all tested trackers and apps, MitM them, and eavesdrop on their communication. We can also remotely eavesdrop and impersonate the Mi Fit and Zepp apps for Android via a malicious app. All attacks work *regardless of* the hardware and software details of the victim device (e.g., SoC, firmware, app, tracker, and BLE versions).

| Smartphone | Android | SB Eaves. | SB App Imp. |
|---|---|---|---|
| Pixel 4A | 12 (n/a) | ✓* | ✓* |
| Pixel 2XL | 11 (34.7%) | ✓ | ✓ |
| Pixel XL | 10 (27.7%) | ✓ | ✓ |
| Galaxy J5 | 9 (13.9%) | ✓ | ✓ |
| Redmi 5 Plus | 8 (10.56%) | ✓ | ✓ |
| Galaxy S5 | 6 (3.36%) | ✓ | ✓ |

* Attack requires Android `BLUETOOTH_CONNECT` dangerous permission

Table 3.5: Evaluation results for SB attacks against the latest six Android versions. All attacks were tested using a MB 4 as a tracker. The first column shows the smartphone model, the second one the Android version and its market share according to [84]. Market share numbers for Android 12 are not available (n/a) as it is too recent. The third and fourth columns show that all Android versions we test are vulnerable to SB attacks (✓). If we sum the markets share numbers, our attacks are effective on at least 90.22% of Android devices.

The SB remote attacks target Android, so we test them on six popular Android versions, as we show in Table 3.5. We confirm that *all* six tested Android versions are vulnerable to our attacks. According to [84], this means that (at least) 90.22% of Android devices are vulnerable. Since Android 12 was recently released in October 2021, statistical market share data is not available yet. We highlight that, up until Android 11, our attacks only ask for standard Bluetooth permissions as a requirement to access the `getConnectedDevices` method (that we exploit). Android 12 introduces the `BLUETOOTH_CONNECT` dangerous-level runtime permission, which must be declared to access `getConnectedDevices`. As a consequence, on Android 12, our malicious app must show the user a Nearby Devices dialog that explicitly states our intent to find, connect to and determine the location of nearby devices.

## 3.7 Countermeasures

We now discuss five countermeasures addressing the vulnerabilities presented in Section 3.3.2 and the attacks presented in Section 5.4. The first four apply to the application-layer and the fifth to the link-layer.

**C1 (Authenticated) Key Establishment** Pairing v1/v2 should use an Authenticated Key Establishment (AKE) to prevent impersonation, man-in-the-middle, and eavesdropping attacks. Figure 3.9 illustrates an updated version of Pairing v1/v2 to support C1. The tracker and the app first generate a public-private key pair and share their public keys (`App_Pk` and `TR_Pk`). They proceed with the calculation of a key (`K`) through a Diffie-Hellman (DH) function and the sharing of two nonces (`App_N` and `TR_N`). Finally, the tracker and the app calculate a confirmation value (`V`) displayed on each screen and wait for user confirmation concerning the match of the displayed values.

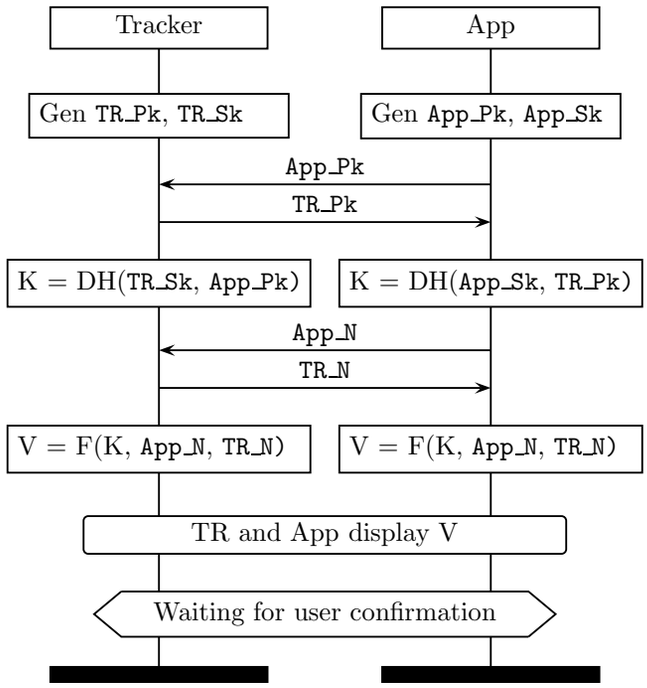



Figure 3.9: Authenticated key establishment (i.e., pairing) protocol providing C1+C2. Tracker and app generate a public-private key pair and share the public keys (App_Pk and TR_Pk) with each other. They calculate a new key through a Diffie–Hellman (DH) function and share two nonces (App_N and TR_N). Then, the tracker and the app both calculate a confirmation value (V) displayed on each screen and wait for user confirmation concerning the match of the displayed values. The DH function guarantees C1, while the user verifying tracker and app during pairing confirmation guarantees C2.

**C2: Strong Pairing Confirmation** The updated Pairing v1/v2 protocol, shown in Figure 3.9, guarantees C2 thanks to the numeric comparison performed by the user. In particular, while pairing, a MitM attacker is not capable of generating V as she does not know K pairing. Moreover, during an app impersonation attack, the adversary would trigger an unexpected user interaction while re-pairing with the app.

**C3: Strong Key Authentication** The Authentication procedure should be mutual and resistant to replay attacks. Mutual authentication is easy to implement by letting the app and the tracker send their challenges and then verifying them on both ends. Replay protection is also straightforward and can be achieved by using nonces and generating a response from a challenge and a nonce. These measures raise the bar for tracker impersonation and app impersonation. Figure 3.10 illustrates the updated Xiaomi Authentication protocol. Tracker and app already share the pairing key (K) and exchange a challenge (App_Ch and TR_Ch). They calculate the solutions through a hash function H, which relies on the challenges and the pairing key. They verify the correctness of the received challenge solution, and if both checks are successful, the tracker unlocks its data. Finally, they start an AES-CCM encrypted communication session using a session key SK obtained through a HKDF key derivation function from the pairing key K, and two exchanged nonces (App_N and TR_N).

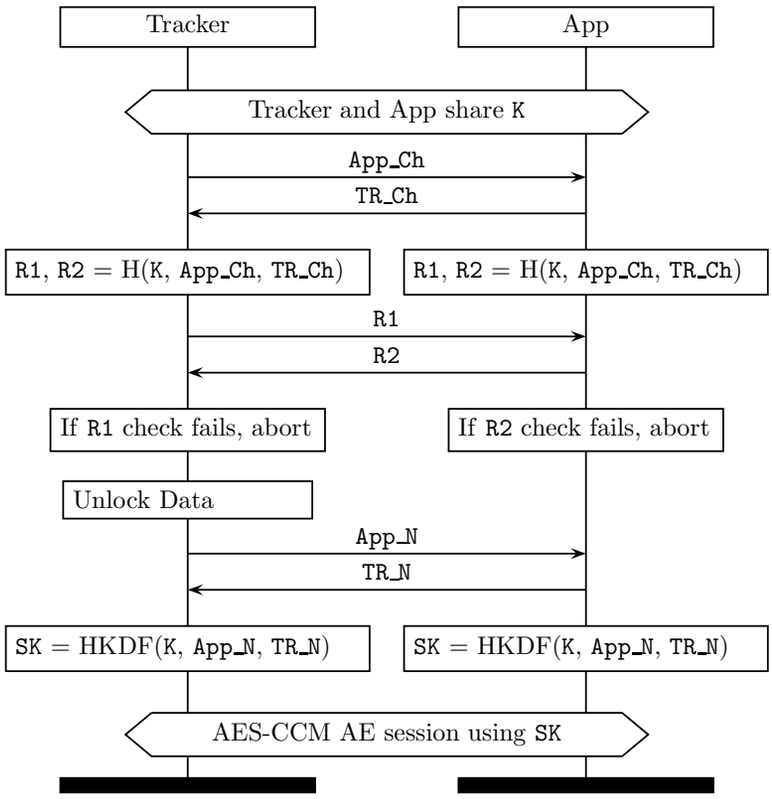



Figure 3.10: Mutual authentication protocol providing C3+C4. Tracker and app already share the pairing key (K) and exchange a challenge (App_Ch and TR_Ch). They calculate the solutions through a hash function H, that relies on the challenges and on the pairing key. They verify the correctness of the received challenge solution, and if both checks are successful, the tracker unlocks its data. Finally, they start an AES-CCM encrypted communication session using a session key SK obtained through a HKDF key derivation function from the pairing key K, and two exchanged nonces (App_N and TR_N). The mutual verification of challenges guarantees C3 and AES-CCM with a fresh session key provides C4.

**C4: Authenticated-encryption** The Communication protocol should use a fresh session key derived from the pairing key K to encrypt and integrity-protect the data exchanged between app and tracker. Devices can rely on AES-CCM and HKDF to introduce this countermeasure, as both functions are already supported by the devices SoC. C4 protects against eavesdropping and MitM attacks during Communication.

**C5: BLE Link-Layer Security** To complement the security at the application-layer, Xiaomi might also enable the BLE link-layer security mechanisms already supported by all its devices. The robustness of BLE link-layer security mechanisms depends on the BLE version of the device. The MB 4/5/6 implement BLE SC, so that they would benefit from secure protocols for pairing and session establishment. Instead, the MB 2/3 implement legacy BLE security, which is known to be insecure [112].

## 3.8 Comparison with Fitbit

In this section, we compare our findings of Xiaomi with the *Fitbit* [113] ecosystem, which is Xiaomi's main competitor in the fitness tracker market. Our motivation for the comparison is twofold. Firstly, to assess if Fitbit is affected by similar vulnerabilities and attacks found on Xiaomi devices. Secondly, to evaluate how effective `breakmi` is on other large fitness tracking ecosystems. We used the same RE methodology adopted for Xiaomi and described in Section 3.9 for this analysis, but we targeted different proprietary protocols.

In particular, we had a look at a Charge 2 tracker and the latest version of the Fitbit app for Android [114], as their security mechanisms were already reversed [18, 26, 27, 14]. The Charge 2 is from 2016, supports Bluetooth 4.1 and BLE link-layer security, and is powered by a BLUENRGCSP SoC from ST Microelectronics. We also considered the Charge 4 from 2020, but unlike the Charge 2, it uses unknown protocols, and reversing them is out of the scope of this work.

We now summarize what is known about the Fitbit protocols. Then we discuss their vulnerabilities and how the attacks presented in Section 5.4 apply to them. We describe how we extended `breakmi` to deploy five attacks on actual Fitbit devices successfully. Finally, we discuss how to port the countermeasures discussed for Xiaomi to the Fitbit ecosystem.

### 3.8.1 Architecture and Protocols

Fitbit uses the same system architecture as Xiaomi (see Figure 3.1). A tracker communicates over BLE with the Fitbit companion app, and the app communicates via Wi-Fi or a mobile network with the Fitbit backend. The devices use proprietary application-layer protocols (e.g., Pairing, Authentication, and Communication) and ignore available BLE link-layer security mechanisms. Unlike Xiaomi, which utilizes public BLE addresses, Fitbit trackers use *random static* addresses. Regardless, Fitbit devices are still trackable because their BLE address never changes for their entire lifetime.

**Pairing** As described in [26, 14], Fitbit employs a Pairing protocol to establish a pairing key (authentication key in Fitbit terms) between the trackers and the app. During Pairing, the backend (which pre-shares a device key with a tracker) computes the pairing key from the device key and a salt, and sends them to the app. The app sends the salt to the tracker, and the tracker computes the pairing key from the salt and the device key. We note that, unlike Xiaomi, the key or its seed is *not* sent in cleartext, and pairing confirmation is strong. Pairing is accepted only when the user confirms on the app that she sees the same numeric sequence on the tracker and app screens. As such, Fitbit adopts a stronger user confirmation strategy than Xiaomi.

**Authentication** Fitbit Authentication, unlike Xiaomi, is *mutual* and works as follows. The app sends a random challenge together with a constant key salt (provided by the backend during Pairing). The tracker sends back a MAC and a counter value, where the MAC is computed using the counter and the random challenge and is keyed using the pairing key corresponding to the key salt. The app checks that the MAC is valid and sends back a different MAC computed using the counter and keyed with the pairing key. The tracker checks the MAC, and then mutual authentication of the pairing key is achieved. The counter is updated on each run of the Authentication protocol.

**Communication** Fitbit sessions, unlike Xiaomi, have two different modes: live and normal.

Live mode is *not encrypted* and monitors the tracker readings in real-time. Live mode data stays in the app, and is not relayed to the backend. On the other hand, normal mode synchronizes the data from the tracker to the backend. The synchronization process is *encrypted* with the shared pairing key by using either XTEA (extended TEA) encryption [115] or AES-EAX authenticated-encryption [116].

### 3.8.2 Vulnerabilities and Attacks

We compare Xiaomi and Fitbit security mechanisms, investigate if Xiaomi vulnerabilities can be found in the Fitbit ecosystem, and evaluate how our OTA and SB attacks perform on Fitbit.

**Vulnerabilities** Fitbit proprietary security mechanisms are slightly better than Xiaomi's, but severe vulnerabilities still affect them. In particular, the Fitbit Pairing protocol does not send the pairing key (seed) in the clear, has strong user confirmation but still *lacks* device authentication. The Authentication protocol is mutual but is *replayable*. The Communication protocol *partially* uses encryption and integrity protection. For example, only normal mode data is encrypted.

**OTA attacks** The OTA app impersonation and MitM attacks from Figure 3.7 are still effective, as Fitbit Authentication is not replay-protected. Instead, the OTA tracker impersonation presented in Figure 3.6 does not work as Fitbit Authentication is mutual. However, the attacker can still impersonate a tracker using a replay attack similar to the one described in Figure 3.7. The OTA eavesdropping works only with unencrypted data (e.g., live mode data).

**SB attacks** The SB app impersonation attack on Android in Figure 3.8 is still effective. A malicious app can get valid authentication credentials from the backend and re-pair with the victim tracker (see Fig 4a in [14]). Thus, we discovered a novel technique to steal trackers virtually. The SB eavesdropping suffers the same limitations as OTA eavesdropping.

Overall, the attacks' impact on Fitbit is lower than the one on Xiaomi but remains significant. For example, during impersonation or MitM, the attacker can only manipulate and tamper with the packets sent in cleartext. Nevertheless, the attacker can still abuse the unprotected live mode data to report worrisome health conditions to the user.

### 3.8.3 Attacking Fitbit with `breakmi`

We extended `breakmi` to analyze and attack the Fitbit ecosystem. We can clone Charge 2 trackers (including their advertised data and GATT servers) and the Fitbit app. Moreover, we can speak the Pairing, Authentication, and Communication protocols described before. We created custom scapy dissection classes to generate valid packets just like we did for Xiaomi. Fitbit uses *static* (random) BLE addresses, and we updated `breakmi` to support this privacy feature.

We use `breakmi` to perform the OTA impersonation and MitM attacks by replaying packets during the Fitbit Authentication. We also developed an extra module for our malicious Android app that performs the SB eavesdropping and app impersonation attacks. The latter provides the trackers' serial number and BLE address to the backend to reset the tracker's owner and then triggers pairing from the malicious app. This strategy is different from those we presented in Section 3.4.5, targeting Xiaomi.

### 3.8.4 Porting our Xiaomi Countermeasures to Fitbit

Fitbit Pairing, Authentication, and Communication protocols described in Section 3.8.1 can be strengthened by using a subset of the countermeasures proposed for Xiaomi in Section 5.7. In particular, unencrypted live mode data can be protected with C1 (authenticated-encryption), Pairing can be enhanced with C2 (authenticated key establishment), Authentication can be improved by adding replay protection as in C3. In addition, defense in depth can be achieved using C4 (BLE link-layer security). C5 (strong pairing confirmation) is not needed as is already provided by Fitbit pairing confirmation.

## 3.9 Reverse-Engineering Methodology

In this section, we describe our reverse-engineering methodology and how we applied it to perform our security analysis of Xiaomi and Fitbit ecosystems. We describe how we perform reconnaissance on a fitness tracking ecosystem. We explain how we analyze the traffic exchanged between tracker, app, and backend and how we apply static and dynamic analysis techniques to a mobile app. We also discuss the development of automated scripts for reverse-engineering and security assessment.

### 3.9.1 Trackers and Apps Reconnaissance

We now describe how we performed reconnaissance of the Xiaomi trackers and apps.

Regarding the tracker, we inspect its BLE GATT server using the "nRF Connect for Mobile" app [117]. The app allows to scan, explore and communicate with BLE devices. We extract data from every service and characteristic, identifying Xiaomi proprietary GATT services (i.e., `0xFEE0`, `0xFEE1`). We find a set of characteristics protected by Authentication. For example, the Auth characteristic manages Pairing and Authentication, and the Steps and Heart Rate characteristics contain sensitive data.

We install Mi Fit and Zepp apps and interact with them. The apps are very similar as they share the same UI, communicate with the same backend, and provide the same interface to the tracker. The apps' UI does not show any information about the Pairing, Authentication, and Communication protocols that we RE. We also find that their codebase is very similar despite being closed-source. GadgetBridge [118], an open-source project, provides some insights into the apps' internals.

### 3.9.2 BLE and Web Traffic Analysis

First, we intercept over-the-air BLE traffic with a BLE sniffer and confirm that Xiaomi BLE traffic is not encrypted. Then, since we control the smartphone running the app, we simply enable the "Bluetooth HCI Snoop Log" under the "Developer Options" and directly access BLE capture files. This option does not work correctly on some smartphones, but other alternatives exist (e.g., hcidump [119], adb bugreport [120]). We visualize and inspect BLE packets using Wireshark [103], a network protocol analyzer. We find several recurring opcodes, shown in Table 3.2. We define the Pairing, Authentication, and Communication protocols and implement them in our automated scripts.

We run mitmproxy [121] on our machine, a MitM proxy tool that intercepts and logs Wi-Fi and cellular networks traffic. We also configure our smartphone to redirect its web traffic to mitmproxy, and we install the mitmproxy CA (Certificate Authority) certificate. We intercept traffic going from the app to the Xiaomi backend while performing various operations with the tracker (e.g., adding a new tracker, synchronizing user activity data, completing workout sessions). We discover several API endpoints (e.g., `account.xiaomi.com/oauth2/authorize`, `account.huami.com/v2/client/login`, `api-mifit-de2.huami.com/v1/device/binds.json`). We inspect the traffic looking for interesting requests. For example, we reverse-engineer how Xiaomi registers new trackers on its backend. Then, we test the API endpoints by sending custom-made requests and monitoring their responses (the tests were performed according to Xiaomi's Bug Bounty program guidelines).

We merge BLE and Web capture files using a Wireshark utility better to visualize the sequentiality of the packets in Xiaomi protocols. We discover how the app acts as a proxy between the tracker and the backend during Pairing v2 and how the app sends security packets to the unprotected (custom) Chunked Transfer characteristic.

### 3.9.3 Mobile Companion Apps Analysis

We examine Mi Fit and Zepp apps' code to uncover the implementation of Xiaomi proprietary protocols from the source. We describe which static and dynamic analysis techniques we applied to our app analysis.

We start our static code analysis by extracting Mi Fit and Zepp APKs and decompiling them with JADX [105], a Dex to Java decompiler. We discover that the Java code is obfuscated and difficult to navigate. We experiment with different deobfuscation tools, either by directly acting on the APK files (i.e., JADX deobfuscation utility, DeGuard [122], simplify [123]) or by converting them into JAR files first (i.e., Java Deobfuscator [124]), but they were unable to deobfuscate it. We utilize apktool [125] to reverse-engineer the APK, inspect its resources and experiment with repackaging.

We manually inspect the apps' code. We search for keywords, trying several interesting words (e.g., Pairing, Bonding, Authentication, and Characteristic strings) and cryptographic functions supported by the tracker (e.g., MD5 and AES-ECB). We utilize the Mobile Security Framework (MobSF) [126], an automated pen-testing and malware analysis tool, to inspect app components and for its automated binary analysis. We create control-flow graphs with Androguard [127], a Python reverse-engineering tool. Then, we perform dataflow analysis to help us to track the pairing `Key`. Ultimately, we find the code sections responsible for Pairing, Authentication, and Communication.

We apply dynamic analysis techniques to confirm that those code sections are actually executed at runtime. We rely on Frida [106], a dynamic binary instrumentation toolkit that allows us to inject code during runtime execution. We use Frida hooks to log runtime variables and confirm they match with the values found in the capture files. We also experiment with editing variables at runtime and test the robustness of Xiaomi's security mechanisms when receiving unexpected values.

### 3.9.4 Development of Scripts

Throughout our RE efforts, we develop a set of scripts that automate time-consuming tasks. We aggregate and upgrade those scripts in `breakmi`, an extensible modular toolkit.

First, we build automated scripts to interact with a tracker's GATT server using a Python library called Bleak [128]. Our scripts automatically connect, explore, and display information about any BLE device. Then, we build scripts that replicate interesting operations on the tracker (e.g., read requests, enable notifications, firmware update, and factory reset).

We automate BLE traffic analysis by developing protocol dissectors on Pyshark [102], a Python wrapper for packet parsing, and by using the scapy [101] packet crafting library. We reverse-engineer the binary structure of Xiaomi firmware so that we can extract firmware from capture files containing a firmware update. We also automate web traffic analysis via mitmproxy by developing scripts with the mitmdump utility. Our mitmdump scripts analyze hundreds of web requests to find pairing-related messages, identify their purpose and retrieve their parameters.

We automate our OTA attacks using Bleno [107] and Noble [108]. We create spoofed BLE peripherals that mirror services, characteristics, and advertising from legitimate MB 2/3/4/5/6 and Cor 2. We implement the Pairing, Authentication, and Communication protocols on the Auth, Steps, Heart Rate, and Chunked Transfer characteristics. We also implement the Pairing, Authentication, and Communication protocols on a BLE central and configure it to scan for and connect to the target trackers. During OTA MitM, the malicious BLE central and peripheral communicate with each other through websockets.

## 3.10 Related Work

We discuss the current state of the literature concerning the security and cryptographic analysis of Mi Band, Fitbit, and other embedded devices, the attacks on BLE protocols, and Android vulnerabilities and compare it with our work.

**Attacks against Mi Band devices** Fereidooni et al. [10] looked at ways to inject false data from the tracking application to the backend of 17 popular trackers, including a Mi Band device. Hilts et al. [24] presented a security and privacy analysis of six trackers, including a Mi Band. These analyses are useful yet orthogonal to ours as they do *not* cover Xiaomi's proprietary security protocols. Some developers released tools for Mi Band 2 [129] and Mi Band 3 [130] able to trigger Xiaomi's Pairing v1 to unlock private data as described in [15]. Mi Band 4 tools such as [131] require the knowledge of the pairing key, because nothing is known about Pairing v2 apart from it being "server-based" [118]. Our attacks are much stronger and stealthy. They do not require knowledge of the pairing key, no specific action from the victim, do not disrupt the pairing between the victim's app and the victim's tracker, and do not reset the data (as they completely skip Pairing). In fact, we improved and corrected the attack in [15], that claims to be targeting Authentication when it is actually targeting Pairing v1. The Xiaomi protocols reverse-engineered in [11] belong to Mi Home [132, 133], the Xiaomi app that manages smart home devices, which does *not* support fitness trackers. To conclude, existing attacks were ad-hoc and partial. Our work is the first to systematize and generalize attacks against the Xiaomi fitness tracking ecosystem.

**Attacks against Fitbit devices** Rahman et al. [78, 81] attacked the legacy ANT Communication protocol used by Fitbit Ultra and Garmin Forerunner and provided fixes for them. Cyr et al. [18] utilized Ubertooth to sniff OTA BLE traffic from a Fitbit Flex and an HTTP/HTTPS proxy, underlining several privacy-related issues, including device identification attacks. As described in [134], Fitbit Charge was patched for being vulnerable to non-authenticated reads and a replay attack. Goyal et al. [83] presented a comparative analysis of a Jawbone UP Move and Fitbit Charge. They discovered security issues in the GATT server, the mobile app, and the backend. Schellevis et al. [26], reversed the Fitbit Charge HR authentication protocol through firmware analysis. Fereidooni et al. [27] found ways to spoof Fitbit Flex and One data by tearing down the devices and analyzing their firmware. Classen et al. [14] demonstrated attacks on Fitbit capable of leaking private data from a tracker, re-flashing a rogue firmware, and redirecting the app to a rogue cloud service. Other researchers could only perform eavesdropping while being near the target device, but we can do it remotely with our SB eavesdropping attack. Our SB app impersonation attack allows us to remotely inject fake data into the Fitbit account of our victim. Instead, previous studies required the ownership of the tracker or physical access. We also verified that Fitbit Charge 2 is still using the same protocols reversed in [14], and we have proven the feasibility of a MitM setup by deploying our OTA MitM attack.

**Attacks against BLE protocol** Fitness trackers communicate with the app using BLE. Privacy-oriented case studies show, by looking at BLE advertising [47, 135, 48, 136] or BLE UUID [77], that one can fingerprint the tracker and even identify the user's activities (e.g., walking or sitting). Several papers discussed standard-compliant attacks on BLE legacy pairing [42], key negotiation [40], Secure Simple Pairing [43, 44, 44], Secure Connections Only Mode [45], associations [137], GATT [138], and reconnections [139]. Bugs specific to implementation and related exploits were also discussed [140, 141]. Wang et al. [41] exploited the BLE features designed for low-cost devices to downgrade the key negotiation and authentication procedures and access the stored BLE data. BLE legacy pairing and Secure Simple Pairing were also found to be vulnerable to misbinding attacks [142]. Works about BLE are orthogonal to what we present as we are targeting proprietary protocols implemented at the application-layer on top of an insecure BLE link layer.

**Cryptographic security on embedded systems** Constrained embedded devices often misuse cryptographic primitives, thus introducing severe vulnerabilities in the whole system. Wouters et al. [52, 53] reverse-engineered the Tesla Model S and Model X key fob and found new vulnerabilities. Our study and methodology are similar to theirs, as they reversed proprietary protocols (i.e., Tesla's PKES and Pairing protocols), analyzed the BLE SoC, discussed cryptographic security measures, developed a proof-of-concept and proposed countermeasures. Their findings could apply to other key fob manufacturers, similar to what we did with `breakmi`. We exploited the same vulnerabilities found both in Xiaomi and Fitbit devices, two of the largest players in the fitness tracker market. An orthogonal approach to reverse-engineering embedded systems is firmware and binary code analysis, adopted by `Incision` [23], an architecture and OS-agnostic RE framework. While similar tools are effective, we believe that our application-layer analysis is necessary to provide a full security assessment of an embedded device.

**Vulnerabilities in Android** It is known that Android does not properly isolate Bluetooth keys among apps. For example, all devices' Bluetooth pairing keys are shared by all apps enabling co-located attacks [143, 144]. This fact might affect fitness trackers that are relying on BLE pairing. However, this is *not* the case for Xiaomi, which only relies on proprietary security mechanisms at the application-layer. As a result, our attacks are still effective even if Android would fix the shared pairing key issue.

**Smartwatches** Prior work also studied the security of smartwatches, with Apple Watch as the main target. In particular, researchers focused on MagicPairing [16], Apple's Bluetooth security mechanism, and reverse-engineering [24, 10]. In this work, we focus only on fitness trackers, as smartwatches represent a separate class of devices with far more capabilities than fitness trackers. For example, smartwatches ship with more capable SoC (e.g., ARM general-purpose and multi-core application processors) than trackers (e.g., ARM microcontroller), so comparing the two devices classes would be unfair.

## 3.11 Conclusion

Xiaomi is a market leader in the fitness tracking industry. Little is known about this ecosystem's security and privacy properties despite managing the sensitive information of millions of users (such as health and personal data). Nonetheless, Xiaomi claims to be "committed to protecting the privacy, confidentiality, and security of personal information" in its Privacy Policy [5]. We address this relevant issue by performing an extensive and up-to-date security evaluation of the Xiaomi fitness tracking ecosystem.

After extensive RE experiments, we uncover several worrisome issues. Xiaomi uses proprietary and undocumented security mechanisms to protect the communication between its trackers and apps. In particular, Xiaomi provides Pairing, Authentication, and Communication application-layer protocols over an insecure BLE connection. Xiaomi's approach is extremely risky as the security of its ecosystem relies on custom protocols that cannot be peer-reviewed in the open by the security community. Moreover, Xiaomi ignores standard BLE link-layer security mechanisms despite being supported by its devices.

We uncovered thirteen severe vulnerabilities (most of which were unknown) in the specification of Xiaomi custom application-layer protocols and exploited ten of them. The issues range from unilateral and replayable authentication to the lack of encryption and integrity protection. Being Xiaomi-compliant, the issues are exploitable on all devices using these protocols.

We demonstrate how to exploit the vulnerabilities with proximity-based (OTA) and remote (SB) attacks. Specifically, we describe over-the-air eavesdropping, impersonation, and MitM attacks, and remote eavesdropping and impersonation threats based on a malicious app co-located with the Xiaomi app.

In our evaluation, we successfully attacked all Xiaomi trackers released since 2016 (e.g., MB 2/3/4/5/6, Cor 2) and the up-to-date versions of the Mi Fit (v 4.8.1) and Zepp (v 5.9.2) companion apps. We also positively test our remote attacks on six popular Android versions (i.e., Android 6/8/9/10/11/12), covering at least 90.22% of Android devices in the market, according to [84].

We develop `breakmi`, a modular toolkit that automates RE experiments and attacks. `breakmi` includes protocol dissector, security mechanisms, and attacks modules. It is based on open-

source software and requires cheap and available hardware. We test our toolkit on the Xiaomi ecosystem, and we extend it to also work on Fitbit. We will open-source `breakmi`.

We propose *five* effective countermeasures that fix the presented vulnerabilities and attacks. Our countermeasures provide stronger Pairing, Authentication, and Communication protocols. Moreover, we show how to integrate our protocols into the Xiaomi ecosystem.

We assess whether the Fitbit ecosystem suffers from the same vulnerabilities as Xiaomi. We find that Fitbit has better (still proprietary) Pairing, Authentication, and Communication protocols. Nevertheless, it is vulnerable to five out of the six attacks presented in this paper. We conduct such attacks on actual Fitbit devices, and we extend our toolkit to be compatible with Fitbit.

Overall, our work required an extensive and time-consuming RE effort. The hardest RE challenge was the server-based Pairing v2. More specifically, it took us six months to understand how the tracker, app, and backend interact among themselves while pairing. We spent two months developing `breakmi`, which automated most of our protocol analysis and reverse-engineering and quickly made up for the time investment. For example, when Xiaomi released the new MB 6, our toolkit immediately detected its similarity to the previous generation of trackers. Using this information, we could deploy our attacks on the MB 6 within a single day. Another example would be the process of adding support for the Fitbit ecosystem. We only spent two weeks on it, while we spent three months working on fully supporting the Xiaomi ecosystem.

# Chapter 4

# E-Spoofer

***This contribution, titled "E-Spoofer: Attacking and Defending Xiaomi Electric Scooter Ecosystem", has been published in the Proceedings of the 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec 2023) [9].***

## 4.1 Introduction

Xiaomi is leading the electric scooter (e-scooter) market [145]. Its ecosystem includes seven e-scooters released in the last seven years (i.e., M365, Pro 1, Pro 2, 1S, Essential, Mi 3, and Mi 4) and the *Mi Home* mobile application for Android [132] and iOS [133]. Mi Home enables a user to manage his e-scooter, e.g., wirelessly locking and unlocking it or setting a password. Mi Home and the e-scooter communicate via *proprietary application-layer* protocols developed by Xiaomi. These protocols are undocumented, not peer-reviewed, and built on top of a *Bluetooth Low Energy (BLE)* link-layer.

Despite their associated security, privacy, and safety risks, no research work evaluated the security protocols used by Xiaomi to secure the interaction between its e-scooters and Mi Home. Instead, recent work focused on the privacy implications of e-scooter rental apps (including Xiaomi) [25] and on the security of Xiaomi's fitness tracking ecosystem [146]. In our work, we find that Xiaomi protocols can be exploited to (remotely) unlock and steal an e-scooter or permanently prevent its owner to manage it from Mi Home.

This work presents the *first* security evaluation of the communication channel between Xiaomi's e-scooters and Mi Home. In particular, we uncover and reverse-engineer all four e-scooter protocols used from 2016 to 2021. We label them as P1, P2, P3, and P4, and we dissect their custom Pairing (i.e., key agreement) and Session phases. We find that P1, P2, and P3 offer no security guarantees but *security through obscurity*. Instead, P4 provides some security properties (e.g., ECDH key agreement and AES-CCM authenticated encryption) but is vulnerable to *downgrade* attacks. Moreover, we find that Xiaomi decided *not* to use standard BLE link-layer security mechanisms (e.g., BLE pairing), despite their devices support them.

We present *four* novel attacks targeting the Xiaomi protocols' specifications. Two attacks enable a proximity-based or remote attacker to pair maliciously with an e-scooter and get authorized access to it *without* spoofing the victim's identity (i.e., MP). The other two attacks allow a proximity-based or remote attacker to downgrade the connection with an e-scooter to an insecure version and send arbitrary commands (i.e., SD). The proximity-based adversary must

be in BLE range of the target e-scooter. Instead, the remote adversary must have installed a malicious app on the victim's smartphone. Our attacks achieve *impactful* goals, such as unlocking and stealing an e-scooter, or preventing a victim from regaining control of the e-scooter via Mi Home. We isolate the *six* attacks' root causes, including the improper authentication and authorization mechanisms, and the unprotected but privileged vendor-specific features of Xiaomi protocols.

We release `E-Spoofer`, a toolkit capable of performing our four attacks by reimplementing and abusing the four reversed Xiaomi protocols. The toolkit includes *three* extensible modules. Two dedicated modules implement the Malicious Pairing and Session Downgrade attacks. The reverse-engineering (RE) module offers protocol dissectors to decode and build custom Xiaomi packets (e.g., P1, P2, P3, and P4). and useful Frida hooks for Mi Home to dynamically intercept and modify the proprietary Xiaomi payloads.

We successfully evaluate the attacks in *eight* different attack scenarios covering P1, P2, P3, and P4. Our setup allows testing multiple e-scooter configurations by using *three* modded e-scooters (e.g., M365, Essential, and Mi 3) with *five* BLE subsystems and *eight* BLE firmware. Our results are alarming. In all attack scenarios, we managed to unlock an e-scooter and steal it, or to lock it and to change its password, preventing its legitimate owner from accessing it via Mi Home. These results lead to millions [147] of exploitable devices.

To fix the four attacks and their six root causes, we developed and tested two usable and low-cost countermeasures and include them in our toolkit. First, we propose a backward-compatible pairing protocol with proper authentication and authorization mechanisms. Second, we provide a script to patch the session downgrade command from an e-scooter BLE firmware. We successfully test our patch on the M65 and Pro 1 e-scooters, whose BLE firmware is no longer updated by Xiaomi.

We summarize our contributions as follows:

- We present the first security evaluation of the proprietary security mechanisms employed by Xiaomi's e-scooters and Mi Home application. We isolate four custom application-layer security protocols on top of an insecure BLE link-layer. After reversing their Pairing and Session phases, we uncover six severe vulnerabilities in their design, including vendor-specific and unauthenticated protocol commands.

- We develop four attacks that steal an e-scooter or prevent its owner from accessing it from the Mi Home app previously paired with that e-scooter. The attacks are effective on P1, P2, P3, and P4, and can be deployed by an attacker in BLE range of a target e-scooter (i.e., proximity-based attacker) or via a malicious application on the victim's smartphone (i.e., remote attacker).

- We open-source `E-Spoofer`, an automated and low-cost toolkit that implements our attacks and tampers with the four Xiaomi protocols. Our toolkit includes the MP and SD attack modules, and a reverse-engineering module with protocol dissectors, firmware analysis tools, and Mi Home Frida hooks.

- We confirm that our four attacks are effective in eight attack scenarios covering five e-scooter BLE subsystems and eight BLE firmware. Our evaluation samples include P1, P2, P3, and P4. Our experimental setup allows to reproduce multiple attack scenarios using three partially disassembled e-scooters and different BLE subsystems. We also release

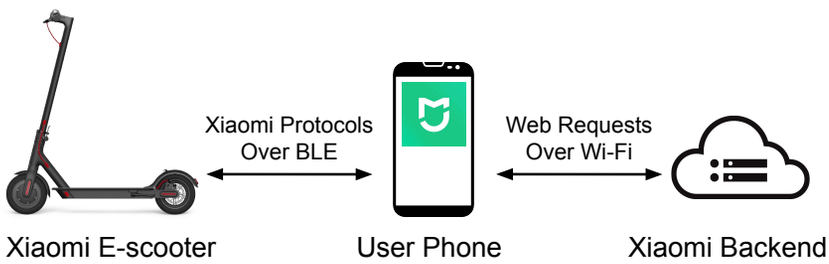



Figure 4.1: Xiaomi e-scooter ecosystem. Xiaomi e-scooter (left), the user smartphone running the Mi Home app (middle), and the Xiaomi backend (right). The e-scooter and the app are paired and connected over BLE. The app associates the e-scooter with the Xiaomi backend over Wi-Fi. We focus on the BLE traffic between the app and the e-scooter.

two effective countermeasures that fix our attacks. The first addresses the MP attacks by implementing a more secure pairing protocol. The second prevents the SD attacks by patching the BLE firmware of an e-scooter.

**Responsible disclosure and ethics** We responsibly disclosed our findings multiple times with Xiaomi via their bug bounty program [148]. In October 2022, we reported a UI password bypass issue with Mi Home, Xiaomi acknowledged it and provided a bug bounty. In November 2022, we shared a technical report and the code to reproduce our findings. In December 2022, we provided them with a video of the attacks on actual devices. Xiaomi did not follow up. We conducted our experiments in a controlled environment without involving third-party users and services. We provide our open-source `E-Spoofer` toolkit at https://github.com/Skiti/ESpoofer.

## 4.2 Xiaomi E-Scooter Ecosystem

Xiaomi is the electric scooter (e-scooter) market leader, sporting the highest number of active users and shipped devices [145]. Currently, it features seven e-scooters, i.e., M365 (2016), Pro 1 (2019), Pro 2 (2020), 1S (2020), Essential (2020), Mi 3 (2021), and Mi 4 (2022). Xiaomi also maintains *Mi Home*, a smartphone application for Android [132] and iOS [133] that manages Xiaomi's smart home devices, including any e-scooter. Xiaom's cloud-based backend service manages the e-scooters and their active Mi Home users.

Figure 4.1 shows a high-level representation of the Xiaomi e-scooter ecosystem. This work focuses on the BLE communication channel *between the e-scooter and Mi Home*. The e-scooter acts as a BLE peripheral (connection responder), while Mi Home is the BLE central (connection initiator). The e-scooter periodically broadcasts BLE advertisement packets to be discovered. These packets contain the e-scooter name, model, security level, and pairing mode activation. Mi Home scans the BLE spectrum and lists all connectable Xiaomi e-scooters nearby. Once connected, the devices exchange data using BLE's Generic Attribute Profile (GATT). The e-scooter exposes a GATT server, which includes the Nordic UART Service and a custom Xiaomi service. On the other hand, Mi Home acts as a GATT client, sending read, write, and subscribe requests to the e-scooter's GATT server. To communicate, Mi Home and the e-scooter establish a BLE link-layer connection. Then, they use *proprietary* application-layer

protocols and mechanisms that cannot be scrutinized with multi-purpose static and dynamic analysis tools.

Mi Home requires the user to register a Xiaomi account to pair, connect, and manage one or more Xiaomi e-scooters. The pairing process is a one-time procedure that requires user interaction and an Internet connection. The user starts pairing via the app UI, scans for nearby Xiaomi e-scooters, selects the correct e-scooter from a list, presses the headlight button to activate pairing mode, and waits. Once the pairing is complete, Xiaomi backend links the user account to the paired e-scooter, and Mi Home remembers the device for future connections. Optionally, the user can set a 6-digit alphanumeric PIN to protect the e-scooter from unauthorized access to Mi Home (e.g., from attackers that have stolen the user's smartphone and want to unlock the e-scooter via Mi Home).

A Xiaomi e-scooter is a high-end embedded device composed of several *proprietary* and *undocumented* subsystems: *radio (BLE)*, *battery management (BMS)*, and *electric motor (DRV)*. Each subsystem has a dedicated system-on-chip (SoC) and firmware. The connection between the subsystems is not standardized and might involve a proprietary bus. The radio subsystem provides BLE connectivity, enabling communication between the e-scooter and Mi Home. It also acts as a gateway to distribute firmware updates to the DRV and BMS. The BMS monitors and manages the e-scooter's battery. The DRV takes care of the electric motor that, when the DRV is not up-to-date, can be patched to change the motor's maximum speed. At the time of writing, all Xiaomi e-scooters are manufactured by *Ninebot*, a Chinese company financed by Xiaomi that acquired Segway (its main competitor in the US) in 2015 [149].

## 4.3 Threat Model

Now we present our system model and our proximity-based and remote attacker models. Please refer to Section 4.2 for their related background material.

### 4.3.1 System Model

We consider a victim who owns a Xiaomi e-scooter and a smartphone equipped with the Mi Home app for Android or iOS, as shown in Figure 4.1. We assume that the Mi Home version number is the *latest* available at the time of submission (e.g., Android v7.11.704 and iOS v7.12.204). We do *not* set a target Android or iOS version as we want to explore Xiaomi-compliant attacks that work regardless of the smartphone OS version.

The victim securely paired the app, and the e-scooter accepted the required permissions and completed the default firmware update. The update process involves the BLE, battery management, and electric motor subsystems (e.g., DRV017, BLE157, BMS141), and the BLE component acts as a gateway. To consider the most secure scenario, we assume that the *password-protection* is enabled to prevent unauthorized access to the e-scooter. Hence, according to common sense, the victim locks and unlocks the e-scooter from the app. Moreover, the victim uses the e-scooter features, such as pressing the power button to activate or deactivate the headlight.

The e-scooter and Mi Home communicate using Xiaomi proprietary application-layer protocols. These protocols run on top of a link-layer connection established using BLE. *Only Xiaomi*

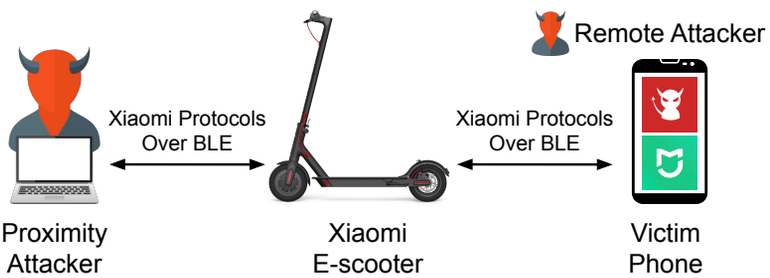



Figure 4.2: Proximity-based (left) and remote (right) attacker models investigated in this work. In the proximity-based threat model, the attacker is within BLE range of a target e-scooter. In the remote threat model, the adversary first installs a malicious Android app on the victim's smartphone. Then, she uses the malicious app (in red) to remotely target an e-scooter within BLE range of the victim's smartphone.

knows the application-layer protocols' details and their security guarantees (e.g., confidentiality, integrity, and authenticity).

### 4.3.2 Attacker Models

Password protection and secure communication at the application-layer and link-layer should protect victims against impactful attacks, including threats effective from BLE proximity or remotely via a malicious app on the victim's smartphone. For example, it should not be possible to (remotely) unlock and steal an e-scooter or (remotely) reset a password to deny the victim access to the e-scooter. Based on this reasoning, and as shown in Figure 4.2, we focus on two relevant threat actors:

**Proximity-based attacker** The proximity-based attacker targets the e-scooter with BLE signals. Hence, she requires being within BLE range of the target device. The proximity attacker has the following goals: (i) unlock and steal a (password-protected) e-scooter, and (ii) prevent the legitimate owner from accessing and controlling the e-scooter via Mi Home.

The proximity adversary has the capabilities of a real-world and low-cost BLE attacker. She can craft custom BLE packets, sniff the traffic over-the-air to get public information (e.g., BLE addresses and advertisements), and replicate the Android and iOS Mi Home apps with her attack equipment. The attacker does not observe the e-scooter while it pairs with Mi Home and does not install malicious software on the victim's devices. Moreover, she does not physically tamper with the e-scooter and the smartphone (e.g., no physical fault injection and side-channel attack).

**Remote attacker** The remote adversary attacks the e-scooter using a malicious application installed on the victim's smartphone. Thus, she requires the victim's smartphone to be within BLE range of the e-scooter, but she can remotely activate the app. For example, the adversary can attack the e-scooter while the victim is parking the e-scooter and walking away from the parking lot. This model differs from a proximity-based attack as the latter involves a BLE attacking machine (e.g., a laptop) in the BLE range of the victim, while the former involves

a malicious smartphone app. The remote attacker has the same goals as the proximity-based attacker.

Capability-wise, we consider a low-cost and real-world remote threat actor targeting the *Android* ecosystem (as opposed to iOS, which is more closed). We assume a malicious Android app that was installed using known (yet practical) social engineering and phishing techniques. The app does not require root privileges but needs basic permissions to interact with the e-scooter, such as Bluetooth and Internet permissions. The attacker develops the app using standard Android tools (e.g., Android Studio) and APIs (e.g., BLE advertisement, scanning, and GATT APIs). The remote attacker has the same limitations as the proximity one, except for installing an app on the victim's smartphone.

**Physical access requirements** Regardless of the attacker model, we assume that the adversary needs minimal (but mandatory) physical access to steal and carry away an e-scooter. For example, in a proximity-based scenario, the attacker can approach the e-scooter when the victim is not present and perform some short interactions with its dashboard (e.g., pressing the headlight and the power buttons). Alternatively, in a remote threat scenario, two adversaries can collude. For example, an adversary unlocks the e-scooter by launching a remote attack via the malicious app. At the same time, the other adversary can press any button (if necessary) and steal the e-scooter.

## 4.4 Reversed Xiaomi Security Protocols

We describe the *four* proprietary Xiaomi protocols that we reverse-engineered (RE). We discover that Mi Home and the e-scooter establish an *insecure* link-layer BLE connection, despite both devices supporting BLE security mechanisms (e.g., BLE Pairing). Instead, Xiaomi uses *proprietary application-layer protocols* to secure their whole e-scooter ecosystem.

Table 4.1 summarizes the details we reversed from the protocols. We label the protocols as P1, P2, P3, and P4, and also assign a descriptive name to each one. P1 is named "No security" because it does not utilize any security mechanism. P2 is named "XOR obfuscation" because it employs an obfuscation strategy exclusively based on XOR. P3 is named "AES-ECB and XOR obfuscation" because it XORs Xiaomi packets with the output of an AES-ECB cipher. P4 is named "ECDH and AES-CCM" because it employs ECDH for Pairing and AES-CCM during Session. Then, we isolate the protocols' phases: *Pairing* (e.g., key agreement), and *Session* (e.g., authenticated encryption). For instance, P2 Pairing is based on a public XOR mask and is unauthenticated. Its Session reuses the XOR mask to obfuscate payloads, is not authenticated, and provides no integrity protection. Our experiments reveal that all Xiaomi e-scooters (more specifically, their BLE subsystems) and all Mi Home versions (from 2016 to 2021) have employed these protocols. Now we describe each protocol in detail.

### 4.4.1 No Security (P1)

P1 provides no security guarantees as it lacks Pairing and Session capabilities. The devices establish a BLE connection and then exchange the application-layer payloads in cleartext without integrity protection. The only roadblock for the attacker to eavesdrop and inject packets into

Table 4.1: The four Xiaomi application-layer security protocols analyzed in this work. The first and second columns show the protocol ID and name. Each protocol has a Pairing and Session phase. P4 has two Session versions, where v2 is equal to v1 but adds downgrade protection. Unil means unilateral.

| ID | Name | Pairing |
|---|---|---|
| P1 | No security | None |
| P2 | XOR obfuscation | Public XOR mask, no auth |
| P3 | AES-ECB and XOR obfuscation | Weak AES-ECB key agreement, no auth |
| P4 | ECDH and AES-CCM | ECDH, AES-CCM unil auth |
| **ID** | **Name** | **Session** |
| P1 | No security | None |
| P2 | XOR obfuscation | XOR mask obfuscation, no auth, no integrity |
| P3 | AES-ECB and XOR obfuscation | XOR obfuscation, implicit auth, no integrity |
| P4 | ECDH and AES-CCM | v1: HKDF, HMAC, AES-CCM, mutual auth<br>v2: v1 with downgrade protection |

the connection is the knowledge of the application-layer packet format. P1 is the prototypical example of *security through obscurity (STO)*.

### 4.4.2 XOR Obfuscation (P2)

P2 offers no security guarantees, but relies on a XOR-based obfuscation strategy. During Pairing, Mi Home reads a twelve-byte *XOR mask* from the e-scooter Hardcopy Data Channel GATT characteristic, different at every reboot of the device. Then, during Session, the devices obfuscate the application-layer payloads by XORing them with the XOR mask. If the payload is longer than the XOR mask, the app asks the e-scooter for the extended version of the same XOR mask and uses that one instead in the XOR operation. Since the attacker can trivially recover the mask (e.g., eavesdropping or reading it from the e-scooter), P2 is insecure and falls into the STO category.

### 4.4.3 AES-ECB and XOR Obfuscation (P3)

P3 uses a weak key establishment protocol based on AES-ECB and XOR obfuscation. Pairing generates a sixteen-byte pairing key (`pk`) by computing `pk=AES-ECB(key=constant,input=escooter_name)`, where `constant` is *hardcoded* both in the Mi Home app and in the e-scooter BLE firmware, and `escooter_name` is publicly advertised by the device. Then, during Session, the devices obfuscate the application-layer payloads by XORing them with `pk`. If the payload is longer than the pairing key, the payload is XORed with an extended pairing key, which is just `pk` repeated as many times as necessary. P3 provides no security guarantees but only STO. An attacker can compute `pk` by extracting `constant` from the reversed code of any Mi Home APK and trivially acquire `escooter_name`. Once `pk` is known, the attacker can de-obfuscate and inject valid P3 packets.

### 4.4.4 ECDH and AES-CCM (P4)

During Pairing, P4 employs *Elliptic Curve Diffie-Hellman (ECDH)* for key agreement and unilateral pairing key authentication. In particular, the e-scooter sends `chal`, a sixteen-byte random challenge. The devices exchange their public keys, using the *SECP256R1* curve, and derive `ss`, an ECDH shared secret. Then, they compute a pairing key (`pk`) and a one-time key (`otk`) using HKDF as follows: `pk||otk=HKDF(key=ss,input="mible-setup-info",salt="")`. The app responds to the e-scooter challenge with `resp=AES-CCM(key=otk,input=chal)`.

During Session, P4 uses HKDF, to derive the directional session keys, and HMAC-based mutual authentication. The devices exchange `rand_esc` and `rand_app`, two sixteen-byte random numbers. The devices derive two directional session keys (`sk_esc` and `sk_app`) and AES-CCM nonces (`n_esc` and `n_app`) as follows: `sk_esc||sk_app||n_esc||n_app=HKDF(key=pk,input="mible-login-info",salt=rand_app||rand_esc)`.

Then, the e-scooter sends `resp_esc=HMAC(key=sk_esc,input=rand_esc||rand_app)` to authenticate its session key. Similarly, the app authenticates its directional key by sending `resp_app=HMAC(key=sk_app,input=rand_esc||rand_app)`. After mutual authentication of both session keys, each device employs AES-CCM to encrypt and integrity protect the application-layer payloads. AES-CCM is keyed with the directional session key and initialized with the directional nonce concatenated with a packet counter.

P4 provides security guarantees (unlike P1, P2, and P3) but can be downgraded. Replay attacks are ineffective against P4 Pairing and Session because the former utilizes a random challenge during pairing key authentication, and the latter utilizes random values and nonces during the HMAC-based authentication. Moreover, the mutual authentication during P4 Session prevents impersonation attacks on Mi Home or the e-scooter. The usage of session keys limits the impact of a compromised key to the current session only, and the usage of a packet counter in the encryption of regular BLE communication protects against nonce reuse attacks.

P4 protocol comes in two versions (i.e., P4v1 and P4v2), depending on the supported version of the Session phase.

## 4.5 Attacks

We present *four novel attacks* targeting the four Xiaomi custom protocols discussed in Section 4.4 that enable stealing a (password-protected) e-scooter or denying a victim from using it via Mi Home. Our attacker can either be *proximity-based* or *remote*, as stated in Section 4.3. The attacks achieve their goals by using one of two spoofing strategies: (i) the attacker pairs with the target e-scooter while impersonating any user, i.e., *Malicious Pairing (MP)* (ii) the adversary connects to the target e-scooter and downgrades the session to an insecure version, i.e., *Session Downgrade (SD)*.

The attacks are critical to the Xiaomi ecosystem as they exploit the four Xiaomi application-layer security protocols at the *architectural level*. Hence, they are effective regardless of the e-scooter's hardware and software details, including its model, and only depend on the BLE firmware being run. Moreover, they defeat the most secure setup, i.e., a password-protected and software-locked e-scooter already paired with a registered Xiaomi user. We even completed the attacks while the e-scooter was in motion (in a controlled environment). We now describe the MP and SD strategies, and we isolate their root causes.

### 4.5.1 Malicious Pairing (MP)

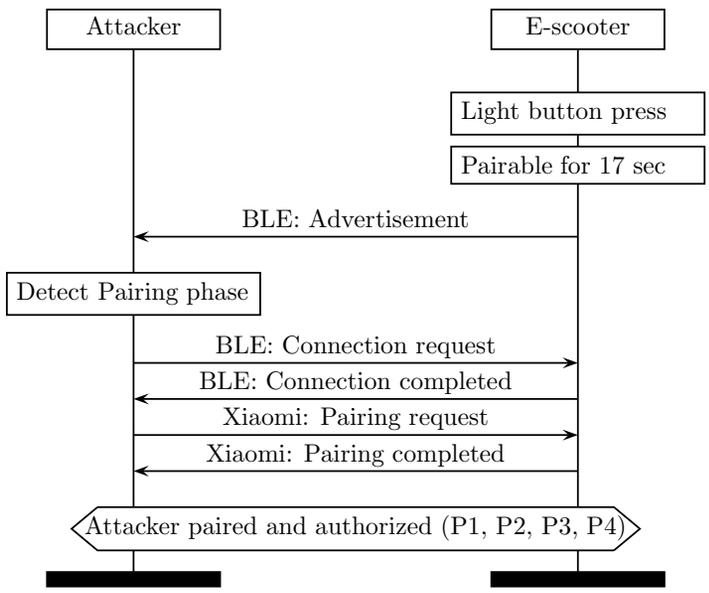



Figure 4.3: Malicious Pairing (MP) attack strategy. The user presses the headlight button. The e-scooter goes into pairable mode for seventeen seconds and advertises it via BLE. The attacker detects the Pairing phase supported by the e-scooter. Then, she establishes a BLE connection without impersonating the victim's smartphone and completes Xiaomi Pairing. As a final result, she is authorized to send any Xiaomi-compliant command to the e-scooter, including lock, unlock, and set or change a password.

Figure 4.3 shows the MP attack strategy that can be used to lock an e-scooter away from its user, or to steal it. The attacker waits until the victim presses the e-scooter headlight button to switch on or off the front light (or presses the button if the e-scooter is unattended). As a *side effect*, the button press activates pairing mode for the e-scooter for *seventeen seconds* without notifying the user. The adversary detects that the e-scooter is pairable from its BLE advertisement packets and detects which Xiaomi protocol it supports (i.e., P1, P2, P3, or P4). Then, she establishes a BLE link-layer connection *without* spoofing the victim's smartphone BLE address. Hence, the adversary can target an e-scooter without knowing any information about its owner (e.g., any e-scooter in a parking lot).

Finally, the attacker sends a Xiaomi-compliant pairing request and completes Pairing, regardless of the supported Xiaomi protocol of the e-scooter. Once paired, she can perform any action requiring authentication. For example, she can lock it and set a new e-scooter password to prevent the victim from accessing it from Mi Home. The takeover is effective, as we discovered that Mi Home does *not* allow resetting the e-scooter password, even with a factory reset. Alternatively, the attacker can use the MP strategy to unlock and steal the e-scooter.

The MP attack strategy is effective for a proximity-based attacker inside the BLE range of the e-scooter, and for a remote attacker controlling a malicious app while the victim's smartphone is within BLE range of the e-scooter. Moreover, the strategy works regardless of the Pairing phase version and the e-scooter password because, while pairing, the attacker does not have to authenticate its identity and provide the password.

### 4.5.2 Session Downgrade (SD)

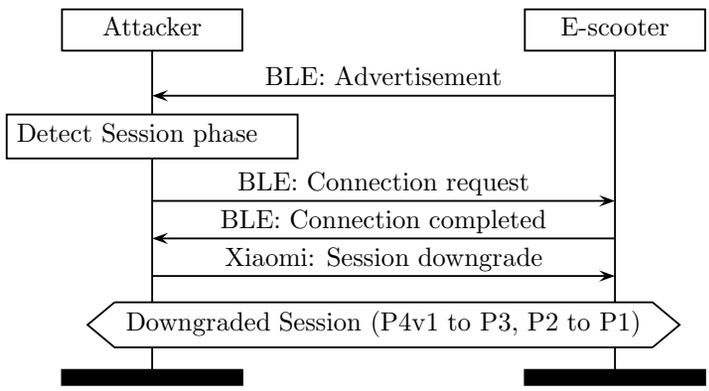



Figure 4.4: Session Downgrade (SD) attack strategy. The app detects a nearby e-scooter vulnerable to SD (i.e., running P4v1 or P2). The attacker skips Pairing and sends the session downgrade command to the e-scooter. The Session is downgraded from P4v1 to P3, or from P2 to P1.

Figure 4.4 shows the SD attack strategy that allows an adversary to lock an e-scooter away from its user, or to steal it. The attacker detects the Session protocol supported by the e-scooter. Then, she looks at the BLE advertisement packets of the e-scooter, and she detects if the target runs P4v1 or P2, being the two Session protocols that expose a session downgrade command. She establishes a BLE link-layer connection without spoofing anything from the victim. Hence, the adversary can target an e-scooter running P4v1 or P2 without knowing any information about its owner. Then, the attacker sends a Xiaomi-compliant session downgrade command, downgrading the Session from P4v1 to P3 or from P2 to P1. The attacker exploits the insecure P3 and P1 to perform dangerous actions on the e-scooter. Similarly to MP, she can lock the e-scooter and prevent access to it from Mi Home by setting a new e-scooter password. She can also use the SD strategy to unlock and steal the e-scooter. The SD strategy can be applied to our proximity-based and remote threat models. The strategy entirely skips Pairing and starts an insecure downgraded Session, removing any authentication requirement from the attacker. Moreover, the strategy is particularly effective on e-scooters running P4v1, as they offer security guarantees that are nullified by downgrading the Session phase to the insecure P3.

### 4.5.3 Root Causes

The four attacks presented above are enabled by the following *six* root causes (i.e., vulnerabilities) that we isolated in the Xiaomi protocols' specification:

**V1: Unauthenticated Pairing** None of the Pairing phases require *device authentication* (e.g., via a certificate signed by Xiaomi). Hence, an attacker can pair with an e-scooter while spoofing an arbitrary Mi Home app without authenticating, regardless of the application-layer protocol used by the e-scooter (i.e., P1, P2, P3, or P4).

Table 4.2: Mapping between the vulnerabilities (rows) and the presented attacks (columns). We put a ✓ if an attack exploits a vulnerabilit. Otherwise, we put an ✗. We split our two attacks, Malicious Pairing (MP) and Session Downgrade (SD), depending on their threat model, either proximity-based or remote.

| | **Proxim.** | | **Remote** | |
|---|---|---|---|---|
| **Vulnerability** | **MP** | **SD** | **MP** | **SD** |
| V1: Unauthenticated Pairing | ✓ | ✗ | ✓ | ✗ |
| V2: Unintentional Pairing mode | ✓ | ✗ | ✓ | ✗ |
| V3: Improper e-scooter passw. enfor. | ✓ | ✓ | ✓ | ✓ |
| V4: Unprotected sensitive memory | ✓ | ✓ | ✓ | ✓ |
| V5: Downgradable and insec. Session | ✗ | ✓ | ✗ | ✓ |
| V6: No BLE sec. despite device support | ✗ | ✓ | ✗ | ✓ |

**V2: Unintentional Pairing mode** Pressing the e-scooter's headlight button activates pairing mode for *seventeen seconds* without notifying the user. Hence, whenever the victim presses the headlight button, an attacker in the BLE range of the e-scooter can detect that the e-scooter is pairable from its BLE advertisements and pair. Alternatively, given physical access, the attacker can trigger pairing mode while the victim is away, by simply pressing the headlight button (even if the e-scooter is software-locked).

**V3: Improper e-scooter password enforcement** The e-scooter does *not* enforce the password set by the user via Mi Home. Only Mi Home checks it to prevent unauthorized access to the e-scooter from the victim's smartphone. Therefore, an attacker can tamper with a password-protected e-scooter without knowing the password. Moreover, Mi Home does not provide a way to *deactivate* the password, and the password does *not* change across factory resets. If the adversary changes the e-scooter password, she prevents the victim from controlling the e-scooter via Mi Home.

**V4: Unprotected sensitive memory** Xiaomi custom protocols include an unauthenticated command to read and write *sensitive* memory regions. For example, the attacker can read and overwrite the victim's password from the e-scooter DRV subsystem memory. Moreover, she can tamper with the BLE subsystem memory to lock, unlock, reboot, and shut down the e-scooter.

**V5: Downgradable and insecure Session** Xiaomi custom protocols include unauthenticated commands to downgrade the Session phase. For instance, the attacker can downgrade a P4v1 Session to a P3 Session and a P2 Session to a P1 Session. At the same time, P1, P2, and P3 Session phases are insecure and provide no confidentiality, authenticity, or integrity guarantees. P1 uses no key, P2 employs XOR-based obfuscation with a constant XOR mask, and P3 uses a slightly more complex, yet predictable, obfuscation based on AES-ECB and XOR operations.

**V6: No BLE security despite device support** Xiaomi does not employ BLE security at the link-layer despite device support but relies solely on its custom security mechanisms at the application-layer. So, there is no defense in depth, and the application-layer is a single point of failure.

Table 4.2 maps the six root causes of the four attacks described earlier. MP attacks exploit V1, V2, V3, and V4. V1 and V3 lower the attack requirements. V2 allows attacks from proximity without requiring physical access to activate pairing mode. V4 enables dangerous operations on the e-scooter by unauthorized attackers. SD attacks exploit V3, V4, V5, and V6. V3 and V6 lower the attack requirements. V4 enables dangerous operations on the e-scooter. V5 makes SD possible.

## 4.6 Implementation

Here, we present `E-Spoofer`, a new toolkit to carry out the four attacks presented in Section 5.4, facilitate further reverse-engineering of Xiaomi protocols and help in future security evaluations in the Xiaomi e-scooter ecosystem.

### 4.6.1 Proximity Attack Module

The `E-Spoofer` proximity attack module performs proximity-based MP and SD *over-the-air*, using BLE. We use Noble [150], a NodeJS module, to create a BLE central that spoofs the Mi Home app and speaks Xiaomi protocols. We replicate P4 Pairing and P4v1 Session, as these protocols are available on all (up-to-date) Xiaomi e-scooters.

Our module reimplements P4 Pairing, including ECDH and pairing key authentication. We perform ECDH and obtain a shared secret. We receive a challenge from the e-scooter. We derive a pairing key and a one-time key from the shared secret by running HKDF-SHA256. We utilize the one-time key and the challenge for the sophisticated pairing key authentication by running AES-CCM-128. Finally, we send the solution to the e-scooter and complete P4 Pairing.

Our module reimplements P4v1 Session, including the HMAC-based mutual authentication and AES-CCM encryption. We send a challenge to the e-scooter, and receive a challenge from him. We retrieve the pairing key generated during Pairing. We derive the directional session keys and IVs from the pairing key and the two challenges by running HKDF-SHA256. Then, we use the directional session keys and IVs to calculate the solution of the e-scooter challenge by running HMAC-SHA256. Finally, we encrypt with AES-CCM-128 the BLE commands (e.g., session downgrade, lock or unlock the e-scooter, setting or changing the password) using the session keys and IVs, and the packet count. The above-mentioned cryptographic operations also require other input values found in the decompiled Mi Home code, identical for all e-scooter models.

### 4.6.2 Remote Attack Module

The `E-Spoofer` remote attack module performs the attacks using a *malicious Android app*. The app acts as a BLE central, spoofing Mi Home and speaking Xiaomi protocols. It detects

a vulnerable e-scooter via its BLE advertisement, by analyzing the info included in the advertisement itself (i.e., e-scooter name, model, security level, and pairing mode activation). When an e-scooter is found in pairing mode, the app will pair and perform MP or SD. We develop the app using the RxAndroidBle library [151], built on RxJava.

Our malicious app requires no root privileges but Bluetooth and location-related permissions. On Android 9 or lower, these permissions are BLUETOOTH, BLUETOOTH_ADMIN, and ACCESS_COARSE_LOCATION. Android 10 and 11 require ACCESS_FINE_LOCATION instead of coarse locatio. On Android 12 or higher, the app requires the BLUETOOTH_CONNECT and BLUETOOTH_SCAN permissions.

### 4.6.3 Reverse-Engineering Module

The `E-Spoofer` reverse-engineering module contains the protocol dissectors, Ghidra utilities, and Frida hooks that we developed while statically and dynamically RE the Xiaomi e-scooter ecosystem. The research community can use these modules to perform other experiments on the Xiaomi ecosystem or adapt them to test similar ecosystems. We now describe each submodule.

**Protocol Dissectors** We develop Pyshark *dissectors* that automatically parse BLE captures and detect custom Xiaomi payloads and advertisement packets. They identify the Xiaomi protocol version from the packet header and dissect the packet accordingly. We also develop Scapy scripts to complement the Pyshark dissectors and offer a more advanced analysis.

We develop an advertisement packet analyzer for Xiaomi e-scooters. Our script extracts the name of the scooter (e.g., MIScooter1234), the scooter model (i.e., 0x20 for M365), the security level (i.e., 0x00 for P1, 0x01 for P2, and 0x02 for P3 and P4) and pairing mode activation (i.e., 0x01 means not active, 0x02 means active).

**Ghidra Utilities** We utilize *Ghidra* [152] to statically RE portions of the e-scooter's BLE firmware. We used the open-source mijia library [153] to identify some compiled functions in the firmware, related to BLE advertisement and cryptographic mechanisms (e.g., AES, HKDF). We manually name the functions related to Pairing and Session and release six YARA [154] rules with their signatures to identify them automatically. We also release our Ghidra project files to reproduce our setup, as part of `E-Spoofer`.

We discover how the session downgrade command is implemented in the BLE firmware, and why P4v2 does not support it. A static memory *flag* decides whether P3 packets (including the downgrade command) are accepted or discarded. Firmware running P4v1 enables this flag, thus becoming vulnerable to SD. Firmware running P4v2 disables this flag, thus discarding the downgrade command and becoming immune to SD. We did not find any way to exploit this flag, unreachable by the *unprotected sensitive memory* (V4) root cause presented in Section 4.5.3.

**Frida Hooks** First, we decompile the Mi Home APK. We navigate the decompiled code to find the classes and functions involved in Xiaomi security mechanisms and we write down their signature. Then, we develop Frida [106] hooks to intercept these calls. We print the input and output values, and we modify them if needed. In particular, we cast the key to their proper classes, before printing or altering them. Our hooks are written in Javascript and can be run by invoking the Frida client from the console, while connected to a rooted smartphone running a Frida server. Operating with Mi Home, will print logs on the console.

## 4.7 Evaluation

In this section, we evaluate the four attacks presented in Section 5.4 against *eight* attack scenarios. We cover P1, P2, P3, and P4 – the proprietary Xiaomi application-layer protocols reversed in Section 4.4, three popular Xiaomi e-scooters models (i.e., M365, Essential, and Mi 3), five Xiaomi BLE subsystems (i.e., M365, Pro 1, Pro 2, Essential, and Mi 3), eight e-scooters' BLE firmware, and the Mi Home app for Android and iOS. We now describe our setup and results.

### 4.7.1 Setup

Our evaluation setup enables experimenting with multiple e-scooters and BLE configurations by using three e-scooters (M365, Essential, and Mi 3) configured to host different BLE subsystems and firmware. We bought the three e-scooters from Amazon for around 1.000 USD. We get access to their BLE subsystem board by unscrewing the dashboard and removing the display. This way, we broaden our evaluation while limiting the evaluation costs. For example, by installing the Pro 1 and Pro 2 BLE subsystems and firmware on the M365 e-scooter, we can test the Pro 1 and Pro 2 subsystems *without* spending hundreds of USD to buy the actual e-scooter.

We test five BLE subsystem boards with eight BLE firmware. Three boards are original parts of M365, Essential, and Mi 3 e-scooters. Two are clone boards for Pro 1 and Pro 2. The M365 and Pro 1 subsystems include an *nRF51822* SoC [155] of the QFAA variant (16 KB of RAM). Instead, the other subsystems use the QFAC variant with 32 KB of RAM. We obtain the BLE firmware from the ScooterHacking repositories [156] or the Mi Home app. We identify each firmware's relative proprietary protocol (i.e., BLE072 runs P1, BLE081 runs P2, BLE090 runs P3, BLE122, BLE129, BLE152, and BLE153 run P4v1, BLE157 runs P4v2).

To debug and manage the BLE subsystems, we use the *ST-Link V2 debugger* [157], which is compatible with the nRF51 SoC family. Attaching the debugger to a subsystem board requires manual effort, such as soldering the data (SWDIO), clock (SWCLK), and power wires. We also remove discrete components to unlock hardware-based debugging (i.e., C16 and R1 on the M365 BLE board, C2 on the Pro 1 BLE board). Once debugging was unlocked, we could run a GDB server for runtime debugging and operate on the SoC RAM with tools such as OPENOCD [158], PySWD [159], MiDu Flasher [160], and nRFSec [161]. Runtime access to the subsystem boards was essential to produce the presented results. For example, via GDB, we discovered that the e-scooters store the cleartext password in RAM, and via firmware flashing, we restored a BLE subsystem in an unbricked state after tampering with it.

On the app side, we test Mi Home for Android and iOS on three smartphones. We evaluate a rooted Pixel 2 running Mi Home v7.11.704 and Android 11, a rooted Oneplus 3 with Android 9 and a Realme GT with Android 12, both running Mi Home v7.6.704, and an iPhone 7 running Mi Home 7.12.204 and iOS v15.7. Our attacks do *not* require rooting a smartphone; we only need root privileges when dynamically instrumenting Mi Home with Frida.

We run `E-Spoofer`, the novel toolkit we present in Section 4.6, from two attacking devices. We deploy our proximity-based MP and SD attacks from a laptop (i.e., Dell Inspiron 15 3000). We select the desired attack from the command line, the victim e-scooter from a list of nearby targets, and the script automatically performs MP or SD, displaying visual feedback. We deploy our remote MP and SD attacks from a smartphone (e.g., Pixel 2). Through the UI of

our malicious app, we scan for nearby targets, connect to a victim e-scooter, and perform MP or SD.

### 4.7.2 Results

Table 5.4 shows our evaluation results. The first two columns indicate the BLE firmware version and the protocol they run. The third column represents the e-scooter model, which hosts the BLE subsystem shown in the fourth column. We specify the SoC variant of the BLE subsystem board in column five. The remaining columns highlight whether a BLE firmware version is vulnerable to MP and SD in their proximity-based and remote variant.

In our attack scenarios, we exploit eight unique BLE firmware, including the latest firmware available on the M365, Essential, and Mi 3. We test the four Xiaomi proprietary protocols we identified, including the two variants of P4 Session (i.e., P4v1 and P4v2), and flash them on five BLE subsystems from different e-scooter models. We confirm that BLE subsystems using the nRF51822 QFAA SoC are incompatible with newer e-scooters models (i.e., Essential, Mi 3), as the latter requires BLE subsystem boards with the nRF51822 QFAC SoC. Similarly, newer boards cannot be installed on the M365. We demonstrate that *all* evaluated BLE subsystems, regardless of their application-layer protocol, are vulnerable to the MP attacks. This happens due to authorization and authentication issues in all four Xiaomi protocols that we discuss and fix in Section 4.8. We also demonstrate that *all* evaluated BLE subsystems running P4v1 or P2 are vulnerable to SD to P3 or P1. We highlight that P1, P2, and P3 have no security guarantees compared to the more secure P4. This fact makes SD from P4v1 to P3 particularly threatening. We confirm that P4v2 is immune from the SD attacks, as discussed in Section 4.6.

Our `E-Spoofer` toolkit proved to be effective on all evaluated Xiaomi e-scooters. Unfortunately, we could not evaluate the Xiaomi Mi 4 e-scooter due to its release time (end of 2022). `E-Spoofer` can be easily extended to support any e-scooter ecosystem that protects their communications with a proprietary application-layer protocol on top of BLE, including the Xiaomi Mi 4 e-scooter. To attain this goal, future researchers will have to reverse-engineer the proprietary application-layer protocols run by that specific e-scooter ecosystem. We also confirm that our toolkit can change the unknown e-scooter password set by an adversary, restoring the user capability of accessing and managing the e-scooter from Mi Home, as a post-attack defence.

During our experiments, we even identified and disclosed a severe *UI authentication* bug in Mi Home for Android and iOS. From Mi Home v7.6.704 onwards, the user can lock or unlock a password-protected e-scooter *without* entering the password. The cause is a 1 second UI delay between the app wake-up and the password prompt. We confirmed this bug using the same smartphones we describe in Section 4.7.1. Since the password is only checked by Mi Home, due to the `improper e-scooter password enforcement` (V3) root cause we discuss in Section 4.5.3, the attacker can bypass app-based password protection, unlock the e-scooter, and steal it. As described in the responsible disclosure paragraph, Xiaomi acknowledged this bug, rewarding us with a bounty, but gave no information about a fix.

## 4.8 Countermeasures

To address the four impactful attacks described in Section 5.4, we design and evaluate two *usable*, *backward-compliant*, and *low-cost* countermeasures. The first countermeasure stops the

Table 4.3: Evaluation results. The first and second columns represent the BLE firmware version and the Xiaomi protocol version. The third column states the e-scooter model, which hosts the BLE subsystem board, specified in the fourth column, indicating if the BLE board is original from Xiaomi or a clone. The fifth column specifies the System-on-Chip present on the BLE subsystem. The last four columns highlight if the evaluated combination is vulnerable to our proximity-based and remote Malicious Pairing (MP) and Session Downgrade (SD) attacks. A hyphen (-) means the attack does not apply to that target.

| | | | | | **Proximity** | | **Remote** | |
|---|---|---|---|---|---|---|---|---|
| **Firmware** | **Protocol** | **E-Scooter** | **BLE Sub. Board** | **SoC** | **MP** | **SD** | **MP** | **SD** |
| BLE072 | P1 | M365 | M365 (Original) | nRF51822 QFAA | ✓ | - | ✓ | - |
| BLE081 | P2 | M365 | M365 (Original) | nRF51822 QFAA | ✓ | ✓ | ✓ | ✓ |
| BLE090 | P3 | M365 | Pro 1 (Clone) | nRF51822 QFAA | ✓ | ✗ | ✓ | ✗ |
| BLE122 | P4v1 | M365 | M365 (Original) | nRF51822 QFAA | ✓ | ✓ | ✓ | ✓ |
| BLE129 | P4v1 | M365 | Pro 2 (Clone) | nRF51822 QFAC | ✓ | ✓ | ✓ | ✓ |
| BLE152 | P4v1 | Essential | Essential (Original) | nRF51822 QFAC | ✓ | ✓ | ✓ | ✓ |
| BLE153 | P4v1 | Mi 3 | Mi 3 (Original) | nRF51822 QFAC | ✓ | ✓ | ✓ | ✓ |
| BLE157 | P4v2 | Mi 3 | Mi 3 (Original) | nRF51822 QFAC | ✓ | ✗ | ✓ | ✗ |

MP attacks by providing a stronger pairing mechanism that is appropriately authorized and authenticated. The second countermeasure fixes the SD attacks by patching away the hidden downgrade command from the vulnerable e-scooter BLE firmware. We now describe them in detail and release them as part of `E-Spoofer`.

### 4.8.1 Authorized and Authenticated Pairing

The MP attacks presented in Section 4.5.1 are enabled by authorization and authentication issues affecting P1, P2, P3, and P4 Pairing phases. We develop a better pairing phase addressing both issues in a backward-compatible way. This countermeasure addresses the *unauthenticated pairing* (V1), *unintentional pairing mode* (V2), and *improper e-scooter password enforcement* (V3) root causes from Section 4.5.3. We now describe how we provide authorization and authentication during pairing.

**Authorized Pairing Mode** We require the Xiaomi Pairing phase to implement a *dedicated* pairing activation command that also *notifies* the user. In particular, to enter pairing mode, the user must press the headlight button while holding down the left brake. Then, the e-scooter's tail light should blink until the completion of Pairing. This fix prevents unexpected and unnotified pairing sessions such as the ones exploited in the MP attacks by waiting until the victim presses the headlight button. The fix is trivial to implement for Xiaomi as it requires minimal modifications to the BLE firmware. On our side is challenging to test as we do not have access to the BLE firmware source code and build tools.

**Password-Protected Authenticated Pairing** We require a password protected pairing protocol to prevent an unauthenticated attacker from pairing with a victim e-scooter. This fix prevents the MP attacks even if the adversary manages to put the e-scooter in pairing mode. This countermeasure is easy to implement by extending the Mi Home password protection functionality. In particular, while pairing an e-scooter with Mi Home for the first time (including after a factory reset), the user should set a password via Mi Home. Then, the password should be stored on Mi Home and the e-scooter and enforced in case of re-pairing. Hence, an attacker cannot maliciously pair with the e-scooter as she cannot provide the password to the e-scooter. We successfully evaluated this fix using our toolkit to replicate P4 Pairing between an e-scooter and Mi Home.

### 4.8.2 Anti-Downgrade BLE Firmware Patching

The SD attacks presented in Section 4.5.2 are enabled by a vendor-specific command, which downgrades Xiaomi Session P4v1 to P3, and P2 to P1. We focus on patching P4v1 because e-scooter running the insecure P2 should update their BLE firmware to the latest version. Regardless, the downgrade command is present even in recent BLE firmware versions, including the latest M365 and Pro 1 BLE firmware. We release a script capable of finding and removing the downgrade command from a vulnerable BLE firmware to fix this issue. Our script addresses the *downgradable and insecure Session* (V5) root cause presented in Section 4.5.3.

The script looks for a specific conditional statement and patches it to allow only P4 Session. Hence, the patch introduces no overheads (e.g., memory, computation). Our scripts opens the binary firmware, finds the function responsible for BLE packet analysis, and alters the conditional statement that accepts either P3 and P4 packets, causing it to only accept P4 packets. More specifically, it replaces the `cmp` instruction `5a2f` with `552f`. As a result, the attacker can neither downgrade P4v1 to P3, nor send any other insecure P3 command.

Developing the script required a one-time manual overhead to understand how to remove the downgrade command. Then we *automated* our binary-patching process. We reuse the BotoX M365 patcher tool [162] to encrypt the patched firmware with the Tiny Encryption Algorithm (TEA). We reuse the third-party M365DownG app [163] to flash the zipped and newly encrypted BLE firmware.

We successfully evaluated our fix on the M365 and Pro 1 e-scooters. We flashed a patched BLE122 firmware on the e-scooters and deployed the proximity-based and remote SD attacks. Both attacks failed, as downgrading the protocol from P4v1 to P3 was impossible with our fix.

## 4.9 Related Work

**E-Scooters Security and Privacy Issues** Academic research on e-scooter security and privacy is scarce, especially on personal e-scooters. Zimperium, a mobile security company, exploits the locking system to stop a running e-scooter [32]. The hacker Lanrat evaluated M365 authentication, discovering that it is not enforced by the e-scooter [31]. Both attacks were publicly disclosed in 2019 and only targeted the Xiaomi M365 model. In our work, we target all Xiaomi e-scooter models from 2016 to 2021.

Security researchers focused on e-scooter rental ecosystems instead of private e-scooters. In [13], the authors identify some vulnerabilities in the APIs exposed by the Bird e-scooter

sharing platform, which utilizes M356 e-scooters [164]. Public e-scooters from the Lime sharing company are weak to a man-in-the-middle attack that allows for arbitrarily swapping audio files [33]. N. Vinayaga-Sureshkanth et al. [25] provide an extended evaluation of Android e-scooter rental applications. In particular, they investigate the user-related data collected and shared with third parties, which could monitor the users' schedules and visited locations. In our work, we perform a security assessment. Therefore, we consider out-of-scope any privacy study on user data.

**E-Scooters Hacking Communities** ScooterHacking [165] is the largest e-scooter hacking community with around 20.000 members. ScooterHacking releases hacking tools [156] and offers a third-party companion app [166] for Xiaomi e-scooters. Expert users can download custom DRV and BLE firmware to alter the e-scooter performances (e.g., maximum speed). Alternatively, users can build their DRV firmware with the ScooterHacking Custom Firmware Toolkit [167] and the BotoX Xiaomi M365 Firmware Patcher [162]. These tools offer limited customizability as they can only binary patch hardcoded and unsigned portions of the firmware.

Third-party researchers provided non-peer-reviewed blog posts about the BLE traffic exchanged by some Xiaomi e-scooters [57, 58, 59, 168]. These resources helped in the initial stage of our work but failed short on the technical details and e-scooter coverage. For example, some report confuses encryption with obfuscation, giving a false sense of security. Or none of the reports cover the session downgrade command, and the flag responsible for it. This work instead provides the first comprehensive and sound description and security evaluation of these protocols.

**Security Analysis of Xiaomi Ecosystems** Xiaomi manages multiple ecosystems, including e-scooters, smartphones, smart home devices, and fitness trackers. In [75], the authors root a Xiaomi vacuum cleaning robot, inspect its internals, assess data privacy, and flash the robot with custom firmware. Another previous work [34] also finds several security issues with Xiaomi vacuum cleaners. Several researchers [146, 41, 169, 24] highlight the limitations of the Xiaomi application-layer protocols run over BLE by the Mi Band fitness trackers. These devices were found vulnerable to eavesdropping, man-in-the-middle, and impersonation. Using a fuzzing approach, X. Du et al. [20] find 95 vulnerabilities in the R1D Xiaomi router. Other Xiaomi IoT devices evaluated in the academic literature are Xiaomi smart speakers [19] and Xiaomi security cameras [35, 36].

**BLE Misuse in Android** Researchers identified multiple flaws in Android BLE APIs. For example, Android saves Bluetooth keys in data structures shared among different apps [49, 50], allowing malicious apps to communicate illegitimately with paired devices. In [170], V. Toubiana et al. present a vulnerability, available from Android 6 to Android 11, that allows an Android app to perform a BLE scan without requiring location permission. Android applications may also misuse the BLE link-layer, allowing attackers to bypass encryption and authentication procedures [17]. In this paper, we focus on application-layer protocols instead and only utilize Android BLE APIs in our remote threat model.

**Attacks on BLE Pairing** Several attacks over the years have targeted BLE link-layer pairing. In 2013, Crackle [171] broke the Just Works and Passkey modes of BLE Legacy pairing

by brute-forcing their temporary key. In 2019, the KNOB [172] attack minimized the entropy of the encryption key in BLE Legacy pairing and Secure Connections, allowing for brute-force attacks on that key. In 2021, Method Confusion [46] performed a man-in-the-middle attack on BLE Secure Connections by separately pairing two devices in two different pairing modes. Xiaomi e-scooters do not utilize BLE link-layer pairing. Instead, we reverse-engineer and attack the proprietary Xiaomi Pairing phase (and Session) at the application-layer.

## 4.10 Conclusion

We present the first security evaluation of the proprietary security protocols employed by Xiaomi to protect its e-scooter ecosystem since 2016. We uncover and reverse-engineer four protocols using ad-hoc Pairing and Session mechanisms at the application-layer on top of an insecure BLE link-layer. We describe their (lack of) security properties.

We show four novel attacks to exploit protocols at the specification level requiring realistic and low-cost attacker models (i.e., a proximity-based adversary with a laptop or remote attacker who installed a malicious app on the victim's smartphone). The attacks enable stealing a software-locked and password-protected e-scooter from its owner or preventing the owner from using the e-scooter via Mi Home. The threats pivot on MP and SD attack strategies and are enabled by six severe root causes that we also uncover.

We open-source `E-Spoofer`, a toolkit implementing our attacks and offering RE utilities for the Xiaomi e-scooter ecosystem (e.g., protocol dissectors, Ghidra scripts, and Frida hooks). We successfully evaluate our attacks in eight relevant scenarios covering five e-scooter BLE subsystems and eight BLE firmware. We empirically demonstrate that our attacks have a critical impact on the Xiaomi ecosystem (e.g., all reversed protocols are affected by at least two of our four attacks), amounting to millions of exploitable devices.

We propose two practical, low-cost, and backward-compliant countermeasures to stop our attacks and release them in our toolkit. We propose Authorized Pairing Mode and Password-Protected Authenticated Pairing to fix the MP attacks and a script to stop the SD attacks by automatically patch the vulnerable e-scooter BLE firmware.

# Chapter 5

# E-Trojans

***This contribution, titled "E-Trojans: Ransomware, Tracking, DoS, and Data Leaks on Battery-powered Embedded Systems", is currently under submission at the IEEE International Symposium on Hardware Oriented Security and Trust (HOST 2025).***

## 5.1 Introduction

*Battery-powered embedded systems (BESs)* are an integral part of our society. They include electric cars, e-scooters, e-bikes, drones, smartphones, and laptops. Electric vehicles alone have an estimated market size of USD 422.8 billion [173]. While, e-scooters scooters have a market of USD 37 billion and an estimated annual growth of 10% [174, 175]. These devices carry sensitive information and if misused can cause security and safety issues. Thus protecting BES against (remote) attacks is crucial.

Real-world BES *internals* (i.e., hardware and firmware) are complex and opaque to researchers. They includes, at least, microcontrollers running closed-source firmware, a rechargeable battery, an electric motor, a wireless communication interface such as Bluetooth Low Energy (BLE), and buses including Universal Asynchronous Receiver-Transmitter (UART) and Inter-Integrated Circuit (I2C). Typically, a user manage a BES directly (e.g., smartphone) or via a companion app that wirelessly connect to the BES (e..g, e-scooter).

BES internals represent an attractive *attack surface* which has received limited attention from the community. There are several research papers about the security of externally reachable automotive electric control units (ECUs) [176] and controller area network (CAN) buses [177]. The research on e-scooters focused on external attack surfaces including proprietary protocols over BLE [178] or studying the privacy of e-scooter rental mobile apps [25]. Drone literature highlighted issues on the physical layer [179] and communication protocols [180, 181], such as firmware update and image transfer. Hence, more research is needed in the area of BES internals security (please refer to Section 5.2.1 for an extended motivation).

We fill this gap by presenting the first evaluation of the internal attack surface of *e-scooters*, an important class of BESs. We focus on *Xiaomi* and its subsidiary Segway-Ninebot [182], as they are two market leaders. Xiaomi has released seven e-scooters in the last seven years: M365, Pro, Pro 2, 1S, Essential, Mi 3 (ES3), and Mi 4. We target the *M365* and *Mi 3 (ES3)* models, as they cover two e-scooter generations (2016 – 2024) and are used daily by millions of people [183]. These users include people owning a private e-scooter or ones using rental e-

scooters sold by Xiaomi, such as Bird and Lime [164, 184] We also study *Mi Home*, the Xiaomi e-scooter companion app available for Android [132] and iOS [133].

We spent ten months reverse-engineering (RE) the M365 and ES3 internals. We report our RE findings, including the e-scooters' hardware block diagram, firmware protections, and battery management details. We discover *four design vulnerabilities*: unsigned and unencrypted BMS (V1, V2), and unencrypted and unauthenticated UART/I2C communication (V3, V4). Being design flaws, they are effective on *any* Xiaomi M365 or Mi 3 e-scooters and possibly other Xiaomi e-scooters.

Based on the identified flaws, we propose **E-Trojans**, four novel BES internals attacks that violate the safety, security, privacy, and availability of the Xiaomi e-scooter ecosystem. The attacks combine the vulnerabilities to get (remote) code execution on the BMS and achieve impactful goals. Among them, we present a *BES battery ransomware*, that progressively and irreversibly destroys the e-scooter's battery via undervoltage, unless the victim pays a ransom. Or a *user tracking* attack taking advantage of immutable e-scooter hardware internals. The attacks are conducted remotely via a malicious app or in proximity of an e-scooter via a BLE device. Moreover, they generalize to any BES device that has similar vulnerabilities.

We present `E-Trojans`, a toolkit implementing our attacks and RE findings with open-source software and cheap hardware. It contains a binary patcher to transform a stock battery controller firmware into a malicious one. It can binary patch ten new capabilities, such as turning OFF firmware updates and disabling safety-critical voltage protections. Our toolkit target STM8 chips, but can be extended to other microcontrollers, including the STM32 [185] family. We also include an Android ransom payment app, a Django backend, and a MongoDB database as part of the battery ransomware proof of concept (PoC).

We successfully tested the E-Trojans attacks in a realistic but controlled scenario against up-to-date M365 and ES3 e-scooters. We empirically confirm they achieve their goals. For instance, the undervoltage battery ransomware permanently degrades the M365 battery's autonomy by 50% in three hours and the tracking attack advertises over BLE a reliable user fingerprint and leaks sensitive e-scooter data. To fix our attacks we propose *four backward-compliant* and *low-cost* countermeasures. Our fixes guarantee the confidentiality and integrity of the battery controller firmware via digital signatures, and protect the UART bus with a secure channel protocol and rate limiting.

We summarize our contributions as follows:

- We present the first security and privacy evaluation of Xiaomi e-scooters internals. We RE Xiaomi M365 and ES3 e-scooters architecture and uncover four critical design vulnerabilities, including unsigned battery controller firmware enabling to (remotely) exploit millions of Xiaomi e-scooters.
- We develop four novel attacks exploitable remotely via a malicious application or in proximity via BLE. The attacks include battery ransomware via undervoltage and user tracking via e-scooter internals.
- We release `E-Trojans`, implementing our attacks with open-source software and cheap hardware. The toolkit is usable to further RE the opaque and proprietary Xiaomi e-scooter ecosystem and to target other BESs. We will release it after responsible disclosure.
- We confirm that our four attacks are practical and stealthy by conducting them on M365 and ES3. We propose four effective and legacy-compliant countermeasures to fix attacks.

**Disclosure, availability, and ethics** In November 2023, we responsibly disclosed our findings and PoCs to Xiaomi [148]. Xiaomi acknowledged the report and closed it as informative in March 2024. In April 2024, we contacted ST Microelectronics (ST) and Texas Instruments (TI), the manufacturers of the battery management chips we attack. ST responded that our attacks do not raise any security concern as they are physical. We disagree with their assessment as our attacks are conducted in wireless proximity or even remotely. TI has not responded. We will release our toolkit after the publication of this paper. We conducted our experiments in an *ethical* way: we tested our own e-scooters and smartphones in a laboratory and minimized any safety risk, especially during the undervoltage and overvoltage experiments.

## 5.2 Motivation, RE, and Vulnerabilities

### 5.2.1 Motivation

The BES internal attack surface has received limited attention from academic researchers. In particular, the security of the hardware and firmware used to manage and configure BES batteries remains largely unexplored. Evaluating these attack surfaces poses significant challenges due to the proprietary and vendor-specific protocols and designs that govern them. BES internals are often undocumented and closed-source, creating substantial barriers for security experts, who must resort to reverse-engineering to analyze them.

Only a few works addressed this research area, such as those exploring a rogue firmware update through physical access on a Tesla BMS [186] or discussing theoretical vulnerabilities in MacBook laptop batteries [187]. However, the internal attack surface extends beyond the BMS and batteries. It also includes other subsystems and components, such as the radio, motor, battery monitor, and communication buses. Moreover, there is an abundance of different BES devices outside of electric cars and laptops, for example electric scooters and bikes, mobile phones, drones, and wireless headphones. Compromising these devices can pose serious safety risks to users, including fire hazards and the release of toxic gas.

Given the current literature's narrow focus on a limited subset of BES devices and insufficient coverage of the internal attack surface, we highlight a pressing need for more research on offensive and defensive techniques, as well as tools specifically designed for the BES security. In this work, we fill this research gap by studying the security of e-scooters.

### 5.2.2 Prior RE on Xiaomi E-Scooters

We now introduce what was known about Xiaomi e-scooters internals and what we discovered via RE, including four architectural internal vulnerabilities.

**Block diagram** As shown in Figure 5.1, a Xiaomi e-scooter is composed of three undocumented and closed-source systems: *Bluetooth Low Energy (BTS)*, *Driving (DRV)*, and *Battery Management (BMS)*. BTS provides low-power and reliable wireless communication, DRV controls the electric motor, and BMS manages the battery. Each system has a dedicated System-On-Chip (SoC) and a closed-source firmware. The systems communicate using a proprietary protocol over a Universal Asynchronous Receiver-Transmitter (UART) bus. UART provides a serial, asynchronous, and full-duplex digital communication channel with two wires (Tx, Rx), typically provided as a SoC hardware peripheral.

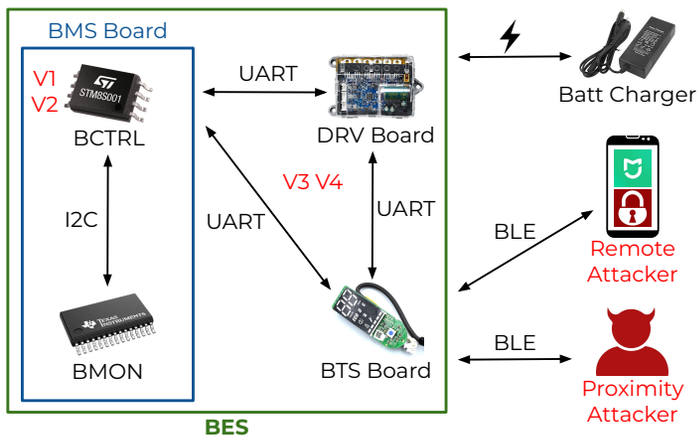



Figure 5.1: E-scooter block diagram and attacker models. The green rectangle shows the e-scooter (BES) internals, such as the BMS, DRV, BTS, and UART communication bus. The blue rectangle (BMS) shows the components of the BMS board, i.e., a BCTRL and a BMON connected via I2C. We consider a proximity-based adversary in BLE range of the e-scooter and a remote adversary who installed a rogue app on the victim smartphone.

**Mi Home** The user interacts with a Xiaomi e-scooter via the *Mi Home* application for Android [132] or iOS [133]. The e-scooter and the app communicate over BLE, using a custom application-layer protocol. There is no way to (directly) talk with the DRV or the BMS, outside from physical access (i.e., through a JTAG or SWIM debugging port).

Device discovery involves BLE advertisement packets from the e-scooter containing its name and BLE address, which the app receives by scanning the BLE spectrum. To connect and communicate with the e-scooter, the user has to log in with their Xiaomi account on Mi Home and pair the app and the e-scooter using the app UI. Then, the paired devices connect whenever they are in BLE range without requiring user interaction. Mi Home offers useful features for controlling the e-scooter. For example, it allows to set a password to lock the e-scooter, perform over-the-air (OTA) firmware updates, and monitor the e-scooter's battery status.

**Prior RE findings** There is limited information about hte internals of Xiaomi e-scooters. In [178], the authors RE the Xiaomi proprietary BLE pairing and session establishment protocols and exploit them with protocol-level attacks (i.e., malicious pairing and session downgrade). Research on the Xiaomi 1S e-scooter [188] reports that Xiaomi encrypts the firmware using TEA with a leaked encryption key [189] and signs it with ECDSA. Firmware binaries include a certificate chain with a public key for signature verification. The BTS is responsible for collecting new firmware from Mi Home and, if needed, decrypting it, verifying its signature, and distributing it to the DRV or the BMS. The developers of the ScooterHacking Utility Android app [190, 191] reversed the Xiaomi firmware update protocol to enable flashing of the modded DRV firmware. Details about the BTS, DRV, and BMS firmware and their communications are still lacking.

### 5.2.3 New Xiaomi E-Scooters RE Details

Starting from the details presented in Section 5.2.2, we RE the internals of the M365 [192] and Mi 3 (ES3) [193] Xiaomi e-scooters. Next, we summarize our RE findings.

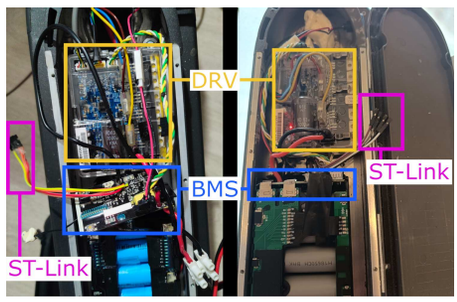



Figure 5.2: Disassembled M365 (left) and ES3 (right). We color code the boxes to visualize the DRV (orange), the BMS (blue), and the soldered ST-Link wires (fuchsia).

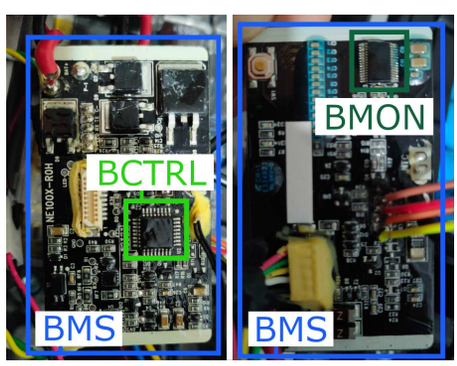



Figure 5.3: M365 BMS: a BCTRL STM8 SoC (left) and a BMON BQ73930 SoC (right).

**Internals** We disassembled the M365 and ES3 to identify their SoCs and internal connections. As shown in Figure 5.2, we tore the lower deck of the e-scooter to access the DRV, BMS, and battery pack. We unscrewed the deck, removed the plastic wrapping and the protective polystyrene from the battery, and pulled out the BMS board. As shown in Figure 5.3, the BMS includes a *battery controller (BCTRL)* and a *battery monitor (BMON)*, located on opposite sides of the BMS board.

The BCTRL is a STM8L151K6 chip [194] from ST with OTA firmware update capabilities [195]. We discovered that the BCTRL has *no hardware debug protection*. Therefore, we identified the BCTRL's VDD, SWIM, GND, and RST pins and soldered them to a *ST-Link v2* programmer/debugger for firmware extraction and debugging. The BMON is a BQ76930 chip [196] from TI and is configurable by the BCTRL. Unlike the BCTRL, the BMON does not support firmware updates [197].

The BCTRL and BMON communicate using a proprietary protocol over I2C, a synchronous serial digital protocol running on two wires (SDA and SCL) and featuring a higher bandwidth than UART. The BCTRL (master) has control over the BMON (slave). We discovered that the proprietary I2C protocol lacks encryption and authentication.

**Firmware signing and encryption** We inspected BMS, BTS, and DRV firmware gathered from Mi Home, the Internet, and the JTAG/SWIM debugging interfaces. Our collection ranges from M365 firmware released in 2019 up to the most recent firmware for the ES3 available in 2024. The M365 firmware is *unencrypted* and *unsigned*. The ES3 BTS firmware is signed with ECDSA since v1.5.2, the DRV firmware is signed with ECDSA and encrypted with TEA since v0.1.7, and the BCTRL firmware is unprotected.

**Battery** We RE the relevant M365 and ES3 battery details by physically inspecting them

Table 5.1: Battery and BMS details. OVT: overvoltage threshold, UVT: undervoltage threshold, LBD: load balancing delta. Prefix d means dangerous, and prefix c means critical.

| | **M365** | **ES3** |
|---|---|---|
| *Battery* | | |
| Manuf. | Ninebot | Ninebot |
| Type | Li-Ion 18650 | Li-Ion 18650 |
| Chem. | 10INR19/66-3 | 10INR19/66-3 |
| Cells | 30 (10x3) | 30 (10x3) |
| Capacity | 7.80Ah/300Wh | 7.65Ah/275Wh |
| Voltage | 42V to 36V | 42V to 36V |
| *BMS* | | |
| BCTRL | STM8L151K6 | STM8L151K6 |
| BMON | BQ76930 | BQ76930 |
| dOVT | 4200mV | 4200mV |
| cOVT | 4700mV | 4700mV |
| dUVT | 2750mV | 2750mV |
| cUVT | 1580mV | 1580mV |
| dLBD | 30mV | 30mV |
| cLBD | 800mV | 800mV |

and reviewing public documentation. As summarized in Table 5.1, the e-scooters have similar lithium-ion (Li-Ion INR) batteries produced by Segway-Ninebot. The battery contains 30 cells configured in 10 series of 3 parallel cells. The M365 battery has more capacity than the ES3 one (i.e., 7.80Ah/300Wh vs. 7.65Ah/275Wh), but supports the same maximum and minimum voltage thresholds (i.e., 42V at 100% and 36V at 0%) and cell voltage (i.e., between 4.2V and 3.6V).

**BMS** We RE the M365 and ES3 BMS by studying the unencrypted BCTRL firmware and the BMON datasheet. The BMON periodically checks the status of the battery and relays data (e.g., voltage and temperature) to the BCTRL. The BCTRL manages the battery based on the data received from the BMON and also initializes the configuration parameters of the BMON, including the *undervoltage* and *overvoltage* battery protection thresholds. The undervoltage threshold is set between 1580mV and 2750mV, and the overvoltage threshold is between 4200mV and 4700mV. We define these intervals as *critical* and *dangerous* undervoltage (cUVT = 1580mV, dUVT = 2750mV) and overvoltage (dOVT = 4200mV, cOVT = 4700mV) intervals. If a battery cell voltage exceeds the established threshold, the BCTRL raises an internal error and powers OFF the BTS and DRV, shutting down the e-scooter.

**Load balancing** The BMS has a proprietary load balancing mechanism that is essential for the battery's health, as imbalanced cells can reach a state of undervoltage or overvoltage. These two conditions can severely and irreversibly damage the battery and cause fire hazards and explosions [198]. Load balancing transfers voltage among battery cells, aiming for a state where all battery cells possess the same voltage. Load balancing occurs when the device is charging or in sleep mode (i.e., e-scooter is powered OFF), never when the e-scooter is running in the streets. According to our experiments, imbalanced cells exceeding the critical load

**Algorithm 1** BCTRL initialization and main loop logic.

```
 1: initBCTRL(serial, fwver, dLBD, cLBD)
 2: initBMON(dUVT, cUVT, dOVT, cOVT)
 3:
 4: loop
 5:     battcells = readBMON(volt, amp)
 6:     battlevel = compBattlevel(battcells)
 7:     if battcells.deltaVolt >= cLBD then
 8:         sleepMode(True)
 9:         loadBalancing(True)
10:     end if
11:     if battcells.minCellVolt < cUVT then
12:         sleepMode(True)
13:     end if
14:     if battcells.maxCellVolt > cOVT then
15:         sleepMode(True)
16:     end if
17:
18:     while (sleepMode and !readWakeUpFromUART()) do
19:         if battcells.deltaVolt >= dLBD then
20:             loadBalancing(True)
21:         end if
22:     end while
23:     readFromUART()
24:     writeToUART()
25: end loop
```

balancing delta (i.e., cLBD = 800mV) cause the BMS to raise an error and power OFF the BTS and DRV.

**BCTRL firmware initialization** We RE the BCTRL firmware initialization and the main loop by decompiling its stripped binary and debugging its execution with JTAG and gdb. As shown in Algorithm 1, during initialization the BCTRL (1) loads its configuration parameters, including serial number, firmware version, and load balancing deltas; and (2) configures the BMON over I2C by writing into specific BMON registers, including setting the undervoltage and overvoltage thresholds.

**BCTRL firmware main loop** Then, the firmware enters a *main loop* where it: (5) reads the battery cells' voltage and current values from special BMON read-only registers; (6) computes the current battery level (percentage); (7) checks voltage levels against cLBD, (9) enables load balancing if needed, and (11) checks undervoltage against cUVT and (14) overvoltage against cOVT. If a critical undervoltage, overvoltage, or balancing check fails, it sends error messages that power OFF the BTS and DRV, and (18) enters sleep mode and performs load balancing. During the main loop, the BCTRL (23) reads from and (24) writes to the UART bus to communicate with the BTS and DRV.

**UART bus** We RE the proprietary UART protocol used by BCTRL, DRV, and BTS. The BCTRL utilizes a USART1 peripheral mapped to RAM (address 0x5230). A UART packet consists of several fields, including the sender and receiver (e.g., 0x22 corresponds to BCTRL),

packet type (e.g., read or write), command to execute, payload, and CRC.

**I2C bus** We RE the proprietary I2C protocol used by BCTRL and BMON. The BCTRL can directly read and write the BMON registers via I2C. For example, we can set dUVT by writing into `UV_TRIP` or read the voltage of specific battery cells via `VC1`, ..., `VC10`.

**DRV firmware** We RE the relevant portions of the DRV firmware. It runs on an STM32F103CxT6 SoC and measures the motor voltage, storing it in a RAM register (offset 0x47). The DRV performs a voltage safety check independent of the BCTRL. If the motor voltage hits cUVT, the DRV raises "Error 24 - Supply Voltage out of range" and powers OFF the e-scooter after ten seconds.

### 5.2.4 RE E-Scooter Vulnerabilities

During our RE experiments, we discovered *four architectural vulnerabilities* affecting the safety, security, availability, and privacy of the Xiaomi e-scooter ecosystem.

**V1: BCTRL firmware is unencrypted** An attacker can retrieve the (stripped) BCTRL firmware in plaintext (e.g., during a firmware update) and RE it.

**V2: BCTRL firmware is unsigned** An attacker can modify the firmware of the BCTRL, including its safety thresholds and how it initializes the BMON, and flash the modified BCTRL on the victim's e-scooter.

**V3: UART bus lacks integrity protection, encryption, and authentication** An attacker with access to the UART bus (e.g., via a malicious BCTRL firmware) can eavesdrop on all UART messages, replay or forge them, and impersonate or MitM the DRV and BTS.

**V4: UART bus lacks protection against DoS** An attacker with access to the UART bus can DoS internal (BMS, BTS, and DRV) and external (Mi Home and battery charger) components.

## 5.3 Threat Model

Before presenting the attacks, we detail our system and attacker models (i.e., our threat model). These models are map to real-world Xiaomi e-scooters as they are based on the RE findings and vulnerabilities described in Section 6.2.

### 5.3.1 System Model

Our system model follows the block diagram in Figure 5.1. It includes a user who owns a Xiaomi e-scooter (BES) and a smartphone running Mi Home for Android or iOS. They paired Mi Home with the e-scooter and locked it with a password. The user runs the latest version of the BTS, DRV, BCTRL firmware, and Mi Home and charges the e-scooter with the original Xiaomi charger.

### 5.3.2 Attacker Model

We consider two protocol-level attacker models: (i) *proximity-based attacker* who interact with the e-scooter wirelessly over BLE (e.g., using a laptop in BLE range with the e-scooter); (ii) *remote attacker* who installs a rogue malicious app on the victim's smartphone and exploits the e-scooter remotely via said app. Implementation-level attacks including side channel and fault injection are out of scope.

**Goals** The attacker wants to violate the *safety*, *security*, *privacy*, and *availability* of the e-scooter by exploiting V1, V2, V3, and V4 (i.e., internals architectural vulnerabilities). For instance, the adversary could try to extort money from the e-scooter driver via a ransomware, or follow their movements via a tracking attack. These two goals are *novel* in the domain of e-scooters and they result in large scale and critical impact as they target the whole Xiaomi e-scooter ecosystem (e.g., millions of users and devices).

**Capabilities** The attacker's knowledge covers the known details (Section 5.2.2) and the uncovered details and vulnerabilities (Sections 5.2.3 and 5.2.4). The attacker can reuse findings and tools from prior works like [178, 190, 188]. For example, they can authenticate to e-scooters via BLE using the `E-Spoofer` toolkit [199]. The attacker cannot physically access the victim's e-scooter, charger, and smartphone.

## 5.4 Attacks

We now present *E-Trojans*, four novel attacks on the internals of Xiaomi e-scooters.

### 5.4.1 Attacks Initialization

Each attack starts by flashing a malicious BCTRL firmware on the victim's e-scooter and turning OFF legitimate firmware updates (only on the BCTRL). The malicious firmware contains custom code enabling malicious functionalities to conduct the attacks described next.

### 5.4.2 Undervoltage Battery Ransomware (UBR)

UBR is a novel *undervoltage ransomware* that progressively and irreversibly damages the e-scooter's battery, until the victim pays a ransom.The damage is caused by flashing a malicious firmware that keeps the battery cell voltages below 2750mV or 1580mV (i.e., battery level below 0%). As a result, the e-scooter's battery longevity is reduced [200, 201] and, by keeping the battery undervolted and imbalanced, the attacker can cause short circuits and thermal runaway [202]. In the optimal scenario, the attacker undervolts a battery cell as low as 0V and quickly destroys it. We highlight that our attack can be used to damage the battery *without* asking for a ransom. Replacing a damaged battery can cost up to USD 139 [203]. UBR has five steps, also shown in Algorithm 2:

1. The attacker flashes a malicious BCTRL firmware, that contains the ransomware, using the remote and proximity-based techniques presented in [178]. The ransomware remains stealthy, letting the user drive as usual. The user rides the e-scooter, discharging it until the battery is low. Although UBR does not require a specific battery level, batteries

closer to 0% are faster to undervolt. Then, the user charges the battery using the stock charger.

2. UBR enters its *initialization* phase (first five lines of Algorithm 2), where it: (1) disables undervoltage protections by increasing dUVT to 1580mV and bypassing cUVT safety checks; (2) disables load balancing by bypassing the dLBD and cLBD safety checks; (3) disables battery charging by setting the `canCharge` flag to `False`; (4) activates the anti-theft software lock to prevent the e-scooter from becoming inactive and automatically turning OFF; (5) enforces faster discharging by disabling sleep mode and, optionally, spoofing a UART command that turns ON the headlight and taillight.

3. UBR enters its *main loop*, where it (8) reads the battery cells' voltages from the BMON, and (9) checks for dead cells. Then, for each of the ten series of cells, it checks (12) if any cell is below cUVT or (14) below dUVT. Since undervoltage and balancing protections are OFF, the battery discharges below the undervoltage threshold and deteriorates. When a programmable condition is met (e.g., a cell reaches cUVT), the ransomware (18) reveals itself. The e-scooter (20) reboots and starts broadcasting over BLE the download link to the ransom payment app. The link appears in a shortened form (e.g., *t.ly/AaBbCc*) due to length restrictions on the e-scooter BLE name.

4. The victim installs the ransom app and can monitor the degradation of the battery. They stop the attack by transferring money to the attacker's crypto wallet. Once the app's backend confirms the payment (in cryptocurrency, making it difficult to trace), it reveals to the app's frontend a secret unlock code, unique to each infected device. Then, the app sends a special BLE packet, with the unlock code, that enables firmware updates on the BCTRL.

5. The ransom app guides the victim through the recovery procedure. The app flashes on the BCTRL a recovery firmware that allows charging even when the battery is critically undervolted. The stock BCTRL firmware does not allow this for safety reasons (e.g., risk of short circuit). As soon as the battery is not critically undervolted anymore (i.e., voltage is higher than cUVT), the app allows the user to flash the stock BCTRL firmware.

### 5.4.3 User Tracking via Internals (UTI)

UTI is a new *user tracking* attack that tracks an e-scooter user and exfiltrates their private data. As shown in Figure 5.4, UTI creates a reliable user fingerprint and leaks sensitive e-scooter data. First, the malicious BCTRL firmware retrieves the 14-byte DRV serial number (Read `DRVId`) by impersonating the BTS and spoofing a read to the DRV. Second, UTI builds a `Fingerprint` from the last 8 bytes of `DRVId` representing the year, month, and day of production, the product revision, and the unit identifier (i.e., nth unit produced that day). UTI optionally retrieves via UART or I2C extra private data from the DRV, BMON, and BTS, like the mileage and battery level (Read `BattLevel` and Read `Mileage`). Then, UTI appends the data to the fingerprint and creates a 14-byte tracking message (`Track`). It sends a command to the BTS while spoofing the DRV to set the tracking message as the BLE name (`EscName = Track`). The e-scooter reboots and advertises its new name (`Track`) over BLE.

**Algorithm 2** UBR undervoltage ransomware logic.

```
disableUndervoltageChecks(cUVT, dUVT)
disableBalancing(cLBD, dLBD)
disableCharging()                                  ▷ Sets canCharge to False
enableLock()                                       ▷ Keeps the e-scooter ON
enableFastDischarge()                              ▷ Battery discharges faster

loop
    battcells = readBMON(volt, amp)
    deadcells = checkCellHealth(battcells)
    log(deadcells)
    for cell in battcells do
        if cell < cUVT then                        ▷ 1580mV
            log(cell, cUVT)
        else if cell < dUVT then                   ▷ 2750mV
            log(cell, dUVT)
        end if
    end for
    if notifyUser then
        log(notifyUser)
        advertiseRansomOverBLE(url)                ▷ Reboot
    end if
end loop
```

The adversary uses `Track` to identify the user via a proximity-based BLE sniffer or remotely via a malicious app. UTI is stealthy as the Mi Home UI never shows the e-scooter name and changing the e-scooter name does not disconnect or unpair the e-scooter from Mi Home, that only checks the BLE address. Moreover, since the fingerprint persists through factory resets and firmware updates, UTI bypasses privacy defenses like BLE address randomization.

### 5.4.4 Denial of E-Scooter Services (DES)

DES is a collection of seven DoS attacks, including an original *denial-of-sleep (DoSL)* which is typically used in sensor networks [204]. DES impacts the availability of internal components (i.e, BTS, DRV, BCTRL, and BMON), internal buses (i.e., UART and I2C), e-scooter functionalities (i.e., battery charging, error handling, and sleep mode), external components (i.e., Mi Home and the battery charger), or even the entire e-scooter. These attacks are particularly effective as they DoS components from the inside and *cannot* be fixed by the victim from the outside (e.g., via Mi Home or by rebooting). We summarize the seven DES attacks.

*DES1* drops all UART and I2C packets destined for the BCTRL by altering the firmware code for internal bus communications. *DES2* forces a supplier-only mode (SHIP mode) that disables the BMON by issuing a special unauthenticated I2C command. *DES3* floods the UART bus with dummy packets by altering the firmware code for the UART transmitter function. *DES4* periodically issues lock or reset commands to the DRV by impersonating the BTS and forging UART packets. As a result, the e-scooter is stuck in a state where the motor is immediately locked whenever the user tries to unlock it or in an infinite loop of power ON and

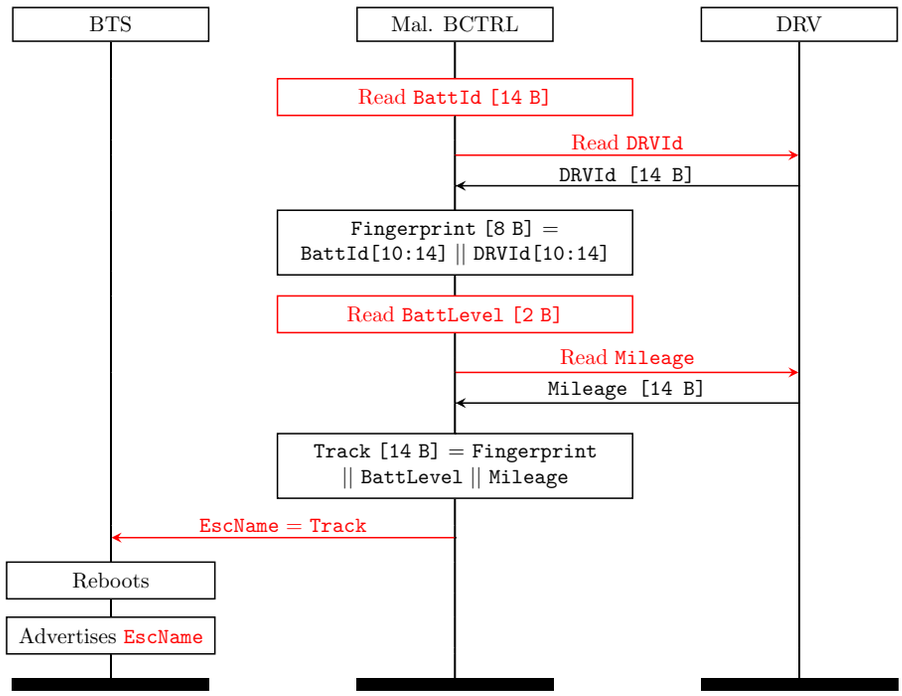



Figure 5.4: User Tracking via Internals (UTI). Mal. BCTRL (the attacker) builds a 8-byte e-scooter fingerprint (`Fingerprint`) from the electric motor serial number. Then, they append other sensitive data like mileage and battery level to the fingerprint, creating a 14-byte tracking message (`Track`). Finally, they enforce as BLE name change to `Track`, allowing anyone to track the user and read their private data in real-time.

OFF. *DES5* disables battery charging by setting the `canCharge` flag in the BCTRL EEPROM to `False`. *DES6* exploits the error handling routines by impersonating BTS, DRV, or BMON and raising fake errors that put the e-scooter in an irregular state (e.g., alarm noises, locked motor, unresponsiveness, and forced power OFF). *DES7* is a denial-of-sleep attack that keeps the BCTRL in a resource-intensive mode, instead of letting it switch to sleep mode, by altering the BCTRL main loop (described in Algorithm 1).

### 5.4.5 Password Leak and Recovery (PLR)

PLR enables remote and offline brute forcing of the e-scooter lock/unlock password. It retrieves the password's SHA256 hash from the DRV by spoofing the BTS, sets the e-scooter BLE name equal to the hash, and exfiltrates it via BLE advertising (similar to UTI). However, the 14-byte long e-scooter name is too short to fit the 32-byte SHA256 hash of the password. To solve this, PLR cycles through three e-scooter names containing parts of the hash, causing a reboot each time the BLE advertisement changes, noticeable by the user. The password is brute-forceable, being a numeric sequence of exactly 6 digits (e.g., `123456`), with a search space of $10^6$ [205]. Hence, the attacker can use known techniques to recover the password offline, including password lists and rainbow tables.

Table 5.2: Mapping between the four attacks and the four vulnerabilities. All attacks exploit V1, V2, and V3. Only DES exploits V4.

| Attack | V1 | V2 | V3 | V4 |
|---|---|---|---|---|
| Undervoltage Battery Ransomware (UBR) | ✓ | ✓ | ✓ | ✗ |
| User Tracking via Internals (UTI) | ✓ | ✓ | ✓ | ✗ |
| Denial of E-Scooter Services (DES) | ✓ | ✓ | ✓ | ✓ |
| Password Leak and Recovery (PLR) | ✓ | ✓ | ✓ | ✗ |

### 5.4.6 Mapping Attacks and Vulnerabilities

Table 5.2 shows the mapping between the discussed four attacks and four vulnerabilities. All attacks take advantage of unencrypted BCTRL firmware (V1) as they implement functionalities we RE from them (e.g., send valid UART messages from the BCTRL). Moreover, all attacks exploit the unsigned BCTRL firmware (V2) to push malicious firmware on the BCTRL. All attacks also abuse the unprotected UART bus (V3) as they utilize arbitrary UART messages without properly encrypting and integrity protecting them and without needing to authenticate on the bus. Finally, DES exploits the lack of DoS protection on the UART bus (V4) to DoS internal and external e-scooter components.

## 5.5 Implementation

We now present `E-Trojans`, a toolkit that implements our RE findings and the four attacks described in Section 5.4. For space reasons, we cannot describe the full implementation. Our toolkit is *extensible* to other BESs from Xiaomi or other vendors employing a similar internal architecture and chips.

### 5.5.1 BCTRL Firmware Patching and Capabilities

Our Python3 script (`payload-patcher.py`) binary patches a BCTRL using STM8 assembly code. During this process, the user can include in the patch up to ten new capabilities unavailable in the stock BCTRL firmware. We devised these capabilities thanks to our extensive RE of the BCTRL firmware. The script writes STM8 assembly code in unused parts of the firmware and then updates the firmware CRC. The resulting binary can be flashed over BLE on the BCTRL of a M365 or ES3 e-scooter. We developed *ten* capabilities that we describe next.

**DFU: Disable Firmware Updates** disables BCTRL firmware updates from Mi Home or third-party apps. We introduce a new `canFwUpdate` flag (address 0x04CB), set to `False` by default. We rewrite the `HandleUART()` function (address 0x94C5) to accept or reject firmware update packets (starting with 0x07, 0x08, 0x09, or 0x0A), depending on `canFwUpdate`. This flag is set to `True` only when the BCTRL receives our new unlock packet (starting with 0xEE and followed by the correct unlock code). For example, in UBR the user retrieves the unique unlock code by paying a ransom and utilizes it to enable firmware updates on the BCTRL.

**MUB: Manage UART Bus** sends arbitrary packets on UART. Due to V3, MUB can craft arbitrary UART messages and impersonate internal components (BTS, DRV, BCTRL,

and BMON) by altering the sender field. We rewrite the `sendUART()` function (address 0xB06C) to send our forged packets, stored from address 0xD3 onward. For example, Password Leak and Recovery uses MUB to spoof the BTS and retrieve the SHA256 hash of the e-scooter password from the DRV.

**MIB: Manage I2C Bus** sends arbitrary packets on I2C. We rewrite the `writeToI2C()` function (address 0x90BE) to send our forged packets, whose data is stored in register `A` and target BMON register in register `X`. This includes special I2C commands such as the one regressing the BMON into SHIP mode (used in DES7).

**DDT: Disable Dangerous Thresholds** alters the default BMON undervoltage and overvoltage thresholds. At BCTRL boot, DDT initializes the BMON `UV_TRIP` (address 0x923D) and `OV_TRIP` (address 0x923D) registers to 1580mV and 4700mV, their lowest and highest values, instead of the default ones. DDT enables UBR to reach voltages below the undervoltage threshold (i.e., 2750mV) recommended by Xiaomi, resulting in permanent battery damage and increased safety risks.

**DCT: Disable Critical Thresholds** bypasses battery-related errors, including undervoltage and overvoltage. We rewrite the BCTRL error handler to replace the critical error check (address 0xB5A9) with NOPs. For example, we emply DCT in DES5 to raise fake errors, such as *Error 23 - Internal BMS not activated* and *Error 24 - Supply Voltage out of range*, that put the e-scooter in an abnormal state.

**DLB: Disable Load Balancing** prevents load balancing of the battery cells. We rewrite the manageLoadBalancing() function (address 0xA81A) to unset (with MOV) the three `CELLBAL` registers that regulate balancing. As a result, battery cells become increasingly more imbalanced over time. DLB enables UBR to create one extremely imbalanced cell that quickly dies, severely impacting the battery's health.

**DBC: Disable Battery Charging** disallows battery charging. We rewrite the `controlCharge()` (address 0x945C) function to always read `canCharge` flag (address 0x42C) as `False`, even if the charger is connected. For instance, without DBC, DES5 would not be able to DoS the battery charger and prevent the battery from charging.

**FBD: Fast Battery Discharge** accelerates battery consumption. We rewrite the BCTRL main loop to permanently assign `False` to the `sleepMode` variable (address 0x400), forever disabling sleep mode and forcing the BCTRL to stay in a more resource-intensive mode. Optionally, FBD can also turn ON the e-scooter's headlight and taillight to further increase battery consumption. UBR uses FBD to accelerate battery depletion and reach the critical undervoltage threshold faster.

**CBA: Change BLE Advertising** changes the e-scooter's BLE advertisements, including its BLE device name. CBA sends an arbitrary e-scooter advertising change packet to the BTS, also causing a reboot. This packet is composed of a destination field (0x20 to indicate BTS), an operation type field (0x50 to indicate a change in BLE advertising), and a payload containing the new advertisement. In UTI, the attacker tracks the user and exfiltrates their private data by changing the e-scooter's BLE advertisements with CBA.

**SCV: Spoof Cell Voltage** spoofs voltage measurements of battery cells. We rewrite the BCTRL main loop to ignore new BMON measurements and only read the spoofed values we inject in register `X`. SCV sends the spoofed voltages to the BTS, which are displayed to the user via the Mi Home UI. By displaying fake cell voltage measurements instead of real ones, SCV enables UBR to conceal an undervolted battery from users that check the battery status via Mi Home.

Table 5.3: Mapping between the ten capabilities and the four attacks. All capabilities enable one or more attacks. All ten capabilities are required to deploy UBR.

| Capability | UBR | UTI | DES | PLR |
|---|---|---|---|---|
| Disable Firmware Updates (DFU) | ✓ | ✓ | ✓ | ✓ |
| Manage UART Bus (MUB) | ✓ | ✓ | ✓ | ✓ |
| Manage I2C Bus (MIB) | ✓ | ✓ | ✓ | ✗ |
| Disable Dangerous Thresh. (DDT) | ✓ | ✗ | ✗ | ✗ |
| Disable Critical Thresh. (DCT) | ✓ | ✗ | ✗ | ✗ |
| Disable Load Balancing (DLB) | ✓ | ✗ | ✗ | ✗ |
| Disable Battery Charging (DBC) | ✓ | ✗ | ✓ | ✗ |
| Fast Battery Discharge (FBD) | ✓ | ✗ | ✓ | ✗ |
| Change BLE Advertising (CBA) | ✓ | ✓ | ✗ | ✓ |
| Spoof Cell Voltage (SCV) | ✓ | ✗ | ✗ | ✗ |

**Capability usage** Table 5.3 shows how we *combine* the ten capabilities to perform UBR, UTI, DES, and PLR. We require DFU in all attacks to prevent the user to replace our malicious BCTRL via BLE firmware updates. We integrate MUB and MIB in all attacks (except MIB in PLR), as they require control over the internal UART and I2C buses. We deploy DDT, DCT, DLB, and SCV in UBR, as they affect the safety of the battery (i.e., undervoltage). We use DBC in UBR and DES5 as, alongside the e-scooter, they also target the battery charger. Similarly, we use FBD in UBR and DES7, as they try to accelerate battery consumption by triggering resource-intensive tasks. For Undervoltage Battery Ransomware, UTI, and PLR, we rely on CBA to exfiltrate data from the e-scooter. UBR requires all capabilities.

### 5.5.2 UBR App and Backend

As part of UBR, we developed an Android app and a Django and MongoDB backend to test our ransomware in a controlled but realistic environment. These components cooperate to generate unique ransomware unlock codes, manage cryptocurrency payments, and handle BCTRL firmware recovery. Next, we provide some related technical details.

**Android App** We developed *ScooterToolkit*, an Android app given to the ransomware victim to pay the ransom in cryptocurrency and rebalance the battery. We wrote the code in Kotlin and we require Bluetooth, Location, and Internet permissions. The app initiates the BCTRL recovery when the back-end returns the unlock code that enables firmware updates (i.e., the user paid the ransom). Under normal battery conditions, the app flashes the stock BCTRL firmware. If the voltage is lower than 1580mV, a special recovery firmware is flashed instead.

**Django and MongoDB Backend** Using Django, we developed a RESTful web service that exposes two APIs to the ransom app, through the API view class. The `simulatePayment()` API simulates a payment made by the user, setting the payment status to `Paid`. The `unlockFirmware(serial)` API requires the e-scooter's motor serial as its input and, if the ransom has been paid, returns their unlock code. We also created a model serializer that converts e-scooter objects into JSON to interface with our MongoDB database. This database stores the

Table 5.4: Evaluation results. All four attacks are effective on the M365 and ES3. ✓[1]: dangerous, but not critical, undervoltage.

| Attack | M365 | ES3 |
|---|---|---|
| Undervoltage Battery Ransomware (UBR) | ✓ | ✓[1] |
| User Tracking via Internals (UTI) | ✓ | ✓ |
| Denial of E-Scooter Services (DES) | ✓ | ✓ |
| Password Leak and Recovery (PLR) | ✓ | ✓ |

e-scooter's motor serial and model, 128-bit ransomware unlock code, and the payment status for each device infected by UBR.

## 5.6 Evaluation

We evaluated our attacks by using the E-Trojans toolkit (described in Section 6.6) on Xiaomi M365 and ES3 e-scooters with up-to-date BTS, DRV, and BCTRL firmware.

### 5.6.1 Setup

We ran our `payload-patcher.py` script to generate the malicious BMS firmware. We select one of our four attacks in the interactive shell of the script and the target e-scooter model. Then, we set up each attack by connecting to the e-scooter, gaining authorized access (i.e., through malicious pairing or session downgrade) and performing a rogue BCTRL firmware update.

We tested proximity-based attacks on a Dell Inspiron 15 3000 running a Linux-based OS, performing the attack setup from the BLE range. We tested remote attacks on a Realme GT (Android 13) and a OnePlus 3 (Android 9), performing the attack setup remotely from the *ScooterToolkit* Android app installed on the victim's smartphone.

Regarding UBR, we monitored the voltage and stopped the experiment upon reaching critical undervoltage (for safety reasons), or after ten hours. We tested the feasibility of retrieving the six-digit e-scooter password when deploying PLR. We considered the seven most common six-digit patterns [206], and randomly generated seventy Xiaomi-compliant passwords (ten for each pattern). Then, we implemented a rainbow table [207] and populated it with 3 million randomly generated Xiaomi-compliant passwords, simulating the most basic attacker setup.

### 5.6.2 Results

As shown in Table 5.4, we successfully conducted our four attacks on the M365 and the ES3. Our experimental results represent a *lower bound* as other e-scooters and BESs might be affected by similar vulnerabilities. For instance, the Xiaomi Pro e-scooter runs BCTRL v1.2.6 (same as the M365), and the 1S, Essential, and Pro 2 run BCTRL v1.4.1 (same as the ES3) [208]. Next, we provide attack-specific insights.

**UBR** irreversibly reduced the battery autonomy of the M365 by approximately 50% in three hours and thirty minutes. We forced a critical undervoltage state and stopped the experiment when some cells reached 0V (i.e., the most critical undervoltage condition). This group of cells

now has limited charging capacity (i.e., 75%) and discharges approximately four times faster than average. Similarly, we reduced the battery autonomy of the ES3 by approximately 10% in ten hours, spending six of them depleting the battery to 0% (three hours for the M365). We could not reach a critical undervoltage due to the DRV independently measuring an abnormal voltage and raising *Error 24 - Supply Voltage out of range*. For this reason, the ES3 is better protected than the M365 against UBR, even though it still sustained significant damage. We also confirmed that our recovery BCTRL firmware rebalances undervolted batteries in five minutes.

**UTI** successfully tracks users and exfiltrates their private data over BLE. By sniffing BLE advertisements and looking for specific fingerprints (e.g., motor serial number), we could track the user across an area where we deployed BLE sniffers. Moreover, we could read their mileage (e.g., 0x01B1 represents 433km), battery level (e.g., 0x2D represents 45%), and any real-time update. Then, we factory reset the e-scooter and we were still able to track the user.

**DES** made the e-scooter irresponsive and undrivable. DES1 and DES3 cause the DRV to raise *Error 21 - No Communication with BMS*, completely halting the motor, and powering OFF the e-scooter after ten seconds. DES2 cuts power from all internal systems, immediately powering OFF the e-scooter and barring it from powering ON again. DES4 prevents battery charging regardless of the battery charger hardware or the e-scooter status (e.g., powered ON or OFF). DES5 induces an endless lock state or reboot cycle, rendering the e-scooter unable to move and forcing it to always stay powered ON. DES6 raises *Error 23 - Internal BMS not activated* which induces a constant beeping noise and *Error 24 - Supply Voltage out of range* which powers OFF the e-scooter. DES7 ignores all sleep cycles even when the e-scooter is powered OFF, thus accelerating battery consumption.

**PLR** successfully leaked the e-scooter password hash and enabled its offline brute forcing. We recovered the hash by sniffing the three rotating BLE advertisements broadcasted by the e-scooter. We also confirmed that retrieving the e-scooter password from the hash is feasible, as we were able to crack our seventy random passwords with an average time of less than one second.

## 5.7 Countermeasures

We propose four *practical* and *backward-compliant* countermeasures, each addressing one of the four vulnerabilities described in Section 5.2.4 (i.e., C1 addresses V1, C2 addresses V2, and so on). To fix UBR, UTI, and PLR, Xiaomi must implement C1, C2, and C3. To fix all DES variants, Xiaomi must also implement C4. Next, we describe each countermeasure in detail.

**C1: Encrypt the BCTRL firmware with TEA** Xiaomi should encrypt the BCTRL firmware to protect its confidentiality at rest (e.g., in Mi Home) and in transit (e.g., during a firmware update over BLE). We recommend reusing the TEA block cipher already used to encrypt the DRV firmware. Its overhead is negligible, as TEA is a lightweight cipher [209].

**C2: Sign and verify the BCTRL firmware with ECDSA** Xiaomi should digitally sign and verify the BCTRL firmware to prevent rogue BCTRL firmware updates. We recommend reusing ECDSA, which is FIPS-compliant [210] and already employed to sign/verify the DRV and BTS firmware. Its overhead is minimal, as the BTS would have to verify the BCTRL firmware during an update using crypto primitives it already supports.

**C3: Protect the UART bus with SCP03** Xiaomi should protect the confidentiality,

integrity, and authenticity of the UART bus. We suggest using standard secure embedded bus protocols such as *Secure Channel Protocol 03 (SCP03)* [211]. SCP03 provides confidentiality, integrity, authenticity, and replay protection using a lightweight scheme based on pre-shared symmetric keys, authenticated encryption, and AES. Since SCP03 was designed for smart cards [212], way more constrained than e-scooters, implementing it adds minimal overhead.

**C4: Protect the UART bus with rate limiting** Xiaomi should protect the UART bus against DoS attacks or at least mitigate them. We recommend implementing a lightweight rate-limiting mechanism for UART, such as the leaky bucket algorithm, where excess packets are discarded if the incoming packet rate exceeds the outgoing rate. C4 can be combined with C2 to achieve better DoS protection, as an authenticated UART bus discards messages that do not pass the integrity check.

## 5.8 Related Work

**E-Scooters Security and Privacy** Research on e-scooters has studied their proprietary protocols and rental ecosystems, but no work has focused on their internals. A recent contribution [178] analyzed the Xiaomi e-scooter ecosystem, identifying issues in the proprietary BLE protocols used by the e-scooter and Mi Home. The researchers identified vulnerabilities [13] in the Bird e-scooter sharing platform and discovered MitM attacks [33] on the rental e-scooters of the Lime company. Other studies focused on the privacy of user-related data in e-scooter rental Android apps [25] and the safety risks of riding e-scooters [213].

**E-Scooters Hacking and Modding** In 2019, Zimperium revealed vulnerabilities in the M365 locking system that could halt a running e-scooter [32]. The same year, the hacker Lanrat found that M365 does not enforce authentication over BLE [31]. ScooterHacking [165], the largest e-scooter modding community, released hacking tools [156, 190] to interact with Xiaomi e-scooters, mod DRV firmware [167] and flash it [162]. In our work, we develop custom BCTRL firmware with capabilities that are not offered by previous tools and research.

**Attacks on Battery** The threats concerning battery overheating, overcharging, and mechanical damage in electric vehicles have been investigated in [214, 215], where the authors acknowledge that thermal runaway, shrapnel ejection, and the release of toxic gas are significant safety hazards. Researchers have proposed a battery drain attack while the ignition is turned OFF [216], an electromagnetic interference attack [217], and a physical access rogue firmware update to a Tesla BMS [186]. Theoretical vulnerabilities in MacBook laptop batteries, capable of overheating or turning it OFF, have been discussed in [187]. The E-Trojans attacks are orthogonal as they target an e-scooter's battery and BMS.

**Attacks on Embedded Firmware** Several studies focused on malicious embedded firmware, but none exploit e-scooter internals. Researchers attacked embedded firmware on programmable control logic (PCL) [218], object trackers [219], printers [220], mice [221], OS management systems [222], mobile bootloaders [223], network equipment [224], botnets [225] and recently industrial wrenches [226].

## 5.9 Conclusion

We present the first evaluation of Xiaomi e-scooters internals and their attack surface. Although we focus on M365 and ES3, two popular e-scooters, our research is generalizable to any BES with

a comparable system model. We RE the internal systems and communication buses and uncover four critical flaws affecting their BCTRL and UART bus. We exploit these vulnerabilities by developing four novel attacks we name E-Trojans. Our attacks violate the safety (UBR), privacy (UTI), availability (DES), and security (PLR) of the Xiaomi e-scooter ecosystem. For example, we deploy the first ransomware for an e-scooter that permanently damages the battery via undervoltage. Our attacks are conducted in BLE proximity or remotely via a malicious Android application.

We develop `E-Trojans`, a toolkit that implements our attacks by leveraging our RE findings (e.g., disable safety thresholds). We successfully carried out the attacks on actual devices, proving their practicality. For instance, UBR irreversibly reduced the M365 battery autonomy by 50% in three hours and the ES3 battery autonomy by 10% in ten hours. We fix attacks and their root causes with four backward-compliant countermeasures.

# Chapter 6

# CTRAPS

***This contribution, titled "CTRAPS: CTAP Impersonation and API Confusion Attacks and Defenses on FIDO2", will be submitted at the 10th IEEE European Symposium on Security and Privacy (EURO S&P 2025).***

## 6.1 Introduction

*Fast IDentity Online v2 (FIDO2)* is the de-facto standard authentication protocol for single-factor (passwordless), second-factor (2FA), and multi-factor (MFA) authentication. Google, Dropbox, and GitHub [227] designed it in a joint effort to offer a practical and scalable solution for authentication. FIDO2 involves three entities: an *authenticator* that generates and asserts possession of authentication credentials (e.g., public-private key pairs), a *relying party* that authenticates the user (e.g., challenge-response protocol based on credentials), and a *client* that manages the communication between the authenticator and the relying party. Typically, the authenticator is a USB dongle, the relying party is a web server, and the client is a web browser or a mobile app. The user owns the authenticator, the client, and the device running the client.

Authenticator and client communicate using the *Client to Authenticator Protocol (CTAP)* over Universal Serial Bus (USB), Near Field Communication (NFC), or Bluetooth Low Energy (BLE). CTAP is an application layer protocol that exposes functionalities via the *CTAP Authenticator API*. For instance, the API creates, manages, and deletes FIDO2 credentials. API calls require specific authorizations such as User Verification ($UV$) and User Presence ($UP$). Client and relying party communicate using the Web Authentication (WebAuthn) protocol over TLS.

The introduction of passkeys (i.e., discoverable credentials) and growth of the FIDO ecosystem broadened the *CTAP attack surface*. Despite the associated challenges [228], passkeys are gradually replacing passwords by using FIDO2 authenticators as the root of trust. Hence, authenticators represent attractive targets as they store valuable credentials [61]. Market forecasts predict the FIDO market to rapidly grow from USD 230.6 million in 2022 to USD 598.6 million in 2031 [229]. Yubico, a FIDO authenticator market leader, sold more than 22 million YubiKey authenticators [230] and many big players are transitioning to FIDO, including the US government [231].

Prior works on FIDO focused on FIDO U2F, client compromises (e.g., malicious FIDO apps), local attacks (e.g., via browser extensions), and WebAuthn. In [232, 233], the authors developed

models to formally verify scenarios where a FIDO U2F client is compromised (e.g., client malware). In [234], the authors presented a social engineering attack on WebAuthn requiring a compromised client running as a malicious application or a browser extension with user-level and root-level privileges. In [235], the authors considered local (i.e., browser extension) and physical (i.e., temporary access) attackers targeting WebAuthn to gain unauthorized access to online accounts registered with FIDO2. Despite its exposure to practical and low-cost attacks, no prior study evaluated the security of the CTAP Authenticator API. We fill this relevant gap by presenting a security and privacy assessment of the CTAP Authenticator API against two realistic and relevant attacker models.

In our work, we propose two novel FIDO2 attack strategies targeting CTAP. We present the first *CTAP client impersonation*, allowing an attacker (e.g., a malicious NFC reader) to call arbitrary APIs without authorization or user interaction. We also propose the novel *API confusion* technique, where the attacker, such as a malicious USB hub, changes (i.e., confounds) the API called by the user to a different one. For example, the user wants to create a new credential through the UI of an uncompromised client, but the attacker leaks the user's private data instead.

We uncover *eleven attacks*, named **CTRAPS**, violating the security, privacy, and availability of the FIDO2 ecosystem via CTAP. For example, our attacks can factory reset FIDO2 authenticators to wipe out all credentials, permanently lock them, or leak private data to profile and track the user. These attacks can be deployed from remote or by the first proximity-based attacker evaluated in FIDO2. No prior academic work considered a proximity-based threat and even the FIDO security reference document [236] ignores them (see Section 6.9.1 for a detailed discussion on its unclear security boundaries and narrow security goals).

Our attacks have *widespread* consequences on the FIDO2 ecosystem as they target *protocol-level* FIDO2 vulnerabilities. They are effective against up to CTAP2.2 and WebAuthnL2, the most recent and supposedly strongest FIDO2 protocols. Since they target the CTAP application-layer, they are effective regardless of the CTAP transport (i.e., USB, NFC, or BLE) or the authenticator's implementative details. Moreover, the attacks are *stealthy* because they employ CTAP-compliant API calls and do not require unexpected user interactions (unlike phishing).

We isolate *seven vulnerabilities*, six of which are novel, that enable the CTRAPS attacks. These root causes include the lack of CTAP client authentication, a single PIN that authorizes both destructive and non-destructive operations, trackable user identifiers and credentials, and improper API *UV* and *UP* authorizations. The vulnerabilities are *critical* as they affect CTAP at the protocol-level and are exploitable on any standard-compliant CTAP implementation.

We implement our Authenticator API attacks in the `CTRAPS` toolkit. The toolkit contains three modules. A CTAP testbed with a virtual relying party and a virtual client, allowing for quick and local testing of authenticators. A CTAP client impersonator implemented for the Proxmark3 RFID tool and Android smartphones. Extended CTAP dissectors for Wireshark that simplify binary analysis of CTAP packets and add new useful features such as status codes and support for credential management.

We exploit popular FIDO2 authenticators and relying parties over different CTAP transports. We attack *six authenticators* from Yubico (including a FIPS-compliant one), Feitian, SoloKeys, and Google. We ran our attacks over USB and NFC. Unfortunately, we could not find FIDO2 authenticators supporting BLE. We empirically confirm that the API confusion attacks are *highly scalable*, while the proximity-based ones are less scalable as they require NFC

range and (cheap) additional devices. We also exploit *ten relying parties* offering passkeys and second-factor authentication, including Microsoft, Apple, GitHub, and Facebook.

We propose *seven countermeasures* fixing the attacks and root causes while being backward-compliant. For example, we add stricter authorization requirements for destructive APIs, introduce a dedicated PIN for destructive operations (e.g., credential deletion), and rotate user identifiers and credentials to mitigate user tracking. Our countermeasures are *practical* and *backward-compliant*, as they rely on mechanisms already available on the authenticator (e.g., PIN and LED). Moreover, we find a severe *implementation flaw* on Yubico authenticators, allowing data leaks and user tracking (see Section 6.9.2 for a detailed discussion).

We summarize our contributions as follows:

- We perform the first security and privacy evaluation of the CTAP Authenticator API. We focus on two practical and relevant attack strategies: CTAP client impersonation and API confusion. We uncover eleven proximity-based and remote attacks exploiting seven CTAP protocol level vulnerabilities. For instance, our attacks erase credentials and master keys, track users, and DoS authenticators.

- We present `CTRAPS`, a comprehensive and portable toolkit that offers unprecedented CTAP testing capabilities and reproduces our attacks.

- We demonstrate our attacks' practicality by conducting a large-scale evaluation. We exploit six authenticators, two transports, and ten relying parties. The list of affected vendors includes key players in the FIDO2 ecosystem, like Google, Apple, Microsoft, and Yubico.

- We fix the attacks and their root causes by proposing seven practical and backward-compliant countermeasures. We discuss a severe vulnerability in Yubico authenticators, three issues in the reference FIDO2 threat model related to CTRAPS, and our lessons learned.

## Disclosure, Ethics, and Availability

We responsibly disclosed our findings to the FIDO Alliance in November 2023 [237]. They acknowledged our report and shared it with their members. In May 2024, the FIDO Alliance provided feedback on our work and we are currently discussing with them. In December 2023, we reported our findings to the affected authenticator manufacturers (i.e., Yubico, Feitian, SoloKeys, and Google). Google assigned priority P2 and severity S2 to our report. Yubico acknowledged the implementation bug we found, pushed a fix in production, published a security advisory [238], and created CVE-2024-35311 [239]. The other manufacturers acknowledged the report without commenting on it. We also contacted Apple and Microsoft regarding their weak credential protection policy that facilitates user tracking and profiling. They responded that our report has no security implications for their products. However, our evaluation shows that stronger policies mitigate our user tracking attacks. We will release our toolkit after this paper's publication.

## 6.2 FIDO2 and CTAP Preliminaries

FIDO2 [240] is an open standard for user authentication based on *asymmetric* cryptography and curated by the FIDO Alliance. Four entities compose the FIDO2 ecosystem: an authenticator, a client, a user, and a relying party. In a typical scenario, a user connects his authenticator to the client in order to access an online service hosted by a relying party.

The FIDO2 specification includes the WebAuthn and CTAP application-layer protocols. WebAuthn provides a secure and private communication channel between a relying party and a client, and its latest version is WebAuthnL2 [241]. CTAP, the focus of this work, enables a secure and private connection between a FIDO2 authenticator and a client via the CTAP Authenticator API. For example, calling `MakeCred` registers a new credential and `GetAssertion` authenticates an existing credential.

A FIDO2 *credential* is a key pair used to sign and verify challenges by applying standard cryptographic techniques, such as the Elliptic Curve Digital Signature Algorithm (ECDSA). Access to the private key of a FIDO2 credential is safeguarded by encryption using a credential master key, which is unique to each authenticator and securely stored within the authenticator's Secure Element. FIDO2 credentials can be discoverable or non-discoverable. Discoverable credentials, also known as passkeys, are *stored on the authenticator.* and used for passwordless authentication. Non-discoverable credentials are stored on the web by the relying party and used for multi-factor authentication.

A FIDO2 credential is bound to three identifiers: the credential identifier (CredId), the relying party identifier (RpId), and the user identifier (UserId). CredId is derived from the credential master key and uniquely identifies a credential. Before deleting a credential, the client needs to specify a CredId. The RpId identifies a relying party, usually coincides with its origin (e.g., `login.microsoft.com`), and should be considered public. A relying party randomly generates a UserId when a user creates his first credential and associates the UserId to all credentials generated by that user. At registration time, a relying party can attach additional data to a credential, including sensitive or personally identifying information, by using the optional *CredBlob* FIDO2 extension.

As part of the FIDO2 specification, the CTAP standard has considerably evolved over time. CTAP1, also known as FIDO U2F (Universal 2nd Factor), provides phishing-resistant second-factor authentication. CTAP2.0 maintains backward compatibility with CTAP1 while introducing passwordless authentication. CTAP2.1 [242] adds the credential protection policy, discoverable credential management (i.e., the `CredMgmt` API), and biometric authentication. The draft for CTAP2.2 [243] is the latest available version, offering new features such as support for hybrid authenticators equipped with cameras to scan QR codes.

CTAP relies on two core user authorization mechanisms to secure API calls: (i) *User Verification (UV)*, which requires the user to enter a PIN or biometric data, and (ii) *User Presence (UP)*, which requires the user to press a button on the authenticator or to bring it into the client's NFC range. Table 6.1 shows the most common *CTAP Authenticator APIs* and their *UV* and *UP* requirements. We describe each API:

- `MakeCred` registers a new credential bound to an online account with a relying party.
- `GetAssertion` authenticates to a relying party by asserting (i.e., proving) possession of a credential.

Table 6.1: CTAP Authenticator API entries, short names (SN), *UV* and *UP* authorization requirements, and support for subcommands. Yes[1]: depends on client and relying party configuration, Yes[2]: depends on API subcommand. In the CTAP standard, `MakeCred` is called `MakeCredential` and `CredMgmt` is called `CredentialManagement`.

| CTAP API | SN | UV | UP | Subcmd |
|---|---|---|---|---|
| `MakeCred` | `MC` | Yes | Yes | No |
| `GetAssertion` | `GA` | Yes[1] | Yes[1] | Yes |
| `CredMgmt` | `CM` | Yes | No | Yes |
| `ClientPin` | `CP` | Yes[2] | No | Yes |
| `Reset` | `Re` | No | Yes | No |
| `Selection` | `Se` | No | Yes | No |
| `GetInfo` | `GI` | No | No | No |

- `CredentialMgmt` manages the authenticator's discoverable credentials (e.g., enumerate, modify, and delete).
- `ClientPin` handles User Verification (*UV*) based on a user PIN to be submitted via the client's UI.
- `Reset` factory resets the authenticator (i.e., wipes all discoverable and non-discoverable credentials by re-generating the credential master key).
- `Selection` selects an authenticator to operate among the available ones.
- `GetInfo` returns the authenticator's details (e.g., manufacturer, transports, extensions, and settings).

CTAP offers other optional security and privacy mechanisms. The authorization requirements for `GetAssertion` depend on the client and relying party configuration. A client can specify the option *up=false* to skip *UP*. At registration time, a relying party can enforce access control by specifying a credential protection policy via the optional *CredProtect* extension. However, the default policy skips *UV*, resulting in a weak privacy protection. Authenticators may also feature additional security mechanisms unrelated to CTAP, such as the FIDO authenticator certification level [244] and the FIPS [245] certification.

The `GetAssertion`, `CredMgmt`, and `ClientPin` APIs offer multiple functionalities through API subcommands. For example, `CredMgmt(GetCredsData)` returns the amount of stored discoverable credentials and `CredMgmt(DelCreds)` deletes all discoverable credentials. Some API subcommands, compared to their original API, have more relaxed requirements. For example, `ClientPin(KeyAgreement)` requests the authenticator's public key without requiring *UV*.

## 6.3 Threat Model

### 6.3.1 System Model

We consider the standard FIDO2 system model composed by an authenticator, a client, and a relying party. The three entities support up to CTAP2.2 and WebAuthnL2 (i.e., the latest and supposedly most secure FIDO2 protocols). The user connects his authenticator to the client in order to access an online service hosted by the relying party.

**Authenticator** The authenticator is a FIDO2 *roaming* authenticator: a physical device carried around by the user that can be connected to the client (e.g., a USB/NFC dongle). The authenticator runs a CTAP server that exposes the Authenticator API over the USB, NFC, and BLE transports. It supports the *UP* (e.g., via a button press) and *UV* (e.g., via a user PIN) user authorization mechanisms and stores discoverable credentials and the credential master key.

**Client** The FIDO2 client handles communication between the authenticator and the relying party. This component runs a CTAP client to speak with the authenticator and a WebAuthn client to speak with the relying party. Typically, the client is a software offering an UI to the user, such as a web browser, a mobile app for Android [246] or iOS [247], or even a command line tool like the Yubico CLI [248].

**Relying party** The relying party is an online service that relies on FIDO2 passwordless or multi-factor authentication, such as Adobe, Apple, Microsoft, and Facebook. This online service runs a WebAuthn server that responds to FIDO2 registration and authentication requests (e.g., over TLS). In case of multi-factor authentication, the relying party stores the non-discoverabl e credential, the user identifier, and the credential identifier. Certain offline operations on the authenticator indirectly affect the relying party. For instance, deleting a discoverable credential will result in the failure of any subsequent authentication attempts.

**User** The user owns the authenticator and the device (e.g., laptop or smartphone) running the client. He utilizes his authenticator to register passwordless and multi-factor FIDO2 credentials and to authenticate to the associated relying party. The user authenticates by connecting his authenticator to the client and providing *UV* and *UP*, if required. The user can also manage and configure the authenticator via the client, without connecting to a relying party. For example, he can check his discoverable credentials and change the authenticator's PIN.

### 6.3.2 Attacker Model

We consider two realistic and relevant attacker models targeting the CTAP Authenticator API, as shown in Figure 6.1: (i) a *CTAP client impersonator* spoofing a CTAP client and (ii) a *man-in-the-middle attacker* positioned between the authenticator and the client. Our two attackers are considered proximity-based if they communicate with the authenticator via CTAP over NFC or remote if they communicate over USB. No prior work evaluated a CTAP client impersonation attacker model or a proximity-based attacker.

**Goals**. The attacker's main goal is to compromise the FIDO2 user's *security* and *privacy* by exploiting the CTAP Authenticator API (introduced in Section 6.2). She wants to tamper with the discoverable credentials stored in the authenticator, delete non-discoverable credentials, track a user via the authenticator, or impersonate a client to an authenticator. She also wants to interfere with the client and relying party by confounding CTAP API calls and altering their

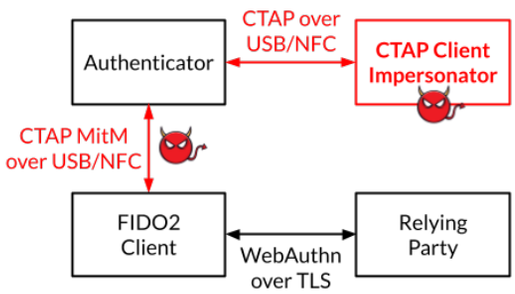



Figure 6.1: Attacker models. The CTAP client impersonator (top-right) communicates with the authenticator using CTAP over NFC/USB. The man-in-the-middle attacker (left) between the client and the authenticator targets the latter using CTAP over NFC/USB.

normal behaviour with forged CTAP messages. This is the first work to evaluate these goals and demonstrate their practical feasibility.

**Capabilities**. The attacker can impersonate a CTAP client, craft valid CTAP commands, and send them to the authenticator and the client over USB and NFC. She can establish a MitM position between the authenticator and the client over USB or NFC. During MitM, she maintains stealthiness by only calling Authenticator APIs when the user is operating the authenticator and only asking for *UV* and *UP* when the user calls an API that requires them.

The CTAP client impersonator can enter the authenticator's NFC range to deploy the proximity attacks. She can extend her reach with specialized equiment [249, 250]. The client impersonator can also remotely communicate with the authenticator, for example via a malicious FIDO2 app installed on the user's smartphone. The MitM attacker can intercept and modify the traffic between the authenticator and the client over NFC, as demonstrated in [251], and over USB (e.g., via a malicious USB hub). The attacker takes advantage of the vulnerabilities we describe in Section 6.5. For example, she abuses the lack of CTAP client authentication for impersonation, the absence of authenticator feedback to perform API confusion, and the *UP* bypass over NFC to perform proximity attacks. She cannot modify the authenticator's software (e.g., flash new firmware) and cannot obtain its credential master key. She cannot compromise a legitimate FIDO2 client or relying party. Physical attackers, including fault injection and side channel, are out of scope.

## 6.4 Attacks

We propose eleven attacks, we name CTRAPS, that abuse the lack of CTAP client authentication and CTAP API bindings and (see Section 6.5). Our attacks exploit CTAP protocol-level vulnerabilities, making them effective regardless of the CTAP transport (i.e., NFC or USB) and the implementation details of the authenticator, the client, and the relying party. They take advantage of novel attack strategies targeting FIDO2, we call CTAP *Client Impersonation (CI)* and the *API Confusion (AC)*.

Our attacks compromise the *integrity* (AC1, CI1, and AC2), *confidentiality* (AC1, CI2, AC3, CI4, and AC7), *availability* (AC4, CI3, AC5, and AC6), and *privacy* (CI2, AC3, CI4, and AC7) of the user and the entire FIDO2 ecosystem. For instance, we lost access to our test Google and Apple ID accounts because we could not pass 2FA after deleting our credentials with AC1. The CTRAPS attacks can be deployed using cheap and non-specialized equipment

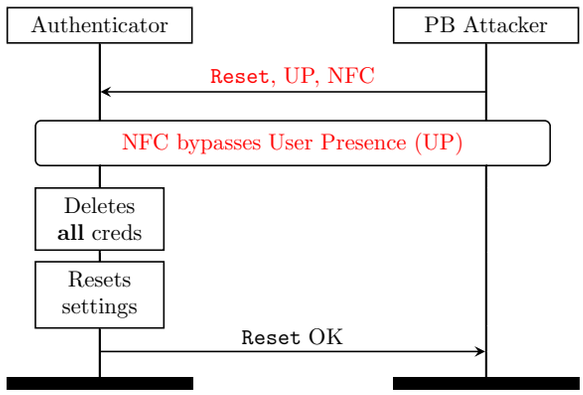



Figure 6.2: Factory reset authenticator attack with proximity CI1. While in NFC range, the attacker calls the `Reset` API. Over NFC, the authenticator skips *UP* and instantly factory resets, deleting all of its discoverable and non-discoverable credentials.

from proximity (e.g., malicious NFC reader) or remotely (e.g., malicious Android application). Remote attacks require different authorizations compared to proximity ones. For example, invoking CTAP APIs over NFC bypasses *UP*.

### 6.4.1 Client Impersonation (CI) Attacks

We present *four* CTAP client impersonation (CI) attacks. We are the first to perform client impersonation in FIDO2, as prior work focused on authenticator impersonation. We are also the first to deploy a NFC reader impersonation on NFC tags as powerful and complex as FIDO2 authenticators. The NFC readers proposed in prior works, such as NFCGate [252], cannot impersonate FIDO2 clients and do not support the advanced cryptography used by FIDO2 authenticators. We now describe the four CI attacks, deployable from proximity or remotely.

**Factory reset authenticator with CI1** Figure 6.2 shows how an attacker abuses the `Reset` API to factory reset the authenticator. In C1, the attacker connects to the authenticator and calls `Reset`, wiping out all discoverable and non-discoverable credentials, user settings and data, none of which can be recovered. The proximity-based attacker enters NFC range to connect to the authenticator. According to the CTAP standard, a proximity-based attacker who enters the authenticator's NFC range is automatically granted *UP*, thereby bypassing the required *UP* check. As a result, proximity CI1 is a *zero-click* factory reset. The remote attacker can achieve the same impact but cannot bypass *UP* and can only attack authenticators recently (i.e., up to ten seconds) plugged in the USB port.

**Profile and track user with CI2** Figure 6.3 shows how an attacker misuses the `GetAssertion` API to profile and track the user. In CI2, the attacker retrieves from the authenticator unique identifiers, employed as fingerprints, that enable her to track the user. The attacker prepared a list of relying parties that utilize the weak *CredProtect=UVOptional* default policy, such as Microsoft and Apple. When connected to the authenticator over USB or NFC, she calls `GetAssertion`, passing the list of weak relying parties as parameters. The remote attacker skips *UP* by also passing *up=false* as an additional parameter. The proximity-based attacker is automatically granted *UP* when entering NFC range. As a consequence, she retrieves, without user consent, all credential and user identifiers registered with the specified relying parties and uses them to fingerprint the user. Each time the user connects his au-

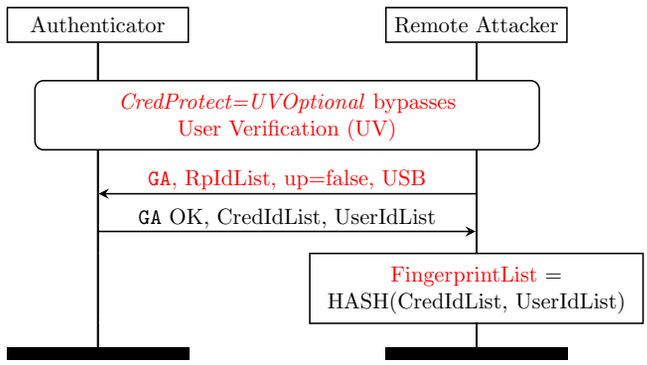



Figure 6.3: Profile and track user attack with remote CI2. The attacker connects over USB and calls the `GetAssertion` API (`GA` in the figure). She bypasses *UV* by only retrieving credential and user identifiers registered with the weak *CredProtect* default policy. Then, she fingerprints the authenticator using the leaked identifiers, allowing her to track the user.

thenticator to a NFC reader or a machine controlled by the attacker, she can track him by performing CI2 again and looking for matching fingerprints. CI2 can even be performed on credentials protected by stronger credential protection policies (i.e., *CredProtect=UVRequired* and *CredProtect=UVOptionalWithCredIDList*). However, this requires *UV* or the knowledge of one or more credential identifiers.

**Force authenticator lockout with CI3** In CI3, the attacker locks the authenticator to prevent its usage, leading to a mandatory factory reset. The attacker abuses the `ClientPin(GetPinToken)` subcommand to submit to the authenticator several wrong PIN guesses in a row. After three wrong guesses, the authenticator enters a soft lock mode preventing further actions until a reboot (i.e., leaving and re-entering a client's NFC range or detaching and re-attaching to a USB port). After a maximum number of failed PIN attempts (CTAP mandates eight), the authenticator enters a hard lock mode only restorable through a factory reset, which wipes out all credentials and can lead to account loss.

**Profile authenticator with CI4** In CI4, the attacker profiles the authenticator through its details. The attacker calls `GetInfo` to retrieve the authenticator's details, such as the manufacturer, model, and FIDO2 version, and the supported algorithms, transports, options, and extensions. The authenticator also leaks some user settings, such as FIDO2 being disabled over a specific transport. All this information can be used to track the user.

### 6.4.2 API Confusion (AC) Attacks

We present *seven* API confusion (AC) attacks, using a novel attack strategy. API confusion is a protocol-level attack strategy that does not depend on FIDO implementation. Consequently, any system speaking CTAP, or another protocol with vulnerabilities similar to the ones we describe in Section 6.5, is susceptible. We are the first to propose API confusion, and we implement it in FIDO2 with a MitM attacker positioned between the client and the authenticator. Prior MitM attacks on FIDO targeted WebAuthn and required a compromised client (e.g., malicious browser extension).

**AC Attack Strategy** In API confusion, the attacker intercepts a call to `API A` and changes (i.e., confounds) it to `API B` with compatible authorization restrictions. We write malicious API

Table 6.2: AC combinations. The rows and columns use API short names (full names in Table 6.1). The user intends to call API A in the first column, instead the attacker calls API B with compatible (same or lower) authorization requirements. ✓[1]: requires proximity-based attacker, ✓[2]: requires default *CredProtect=UVOptional* if non-mandatory credential protection is enabled, n/a: not applicable. The last row counts the API combinations for each AC attack.

| | **AC1** CM | **AC2** Re | **AC3** GA | **AC4** MC | **AC5** CP | **AC6** Se | **AC7** GI |
|---|---|---|---|---|---|---|---|
| MC | ✓ | ✓ | ✓ | n/a | ✓ | ✓ | ✓ |
| GA | ✓ | ✓ | n/a | ✓ | ✓ | ✓ | ✓ |
| CM | n/a | ✓[1] | ✓ | ✓[1] | ✓ | ✓ | ✓ |
| CP | ✓ | ✓[1] | ✓ | ✓[1] | n/a | ✓ | ✓ |
| Re | n/a | n/a | ✓[2] | n/a | ✓ | ✓ | ✓ |
| Se | n/a | ✓ | ✓[2] | n/a | ✓ | n/a | ✓ |
| GI | n/a | ✓[1] | ✓[2] | ✓ | ✓ | ✓ | n/a |
| Total | 3 | 6 | 6 | 4 | 6 | 6 | 6 |

calls in red. We explain API confusion with a *six-step attack strategy*:

① The user calls API A through the client. This API might optionally require *UV* and/or *UP*.

② If required by API A, the attacker obtains *UV* by executing the CTAP PIN/UV authentication protocol v1 (via ClientPin). The user inputs the PIN (i.e., 4 up to 63 Unicode characters) on the client, which encrypts it and submits it to the authenticator. The authenticator responds with an encrypted User Verification Token (UVT), that will be attached to any API call requiring *UV*.

③ The attacker confounds the communication by calling a compatible API B rather than API A (see Table 6.2 for the available AC combinations).

④ If required by API A, the attacker obtains *UP* from the user, unable to realize he is under attack. The attacker can only obtain *UP* once, as multiple requests would alarm the user. This step is skipped over NFC.

⑤ The attacker executes API B. For instance, she deletes all credentials instead of creating a new one. Then, the authenticator returns a success message.

⑥ Optionally, the attacker can perform AC again by calling API C with compatible authorizations. For example, she can perform AC3 to profile the user and then AC5 to lock the authenticator. Finally, the attacker falsely informs the user that API A was successful.

**AC Combinations** Table 6.2 illustrates all 49 confoundable API combinations of the seven CTAP APIs available to all FIDO2 authenticators. Multiple (API A, API B) pairs achieve the same goal. The amount of available pairs depends on their *UV* and *UP* requirements and, in case of AC3, also on the *CredProtect* policy. The first column lists seven APIs the user intends to call (API A), and the remaining columns represent the API called by the attacker (API B). For instance, AC1 is available whenever the user calls MakeCred, GetAssertion, or ClientPin, confounding the call to CredMgmt. Some combinations are only feasible by a proximity-based attacker or under the default *CredProtect* policy. An API cannot be confounded with itself or APIs with incompatible authorization requirements.

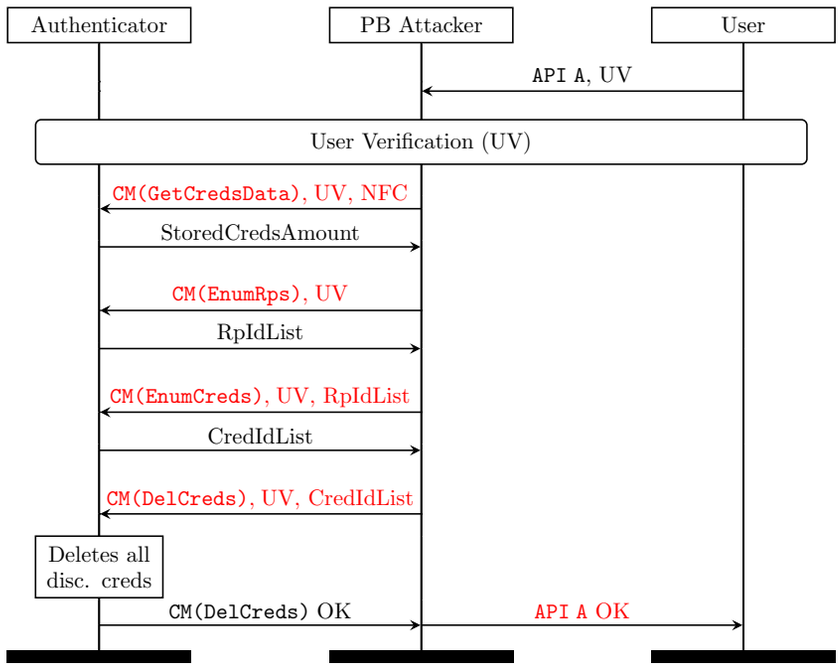



Figure 6.4: Delete discoverable credentials attack with proximity AC1. The user intends to call `API A`, requiring *UV* but not necessarily *UP*. For example, `GetAssertion`, `ClientPin`, or `MakeCred`. The attacker obtains *UV* from the unsuspecting user. Instead of `API A`, she calls `CredMgmt` (`CM` in the figure). She executes four `CredMgmt` subcommands which list and then delete all discoverable credentials on the authenticator.

**Delete discoverable credentials with AC1** Figure 6.4 shows how an attacker abuses the `CredMgmt` API to delete all discoverable credentials stored on the authenticator. The attacker has a MitM position between the authenticator and the client with access to their traffic, allowing her to perform API confusion. The user intends to call `API A`, which requires *UV* but not necessarily *UP*, such as `GetAssertion`, `ClientPin`, or `MakeCred`. Instead, the attacker executes four separate `CredMgmt` subcommands, none of which require *UP* to avoid alarming the user. First, she checks the existance of discoverable credentials to erase (StoredCredsAmount) via `CredMgmt(GetCredsMetadata)`. Second, she retrieves the list of relying parties stored on the authenticator (RpIdList) via `CredMgmt(EnumRps)`. Third, she uses RpIdList to retrieve the list of stored credential identifiers (CredIdList) via `CredMgmt(EnumCreds)`. Fourth, she uses CredIdList to delete all discoverable credentials via `CredMgmt(DelCreds)`. Finally, she falsely returns `API A` OK to the user.

**Factory reset authenticator with AC2** In AC2, the attacker exploits the `Reset` API to factory reset the authenticator, similar to CI1. Since `Reset` only requires *UP*, an attacker over USB can confound `MakeCred`, `GetAssertion`, and `Selection` into a `Reset` call. Additionally, an attacker over NFC can bypass *UP* to also confound `CredMgmt`, `ClientPin`, and `GetInfo`.

**Profile and track user with AC3** In AC3, the attacker misuses the `GetAssertion` API to leak unique identifiers as fingerprints and track the user, similar to CI2. She can confound `MakeCred`, *CredMgmt*, and *ClientPin* into a `GetAssertion` call, if she wants to access to credentials protected by the *CredProtect=UVRequired* or *CredProtect=UVOptionalWithCredIDList* policies. Additionally, the attacker can also confound `Reset`, `Selection`, and `GetIn`

`fo` if she only wants to access to credentials protected by the weak *CredProtect=UVOptional* default policy.

**Fill authenticator credential storage with AC4** In AC4, the attacker repeatedly calls `MakeCred` to register new discoverable credentials, until the authenticator's credential storage is full. She exploits the *rk=true* option to enforce the generation of discoverable credentials over non-discoverable ones. A full storage compromises the authenticator's availability as the user cannot register new discoverable credentials.

**Force authenticator lockout with AC5** In AC5, the attacker abuses the `ClientPin` API to lock the authenticator and force a mandatory factory reset, similar to CI3. Although `ClientPin` requires *UV*, the attacker wants to fail multiple PIN attempts (i.e., she does not need *UV*). Consequently, she can confound any API call into a `ClientPin` call, as she does not need authorization.

**Authenticator DoS with AC6** In AC6, the attacker calls the `Selection` API to trigger an unwanted *UP* check keeping the authenticator busy and denying availability. Since the attacker can detect when the busy state ends (e.g., the user pressed the authenticator's button or 30 seconds have passed), she can prolong the attack indefinitely.

**Profile authenticator with AC7** In AC7, the attacker uses the `GetInfo` API to get the authenticator's details and profile the user, similar to CI4. Due to the lack of authorization, the attacker can confound any API call into a `GetInfo` call.

### 6.4.3 Discussion

The eleven CTRAPS attacks impact the security and privacy of the entire FIDO2 ecosystem as they exploit *protocol level* CTAP vulnerabilities (detailed in Section 6.5). Hence, they can be conducted on any FIDO2-compliant device, regardless of hardware and software details. Our attacks have severe real-world implications such as deleting FIDO2 credentials to make users lose their linked account, tracking users, or disabling the authenticator's functionalities. Moreover, CI and AC attacks can be *combined*. For example, the attacker obtains a fingerprint with AC7 and tracks the user with CI2.

The attacks are *practical* and *low-cost* because we propose attacker models also found in [252, 253] that require minimal equipment (e.g., NFC reader or smartphone). Moreover, the attacks are *stealthy* as rely on expected user interaction (i.e., *UV* and *UP*) or no user interaction. We support our claims with the cheap toolkit we present in Section 6.6 and the experimental results in Section 6.7.

## 6.5 Vulnerabilities

We discuss *seven vulnerabilities* found in the CTAP standard, six of which are novel. Only V2 is known, but prior works focused on its implications for WebAuthn, like enabling authenticator misbinding attacks. In contrast, we find a new method to exploit V2 through API confusion.

Table 6.3 maps the seven vulnerabilities (columns) to our eleven CTRAPS attacks (rows). V1 and V2 are the main root cause for CI and AC attacks, respectively. The remaining vulnerabilities enhance the effectiveness of our attacks, by increasing their impact, scope, and stealthiness or by lowering their requirements.

All CI and AC attacks exploit V1, as the authenticator cannot reject the API called by the attacker. AC1, AC3, and AC4 exploit V2 as they confound APIs to destructive ones. CI1 and AC2 exploit V3 to factory reset the authenticator without *UV*. AC1 exploits V4 to delete discoverable credentials without *UP*. CI2 and AC3 exploit V5 to track a victim via their persistent credential and user identifiers. AC6 exploits V6 to DoS an authenticator via periodic `Selection` calls. All CI and AC attacks exploit V7, as it allows the attacker to pose as a trusted CTAP client. All CI and AC attacks exploit V8 to be stealthy, as the victim receives no feedback from unathorized API calls. Proximity-based AC1, CI1, AC2, CI2, AC3, and AC4 exploit V9 to send unauthorized CTAP commands over NFC.

Table 6.3: Mapping the seven vulnerabilities (columns) to the eleven CTRAPS attacks (rows).

| | **V1** | **V2** | **V3** | **V4** | **V5** | **V6** | **V7** |
|---|---|---|---|---|---|---|---|
| CI1 | ✓ | ✓ | ✓ | ✗ | ✗ | ✓ | ✗ |
| CI2 | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ |
| CI3 | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| CI4 | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| AC1 | ✓ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ |
| AC2 | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ |
| AC3 | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ |
| AC4 | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ |
| AC5 | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| AC6 | ✓ | ✓ | ✗ | ✗ | ✗ | ✓ | ✓ |
| AC7 | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |

**V1**: Unauthenticated CTAP client, **V2**: No authenticator feedback from API calls, **V3**: *UP* over NFC does not require user interaction, **V4**: Same PIN authorizes destructive and non-destructive API calls, **V5**: Trackable and profilable CredId and UserId, **V6:** Insufficient *UV* and *UP* requirements for `Reset` and `CredMgmt`, **V7**: `Selection` can be exploited for DoS

**V1: Unauthenticated CTAP client** The CTAP client does not authenticate to the user, the authenticator, or the relying party. FIDO2 clients (and by extension, CTAP clients) have no identity meaning that the authenticator has no way to distinguish an official client developed by its manufacturer from a third-party client. The authenticator has no choice but to always trust any connecting client, including impersonators. V1 enables all CI attacks and lowers the requirements for all AC attacks (i.e., easier to MitM an unauthenticated channel).

**V2: No authenticator feedback from API calls** Despite having a LED, the authenticator does not provide the user with visual feedback when invoking APIs or granting *UV* and *UP*. The user cannot confirm whether the intended API has been called (or confounded) and which API consumed the *UV* and *UP* authorization he just granted. V2 enables all AC attacks and increases the stealthiness of all CI attacks.

**V3: *UP* over NFC does not require user interaction** Authenticators inside the NFC range of a FIDO2 client automatically obtain *UP* without user interaction (e.g., without the user pressing a button on the authenticator). By removing the *UP* requirement, V3 improves the effectiveness and stealthiness of proximity attacks. For example, V3 enables a zero-click NFC factory reset (i.e., CI1) and increases the combinations of confoundable APIs.

**V4: Same PIN authorizes destructive and non-destructive API calls** The same *UV* PIN to authorizes *destructive* operations, such as credential deletion (`CredMgmt`) or factory reset (`Reset`), and *non-destructive* ones, such as authentication (`GetAssertion`). Moreover, the user has no way to configure the authenticator to disable destructive API calls (e.g., temporarily turn-off factory resets). V2 allows AC attacks to escalate the confusion from non-destructive APIs to destructive ones. For example, the user intends to authenticate (non-destructive) by providing *UV* and *UP*, but the attacker factory resets the authenticator instead (destructive).

**V5: Trackable and profilable CredId and UserId** Discoverable credentials contain *static* and *unique* CredId and UserId, exploitable to reliably profile and track users. By default, `GetAssertion` retrieves CredId and UserId from the authenticator without requiring *UV* or *UP*. V5 enables two zero-click user tracking attacks (i.e., CI2 and AC3). Storing more credentials on the authenticator makes the user more profilable and trackable. A relying party can protect the identifiers with the optional *CredProtect* extension at registration time, preventing CI2 but not AC3.

**V6: Insufficient *UV* and *UP* requirements for `Reset` and `CredMgmt`** The `Reset` and `CredMgmt` should enforce stricter authorization requirements. Despite being destructive, the `Reset` API does *not* require *UV*, but only *UP*. Anyone close to the authenticator can obtain *UP* by pressing its button or putting it inside the client's NFC range. The `CredMgmt` API does *not* require *UP*, but only *UV*, to delete discoverable credentials. The user submits the PIN only once but can delete any number of credentials. In contrast, creating N credentials also requires N *UP* checks. V6 allows for a zero-click NFC factory reset (i.e., CI1) and a one-click discoverable credential deletion (i.e., AC1).

**V7: `Selection` can be exploited for DoS** The `Selection` API does *not* support temporal rate limiting. Hence, an attacker can keep the authenticator in an unresponsive state by periodically requesting unwanted *UP* checks via `Selection`. V7 enables a persistent denial-of-service attack on the authenticator (i.e., AC6).

## 6.6 Implementation

In this section, we present `CTRAPS`, a novel toolkit implementing the CTRAPS attacks, and its *three* modules: CTAP testbed (Section 6.6.1), CTAP Client impersonation (Section 6.6.2), and Wireshark dissectors (Section 6.6.3).

### 6.6.1 CTAP Testbed

We develop a CTAP testbed composed of a *virtual relying party* and a *virtual client* in a Python3 module. The virtual relying party extends `python-fido2` [254], Yubico's open-source Python library for FIDO2. It includes a customizable WebAuthn server to simulate traffic with a backend, and CTAP and WebAuthn clients to simulate traffic with an authenticator and a relying party. This setup has two main benefits. (i) We perform our experiments locally, without interacting with real relying parties thus avoiding the constant creation of new dummy accounts and need for an Internet connection. (ii) We fully customize our relying party to imitate any existing one, including support for discoverable credentials and extensions (e.g., credential protection policy via *CredProtect*).

The virtual client is a custom CTAP client based on a open-source FIDO library [254],

exposing a low-level and customizable CTAP Authenticator API. Our client can send individual CTAP commands in any order with custom payload values, instead of abstracting them with high-level APIs. It also allows to set custom authorization requirements for the APIs. For example, the client app can change the client data (e.g., WebAuthn operation type, challenge, and origin), and CTAP options (e.g., *rk* and *up* to enable/disable discoverable credentials and user presence, respectively).

We also wrote a malicious library that supports the CTAP over USB (i.e., CTAPHID) protocol via the *node-hid* [255] module for Node [256] JavaScript. Our library monitors and hijacks USB Human Interface Device (HID) communications, enabling us to test the interactions between the authenticator and the client. Since CTAPHID encapsulates CTAP messages over USB HID, we had to implement a CTAPHID parser that properly handles CTAPHID packet fragmentation. Moreover, since CTAP messages are encoded using the Concise Binary Object Representation (CBOR), we implemented a custom CBOR parser using the `cbor` [257] module.

Our CTAP testbed requires the user to authorize direct access to the authenticator. Linux requires adding extra `udev` rules, MacOS asks to accept a notification on the screen, and Windows needs admin privileges.

### 6.6.2 CTAP Client Impersonation

We provide two implementations for a malicious CTAP client impersonation over NFC. First, we develop custom Lua scripts for the Proxmark3 [258] that deploy our proximity-based CI attacks. The Proxmark3 is a programmable RFID tool that can function as a NFC reader. We use it with the built-in (i.e., range of 40 to 85 millimeters) and long range (i.e., range of 100 to 120 millimeters) high frequency antennas.

Second, we develop a malicious library that supports the custom CTAP NFC protocol encapsulated with the ISO7816/ISO14443 contacless protocol [259]. We also provide a NFC attacker app that imports our library to impersonate a legitimate FIDO2 app and deploy our proximity-based CI attacks. We use it to test a proximity-based attacker that has no access to specialized hardware (e.g., a Proxmark3), but owns a NFC-enabled Android smartphone.

We wrote our library in Java using the `android.nfc` [260] API. It requires the normal `android.permission.NFC` permission. The library processes the authenticator as an IsoDep NFC tag and utilizes the `transceive` method to exchange binary data over-the-air, that needs to be interpreted and decoded (e.g., with a custom CBOR parser).

### 6.6.3 FIDO Wireshark Dissectors

We release our FIDO2 Wireshark dissectors, that we extensively used to analyze the CTAP traffic during our research and experiments. They simplify the visualization of CTAP transport-specific bindings and their binary content, allowing for faster debugging of communications between the authenticator and the client (e.g., when crafting arbitrary CTAP packets that might be rejected). Our dissectors extend the official ones [261] with new and valuable features. For example, we add support for `CredMgmt` and we parse `WAITING` and `PROCESSING` keepalive status codes that identify when authenticators are unavailable waiting for *UP*. Moreover, we parse the authenticator's capabilities in the `CTAPHID_INIT` message, which are useful for testing AC7. We provide an improved way to display CTAP data when dissecting CTAPHID (USB)

and ISO7816/ISO14443 (NFC). Finally, we add missing vendor and product identifiers to the dissector tables. See `fido2-dissectors.lua` in our toolkit for more details.

## 6.7 Evaluation

We successfully evaluated our eleven attacks against *six* popular authenticators from Yubico, Feitian, SoloKeys, and Google supporting CTAP over USB and NFC, and *ten* well-known relying parties, including Microsoft, Apple, GitHub, and Facebook. We tested our two attacker models by deploying the attacks with our toolkit (presented in Section 6.6).

### 6.7.1 Setup

**Authenticators** Table 6.4 shows the specifications of the six authenticators we exploited. The YubiKey 5 NFC, YubiKey 5 NFC FIPS, and Feitian NFC K9 are *closed-source* and do not support firmware updates, The Solo V1, Solo V2 Hacker, and Open Security Key (OpenSK) have an *open-source firmware* (OSF), that we updated to their latest version. All authenticators support both USB and NFC, except for OpenSK which, despite featuring a NFC module, only supports USB. Unfortunately, we could not find any FIDO2 authenticator supporting BLE. The authenticators store a maximum of 25 (Yubico), 50 (Feitian and SoloKeys), or 150 (OpenSK) discoverable credentials. The YubiKey 5 FIPS is FIPS140-2 compliant, hence its cryptographic modules and secrets have high security guarantees. We run OpenSK on an NRF52840 dongle, but the attacks are effective regardless of its hardware.

Table 6.4: Details about the six authenticators we attack. All authenticators support USB and NFC, except OpenSK, which only supports USB. FVer: firmware version, OSF: open-source firmware, DCred: discoverable credentials.

| **Authenticator** | **Manuf** | **Year** | **FVer** | **OSF** | **DCred** |
|---|---|---|---|---|---|
| YubiKey 5 | Yubico | 2018 | 5.2.7 | No | 25 |
| YubiKey 5 FIPS | Yubico | 2021 | 5.4.3 | No | 25 |
| Feitian K9 | Feitian | 2016 | 3.3.01 | No | 50 |
| Solo V1 | SoloKeys | 2018 | 4.1.5 | Yes | 50 |
| Solo V2 Hacker | SoloKeys | 2021 | 2.964 | Yes | 50 |
| OpenSK | Google | 2023 | 2.1 | Yes | 150 |

**Relying parties** We registered our authenticators with ten FIDO2 relying parties (i.e., Adobe, Apple, DocuSign, Facebook, GitHub, Hancock, Microsoft, NVidia, Synology, and Vault Vision). Our list covers pervasive and heterogeneous web services, including software as a service, social, gaming, cryptographic signing, authentication, and cloud storage. We highlight that a FIDO2 credential can authenticate accounts from multiple online services. So, by deleting a Microsoft credential we also jeopardize other services such as OneDrive, Outlook, and Minecraft. This affects even services *not* directly supporting FIDO2 but relying on it like ChatGPT, which utilizes Single Sign-On (SSO) through FIDO2-enabled Microsoft or Apple credentials.

Table 6.5: CI and AC attacks on six authenticators. The first column lists the authenticators' names. The remaining columns report our four CI and seven AC attacks on CTAP. ✓: attack is effective on the authenticator, **n/a**: attack not applicable as the authenticator does not implement the API.

| Authen. | CI1 | CI2 | CI3 | CI4 | AC1 | AC2 | AC3 | AC4 | AC5 | AC6 | AC7 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| YubiKey 5 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | n/a | ✓ |
| YubiKey 5 FIPS | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | n/a | ✓ |
| Feitian K9 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | n/a | ✓ |
| Solo V1 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | n/a | ✓ |
| Solo V2 Hacker | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| OpenSK | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

**CI1**: Factory reset authenticator, **CI2**: Profile and track user, **CI3**: Force authenticator lockout, **CI4**: Profile authenticator, **AC1**: Delete discoverable credentials, **AC2**: Factory reset authenticator, **AC3**: Profile and track user, **AC4**: Fill authenticator credential storage, **AC5**: Force authenticator lockout, **AC6:** Authenticator DoS, **AC7**: Profile authenticator.

**CTAP Testbed** We installed our CTAP testbed on a Dell Inspiron 15 3502 laptop (OSes: Ubuntu 22.04.3 LTS and Windows 11 Home) and on a MacBook Pro M1 (OS: MacOs Ventura 13.4). We installed our Android NFC attacking app on a rooted Google Pixel 2 (OS: Android 11), a non-rooted RealMe 11 Pro (OS: Android 14) and Xiaomi Redmi Plus 5 (OS: Android 8.1). We ran our custom Lua scripts on a Proxmark3 RDV2 using the built-in and the long range high frequency antennas. We deployed our proximity-based attacks using the Proxmark3 and the Android NFC attacker app by bringing the authenticators inside the malicious device's NFC range. We tested multiple NFC attack scenarios such as attacking an authenticator hidden in the user's pockets (e.g., a user seated on a bus) or sitting on top of a table that conceals a malicious NFC reader. We deployed our remote attacks using the CTAP testbed on our two laptops by plugging the authenticators in one of their USB ports.

### 6.7.2 Authenticators Results

Table 6.5 shows the evaluation results for the CTAP client impersonation and MitM API confusion attacks. In total, we tested four CI attacks and seven AC attacks from proximity and remotely. Despite being a standard CTAP API, only the Solo V2 Hacker and the OpenSK implement `Selection`. Consequently, AC6 does not work on the other four authenticators. The remaining CI and AC attacks are effective on all tested authenticators, including the FIPS-compliant YubiKey.

The proximity-based CI and AC attacks required 50 to 500 milliseconds within NFC range to complete, depending on their complexity (e.g., AC1 takes the longest). Allegedly, the NFC range of smartphones is four centimeters or less [262]. In our experiments, we achieved up to three centimeters of range, depending on the smartphone model. This reduction is expected, as our NFC signals travel through solid material (e.g., clothes). The Proxmark3 built-in antenna also achieved up to three centimeters of range, that we extended up to ten centimeters by using its long range antenna. However, prior work demonstrated that, with specialized equipment, the NFC range can be extended up to 50 centimeters [263].

Table 6.6: CTRAPS attacks on ten relying parties. The first and second columns list the relying parties' names and identifiers. The third column highlights whether they register discoverable (D, DW) or non-discoverable (ND) credentials. We indicate with DW a relying party using the default and weak *CredProtect=UVOptional* policy. Columns four, five, and six specify the effect of each attack. n/a: the attack is not applicable because the relying party currently does not support discoverable credentials.

| Rp | RpId | Cred | Delete Creds | Track User | DoS Authen |
|---|---|---|---|---|---|
| Adobe | `adobe.com` | D | AC1,CI1,AC2 | CI2,AC3 | AC4,CI3,AC5,AC6 |
| Apple | `apple.com` | DW | AC1,CI1,AC2 | CI2,AC3 | AC4,CI3,AC5,AC6 |
| DocuSign | `account.docusign.com` | ND | CI1,AC2 | n/a | CI3,AC5,AC6 |
| Facebook | `facebook.com` | ND | CI1,AC2 | n/a | CI3,AC5,AC6 |
| GitHub | `github.com` | D | AC1,CI1,AC2 | CI2,AC3 | AC4,CI3,AC5,AC6 |
| Hancock | `hancock.ink` | D | AC1,CI1,AC2 | CI2,AC3 | AC4,CI3,AC5,AC6 |
| Microsoft | `login.microsoft.com` | DW | AC1,CI1,AC2 | CI2,AC3 | AC4,CI3,AC5,AC6 |
| NVidia | `login.nvgs.nvidia.com` | D | AC1,CI1,AC2 | CI2,AC3 | AC4,CI3,AC5,AC6 |
| Synology | `account.synology.com` | D | AC1,CI1,AC2 | CI2,AC3 | AC4,CI3,AC5,AC6 |
| Vault Vision | `auth.vaultvision.com` | D | AC1,CI1,AC2 | CI2,AC3 | AC4,CI3,AC5,AC6 |

As expected, since we attack CTAP at the protocol level, the attacks are effective regardless of the CTAP transport (i.e., USB or NFC), or the authenticator's software and hardware. We discovered that the Solo V1 requires a button press to activate the NFC interface, as opposed to all other authenticators that do not require any user interaction to activate NFC. We also found an implementation vulnerability on the YubiKey 5 and YubiKey 5 FIPS, that we discuss in Section 6.9.2.

### 6.7.3 Relying Parties Results

Our attacks have a *direct* (AC1, CI1, AC2, CI2, and AC3) or *indirect* (AC4, CI3, AC5, and AC6) impact on relying parties, despite the lack of direct communications (via WebAuthn). AC1, CI1, and AC2 prevent access to a relying party via credential deletion. CI2 and AC3 utilize the user identifiers provided by a relying party to track users. AC4, CI3, AC5, and AC6 indirectly affect relying parties by preventing them from authenticating a registered authenticator. AC1 and AC4 are not applicable to non-discoverable credentials (i.e., DocuSign and Facebook).

Table 6.6 shows that all evaluated relying parties are directly or indirectly affected by our attacks regardless of their credential type (i.e., D or ND). We analyzed eight relying parties supporting discoverable credentials and two employing non-discoverable ones. We also discovered that Microsoft and Apple ignore the *CredProtect* extension, making it non-enforceable by the user. As a consequence, their weak *UVOptional* credential policy (DW) enables the more severe (*UV-less*) CI2 and AC3 attacks.

Since users cannot recover deleted FIDO2 credentials, victims of CTRAPS attacks risk losing access to their online accounts. For example, when testing AC1, CI1, and AC2, we lost access to our dummy Apple ID account which we could only recover after two weeks due to Apple's policies.

## 6.8 Countermeasures

We present the design and evaluation of *seven* practical and backward-compliant countermeasures fixing our eleven attacks and their seven root causes. Each countermeasure reduces the CTAP attack surface and maps to a specific vulnerability (e.g., C1 addresses V1). Our countermeasures are implementable as amendments to the FIDO2 standard or as FIDO2 extensions.

**C1: Trusted CTAP clients** We address V1 by recommending that FIDO provide a list of trusted CTAP clients. FIDO offers several certifications, including the FIDO Functional Certification [264] which only attests the *interoperability* of clients, servers, and authenticators. We suggest extending this certification to also cover the *trustworthiness* of CTAP clients. For instance, FIDO could implement a Software Bill Of Materials (SBOM) solution to monitor trusted CTAP clients and their vulnerabilities.

**C2: Authenticator visual feedback** We address V2 by requiring the authenticator to provide visual feedback of the called APIs. For instance, the authenticator's LED could blink *once* for non-destructive API calls and *twice* for destructive ones. The CTAP *wink* command, which blinks the LED, must be disabled during this visual feedback.

**C3: User interaction for *UP* over NFC** We address V3 by requiring user interaction during *UP* checks over NFC. For example, the user could press a button on the authenticator to grant *UP* over NFC, similar to *UP* checks over USB.

**C4: Dedicated PIN for destructive APIs** We address V4 by introducing a dedicated PIN to authorize destructive API calls (e.g., `CredMgmt` and `Reset`) and by repurposing the current PIN to authorize non-destructive API calls (e.g., `Selection` and `GetInfo`). The new PIN should have the same or stricter requirements as the non-destructive PIN (i.e., four to sixty-three Unicode characters [242]).

**C5: Dynamic and *UV*-protected CredId and UserId** We address V5 by implementing dynamic CredId and UserId and mandating *CredProtect=UVRequired*. CredId and UserId should rotate after a set amount of logins (e.g., every ten logins) or a time interval (e.g., once per month). Hence, we raise the bar for user profiling and tracking attacks on authenticators. Currently, the user can indirectly change a CredId by calling `MakeCred` to generate a new credential for his account, replacing the old one. However, the user cannot change the UserId, which is determined by the relying party and, based on our experience, remains fixed to the user account.

**C6: `Reset` and `CredMgmt` must require *UV* and *UP*** We address V6 by requiring both *UV* and *UP* to call `Reset` and `CredMgmt`. Hence, the user must validate a factory reset by entering a valid PIN and authorize credential deletion one by one, by pressing the authenticator's button, making it impossible to delete multiple credentials with a single API call.

**C7: Rate limiting `Selection` calls** We address V7 by enforcing temporal rate limiting on `Selection` calls to a maximum of three calls within two minutes. We are not expecting issues with our rate limiting, akin to the limiting already existing for `ClientPin(GetPinToken)`, as a client typically calls `Selection` once per session.

**Usability** We believe that the stronger security granted by our countermeasures is worth the inevitable usability trade-offs. C1, C2, and C7 do not affect usability. C5 only introduces one additional *UV* and *UP* check every time the credential and user identifier need to rotate out (e.g., once per month), barely affecting usability. On the other hand, C4 requires the user to remember a second PIN, and C6 add more authorization requirements to `Reset` and `CredM`

`gmt`. C3 also adds user interaction when connecting to the authenticator over NFC.

**Adding a display** We do not consider adding a display to a roaming authenticator an optimal solution as it is *not backward-compliant*. Millions of deployed authenticators would remain vulnerable. Moreover, it would require significant hardware and software modifications, such as adding a secure display, a display controller firmware, and a battery, that would introduce usability, performance, and cost issues.

## 6.9 Discussion

### 6.9.1 FIDO2 Reference Threat Model Issues

FIDO2 has a non-normative *reference threat model* [236] that includes security assumptions, goals, and threats against clients, authenticators, and relying parties. We found *three issues* (IS1, IS2, and IS3) with their threat model:

**IS1: Unclear security boundaries** The threat model presents six broad security assumptions but then violates them when discussing threats. For example, SA-4 states that the FIDO user device and applications involved in a FIDO operation act as trustworthy agents of the user. This implies that the FIDO client (e.g., the user's browser or mobile app) must be inherently trusted. However, the threat model includes threats breaking SA-4 like *T-1.2.1: FIDO client corruption* that identifies an attacker with code execution on a FIDO client. Instead, security assumptions should hold to enable a security analyst to draw security boundaries (i.e., differentiate what we trust from what can be attacked).

**IS2: Proximity threats are missing** Despite FIDO supporting proximity transports like NFC and BLE, the threat model classifies proximity-based threats in the same category as physical access threats, even though these two threats have significant differences. For example, compared to physical access, the range of a proximity attack can be extended. Hence, our proximity attacks, which do not require physical access, cannot be accurately described within this reference threat model.

**IS3: Security goals are narrow** The threat model has narrow security goals based on [265] (2006) and [266] (2012). The security goals focus on web authentication but overlook FIDO clients and roaming authenticators. For example, there are no security goals for the Authenticator API (i.e., addressing all AC attacks) or discoverable credentials (i.e., addressing AC1, CI1, and AC2).

### 6.9.2 Yubico `CredMgmt` Implementation Vulnerability

By testing API calls from our CTAP virtual client, we found a `CredMgmt` implementation vulnerability on the YubiKey 5 and YubiKey 5 FIPS that violates the CTAP specification. For example, the subcommand `CredMgmt(EnumRpsGetNextRp)` should be executed by the authenticator only if `CredMgmt(EnumRpsBegin)` was called first. Instead, such subcommand is always available on Yubico's authenticators due to an incorrect handling of their state.

We exploit this flaw to *leak* confidential data and *track* users. In particular, we skip calling `CredMgmt(EnumRpsBegin)`, which requires *UV*, and call `CredMgmt(EnumRpsGetNextRp)`, which does not require *UV*. As a consequence, we retrieve all but one relying parties linked to discoverable credentials stored on the authenticator, without needing *UV* and regardless of the *CredProtect* policy.

## 6.10 Related Work

We review and summarize FIDO and FIDO2 literature. In particular, we focus on attacks, formal analysis, proposed extensions and enhancements, usability studies, and surveys.

Table 6.7 quantitatively compares our work with a selection of prior attacks on FIDO. Our CI attacks present the first NFC reader impersonation on FIDO and our AC attacks introduce a new strategy called CTAP API confusion Compared to other works that analyzed contactless payment and keyless entry, FIDO is a unique case study where the NFC tag is smart and could, but does not, authenticate to the reader, allowing for impersonation. We are the first to target all CTAP versions and also all transports. We are the first to exploit the whole CTAP Authenticator API, instead of focusing on a specific API. Our CI attacks are low-cost as they do not require a compromised client or user device, or prior knowledge. Our AC attacks have a moderate cost, as they require a man-in-the-middle position between the authenticator and the client. The CTRAPS attacks have higher impact than most other previous attacks, as, for example, they can permanently destroy all credentials and track users.

**Attacks on FIDO and FIDO2** Researchers demonstrated practical attacks on older FIDO versions, such as authenticator rebinding, parallel sessions, and multi-user attacks [268, 269], USB HID man-in-the-middle attacks [253], BLE pairing [270], relying party public key substitution [61], bypassing push-based 2FA [271], real-time phishing [272], and side channel attacks [37, 38]. FIDO2 was also found vulnerable to side channel attacks [39] and rogue key or impersonation threats [267]. Moreover, attacks on lower layers trusted by FIDO2 were presented including IV reuse on Samsung Keystore [62]. No prior attack investigated *API confusion* on CTAP, including its *latest* version.

**Formal Analysis** The formal analysis and verification community extensively researched the FIDO standards. The community formally verified FIDO's Universal Authentication Framework (UAF) [28, 273], FIDO2 (including its privacy, revocation, attestation, and post-quantum crypto) [60, 274, 68, 69]. Yubico proposed a key recovery mechanism based on a backup authenticator that was proven secure using the asynchronous remote key generation (ARKG) primitive [67]. The formal analyses are *not* covering our CI and AC attacks.

**Extensions** FIDO supports extensions to add optional features in a backward-compliant way. For instance, FeIDO [275] proposes an extension to recover a FIDO2 credential using an electronic identifier. Extensions are not secure by default, and researchers proposed a fix to protect them against MitM attacks [276]. We suggest to *update* the CTAP specification rather than implementing our countermeasures as FIDO extensions that would be optional and insecure by design.

**Enhancements** Researchers proposed (cryptographic) enhancements to FIDO protocols. In [277], the authors present a hybrid post-quantum signature scheme for FIDO2 and tested it using OpenSK [278] (which we exploit in this work). In [279], the authors propose a global key revocation procedure for WebAuthn that revokes credentials without communicating to each individual relying party WebAuthn server. Proposed enhancements are *not* addressing our attacks, which are effective *regardless of* the FIDO2 cryptographic primitives.

**Usability** Researchers performed extensive usability studies on FIDO U2F [280, 281, 282, 283], FIDO2 roaming authenticators [284, 285], passkeys [286], and cross-site 2FA [287]. Our paper is *orthogonal* to usability studies.

**Surveys** There are several FIDO survey papers. In [288] the authors describe the evolution of FIDO protocols, security requirements, and adoption factors. In [289], the authors surveyed

Table 6.7: Comparison between recent attacks on the FIDO protocol. We consider the attack class, attacked protocols, transports, and surface, whether the attack was implemented and its requirements. We also assign them a cost and impact. For example, the cost for a MitM attacker is Mid and for a proximity attacker is Low. Similarly, hijacking a session has a Medium impact, and permanently destroying credentials has a High impact.

| **Attack** | **Class** | **Protocol** | **Transp** | **Surface** |
|---|---|---|---|---|
| Auth MitM [233] | DH MitM | CTAP2.0 | All | ClientPin |
| Priv leak [233] | Eavesdropping | CTAP2.0 | All | MakeCred |
| Auth rebind [233] | Auth rebind | WebAuthn | All | Creds.create |
| Parallel sess [233] | Session hijack | WebAuthn | All | Creds.get |
| Evil maid [39] | Phys access | n/a | n/a | Auth TEE |
| Titan phone imp [39] | Impersonation | U2F | BLE | Android |
| Titan key imp [39] | Impersonation | U2F | BLE | Google Acc |
| Auth MitM [267] | DH MitM | CTAP2.0/2.1 | USB | ClientPin |
| Web MitM [267] | Session hijack | WebAuthn | USB | Creds.get |
| Rogue key [267] | Auth rebind | WebAuthn | USB | MakeCred |
| *CTRAPS CI* | Impersonation | CTAP2.0/2.1/2.2 | All | Auth API |
| *CTRAPS AC* | MitM API confusion | CTAP2.0/2.1/2.2 | All | Auth API |
| **Attack** | **Impl** | **Reqs** | **Cost** | **Impact** |
| Auth MitM [233] | ✗ | n/a | Mid | Mid |
| Priv leak [233] | ✗ | n/a | Low | Low |
| Auth rebind [233] | ✗ | n/a | High | High |
| Parallel sess [233] | ✗ | n/a | Mid | Mid |
| Evil maid [39] | ✗ | Phys access | High | High |
| Titan phone imp [39] | ✓ | Proximity | Low | Mid |
| Titan key imp [39] | ✓ | Proximity | Low | Mid |
| Auth MitM [267] | ✓ | Mal browser | Mid | Mid |
| Web MitM [267] | ✓ | Mal browser | Mid | Mid |
| Rogue key [267] | ✓ | Mal browser | Mid | High |
| *CTRAPS CI* | ✓ | Proximity | Low | High |
| *CTRAPS AC* | ✓ | MitM | Mid | High |

the adoption of passwordless authentication among a large user base, considering users' perception, acceptance, and concern with single-factor authentication without passwords. Our paper is *orthogonal* to surveys.

## 6.11 Conclusion

No prior work assessed the security of the CTAP Authenticator API, despite being a core protocol of the FIDO2 standard. To address this gap, we present the first security and privacy evaluation of FIDO2's CTAP Authenticator API. We deploy the first CTAP client impersonation in FIDO2, enabling an attacker to call CTAP APIs without authorization or user interaction. We also introduce a novel attack strategy called API confusion, that changes, without user consent, the API called by the user to an API chosen by the attacker.

We uncover eleven new proximity-based and remote attacks that can severely impact millions of FIDO2 users. For example, our attacks delete FIDO2 credentials and master keys (security breach) and track users through their credentials (privacy breach). The attacks are effective on the entire FIDO2 ecosystem as they target seven vulnerabilities we discovered in the CTAP specification. These flaws include the lack of CTAP client authentication and improper API authorizations. CTRAPS attacks are low-cost, as they do not require specialized or expensive equipment, and stealthy, as they do not trigger unexpected user interactions.

We develop the `CTRAPS` toolkit to test our attacks with a low-cost setup and on a large scale. It includes a CTAP testbed with a virtual relying party and a virtual client, a CTAP client NFC impersonator (i.e., malicious Proxmark scripts and Android NFC app), and enhanced Wireshark dissectors for CTAP. We successfully exploit six authenticators and ten relying parties from leading FIDO2 players such as Yubico, Feitian, Google, Microsoft, and Apple. We develop seven effective and legacy-compliant countermeasures to fix our attacks and their root causes.

We share *three lessons* we learned about FIDO2 *credential storage* and *passwordless-ness*, which are valuable for the current transition from single-factor authentication to 2FA and passkeys [290, 291]: (i) Being stored on the authenticator, FIDO2 discoverable credentials are protected from third-party data breaches. However, this introduces new attacks that work exclusively on discoverable credentials (i.e., AC1, CI2, AC3, and AC4); (ii) FIDO2 users cannot prevent attacks targeting discoverable credentials, as they cannot choose the type of credentials they register and their protection policies, decided by the relying party and the client instead. (iii) FIDO2 core message is to steer away from passwords because they are vulnerable to phishing. However, digging deeper, we realized that FIDO2 still relies on phishable mechanisms, even for passwordless authentication. For instance, a passwordless credential is protected by an alphanumeric PIN (i.e., a phishable sequence the user must remember).

# Chapter 7

# Conclusion

In this thesis, we analyzed the security of multiple IoT protocols, underscoring the immaturity of security and privacy practices. We developed a security testing approach extensible to fitness trackers, e-scooters, and FIDO2 authenticators. As a result, we release open-source tools for virtualizing IoT devices (i.e., fitness trackers and e-scooters), clients (i.e., Xiaomi companion apps and FIDO2 clients), and backend (i.e., relying party) and attack them, analysis tools for firmware and code (i.e., e-scooter firmware patcher and Frida hooks), and protocol dissectors. We find twenty-seven vulnerabilities that enable twenty-three impactful and low-cost attacks at the protocol-level, utilizing novel attack strategies (e.g., API confusion) and targeting unexplored attack surfaces (e.g., e-scooter internals). Among them, we highlight four vulnerabilities and nine attacks that break user privacy, to the point of reading health data protected by GDPR. We improve insecure protocols, applying state-of-the-art security mechanisms while preserving their backward-compatibility, release vulnerability assessment tools to test devices with known issues, and firmware patches to protect users utilizing vulnerable devices. We explored four challenging research questions about improving security testing, finding new vulnerabilities and attacks, securing IoT ecosystems, and reviewing the state of privacy. As a result to our journey, we now discuss four key takeways from our research.

First, IoT security testing still demands significant manual effort. Despite the variety of existing techniques, none provided a reliable solution for analyzing closed-source protocols, especially when working with a limited set of protocol packets and seeking sophisticated attack outcomes. These criteria were essential to our work with proprietary protocols, where protocols like Mi Band pairing may only involve four messages, and our focus was on critical security properties like confidentiality and integrity. We envision a framework that: (i) identifies high-level device operations (e.g., pairing, session establishment, authentication, and firmware updates); (ii) collects data on the IoT attack surface (e.g., device-to-app and app-to-backend traffic, app execution traces, API endpoints, and firmware); and (iii) matches this data to known patterns to reverse-engineer protocol logic and reveal its design. For example, this framework could detect the use of cryptographic primitives through instrumentation in companion apps, track session key generation, and decrypt traffic based on observed patterns. It could also infer that unencrypted packets containing public keys are part of pairing, while encrypted messages represent session establishment.

Second, the rapid evolution of IoT devices is not met with corresponding security improvements. This results in an ever-expanding range of new features that introduce fresh vulnerabilities, making devices more susceptible to stronger attacks over time. For instance, FIDO2's

introduction of discoverable credentials exposed a new attack surface, allowing attackers to erase credentials and prevent users from accessing their accounts. We advocate for greater transparency from vendors, many of whom make bold claims about the security of their products primarily for marketing purposes, rather than to genuinely protect users. FIDO2 advertises phishing protection, while still relying on a PIN (i.e., a phishable code) and being vulnerable to many social engineering and UI deception attacks. We also notice that manufacturers offload security responsibilities onto user instead of addressing the risks themselves. For example, Xiaomi's security approach requires users to be in a secure environment every time they activate the headlight on their e-scooter, as this also triggers pairing mode, leaving them vulnerable to malicious pairing attacks. Similarly, the FIDO2 reference threat model assumes the presence of a legitimate, uncompromised client, overlooking the realistic threat of malware infecting devices.

Third, despite the availability of robust and cost-effective security mechanisms, manufacturers still largely rely on security through obscurity. Even with widespread awareness of the need for strong authentication, encryption, and integrity protection, many IoT protocols fail to implement these basic measures. Devices do not encrypt traffic data, lack mutual authentication, and leave firmware unsigned, despite modern System-on-Chip (SoC) supporting symmetric and asymmetric cryptography. Furthermore, manufacturers ignore the security protocols provided by technologies such as BLE and NFC, even with respect to critical operations involving confidential or sensitive data.

Finally, privacy remains an afterthought in IoT design. Even with some progress in security, manufacturers show little interest in limiting the data they collect from users. Closed-source ecosystems further complicate regulatory oversight, making it difficult for authorities to audit compliance with GDPR and other privacy regulations. As a result, users have little control over their privacy, often forced to choose between using a device or not, with no alternatives to reduce data exposure. Strengthening user awareness of privacy issues and empowering users with tools to protect their data are critical next steps for improving privacy in IoT environments.

In conclusion, this thesis highlights the significant gaps in IoT security and privacy, underscoring the need for more effective testing, proactive vulnerability mitigation, robust defense mechanisms, and greater transparency from manufacturers. Our research contributions offer valuable insights into the weaknesses of current practices and pave the way for future improvements. However, the road ahead requires collaboration between researchers, industry, and regulatory bodies to address these persistent challenges and ultimately create a more secure and privacy-respecting IoT ecosystem. By building on the foundations laid in this work, we hope to inspire further advancements in securing the rapidly evolving IoT landscape.

# Résumé en Français

Les dispositifs de l'Internet des objets (IoT) sont des systèmes embarqués conçus pour se connecter et échanger des données avec d'autres systèmes via Internet. Ces dispositifs couvrent une large gamme, allant des petits gadgets du quotidien comme les bracelets connectés, aux systèmes sophistiqués comme les trottinettes électriques. Ils sont généralement construits pour effectuer des tâches spécifiques et limitées, tout en donnant la priorité à l'efficacité énergétique et à la rentabilité. Par exemple, ils collectent des données des utilisateurs et de leur environnement grâce à des capteurs. Cependant, leurs contraintes de conception signifient que les dispositifs IoT fonctionnent souvent avec une puissance de calcul, une mémoire et un stockage minimaux afin de réduire les coûts et prolonger la durée de vie des batteries. Pour assurer la connectivité tout en économisant de l'énergie, ils s'appuient sur des protocoles sans fil basse consommation tels que le Bluetooth Low Energy (BLE) et la communication en champ proche (NFC). Leur format compact limite leur capacité à intégrer du matériel haut de gamme comme des écrans, le GPS ou des modules cryptographiques avancés.

Compte tenu de ces contraintes, les dispositifs IoT s'appuient généralement sur un *écosystème* de soutien pour étendre leurs fonctionnalités. Un backend basé sur le cloud, par exemple, peut prendre en charge des tâches gourmandes en ressources telles que le stockage de données, l'analyse complexe et l'apprentissage automatique sur les données collectées par les capteurs de l'appareil. Parallèlement, une application mobile compagnon sert souvent de proxy, permettant l'interaction avec l'utilisateur et fournissant à l'appareil une connectivité Internet. Cet écosystème permet à l'appareil d'offrir des fonctionnalités avancées grâce à une infrastructure externe.

L'un des plus grands défis en matière de sécurité des IoT est de trouver un équilibre entre efficacité, rentabilité et sécurité, cette dernière étant souvent reléguée au second plan. Les dispositifs IoT doivent s'appuyer sur une cryptographie légère en raison de la puissance de traitement et de la mémoire significatives requises par les algorithmes de chiffrement forts, ainsi que du coût et du manque d'espace pour des modules cryptographiques. La plupart des dispositifs ne prennent pas en charge l'authentification basée sur les certificats ou la cryptographie à clé publique. Cela pose particulièrement problème lorsqu'il s'agit de protéger des opérations critiques en matière de sécurité telles que l'appariement, l'établissement de session et les mises à jour de firmware.

Le déploiement massif de dispositifs IoT dans des fonctions critiques — allant de la santé personnelle à la sécurité en ligne et à la mobilité — signifie que des attaques réussies peuvent avoir des conséquences considérables, affectant non seulement les utilisateurs individuels mais potentiellement *des millions* de personnes ainsi que l'ensemble de l'écosystème du fournisseur. Voici quelques exemples mettant en évidence les effets significatifs des violations de la sécurité sur la confidentialité, la disponibilité et la sécurité.

Les trackers d'activité physique stockent et gèrent des données personnelles très sensibles, telles que la fréquence cardiaque, l'activité physique, et même les notifications de SMS ou d'appels entrants. Un attaquant pourrait divulguer ou détourner ces données de santé sensibles, compromettant ainsi la vie privée de l'utilisateur et l'exposant potentiellement au vol d'identité ou à une surveillance non désirée. Les authentificateurs FIDO2 gèrent des clés cryptographiques et des identifiants qui sécurisent l'accès aux comptes en ligne. Si un attaquant parvient à compromettre ces dispositifs, cela pourrait entraîner la perte d'accès à des services en ligne essentiels tels que les emails, les comptes bancaires ou les comptes professionnels. Cela perturbe non seulement la vie quotidienne de l'utilisateur, mais ouvre également la voie à d'autres risques de sécurité, notamment des prises de contrôle de comptes ou des fraudes financières. Les trottinettes électriques surveillent en continu l'état de leur batterie lithium-ion pour assurer la sécurité du conducteur. Si elles sont compromises, les mécanismes de sécurité d'une trottinette pourraient être désactivés, ou des données erronées sur l'état de la batterie pourraient être transmises à l'utilisateur, entraînant des dysfonctionnements voire des incendies de batterie. Cela met non seulement en danger le conducteur, mais aussi les personnes à proximité. De plus, les dispositifs IoT sont conçus pour une utilisation en déplacement, et ne pas pouvoir y accéder au moment voulu peut entraîner des désagréments importants. Par exemple, une trottinette ou un authentificateur verrouillé pourrait empêcher les utilisateurs d'accéder à des services ou des moyens de transport essentiels à des moments inopportuns.

# Énoncé du Problème

Les dispositifs IoT s'appuient sur des transports de communication standards, tels que le BLE, le NFC et l'USB. Cependant, ces standards ne répondent souvent pas aux exigences spécifiques des fabricants, les incitant à développer des protocoles *ad-hoc* sur ces transports. Par exemple, les services GATT standards de BLE manquent de mécanismes d'authentification adéquats pour répondre aux besoins d'écosystèmes comme celui de Xiaomi, qui utilise une authentification côté serveur. De même, dans la norme FIDO2, la communication NFC entre un authentificateur et un client utilise le protocole Client-To-Authenticator (CTAP) sur ISO14443 (une norme de communication sans contact) et ISO7816-4 (une spécification pour les cartes à puce).

Les fabricants manquent des ressources nécessaires pour concevoir des systèmes IoT sécurisés tout en réimplémentant les mécanismes de sécurité standards à travers plusieurs protocoles *couches*. En comparaison, le groupe de travail Constrained Restful Environments (CoRE [1]) a mis quatre ans à publier leur protocole de couche applicative, c'est-à-dire le Constrained Application Protocol (CoAP [2]), et quatre années supplémentaires pour étendre [3] leur pile réseau IoT à TCP et TLS. De plus, les fabricants contribuent à la prolifération des protocoles IoT non sécurisés en refusant de développer un cadre IoT unique et robuste, préférant *fragmenter* leur écosystème en plus petits sous-ensembles, chacun avec son propre protocole mal conçu. En conséquence de la complexité et de la fragmentation des protocoles en couches, les fabricants introduisent fréquemment des vulnérabilités critiques dans des opérations essentielles à la sécurité telles que l'appariement, l'établissement de session et les mises à jour de firmware, mettant ainsi en danger l'ensemble de l'écosystème. Ces problèmes peuvent être présents dans différentes couches et composants de l'écosystème, tels que la conception du protocole, le code de l'appareil, le matériel, ou encore l'interaction utilisateur.

Nous nous concentrons sur les problèmes *au niveau du protocole*, que l'on retrouve dans

la conception et la logique des protocoles de communication des IoT. Une vulnérabilité au niveau du protocole découle de failles dans la manière dont le protocole est défini, telles que l'absence d'authentification ou des schémas de chiffrement faibles. Ces vulnérabilités, ainsi que les attaques les exploitant, ont une portée large qui affecte tous les dispositifs exécutant le protocole, indépendamment de leur logiciel et matériel. Cela signifie que même les versions futures des dispositifs concernés restent vulnérables. Par exemple, nos attaques sur le Mi Band 5 étaient tout aussi efficaces sur le Mi Band 6, même si elles ont été développées avant que nous ne soyons conscients de l'existence du Mi Band 6.

Corriger les problèmes au niveau du protocole est particulièrement complexe, surtout lorsqu'il faut tenir compte de la *compatibilité descendante*. Résoudre ces vulnérabilités nécessite souvent de redéfinir ou de mettre à jour la spécification du protocole, ce qui, dans le cas de standards largement adoptés comme BLE et FIDO2, demande une collaboration entre différentes industries et organismes de normalisation. Par exemple, nous avons préconisé des améliorations aux protocoles d'appariement et d'établissement de session utilisés par les bracelets connectés Xiaomi, et avons participé à des discussions avec la FIDO Alliance pour mettre à jour leur modèle de menace de référence.

# Questions de Recherche

Cette thèse explore quatre questions de recherche fondamentales qui posent des défis majeurs et non résolus dans le paysage en évolution de la sécurité des objets connectés (IoT). Lors du choix de nos questions de recherche, nous avons cherché à combler des lacunes critiques dans la compréhension et la sécurité des écosystèmes IoT, en nous concentrant sur des domaines à la fois peu développés et ayant un impact significatif.

**RQ1 - Comment améliorer les tests de sécurité des protocoles IoT ?**

Nous estimons qu'il est essentiel de tenir les fabricants responsables pour une sécurité insuffisante, en particulier lorsqu'ils présentent de manière trompeuse les garanties offertes par leurs produits. Pour ce faire, il est indispensable de disposer d'une base solide en techniques d'analyse des protocoles et en outils disponibles. Dans l'évaluation d'un écosystème IoT, nous identifions deux activités principales : analyser la surface d'attaque et comprendre la logique sous-jacente de l'appareil, y compris par rétro-ingénierie si nécessaire.

La surface d'attaque des dispositifs IoT varie considérablement en fonction de facteurs tels que les canaux de communication (par exemple, radio BLE et connecteurs USB), les composants matériels (par exemple, capteurs et écrans), et l'architecture système (par exemple, la présence de serveurs backend et de sous-systèmes internes). Par exemple, les trackers d'activité physique et les trottinettes électriques sont vulnérables à des attaques à longue portée via BLE, tandis que les authentificateurs FIDO2 sont exposés à des menaces de proximité via NFC ou à des menaces locales via des connexions USB. Certains vecteurs d'attaque exploitent des faiblesses spécifiques, telles que les attaques d'usurpation d'identité de clients FIDO2, qui tirent parti de l'absence de retour visuel sur les authentificateurs. De même, le protocole d'appariement côté serveur de Xiaomi peut être manipulé pour dissocier à distance des appareils des comptes de leurs propriétaires. Le choix du mécanisme de transport influence également la manière dont le trafic est capturé, déchiffré et analysé. Par exemple, le journal HCI snoop d'Android permet de surveiller le trafic BLE sans chiffrement de la couche liaison, tandis que le déchiffrement d'une

session TLS nécessite l'obtention de la clé pré-master. Les tests de sécurité à grande échelle doivent prendre en compte tous ces scénarios et variables.

De nombreux fabricants s'appuient sur la sécurité par l'obscurité, en gardant leurs protocoles fermés et non documentés. Cette pratique rend l'analyse des protocoles fortement dépendante de l'efficacité de la rétro-ingénierie (RE). La RE est nécessaire pour déchiffrer les messages de la couche application (par exemple, ceux échangés après l'accord de clés) et comprendre le format binaire et la sémantique des communications du protocole. De plus, la RE est un effort continu, car les protocoles propriétaires peuvent évoluer au fil du temps avec la sortie de nouveaux correctifs ou de nouvelles générations de dispositifs. Par exemple, le Mi Band 4 utilise un protocole d'appariement différent du Mi Band 3 et a subi une autre mise à jour majeure en 2021. Malgré l'existence d'outils tiers développés par des experts en sécurité, ceux-ci sont trop spécifiques à certains dispositifs, offrent des capacités de test limitées, et deviennent obsolètes dès que les protocoles évoluent. Il est nécessaire de disposer de meilleurs outils et d'un environnement de test stable qui ne nécessite pas l'achat ou l'exécution du matériel.

**RQ2 - Comment trouver des vulnérabilités et des attaques au niveau des protocoles IoT ?**

Les écosystèmes IoT présentent une gamme de défis de sécurité qui, malgré leur apparente simplicité, se sont avérés difficiles à résoudre et persistent depuis plus d'une décennie de recherches. Ces défis proviennent de problèmes inhérents tels que les ressources limitées des appareils, les protocoles fragmentés, et la délégation injuste des responsabilités de sécurité aux utilisateurs finaux. Faire progresser la sécurité des IoT nécessite une compréhension approfondie de ces problèmes persistants et l'élaboration de solutions à leurs causes profondes. Ci-dessous, nous soulignons une sélection des risques de sécurité critiques que nous avons évalués.

Une authentification faible ou non-mutuelle ouvre la porte aux attaques d'usurpation d'identité et de spoofing. Un chiffrement inadéquat et une protection insuffisante de l'intégrité exposent les communications à l'écoute clandestine, à des messages falsifiés, et à des attaques par rejeu. Des mises à jour de firmware non sécurisées exposent les dispositifs à des compromissions par des logiciels malveillants ou des modifications non autorisées. Un protocole défectueux ou une négociation de version de firmware vulnérable les rendent sensibles aux attaques de rétrogradation, réduisant leurs garanties de sécurité à celles d'une version plus faible du protocole ou du firmware. Une prise de conscience insuffisante de l'utilisateur et des confirmations trompeuses dues à une mauvaise interface utilisateur et à des avertissements non transparents lors de décisions critiques en matière de sécurité. La dépendance excessive des fabricants à la sécurité par l'obscurité plutôt qu'à des mécanismes de sécurité solides conduit à des protocoles sans garanties de sécurité dès qu'ils sont rétro-ingéniés.

**RQ3 - Comment concevoir des protocoles IoT sécurisés ?**

De nombreux travaux académiques ont tenté de résoudre les risques de sécurité précédemment discutés. Une stratégie courante consiste à adapter des versions allégées de mécanismes de sécurité établis. Par exemple, l'introduction de CoAP, un protocole web similaire à HTTP mais optimisé pour les environnements contraints, ainsi que le développement de DTLS pour les environnements contraints (DICE [4]), qui implémente des primitives cryptographiques comme Elliptic Curve Diffie-Hellman Ephemeral (ECDHE), également utilisé dans TLS. Une deuxième approche consiste en la création de solutions ad hoc adaptées à des écosystèmes IoT spécifiques. Un exemple est le développement d'extensions FIDO2 pour améliorer la sécurité, ou la refonte

de protocoles afin d'utiliser plus efficacement les primitives cryptographiques existantes. Par exemple, nous avons exploité l'implémentation par Xiaomi de l'algorithme de chiffrement Tiny Encryption Algorithm (TEA) pour chiffrer le firmware.

La principale limitation de ces contre-mesures réside dans leur manque de compatibilité ascendante et d'interopérabilité. Les chercheurs conçoivent souvent de nouveaux protocoles entièrement incompatibles avec les existants ou proposent l'ajout de nouveaux composants matériels aux appareils. Par exemple, plusieurs travaux liés à FIDO2 suggèrent de résoudre l'usurpation d'identité des clients en intégrant un affichage sécurisé. Cependant, de nombreuses solutions ne sont pas (facilement) portables entre les appareils en raison des différences de logiciels, d'architectures ou de transport. Par exemple, bien que nous ayons pu corriger une vulnérabilité de rétrogradation de session dans le firmware des trottinettes électriques, l'extension de cette correction à d'autres versions du firmware a nécessité des efforts de codage supplémentaires et de rétro-ingénierie. Il est irréaliste d'attendre des utilisateurs qu'ils remplacent leurs appareils et passent à de nouveaux modèles simplement pour corriger des failles de sécurité qu'ils n'ont pas causées. Les chercheurs devraient aider à développer des contre-mesures applicables à un large éventail d'appareils, idéalement sans nécessiter l'intervention des fabricants. Améliorer les mécanismes d'interaction utilisateur et publier des outils pour corriger les appareils permettraient aux utilisateurs de prendre le contrôle de leur propre sécurité, au lieu de dépendre uniquement des mises à jour officielles. Nous ne traitons pas de la fragmentation actuelle des protocoles IoT, mais nous nous concentrons sur la résolution de concepts de sécurité de haut niveau adaptables à divers protocoles, laissant les détails d'implémentation aux développeurs.

**RQ4 - Comment concevoir des protocoles IoT respectueux de la vie privée ?**

Garantir la confidentialité dans l'IoT est un défi critique, car la prolifération des dispositifs interconnectés génère d'énormes quantités de données personnelles, souvent collectées et transmises sans consentement explicite des utilisateurs. L'importance de traiter les préoccupations relatives à la confidentialité réside dans les conséquences potentielles des violations de données, qui peuvent entraîner le suivi des utilisateurs, le profilage, une infraction au RGPD, et la manipulation du comportement des utilisateurs. Actuellement, la confidentialité dans l'IoT n'est pas régulée, les fabricants se contentant de publier un document arbitraire (par exemple, la notice de confidentialité des trackers d'activité Xiaomi [5]) qui ne les rend pas responsables en cas d'atteinte à la vie privée (c'est-à-dire que Xiaomi n'a jamais reconnu nos nombreuses attaques divulguant des données de santé et privées des utilisateurs). La Fondation Mozilla s'est engagée dans l'initiative "Privacy Not Included" [6], publiant sa propre évaluation de la confidentialité des produits IoT.

Nous considérons qu'il est inacceptable qu'à l'heure actuelle, les appareils qui collectent le plus de données personnelles soient ceux qui les protègent le moins et les exposent à des acteurs malveillants, à des fabricants peu scrupuleux et à des publicitaires. Plusieurs tentatives de systématisation de la sécurité IoT ont été faites, telles que le cadre Fitbit Golden Gate [7] qui promeut des communications fiables et sécurisées, mais sans volonté explicite de garantir la confidentialité. Nous considérons les attaques contre la vie privée aussi importantes que celles contre la sécurité, et nous les corrigeons, par exemple en proposant une rotation dynamique des annonces BLE sur les trackers d'activité et des identifiants d'authentification FIDO2. Nous soulignons l'absence de mécanismes permettant à l'utilisateur de gérer quelles données sont collectées par ses appareils et quelles données sont envoyées au backend pour traitement. Ce

serait une étape cruciale pour permettre aux utilisateurs de prendre le contrôle de leur propre vie privée.

# Contributions

Cette thèse apporte quatre contributions clés à l'étude des techniques d'analyse, des vulnérabilités, des attaques et des contre-mesures dans les systèmes IoT. Nous mettons particulièrement l'accent sur les vulnérabilités au niveau du protocole en raison de leur impact significatif, tant académique que dans le monde réel, ce qui a conduit à des améliorations fondamentales en matière de sécurité et de confidentialité. Nos recherches exposent des failles critiques dans la logique et la structure sous-jacentes des protocoles utilisés par des dispositifs tels que les trackers d'activité, les trottinettes électriques et les authentificateurs FIDO2, et plaident pour une transition vers des principes de sécurité dès la conception de la part des fabricants. Nous résumons maintenant les principales contributions de cette thèse et discutons de la manière dont chaque travail répond aux questions de recherche dans son contexte respectif.

## BreakMi

La contribution "BreakMi: Reversing, Exploiting and Fixing Xiaomi Fitness Tracking Ecosystem" [8] a été publiée dans les IACR Transactions on Cryptographic Hardware and Embedded Systems (CHES 2022).

Dans BreakMi, nous analysons les trackers d'activité, démontrant comment même les appareils les plus petits et les moins chers peuvent avoir un impact significatif sur les utilisateurs et l'écosystème du fabricant. Nous répondons à *RQ1* en publiant un tracker d'activité virtuel et une application compagnon qui communiquent en utilisant le protocole Xiaomi. Nous publions les services GATT de Xiaomi, leurs caractéristiques et fonctions, ainsi que des dissectors pour les paquets Xiaomi et leur format binaire. Nous répondons à *RQ2* en identifiant six attaques à distance et de proximité issues de dix vulnérabilités. Nos attaques contournent les mécanismes de sécurité et permettent l'injection de fausses données dans le tracker ainsi que dans le système backend. Nous répondons à *RQ3* en améliorant les protocoles d'appairage et d'authentification de Xiaomi, renforçant ainsi les garanties de sécurité sans compromettre la compatibilité ascendante ou les performances. Enfin, nous répondons à *RQ4* en démontrant comment les trackers d'activité divulguent des données confidentielles, telles que les données de santé et les horaires de sommeil, et peuvent être utilisés pour suivre les utilisateurs.

## E-Spoofer

La contribution "E-Spoofer: Attacking and Defending Xiaomi Electric Scooter Ecosystem" [9] a été publiée dans les actes de la 16e Conférence ACM sur la sécurité et la vie privée dans les réseaux mobiles et sans fil (WiSec 2023).

Dans E-Spoofer, nous effectuons la rétro-ingénierie et exploitons les protocoles d'appairage et d'établissement de session des trottinettes personnelles Xiaomi. Nous répondons à *RQ1* en publiant une trottinette virtuelle et une application compagnon qui répliquent exactement

les communications BLE d'origine, permettant ainsi des tests sans avoir besoin d'acheter du matériel. Nous systématisons les quatre protocoles de Xiaomi et fournissons des outils pour vérifier les primitives cryptographiques en usage (par exemple, aucune, AES ou ECDH). Nous répondons à *RQ2* en présentant deux attaques à distance et de proximité exploitant six vulnérabilités. Une attaque d'appairage malveillant permet à un attaquant d'usurper l'application et de déverrouiller une trottinette à distance, facilitant ainsi le vol. Une attaque de rétrogradation de session exploite un oubli des développeurs concernant la rétrogradation de version du protocole. Nous répondons à *RQ3* en repensant les protocoles de Xiaomi avec un geste d'appairage intentionnel et une authentification mutuelle. Nous publions également un outil pour détecter les firmwares vulnérables à nos attaques, ainsi qu'un firmware corrigé pour un modèle qui n'est plus mis à jour par Xiaomi. Enfin, nous répondons à *RQ4* en obtenant un accès non autorisé à la trottinette et à toutes les informations privées qu'elle contient, y compris le kilométrage et le temps passé sur la course en cours.

## E-Trojans

La contribution "E-Trojans: Ransomware, Tracking, DoS, and Data Leaks on Battery-powered Embedded Systems" est actuellement en cours de soumission au Symposium international IEEE sur la sécurité matérielle et la confiance (HOST 2025).

Dans E-Trojans, nous démontrons comment un attaquant interne peut compromettre la sécurité du conducteur d'une trottinette électrique, entraînant des dangers tels que des courts-circuits et des risques d'incendie. Nous répondons à *RQ1* en publiant un outil d'analyse et de correction de trottinettes électriques capable d'introduire des capacités malveillantes dans le firmware du système de gestion de batterie légitime. Nous répondons à *RQ2* en présentant quatre attaques exécutables depuis le firmware du système de gestion de batterie des trottinettes. Nos découvertes incluent le premier cas de ransomware par sous-tension de batterie ainsi qu'une série d'attaques par déni de service (DoS). Nous identifions quatre vulnérabilités facilitant ces attaques, y compris l'absence de vérification de la signature du firmware. Nous répondons à *RQ3* en sécurisant le firmware grâce au chiffrement et à la vérification de la signature, en protégeant le bus UART interne avec un protocole sécurisé tel que SCP03, et en mettant en œuvre des mesures de limitation de débit. Enfin, nous répondons à *RQ4* en proposant une attaque de suivi des utilisateurs qui utilise l'empreinte des détails de la batterie et divulgue des données personnelles via la publicité BLE.

## CTRAPS

La contribution "CTRAPS: CTAP Impersonation and API Confusion Attacks and Defenses on FIDO2" sera soumise au 10e Symposium européen IEEE sur la sécurité et la vie privée (EURO S&P 2025).

Dans CTRAPS, nous proposons deux stratégies d'attaque pour compromettre la sécurité et la confidentialité de FIDO2, une norme d'authentification open-source et robuste. Nous répondons à *RQ1* en développant un banc d'essai FIDO2 virtuel, comprenant un client virtuel pour l'invocation arbitraire de l'API Authenticator et une partie de confiance virtuelle capable d'enregistrer tout type d'identifiants sur un authentificateur. Nous améliorons également les

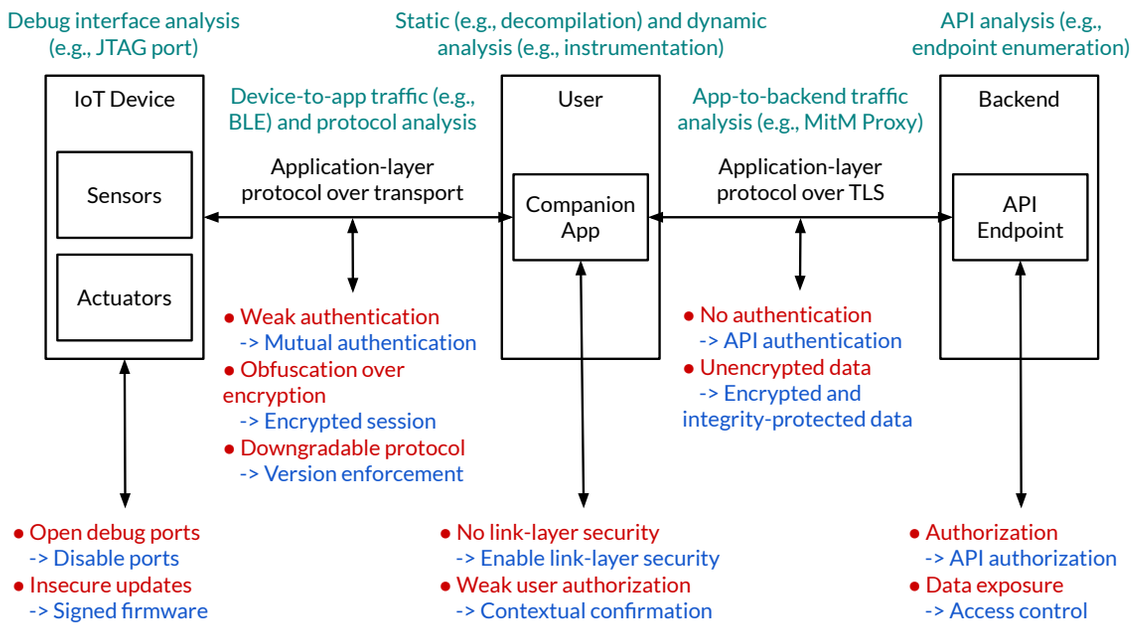



Figure 7.1: High-level system model for an IoT ecosystem. The IoT device (left) collects data via sensors and actuators. Users interact with the device through a companion app (middle), connected to the vendor's backend (right). We indicate available analysis techniques in teal. We report common vulnerabilities in red and a possible defense against them in blue.

dissectors FIDO2 existants pour capturer plus de paquets et mieux visualiser les informations. Nous répondons à *RQ2* en présentant onze attaques de proximité et à distance et en discutant de leurs sept causes racines, ainsi que des limites du modèle de menace de référence FIDO2, irréaliste. Nos stratégies d'attaque incluent l'usurpation d'identité du client (par exemple, lecteur NFC) capable d'attaques zero-click, ainsi qu'une nouvelle technique de confusion API, où les autorisations accordées par l'utilisateur pour certaines requêtes API sont détournées pour d'autres requêtes potentiellement destructrices. Nos attaques démontrent que l'introduction de nouvelles fonctionnalités (par exemple, les identifiants découvrables conçus pour protéger contre les violations de données tierces) peut involontairement créer de nouveaux vecteurs d'attaque (par exemple, la suppression d'identifiants découvrables via l'API de gestion des identifiants). Nous répondons à *RQ3* en proposant sept contre-mesures, y compris l'authentification du client CTAP et l'amélioration des mécanismes d'autorisation. Nous analysons comment nos solutions impactent l'utilisabilité tout en maintenant la compatibilité ascendante et des coûts faibles, contrairement aux solutions matérielles (par exemple, ajouter un écran). Enfin, nous répondons à *RQ4* en démontrant une extraction non autorisée, sans interaction de l'utilisateur, de toutes les parties de confiance et des identifiants d'authentification stockés sur l'authentificateur.

# Contexte

Nous fournissons des informations contextuelles pertinentes sur l'écosystème IoT afin d'établir une compréhension fondamentale des contenus de cette thèse.

La Figure 7.1 présente un aperçu général d'un écosystème IoT typique. Dans ce modèle, un appareil IoT collecte des données de son environnement via des capteurs (par exemple,

accéléromètre et moniteur de fréquence cardiaque) et interagit avec l'utilisateur par le biais d'actionneurs (par exemple, boutons tactiles). Le propriétaire de l'appareil interagit principalement avec le système à travers une application compagnon installée sur son smartphone. La communication entre l'appareil IoT et le smartphone s'effectue par des transports sans fil (par exemple, BLE et NFC) ou physiques (par exemple, USB), utilisant soit des protocoles propriétaires (par exemple, Xiaomi), soit des normes ouvertes (par exemple, FIDO2). L'application compagnon se connecte au backend du fabricant via Wi-Fi, généralement sécurisé par un canal TLS. Le backend est responsable de la gestion des opérations gourmandes en ressources qui dépassent les capacités de calcul de l'appareil ou du smartphone, tout en gérant également le stockage et le traitement des données utilisateur. Ce modèle de système fournit un cadre clair pour identifier les points clés où l'analyse des protocoles, l'évaluation des vulnérabilités et les améliorations de sécurité peuvent être appliquées.

Dans les sections suivantes, nous comparons les travaux existants liés au contenu de notre thèse. Nous examinons comment d'autres chercheurs ont tenté de répondre à nos questions de recherche, discutons de leurs résultats et les comparons à notre contribution. Pour plus d'informations concernant les travaux connexes spécifiques à chaque contribution présentée dans cette thèse, veuillez vous référer à la Section 3.10 pour BreakMi, à la Section 4.9 pour E-Spoofer, à la Section 5.8 pour E-Trojans, et à la Section 6.10 pour CTRAPS.

## RQ1

*Comment pouvons-nous améliorer les tests de sécurité des protocoles IoT ?* Les chercheurs ont analysé et testé les protocoles en utilisant diverses approches que nous catégorisons en fonction de la surface ciblée, comme l'illustre la Figure 7.1. Nous passons en revue les points d'entrée pour effectuer une analyse sur les écosystèmes IoT.

**Analyse du trafic entre l'application et le backend et analyse des API.** Le trafic entre l'application et le backend contient souvent des opérations critiques pour la sécurité, telles que le couplage, et des données sensibles comme les mesures de santé et les routines quotidiennes de l'utilisateur. Dans [10], les auteurs ont vérifié l'intégrité et le chiffrement des communications des traqueurs d'activité avec le backend du fabricant en utilisant une attaque de type homme du milieu (MitM) via un proxy HTTPS. De même, dans [11], les chercheurs ont reverse-engineered la mise à jour du firmware et le trafic backend des appareils Xiaomi pour la maison intelligente, y compris les robots aspirateurs, leur permettant de déconnecter les appareils du cloud de Xiaomi et de les ajouter à un cloud isolé. BurnFit [12] a testé la résistance de trois backends de suivi d'activité aux attaques par spoofing DNS, découvrant des vulnérabilités dans les trois. L'analyse des API a également été un axe d'analyse du trafic. Par exemple, un hacker a exploité des points d'API non sécurisés du backend du service de location de trottinettes Bird pour réserver à distance des trottinettes ou déclencher des alarmes [13]. Dans [14], les chercheurs ont exploité une vulnérabilité dans l'API de Fitbit pour effectuer un couplage non autorisé côté serveur, déconnectant ainsi l'utilisateur d'origine de son Fitbit Charge 2.

**Analyse du trafic entre le dispositif et l'application.** L'analyse du trafic entre le dispositif et l'application varie en difficulté en fonction du protocole de transport. Le Bluetooth Low Energy (BLE) est un transport courant pour les appareils IoT, et la capture du trafic BLE est relativement simple si l'utilisateur a le contrôle sur le smartphone de communica-

tion. Sur Android, les utilisateurs peuvent activer le journal de surveillance HCI pour collecter le trafic BLE. En utilisant cette technique, un chercheur a appris à simuler des notifications sur le Mi Band 3 et a reverse-engineered son serveur GATT, y compris des services de transfert BLE standard (par exemple, fréquence cardiaque, pas) et personnalisés de Xiaomi [15]. Le protocole de couplage Magic Pairing d'Apple pour un couplage BLE transparent a été reverse-engineered dans [16], révélant des détails sur la publicité BLE et la communication avec iCloud. Periscope [17] a examiné des applications Android qui configurent des smartphones en périphériques BLE, utilisant des techniques d'analyse statique comme la décompilation de code pour étudier le trafic BLE. Lorsque l'accès physique à l'appareil n'est pas disponible, le trafic BLE peut être intercepté sans fil à l'aide d'outils de capture comme Ubertooth, comme cela a été fait dans [18] pour découvrir des attaques par corrélation.

**Analyse automatisée des protocoles.** Le reverse-engineering de protocoles binaires fermés et non documentés est une tâche difficile. Les méthodes automatisées comme le fuzzing et la criminalistique s'adaptent bien à l'augmentation des données d'entrée, permettant aux chercheurs de détecter rapidement des vulnérabilités et d'explorer divers chemins d'exécution. Dans [19], les auteurs ont appliqué la criminalistique numérique aux haut-parleurs intelligents Xiaomi, utilisant le traitement du langage naturel et l'extraction de texte pour associer des modèles de discours aux intentions des utilisateurs. En analysant les protocoles propriétaires de Xiaomi, ils ont identifié des événements spécifiques comme la lecture de musique. Les auteurs de [20] ont développé AFLIoT, un fuzzer greybox pour les binaires IoT basés sur Linux, leur permettant de fuzz des dispositifs du monde réel tels que le routeur Xiaomi R1D et de découvrir des plantages uniques. Le cadre Avatar [21] était capable d'une analyse statique en temps réel et d'une émulation d'appareil, indépendamment de la présence de périphériques. Frankenstein [22] a émulé des dispositifs BLE dans un environnement virtuel, exécutant de larges portions de code et permettant l'injection de trames sans fil. Incision [23] a élargi ces cadres en utilisant une boucle de rétroaction améliorant progressivement le reverse-engineering du protocole, de manière automatisée.

**Analyse manuelle des protocoles.** Les méthodes manuelles, telles que la décompilation de code et l'instrumentation dynamique, impliquent plus d'efforts humains mais révèlent souvent des vulnérabilités plus complexes. Dans [24], les chercheurs ont effectué une analyse statique de code sur des applications de suivi d'activité en décompilant le bytecode de l'application en Java ou en Smali si nécessaire. De même, les auteurs de [25] ont analysé les préoccupations en matière de confidentialité dans les applications de trottinette électrique en utilisant des outils d'analyse statique comme MobSF pour examiner les autorisations de l'application et les bibliothèques tierces. Le débogage du firmware a été utilisé par [26] pour trouver une porte dérobée dans le firmware de Fitbit Charge HR, et par [27], où des ports JTAG dans un Fitbit Charge 2 ont été accessibles pour extraire son firmware et ses clés de chiffrement. Les méthodes de vérification formelle, bien que partiellement automatisées, nécessitent une modélisation du protocole pour prouver des garanties de sécurité. Par exemple, dans [28], les auteurs ont démontré que l'authentification FIDO U2F est compromise lorsque l'identifiant de l'application serveur n'est pas validé. De plus, la vérification formelle a été utilisée dans d'autres protocoles IoT comme EDHOC et MQTT pour découvrir des problèmes de sécurité [29, 30].

**Nos améliorations.** Dans notre recherche, nous avons appliqué une variété de ces techniques pour reverse-engineerer et analyser les protocoles dans des traqueurs d'activité, des trot-

tinettes électriques et des authentificateurs FIDO2. Nous avons développé des outils d'analyse et de test de sécurité personnalisés avec des capacités uniques non disponibles auparavant, et les avons rendus publics. Nos améliorations incluent : (i) *environnements virtuels* permettant des tests de sécurité sans nécessiter de dispositifs IoT physiques ou leurs applications compagnon ; (ii) *kits d'outils d'attaque* implémentés sous forme de scripts en ligne de commande et d'applications Android, facilitant la reproductibilité de nos attaques ; (iii) *dissécteurs Wireshark* pour améliorer la visualisation et la compréhension des paquets de protocoles personnalisés ; et (iv) *crochets Frida* qui extraient des informations en temps réel sur l'exécution du protocole en temps réel.

## RQ2

*Comment pouvons-nous trouver des vulnérabilités et des attaques au niveau des protocoles IoT ?* La communauté académique a largement étudié et rapporté des vulnérabilités et des attaques dans la sécurité IoT. Cependant, la diversité et l'évolution continue des écosystèmes IoT entraînent constamment de nouvelles découvertes. Ci-dessous, nous présentons une sélection des attaques les plus pertinentes pour cette thèse.

**Attaques sur les écosystèmes IoT.** Les chercheurs en sécurité ont exploité des vulnérabilités à travers diverses couches des écosystèmes IoT, y compris les dispositifs, les applications compagnon et les backends. De nombreux traqueurs de fitness, une cible populaire, ont été victimes d'écoutes, d'usurpation d'identité, d'attaques de l'homme du milieu et de fuites de données. Les causes profondes de ces attaques proviennent généralement d'un manque d'authentification et de chiffrement dans les communications entre le traqueur de fitness, son application compagnon et le backend. Ces problèmes ont été identifiés soit au niveau du protocole, soit causés par des négligences dans l'implémentation de la part des fabricants. Par exemple, les protocoles d'authentification du Fitbit Charge HR ont été rétro-ingénierés dans [26], tandis que des chercheurs dans [14] ont pu fuites des données privées du Fitbit Charge 2 et effectuer une mise à jour de firmware malveillante sur l'appareil. Les trottinettes électriques, bien que moins fréquemment étudiées, ont également été ciblées. La Xiaomi M365, par exemple, a montré des failles d'authentification mutuelle entre l'appareil et l'application compagnon [31]. D'autres exploits comprenaient des mécanismes de verrouillage qui freinent brusquement la trottinette [32] et une attaque de l'homme du milieu (MitM) pour insérer des fichiers audio personnalisés dans les trottinettes Lime [33]. Des API vulnérables dans l'application Bird ont permis aux attaquants de contourner la vérification du code QR [13]. L'écosystème IoT diversifié de Xiaomi a également attiré l'attention. Des chercheurs ont rétro-ingénéré des aspirateurs Xiaomi dans [34], contournant le démarrage sécurisé, déchiffrant les communications avec le backend, fuyant des données et injectant de faux enregistrements. D'autres cibles de Xiaomi comprennent des haut-parleurs intelligents [19], des caméras de sécurité [35, 36] et des routeurs [20]. Les attaques par canaux auxiliaires se sont également révélées efficaces sur les authentificateurs, comme le montrent [37, 38, 39].

**Attaques sur la sécurité BLE.** En tant que couche de transport la plus répandue dans les dispositifs IoT, le BLE a été soumis à de nombreuses attaques. La négociation de clés dans le BLE a été compromise par des attaques de rétrogradation qui forcent un chiffrement à faible entropie [40] ou exploitent des fonctionnalités du BLE [41]. Toutes les itérations des mécanismes de couplage BLE, y compris le couplage hérité [42], le couplage simple [43, 44],

et le mode uniquement sécurisé [45], ont été attaquées avec succès. La confusion de méthode [46] permet des attaques de type homme du milieu en associant des dispositifs utilisant deux modes différents, ce qui est difficile à détecter pour les utilisateurs. Les attaques sur la vie privée, telles que le suivi des utilisateurs via des adresses MAC, ont été démontrées par le biais de botnets [47] ou en exploitant la publicité BLE à travers des applications [48]. Une étude à grande échelle dans [49] a examiné des applications Android partageant des canaux BLE, révélant le potentiel d'abus par des applications malveillantes. BLECryptracer [50] a exploité cette faille pour effectuer des mises à jour de firmware indésirables sur des dispositifs BLE. Les protocoles de couche application construits sur le BLE ont également été examinés. Par exemple, des chercheurs dans [51] ont compromis le protocole ZeroConf dans l'écosystème d'Apple, leur permettant d'intercepter les canaux Bluetooth utilisés par les applications Tencent QQ et Scribe.

**Mauvaise utilisation de la cryptographie dans l'IoT.** Malgré son rôle critique dans la sécurisation de l'IoT, la cryptographie est souvent mal appliquée. L'extraction de clés cryptographiques a été un domaine d'attention majeur, en particulier dans les systèmes automobiles. Des vulnérabilités ont été trouvées dans les systèmes d'entrée sans clé, y compris les Tesla Model S [52] et X [53], et les systèmes d'entrée basés sur KeeLoq [54]. Les faiblesses des transpondeurs qui immobilisent les voitures, telles que Hitag2 [55] et Megamos Crypto [56], ont également été exploitées. Les dispositifs Xiaomi ont montré des faiblesses cryptographiques, allant d'une dépendance excessive à l'obfuscation [57, 58] à des fuites de clés de déchiffrement de firmware [59]. Les protocoles cryptographiques de FIDO2, y compris la vie privée, la révocation, l'attestation et la sécurité post-quantique, ont été formellement vérifiés dans [60]. Cependant, une attaque de substitution de clé publique a été identifiée dans [61] pour FIDO2, exploitant l'accès en écriture à la base de données du serveur. Dans [62], le Keymaster de Samsung TrustZone, responsable de la gestion des identifiants FIDO2, a été trouvé vulnérable à une attaque de réutilisation de IV AES-GCM en raison d'une rétrogradation qui permettait d'utiliser un IV fixe au lieu d'un aléatoire.

**Nos améliorations.** Dans notre recherche, nous avons trouvé de nouvelles attaques et vulnérabilités impactantes et peu coûteuses dans les traqueurs de fitness, les trottinettes électriques et les authentificateurs FIDO2. Nos améliorations comprennent : (i) *vingt-sept vulnérabilités IoT* telles que le couplage non authentifié, la session reproductible, le firmware non signé et les dispositifs traçables ; (ii) *vingt-trois attaques IoT* telles que le premier ransomware de batterie et une réinitialisation d'authentificateur à un clic à distance et de proximité ; et (iii) *la stratégie d'attaque de confusion API* qui dirige l'utilisateur vers l'appel d'une API CTAP non intentionnelle.

## RQ3

*Comment concevoir des protocoles IoT sécurisés ?* Le domaine de l'IoT a un besoin constant de contre-mesures robustes face aux menaces croissantes auxquelles il fait face. La direction de la recherche ne consiste pas à corriger les protocoles, mais à détecter des signes de manipulation ou la présence d'un acteur malveillant. C'est une approche sensée pour l'utilisateur afin d'atténuer les dommages, mais cela ne résout pas les problèmes fondamentaux au niveau des protocoles, laissant le système éternellement vulnérable. Nous discutons de deux moyens courants de défendre les dispositifs IoT.

**Détection d'intrusions et d'anomalies.** Les systèmes de détection d'intrusions (IDS) atténuent l'impact d'un attaquant déjà infiltré dans un système, acceptant qu'il soit parfois impossible de corriger la conception du protocole. Les auteurs de [63] ont proposé un IDS supervisé pour les maisons intelligentes, utilisant l'apprentissage automatique sur des captures de trafic pour classer les paquets légitimes et malveillants. Passban [64] a tenté de résoudre le problème selon lequel les dispositifs IoT en périphérie ne peuvent pas déployer efficacement des IDS basés sur des signatures en raison de leurs ressources limitées et des mises à jour peu fréquentes, en s'appuyant sur la détection d'anomalies. La détection d'anomalies a également été utilisée par [65], un IDS pour les environnements IoT qui détecte les attaques classiques sur les réseaux, telles que l'empoisonnement ARP, le spoofing DNS et les inondations via COAP/HTTP. Oasis [66] est un cadre BLE capable d'injecter des algorithmes de détection d'intrusion dans le firmware des contrôleurs BLE. Son module de détection repose sur l'instrumentation et utilise des heuristiques pour chaque attaque spécifique.

**Attestation à distance.** L'objectif de l'attestation à distance est de garantir que le logiciel exécuté sur le dispositif IoT est légitime et n'a pas été manipulé. Cette pratique n'est souvent pas prise en charge par les dispositifs ou d'autres composants de l'écosystème. Des chercheurs ont analysé l'attestation à distance dans les authentificateurs Yubico [67], constatant que le mécanisme proposé de génération de clés à distance asynchrone produit des signatures défiables et impossibles à falsifier. D'autres modes de mécanismes d'attestation ont été évalués, tels que [68], fournissant une révocation de clé globale et l'anonymat au schéma de chiffrement sous-jacent à l'emballage de clé, et [69], proposant différents modes d'attestation. Des alternatives à l'attestation à distance ont été discutées dans la littérature. Par exemple, dans [70], les chercheurs présentent des mécanismes de confiance efficaces qui empêchent les attaques Sybil dans les réseaux IoT ayant un accès constant au backend, comme les réseaux intelligents en agriculture. Des environnements d'exécution de confiance ont été utilisés pour garantir un accès autorisé aux ressources, comme dans [71], où des politiques de contrôle d'accès sont mises en place et les applications enfreignant les politiques au sein de TEE sont examinées. L'Internet Engineering Task Force (IETF) a proposé Software Updates for Internet of Things (SUIT) pour des mises à jour logicielles sécurisées, qui ont été testées avec succès sur des dispositifs à faibles ressources par [72]. Les auteurs de [73] ont conçu un moniteur d'intégrité des données en ligne pour les systèmes de capteurs gérant les défaillances de données et la récupération.

**Nos améliorations.** Dans notre recherche, nous avons proposé des défenses pratiques et rétrocompatibles pour corriger nos attaques et résoudre leurs causes profondes au niveau des protocoles. Nous avons divulgué de manière responsable nos découvertes aux vendeurs, qui les ont ignorées (c'est-à-dire, Xiaomi), corrigées (c'est-à-dire, Google) ou qui évaluent actuellement (c'est-à-dire, FIDO2 et Microsoft) nos retours. Nos améliorations comprennent : (i) *des améliorations protocolaires fortes* qui augmentent leur sécurité et leur confidentialité avec des contre-mesures robustes, telles que l'authentification mutuelle, la confirmation contextuelle de l'utilisateur, la vérification du firmware et les clients de confiance ; (ii) *des outils d'évaluation des vulnérabilités* qui vérifient si un dispositif est affecté par nos attaques, comme nos signatures Yara pour le firmware de la trottinette Xiaomi et notre application Android testant une vulnérabilité d'implémentation Yubico ; et (iii) *des correctifs pour les utilisateurs* qui protègent leurs dispositifs sans s'appuyer sur des mises à jour de sécurité du vendeur, comme notre correctif pour le firmware Xiaomi M365.

## RQ4

*Comment concevoir des protocoles IoT respectueux de la vie privée ?* Le domaine de l'IoT a un besoin constant de contre-mesures robustes face aux menaces croissantes auxquelles il fait face. La direction de la recherche ne consiste pas à corriger les protocoles, mais à détecter des signes de manipulation ou la présence d'un acteur malveillant. C'est une approche sensée pour l'utilisateur afin d'atténuer les dommages, mais cela ne résout pas les problèmes fondamentaux au niveau des protocoles, laissant le système éternellement vulnérable. La vie privée peut être compromise par le fabricant, qui n'est pas intrinsèquement malveillant, ou par des attaquants tiers ayant des intentions malveillantes.

**Violations de la vie privée par les fabricants.** Les dispositifs téléchargent fréquemment des données vers le backend du fabricant en raison de limitations de stockage et pour le traitement des données côté serveur. Cependant, il n'existe souvent pas de méthode transparente pour vérifier quelles informations sont transmises, ce qui soulève des préoccupations concernant un éventuel partage excessif des données privées des utilisateurs. Par exemple, les auteurs de [18] ont découvert que les traqueurs Fitbit Flex collectent plus de données que ce qui est officiellement divulgué. Ils ont trouvé des preuves d'enregistrements d'activité par minute envoyés au backend, informations qui ne sont jamais partagées avec l'utilisateur. De plus, les fabricants ne respectent pas toujours les normes de confidentialité qu'ils annoncent. Dans une étude évaluant onze vendeurs, les chercheurs de [74] ont identifié deux vendeurs qui ne respectaient pas leurs propres politiques de confidentialité. Une solution potentielle pour atténuer l'interférence des vendeurs est de découpler les dispositifs de leurs backends. Par exemple, dans [75], un expert en sécurité a développé un firmware personnalisé pour un aspirateur Xiaomi qui se connecte à des points d'extrémité cloud personnels alternatifs, plus sécurisés. Cependant, la mise en œuvre de telles modifications nécessite généralement un accès root, ce qui n'est pas faisable pour la plupart des dispositifs. De plus, les applications compagnon IoT partagent souvent des informations sensibles avec des entreprises de publicité, qui utilisent ces données pour des campagnes de marketing ciblées. Les auteurs de [76] soulignent le volume alarmant de données confidentielles pouvant être extraites des réseaux de capteurs.

**Violations de la vie privée par des tiers.** Des tiers malveillants exploitent les canaux de communication des dispositifs IoT pour divulguer des données confidentielles et suivre les utilisateurs. Les auteurs de [47] ont démontré comment la publicité BLE peut être utilisée pour identifier les utilisateurs en fonction des schémas de trafic de leurs traqueurs de fitness. L'outil d'analyse BLEScope [77] extrait des UUID de services et de caractéristiques des applications BLE et les identifie en fonction d'informations statiques. Il évalue également si les applications chiffrent et authentifient le trafic de couche application en analysant les appels d'API cryptographiques. Dans une autre étude [10], les chercheurs ont évalué la sécurité du trafic Wi-Fi entre les traqueurs de fitness et le backend, trouvant un manque de chiffrement et de protection de l'intégrité, ce qui permet aux attaquants d'écouter les informations de connexion Web des utilisateurs et des données sensibles telles que des dossiers de santé. De même, les auteurs de [78] ont révélé que les attaquants peuvent lire, modifier et supprimer des enregistrements stockés dans les banques de mémoire de Fitbit, exposant toutes les données qu'elles contiennent. Comme démontré dans [79], un FAI malveillant ou un observateur du réseau pourrait déduire des activités privées à domicile en analysant le trafic Internet chiffré des appareils intelligents. Dans [80], les auteurs ont découvert qu'exploiter des états illégaux et des connexions non au-

torisées pourrait même donner aux attaquants accès aux enregistrements vidéo de caméras de sécurité privées.

**Nos améliorations.** Dans notre recherche, nous trouvons et discutons des vulnérabilités, des attaques et des contre-mesures impliquant la vie privée et les données de santé protégées par le RGPD. Nos améliorations comprennent : (i) *quatre vulnérabilités liées à la vie privée*, telles que la mémoire sensible non protégée dans le firmware de la trottinette et les identifiants de credential traçables dans FIDO2 ; et (ii) *neuf attaques portant atteinte à la vie privée*, telles que l'imitation de traqueurs de fitness (c'est-à-dire, lire les messages SMS depuis le téléphone) et le suivi des utilisateurs via des trottinettes et des authentificateurs FIDO2.

# Conclusion

Dans cette thèse, nous avons analysé la sécurité de plusieurs protocoles IoT, soulignant l'immaturité des pratiques en matière de sécurité et de vie privée. Nous avons développé une approche de test de sécurité extensible aux trackers d'activité, aux trottinettes électriques et aux authentificateurs FIDO2. En conséquence, nous avons publié des outils open-source pour virtualiser les appareils IoT (c'est-à-dire, les trackers d'activité et les trottinettes électriques), les clients (c'est-à-dire, les applications compagnon Xiaomi et les clients FIDO2), ainsi que le backend (c'est-à-dire, la partie de confiance) et les attaquer, ainsi que des outils d'analyse pour le firmware et le code (c'est-à-dire, le patcher de firmware pour trottinettes électriques et les hooks Frida), et des disecteurs de protocoles.

Nous avons découvert vingt-sept vulnérabilités qui permettent vingt-trois attaques impactantes et à faible coût au niveau des protocoles, utilisant des stratégies d'attaque novatrices (par exemple, la confusion d'API) et ciblant des surfaces d'attaque inexplorées (par exemple, les internals des trottinettes électriques). Parmi elles, nous soulignons quatre vulnérabilités et neuf attaques qui compromettent la vie privée des utilisateurs, au point de lire des données de santé protégées par le RGPD. Nous améliorons les protocoles non sécurisés, appliquant des mécanismes de sécurité à la pointe de la technologie tout en préservant leur compatibilité descendante, publions des outils d'évaluation des vulnérabilités pour tester les appareils avec des problèmes connus, et des correctifs de firmware pour protéger les utilisateurs utilisant des appareils vulnérables.

Nous avons exploré quatre questions de recherche difficiles concernant l'amélioration des tests de sécurité, la découverte de nouvelles vulnérabilités et attaques, la sécurisation des écosystèmes IoT, et la révision de l'état de la vie privée. À la suite de notre parcours, nous discutons maintenant de quatre enseignements clés de notre recherche.

Tout d'abord, les tests de sécurité IoT nécessitent encore un effort manuel significatif. Malgré la variété des techniques existantes, aucune n'a fourni une solution fiable pour analyser les protocoles fermés, surtout lorsque l'on travaille avec un ensemble limité de paquets de protocole et en recherchant des résultats d'attaque sophistiqués. Ces critères étaient essentiels à notre travail avec des protocoles propriétaires, où des protocoles comme l'appairage Mi Band peuvent impliquer seulement quatre messages, et notre attention était portée sur des propriétés de sécurité critiques comme la confidentialité et l'intégrité. Nous envisageons un cadre qui : (i) identifie les opérations de haut niveau des appareils (par exemple, l'appairage, l'établissement de session, l'authentification, et les mises à jour de firmware) ; (ii) collecte des données sur la surface d'attaque IoT (par exemple, le trafic entre appareil et application et entre application et

backend, les traces d'exécution d'application, les points de terminaison d'API, et le firmware) ; et (iii) associe ces données à des modèles connus pour rétroconcevoir la logique des protocoles et révéler leur conception. Par exemple, ce cadre pourrait détecter l'utilisation de primitives cryptographiques par instrumentation dans les applications compagnon, suivre la génération de clés de session et déchiffrer le trafic en fonction des modèles observés. Il pourrait également déduire que les paquets non chiffrés contenant des clés publiques font partie de l'appairage, tandis que les messages chiffrés représentent l'établissement de session.

Deuxièmement, l'évolution rapide des appareils IoT n'est pas accompagnée d'améliorations correspondantes en matière de sécurité. Cela entraîne une gamme toujours croissante de nouvelles fonctionnalités qui introduisent de nouvelles vulnérabilités, rendant les appareils plus susceptibles d'être la cible d'attaques plus puissantes au fil du temps. Par exemple, l'introduction par FIDO2 de credentials découvrables a exposé une nouvelle surface d'attaque, permettant aux attaquants d'effacer des identifiants et d'empêcher les utilisateurs d'accéder à leurs comptes. Nous plaidons pour une plus grande transparence de la part des fournisseurs, dont beaucoup font des déclarations audacieuses sur la sécurité de leurs produits principalement à des fins marketing, plutôt que pour protéger réellement les utilisateurs. FIDO2 annonce une protection contre le phishing, tout en s'appuyant toujours sur un code PIN (c'est-à-dire, un code susceptible d'être phished) et en étant vulnérable à de nombreuses attaques d'ingénierie sociale et de tromperie d'interface utilisateur. Nous constatons également que les fabricants déchargent les responsabilités en matière de sécurité sur les utilisateurs au lieu de traiter les risques eux-mêmes. Par exemple, l'approche de sécurité de Xiaomi exige que les utilisateurs se trouvent dans un environnement sécurisé chaque fois qu'ils activent le phare de leur trottinette électrique, car cela déclenche également le mode d'appairage, les rendant vulnérables à des attaques d'appairage malveillantes. De même, le modèle de menace de référence FIDO2 suppose la présence d'un client légitime et non compromis, négligeant la menace réaliste d'un logiciel malveillant infectant les appareils.

Troisièmement, malgré la disponibilité de mécanismes de sécurité robustes et rentables, les fabricants continuent de s'appuyer principalement sur la sécurité par obscurité. Même avec la prise de conscience généralisée de la nécessité d'une authentification forte, d'un chiffrement et d'une protection de l'intégrité, de nombreux protocoles IoT échouent à mettre en œuvre ces mesures de base. Les appareils ne chiffrent pas les données de trafic, manquent d'authentification mutuelle, et laissent le firmware non signé, malgré le soutien des systèmes sur puce (SoC) modernes au chiffrement symétrique et asymétrique. De plus, les fabricants ignorent les protocoles de sécurité fournis par des technologies telles que le BLE et le NFC, même pour des opérations critiques impliquant des données confidentielles ou sensibles.

Enfin, la vie privée reste une préoccupation secondaire dans la conception des IoT. Même avec certains progrès en matière de sécurité, les fabricants montrent peu d'intérêt à limiter les données qu'ils collectent auprès des utilisateurs. Les écosystèmes fermés compliquent encore la surveillance réglementaire, rendant difficile pour les autorités d'auditer la conformité avec le RGPD et d'autres réglementations en matière de vie privée. Par conséquent, les utilisateurs ont peu de contrôle sur leur vie privée, souvent contraints de choisir entre utiliser un appareil ou non, sans alternatives pour réduire leur exposition aux données. Renforcer la sensibilisation des utilisateurs aux problèmes de vie privée et leur fournir des outils pour protéger leurs données sont des étapes cruciales pour améliorer la vie privée dans les environnements IoT.

En conclusion, cette thèse met en évidence les lacunes significatives dans la sécurité et la vie privée des IoT, soulignant la nécessité de tests plus efficaces, de mesures proactives d'atténuation

des vulnérabilités, de mécanismes de défense robustes, et d'une plus grande transparence de la part des fabricants. Nos contributions à la recherche offrent des perspectives précieuses sur les faiblesses des pratiques actuelles et ouvrent la voie à des améliorations futures. Cependant, le chemin à parcourir nécessite la collaboration entre chercheurs, industriels et autorités réglementaires pour aborder ces défis persistants et, en fin de compte, créer un écosystème IoT plus sûr et respectueux de la vie privée. En nous basant sur les fondations posées dans ce travail, nous espérons inspirer de nouvelles avancées pour sécuriser le paysage IoT en rapide évolution.

# References


[1] CoRE Working Group. IETF CoRE Working Group. https://core-wg.github.io, 2024.

[2] IETF CoRE Working Group. RFC 7252 - The Constrained Application Protocol (CoAP). https://datatracker.ietf.org/doc/html/rfc7252, 2014.

[3] IETF CoRE Working Group. RFC 8323 - CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets. https://datatracker.ietf.org/doc/html/rfc8323, 2018.

[4] IETF CoRE Working Group. DTLS In Constrained Environments (DICE). https://datatracker.ietf.org/wg/dice, 2014.

[5] Huami. Huami Privacy Note. https://upload-cdn.huami.com/tposts/9250, 2020.

[6] Mozilla Foundation. Privacy Not Included - Mi Band 5. https://foundation.mozilla.org/en/privacynotincluded/mi-band-5/, 2020.

[7] Fitbit. Golden Gate. https://github.com/Fitbit/golden-gate, 2023.

[8] *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(3):330–366, 2022.

[9] Marco Casagrande, Riccardo Cestaro, Eleonora Losiouk, Mauro Conti, and Daniele Antonioli. E-spoofer: Attacking and defending xiaomi electric scooter ecosystem. In *Proceedings of the 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, page 85–95, 2023.

[10] Hossein Fereidooni, Tommaso Frassetto, Markus Miettinen, Ahmad-Reza Sadeghi, and Mauro Conti. Fitness Trackers: Fit for Health but Unfit for Security and Privacy. In *2017 IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE '17)*, pages 19–24. IEEE, 2017.

[11] Dennis Giese. Having fun with IoT: Reverse Engineering and Hacking of Xiaomi IoT Devices. https://dontvacuum.me/talks/DEFCON26/DEFCON26-Having_fun_with_IoT-Xiaomi.html, 2018.

[12] Dongkwan Kim, Suwan Park, Kibum Choi, and Yongdae Kim. Burnfit: Analyzing and exploiting wearable devices. In Ho-won Kim and Dooho Choi, editors, *Information Security Applications*, pages 227–239. Springer International Publishing, 2016.

[13] The App Analyst. App Analysis: Bird. https://theappanalyst.com/bird.html, 2019.

[14] Jiska Classen, Daniel Wegemer, Paul Patras, Tom Spink, and Matthias Hollick. Anatomy of a Vulnerable Fitness Tracking System: Dissecting the Fitbit Cloud, App, and Firmware. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT '18)*, 2(1):1–24, 2018.

[15] Yogesh Ojha. I hacked MiBand 3, and here is how I did it. Part I. https://medium.com/@yogeshoa/i-hacked-xiaomi-miband-3-and-here-is-how-i-did-it-43d68c272391, 2018.

[16] Dennis Heinze, Jiska Classen, and Felix Rohrbach. MagicPairing: Apple's Take on Securing Bluetooth Peripherals. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 111–121, 2020.

[17] Qingchuan Zhao, Chaoshun Zuo, Jorge Blasco, and Zhiqiang Lin. Periscope: Comprehensive vulnerability analysis of mobile app-defined bluetooth peripherals. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, page 521–533, New York, NY, USA, 2022. Association for Computing Machinery.

[18] Britt Cyr, Webb Horn, Daniela Miao, and Michael Specter. Security Analysis of Wearable Fitness Devices (Fitbit). *Massachusetts Institute of Technology*, 1, 2014.

[19] Li Lin, Xuanyu Liu, Xiao Fu, Bin Luo, Xiaojiang Du, and Mohsen Guizani. A non-intrusive method for smart speaker forensics. In *ICC 2021 - IEEE International Conference on Communications*, pages 1–6, 2021.

[20] Xuechao Du, Andong Chen, Boyuan He, Hao Chen, Fan Zhang, and Yan Chen. Afliot: Fuzzing on linux-based iot device with binary-level instrumentation. *Computers & Security*, 122:102889, 2022.

[21] Jonas Zaddach, Luca Bruno, Aurelien Francillon, and Davide Balzarotti. Avatar: A framework to support dynamic security analysis of embedded systems' firmwares. In *NDSS 2014, Network and Distributed System Security Symposium, 23-26 February 2014, San Diego, USA*, 2014.

[22] Jan Ruge, Jiska Classen, Francesco Gringoli, and Matthias Hollick. Frankenstein: advanced wireless fuzzing to exploit new bluetooth escalation targets. In *Proceedings of the 29th USENIX Conference on Security Symposium*, 2020.

[23] Sam L. Thomas, Jan V. den Herrewegen, Georgios Vasilakis, Zitai Chen, Mihai Ordean, and Flavio D. Garcia. Cutting Through the Complexity of Reverse Engineering Embedded Devices. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021:360–389, 2021.

[24] Andrew Hilts, Christopher Parsons, and Jeffrey Knockel. Every step you fake: A comparative analysis of fitness tracker privacy and security. *Open Effect Report*, 76(24):31–33, 2016.

[25] Nisha Vinayaga-Sureshkanth, Raveen Wijewickrama, Anindya Maiti, and Murtuza Jadliwala. An investigative study on the privacy implications of mobile e-scooter rental apps.

In *Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, page 125–139, New York, NY, USA, 2022. Association for Computing Machinery.

[26] Maarten Schellevis, Bart Jacobs, Carlo Meijer, and Joeri de Ruiter. Getting Access to Your Own Fitbit Data. Master's thesis, Radboud University, 2016.

[27] Hossein Fereidooni, Jiska Classen, Tom Spink, Paul Patras, Markus Miettinen, Ahmad-Reza Sadeghi, Matthias Hollick, and Mauro Conti. Breaking Fitness Records Without Moving: Reverse Engineering and Spoofing Fitbit. In *Research in Attacks, Intrusions, and Defenses - 20th International Symposium (RAID '17)*, pages 48–69, 2017.

[28] Olivier Pereira, Florentin Rochet, and Cyrille Wiedling. Formal analysis of the FIDO 1. x protocol. In *Foundations and Practice of Security: 10th International Symposium, FPS 2017, Nancy, France, October 23-25, 2017, Revised Selected Papers 10*, pages 68–82. Springer, 2018.

[29] Karl Norrman, Vaishnavi Sundararajan, and Alessandro Bruni. Formal analysis of edhoc key establishment for constrained iot devices, 2021.

[30] Mujahid Mohsin, Zahid Anwar, Ghaith Husari, Ehab Al-Shaer, and Mohammad Ashiqur Rahman. Iotsat: A formal framework for security analysis of the internet of things (iot). In *2016 IEEE Conference on Communications and Network Security (CNS)*, pages 180–188, 2016.

[31] Ian D. Foster. Xiaomi M365 Scooter Authentication Bypass. https://lanrat.com/xiaomi-m365/, 2019.

[32] Rani Idan (Zimperium). Don't Give Me A Brake – Xiaomi Scooter Hack Enables Dangerous Accelerations And Stops For Unsuspecting Riders. https://blog.zimperium.com/dont-give-me-a-brake-xiaomi-scooter-hack-enables-dangerous-accelerations-and-stops-for- 2019.

[33] BBC News. Scooters Hacked To Play Rude Messages To Riders. https://www.bbc.com/news/technology-48065432, 2019.

[34] Fabian Ullrich, Jiska Classen, Johannes Eger, and Matthias Hollick. Vacuums in ihe cloud: Analyzing security in a hardened iot ecosystem. In *13th USENIX Workshop on Offensive Technologies (WOOT 19)*, Santa Clara, CA, 2019. USENIX Association.

[35] Mathy Vanhoef. Fragment and Forge: Breaking Wi-Fi Through Frame Aggregation and Fragmentation. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 161–178. USENIX Association, 2021.

[36] Zouheir Trabelsi. Investigating the robustness of iot security cameras against cyber attacks. In *2022 5th Conference on Cloud and Internet of Things (CIoT)*, pages 17–23, 2022.

[37] Thomas Roche, Victor Lomné, Camille Mutschler, and Laurent Imbert. A side journey to titan. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 231–248, 2021.

[38] Michal Kepkowski, Lucjan Hanzlik, Ian Wood, and Mohamed Ali Kaafar. How Not to Handle Keys: Timing Attacks on FIDO Authenticator Privacy. In *Proceedings on Privacy Enhancing Technologies*, volume 4, pages 705–726, 2022.

[39] Victor Lomne. An Overview Of The Security Of Some Hardware FIDO(2) Tokens. https://www.youtube.com/watch?v=hpOp9X4sMaE, 2022.

[40] Daniele Antonioli, Nils O. Tippenhauer, and Kasper Rasmussen. Key Negotiation Downgrade Attacks on Bluetooth and Bluetooth Low Energy. *ACM Transactions on Privacy and Security*, 23(3), 2020.

[41] Jiliang Wang, Feng Hu, Ye Zhou, Yunhao liu, Hanyi Zhang, and Zhe Liu. BlueDoor: Breaking the Secure Information Flow via BLE Vulnerability. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services (MobiSys '20)*, page 286–298. Association for Computing Machinery, 2020.

[42] Mike Ryan. Bluetooth: With Low Energy Comes Low Security. In *Proceedings of the 7th USENIX Conference on Offensive Technologies (WOOT'13)*, page 4. USENIX Association, 2013.

[43] Eli Biham and Lior Neumann. Breaking the Bluetooth Pairing – The Fixed Coordinate Invalid Curve Attack. In *Selected Areas in Cryptography (SAC '19)*, pages 250–273. Springer International Publishing, 01 2020.

[44] Yue Zhang, Jian Weng, Rajib Dey, Yier Jin, Zhiqiang Lin, and Xinwen Fu. Breaking Secure Pairing of Bluetooth Low Energy Using Downgrade Attacks. In *29th USENIX Security Symposium (USENIX Security '20)*, pages 37–54. USENIX Association, 2020.

[45] Yue Zhang, Jian Weng, Rajib Dey, Yier Jin, Zhiqiang Lin, and Xinwen Fu. Breaking Secure Pairing of Bluetooth Low Energy Using Downgrade Attacks. In *29th USENIX Security Symposium (USENIX Security '20)*, pages 37–54. USENIX Association, 2020.

[46] Maximilian von Tschirschnitz, Ludwig Peuckert, Fabian Franzen, and Jens Grossklags. Method confusion attack on bluetooth pairing. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1332–1347, 2021.

[47] Aveek K. Das, Parth H. Pathak, Chen-Nee Chuah, and Prasant Mohapatra. Uncovering Privacy Leakage in BLE Network Traffic of Wearable Fitness Trackers. In *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications (HotMobile '16)*, page 99–104, 2016.

[48] Aleksandra Korolova and Vinod Sharma. Cross-App Tracking via Nearby Bluetooth Low Energy Devices. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy (CODASPY '18)*, page 43–52. Association for Computing Machinery, 2018.

[49] Muhammad Naveed, Xiaoyong Zhou, Soteris Demetriou, Xiaofeng Wang, and Carl Gunter. Inside job: Understanding and mitigating the threat of external device mis-bonding on android. In *NDSS*, 2014.

[50] Pallavi Sivakumaran and Jorge Blasco. A study of the feasibility of co-located app attacks against BLE and a Large-Scale analysis of the current Application-Layer security landscape. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1–18, Santa Clara, CA, 2019. USENIX Association.

[51] Xiaolong Bai, Luyi Xing, Nan Zhang, XiaoFeng Wang, Xiaojing Liao, Tongxin Li, and Shi-Min Hu. Staying secure and unprepared: understanding and mitigating the security risks of Apple Zeroconf. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 655–674, 2016.

[52] Lennert Wouters, Eduard Marin, Tomer Ashur, Benedikt Gierlichs, and Bart Preneel. Fast, Furious and Insecure: Passive Keyless Entry and Start Systems in Modern Supercars. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(3):66–85, 2019.

[53] Lennert Wouters, Benedikt Gierlichs, and Bart Preneel. My Other Car is Your Car: Compromising the Tesla Model X Keyless Entry System. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(4):149–172, 2021.

[54] Andrey Bogdanov. *Linear Slide Attacks on the KeeLoq Block Cipher*, page 66–80. 2007.

[55] Roel Verdult, Flavio D. Garcia, and Josep Balasch. Gone in 360 seconds: Hijacking with hitag2. page 37, 2012.

[56] Roel Verdult, Flavio D. Garcia, and Baris Ege. Dismantling megamos crypto: Wirelessly lockpicking a vehicle immobilizer. In *22nd USENIX Security Symposium (USENIX Security 13)*, 2013.

[57] CamiAlfa. M365 55AA BLE Protocol. https://github.com/CamiAlfa/M365-BLE-PROTOCOL, 2017.

[58] Piotr Dobrowolski. M365 5AA5 BLE Protocol. https://github.com/Informatic/py9b, 2019.

[59] Daljeet Nandha and Florian Bruhin. EC MiAuth Library. https://github.com/dnandha/miauth, 2022.

[60] Manuel Barbosa, Alexandra Boldyreva, Shan Chen, and Bogdan Warinschi. Provable security analysis of FIDO2. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part III 41*, pages 125–156. Springer, 2021.

[61] Michael Scott. FIDO–That Dog Won't Hunt. In *Security and Privacy in New Computing Environments: EAI Conference, SPNCE 2020*, pages 255–264. Springer, 2021.

[62] Alon Shakevsky, Eyal Ronen, and Avishai Wool. Trust Dies in Darkness: Shedding Light on Samsung's TrustZone Keymaster Design. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 251–268, 2022.

[63] Eirini Anthi, Lowri Williams, Małgorzata Słowińska, George Theodorakopoulos, and Pete Burnap. A supervised intrusion detection system for smart home iot devices. *IEEE Internet of Things Journal*, 6(5):9042–9053, 2019.

[64] Mojtaba Eskandari, Zaffar Haider Janjua, Massimo Vecchio, and Fabio Antonelli. Passban ids: An intelligent anomaly-based intrusion detection system for iot edge devices. *IEEE Internet of Things Journal*, 7(8):6882–6897, 2020.

[65] Parth Bhatt and Anderson Morais. Hads: Hybrid anomaly detection system for iot environments. In *2018 International Conference on Internet of Things, Embedded Systems and Communications (IINTEC)*, pages 191–196, 2018.

[66] Romain Cayre, Vincent Nicomette, Guillaume Auriol, Mohamed Kaâniche, and Aurélien Francillon. Oasis: An intrusion detection system embedded in bluetooth low energy controllers. In *Proceedings of the 2024 ACM Asia conference on Computer and Communications Security (ASIACCS)*, 2024.

[67] Nick Frymann, Daniel Gardham, Franziskus Kiefer, Emil Lundberg, Mark Manulis, and Dain Nilsson. Asynchronous Remote Key Generation: An Analysis of Yubico's Proposal for W3C WebAuthn. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 939–954, 2020.

[68] Lucjan Hanzlik, Julian Loss, and Benedikt Wagner. Token meets wallet: Formalizing privacy and revocation for FIDO2. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1491–1508. IEEE, 2023.

[69] Nina Bindel, Nicolas Gama, Sandra Guasch, and Eyal Ronen. To attest or not to attest, this is the question–provable attestation in fido2. *Cryptology ePrint Archive*, 2023.

[70] Jawad Hassan, Adnan Sohail, Ali Ismail Awad, and M. Ahmed Zaka. Letm-iot: A lightweight and efficient trust mechanism for sybil attacks in internet of things networks. *Ad Hoc Networks*, 163, 2024.

[71] Anum Khurshid, Sileshi Demesie Yalew, Mudassar Aslam, and Shahid Raza. Tee-watchdog: Mitigating unauthorized activities within trusted execution environments in arm-based low-power iot devices. *Security and Communication Networks*, 2022(1), 2022.

[72] Koen Zandberg, Kaspar Schleiser, Francisco Acosta, Hannes Tschofenig, and Emmanuel Baccelli. Secure firmware updates for constrained iot devices using open standards: A reality check. *IEEE Access*, 7:71907–71920, 2019.

[73] Gong-Xu Liu, Ling-Feng Shi, and Dong-Jin Xin. Data integrity monitoring method of digital sensors for internet-of-things applications. *IEEE Internet of Things Journal*, 7(5):4575–4584, 2020.

[74] Alanoud Subahi and George Theodorakopoulos. Ensuring Compliance of IoT Devices with Their Privacy Policy Agreement. In *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 100–107, 2018.

[75] Daniel AW Dennis Giese. Unleash Your Smart-Home Devices: Vacuum Cleaning Robot Hacking. https://media.ccc.de/v/34c3-9147-unleash_your_smart-home_devices_vacuum_cleaning_robot_hacking, 2017.

[76] Dong Chen, Phuthipong Bovornkeeratiroj, David Irwin, and Prashant Shenoy. Private memoirs of iot devices: Safeguarding user privacy in the iot era. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1327–1336, 2018.

[77] Chaoshun Zuo, Haohuang Wen, Zhiqiang Lin, and Yinqian Zhang. Automatic Fingerprinting of Vulnerable BLE IoT Devices with Static UUIDs from Mobile Apps. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19), London, United Kingdom*, page 1469–1483, 2019.

[78] Mahmudur Rahman, Bogdan Carbunar, and Madhusudan Banik. Fit and Vulnerable: Attacks and Defenses for a Health Monitoring Device. *ArXiv*, abs/1304.5672, 2013.

[79] Noah J. Apthorpe, Dillon Reisman, Srikanth Sundaresan, Arvind Narayanan, and Nick Feamster. Spying on the Smart Home: Privacy Attacks and Defenses on Encrypted IoT Traffic. *CoRR*, 2017.

[80] Wei Zhou, Yan Jia, Yao Yao, Lipeng Zhu, Le Guan, Yuhang Mao, Peng Liu, and Yuqing Zhang. Discovering and Understanding the Security Hazards in the Interactions between IoT Devices, Mobile Apps, and Clouds on Smart Home Platforms. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1133–1150, 2019.

[81] Mahmudur Rahman, Bogdan Carbunar, and Umut Topkara. Secure Management of Low Power Fitness Trackers. *IEEE Transactions on Mobile Computing*, 15(2):447–459, 2016.

[82] International Data Corporation. Shipments of Wearable Devices Leap to 125 Million Units, Up 35.1% in the Third Quarter, According to IDC. https://www.idc.com/getdoc.jsp?containerId=prUS47067820, 2020.

[83] Rohit Goyal, Nicola Dragoni, and Angelo Spognardi. Mind the Tracker You Wear: A Security Analysis of Wearable Health Trackers. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing (SAC '16), Pisa, Italy*, page 131–136, 2016.

[84] Statcounter. Mobile and Tablet Android Version Market Share Worldwide (Nov 2020 - Nov 2021). https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide, 2021.

[85] Bluetooth SIG. Bluetooth Core Specification v5.2. https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=478726, 2019.

[86] Xiaomi Inc. Mi Band Homepage. https://www.mi.com/global/miband.

[87] Huami Inc. Amazfit Homepage. https://www.amazfit.com/en/.

[88] Huami Inc. Huami Homepage. https://www.huami.com/investor.

[89] Anhui Huami Information Technology Co. Mi Fit for Android. https://play.google.com/store/apps/details?id=com.xiaomi.hm.health&hl=en_US&gl=US.

[90] Huami Inc. Mi Fit for iOS. https://apps.apple.com/us/app/mi-fit/id938688461.

[91] Huami Inc. Zepp (formerly Amazfit) for Android. https://play.google.com/store/apps/details?id=com.huami.watch.hmwatchmanager&hl=en_US&gl=US.

[92] Huami Inc. Zepp (formerly Amazfit) for iOS. https://apps.apple.com/us/app/zepp-formerly-amazfit/id1127269366.

[93] Mozilla Foundation. Privacy Not Included - Mi Band 5. https://foundation.mozilla.org/en/privacynotincluded/mi-band-5/, 2020.

[94] Mozilla Foundation. Privacy Not Included - Mi Band 6. https://foundation.mozilla.org/en/privacynotincluded/mi-band-6/, 2020.

[95] Mozilla Foundation. Privacy Not Included - Amazfit Fitness trackers. https://foundation.mozilla.org/en/privacynotincluded/amazfit-fitness-trackers/, 2020.

[96] Rene Mayrhofer, Jeffrey V. Stoep, Chad Brubaker, and Nick Kralevich. The Android Platform Security Model. *ACM Transactions on Privacy and Security*, 24(3), 2021.

[97] Ravie Lakshmanan. Over 750.000 Users Downloaded New Billing Fraud Apps From Google Play Store. https://thehackernews.com/2021/04/over-750000-users-download-new-billing.html, 2021.

[98] Ravie Lakshmanan. Attention! FluBot Android Banking Malware Spreads Quickly Across Europe. https://thehackernews.com/2021/04/attention-flubot-android-banking.html, 2021.

[99] Ravie Lakshmanan. WhatsApp-based wormable Android malware spotted on the Google Play Store. https://thehackernews.com/2021/04/whatsapp-based-wormable-android-malware.html, 2021.

[100] Platon Kotzias, Juan Caballero, and Leyla Bilge. How Did That Get In My Phone? Unwanted App Distribution on Android Devices. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 53–69, 2021.

[101] Philippe Biondi and the Scapy Community. Scapy. https://scapy.net/, 2021.

[102] Kimi Newt. Pyshark. https://pypi.org/project/pyshark/, 2021.

[103] Wireshark. Wireshark. https://www.wireshark.org/, 2021.

[104] Python Cryptographic Authority. Python Cryptography. https://cryptography.io/en/latest/, 2021.

[105] Skylot. JADX. https://github.com/skylot/jadx, 2021.

[106] Ole André Vadla Ravnås. Frida. https://frida.re/, 2023.

[107] Sandeep Mistry. Bleno. https://github.com/noble/bleno, 2021.

[108] Sandeep Mistry. Noble. https://github.com/noble/noble, 2021.

[109] Freezed or frozen. Pymb1a. https://github.com/freezed-or-frozen/pymb1a, 2019.

[110] Dialog Semiconductor. Dialog Semiconductor. https://www.dialog-semiconductor.com/, 2021.

[111] Virtualabs. BtleJack: a new Bluetooth Low Energy swiss-army knife. https://github.com/virtualabs/btlejack, 2021.

[112] Mike Ryan. Bluetooth: With Low Energy Comes Low Security. In *7th USENIX Workshop on Offensive Technologies (WOOT 13)*. USENIX Association, 2013.

[113] Inc. Fitbit. Fitbit Homepage. https://www.fitbit.com/global/it/home.

[114] Inc. Fitbit. Fitbit app for Android. https://play.google.com/store/apps/details?id=com.fitbit.FitbitMobile&hl=en_US&gl=US.

[115] Roger M. Needham and David J. Wheeler. TEA Extensions. *Report, Cambridge University*, 1997.

[116] Mihir Bellare, Phillip Rogaway, and David Wagner. The EAX Mode of Operation. In *International Workshop on Fast Software Encryption*, pages 389–407. Springer, 2004.

[117] Nordic Semiconductor ASA. nRF Connect for Mobile. https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp, 2021.

[118] Freeyourgadget. Gadgetbridge a free and cloudless replacement for your gadget vendors' closed source Android applications. https://codeberg.org/Freeyourgadget/Gadgetbridge, 2021.

[119] Maxim Krasnyansky, Marcel Holtmann, and Fabrizio Gennari. Hcidump. https://manpages.debian.org/testing/bluez-hcidump/hcidump.1.en.html, 2021.

[120] Android. Capture and Read Bug Reports. https://developer.android.com/studio/debug/bug-report, 2021.

[121] Mitmproxy Project. Mitmproxy. https://mitmproxy.org/, 2022.

[122] Benjamin Bichsel, Veselin Raychev, Petar Tsankov, and Martin Vechev. Statistical Deobfuscation of Android Applications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, page 343–355. Association for Computing Machinery, 2016.

[123] Caleb Fenton. Simplify Android Deobfuscator. https://github.com/CalebFenton/simplify, 2020.

[124] Samczsun. Java Deobfuscator. https://github.com/java-deobfuscator/deobfuscator, 2020.

[125] iBotPeaches. Apktool. https://ibotpeaches.github.io/Apktool/, 2021.

[126] Ajin Abraham. Mobile Security Framework. https://github.com/MobSF/Mobile-Security-Framework-MobSF, 2021.

[127] Anthony Desnos and Geoffroy Guegue. Androguard. https://github.com/androguard/androguard, 2019.

[128] Henrik Blidh and David Lechner. Bleak. https://pypi.org/project/bleak/, 2021.

[129] Creotiv. Mi Band 2 - Python Library. https://github.com/creotiv/MiBand2, 2019.

[130] Yogeshojha. Mi Band 3 - Python Library. https://github.com/yogeshojha/MiBand3, 2019.

[131] Satcar77. Mi Band 4 - Python Library. https://github.com/satcar77/miband4, 2021.

[132] Xiaomi Inc. Mi Home for Android. https://play.google.com/store/apps/details?id=com.xiaomi.smarthome&hl=it&gl=US.

[133] Ltd Beijing Xiaomi Co. Mi Home for iOS. https://apps.apple.com/us/app/mi-home-xiaomi-smart-home/id957323480.

[134] AV-TEST Team. Analysis of Fitbit Vulnerabilities. https://www.av-test.org/fileadmin/pdf/avtest_2016-04_fitbit_vulnerabilities.pdf, 2015.

[135] Taher Issoufaly and Pierre U. Tournoux. BLEB: Bluetooth Low Energy Botnet for Large Scale Individual Tracking. In *1st International Conference on Next Generation Computing Applications (NextComp '17)*, pages 115–120, 2017.

[136] Kassem Fawaz, Kyu-Han Kim, and Kang G. Shin. Protecting Privacy of BLE Device Users. In *25th USENIX Security Symposium (USENIX Security '16)*, pages 1205–1221. USENIX Association, 2016.

[137] Maximilian von Tschirschnitz, Ludwig Peuckert, Franzen Franzen, and Jens Grossklags. Method Confusion Attack on Bluetooth Pairing. In *2021 IEEE Symposium on Security and Privacy (SP '21)*, pages 213–228. IEEE Computer Society, 2021.

[138] Sławomir Jasek. Gattacking Bluetooth Smart Devices. Black Hat USA Conference, 2016.

[139] Jianliang Wu, Yuhong Nan, Vireshwar Kumar, Dave J. Tian, Antonio Bianchi, Mathias Payer, and Dongyan Xu. BLESA: Spoofing Attacks against Reconnections in Bluetooth Low Energy. In *14th USENIX Workshop on Offensive Technologies (WOOT '20)*. USENIX Association, 2020.

[140] Armis Inc. BLEEDINGBIT: The Hidden Attack Surface Within BLE Chips. https://armis.com/bleedingbit/, 2019.

[141] Matheus Garbelini, Sudipta Chattopadhyay, and Chundong Wang. SweynTooth: Unleashing Mayhem over Bluetooth Low Energy. https://asset-group.github.io/disclosures/sweyntooth/sweyntooth.pdf, 2020.

[142] Mohit Sethi, Aleksi Peltonen, and Tuomas Aura. Misbinding Attacks on Secure Device Pairing and Bootstrapping. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security (Asia CCS '19)*, page 453–464. Association for Computing Machinery, 2019.

[143] Muhammad Naveed, Xiao yong Zhou, Soteris Demetriou, XiaoFeng Wang, and Carl A. Gunter. Inside Job: Understanding and Mitigating the Threat of External Device Mis-Binding on Android. In *21st Annual Network and Distributed System Security Symposium (NDSS '14)*, 2014.

[144] Pallavi Sivakumaran and Jorge Blasco. A Study of the Feasibility of Co-located App Attacks against BLE and a Large-Scale Analysis of the Current Application-Layer Security Landscape. In *28th USENIX Security Symposium (USENIX Security '19)*, pages 1–18. USENIX Association, 2019.

[145] iResearch. iResearch Coverage On Xiaomi. http://www.iresearchchina.com/Upload/201808/20180824143739_3256.pdf, 2018.

[146] Marco Casagrande, Eleonora Losiouk, Mauro Conti, Mathias Payer, and Daniele Antonioli. Breakmi: Reversing, exploiting and fixing xiaomi fitness tracking ecosystem. *IACR Transactions On Cryptographic Hardware And Embedded Systems*, 2022(3):330–366, 2022.

[147] Enrico Punsalang from InsideEVs. Segway-Ninebot Has Sold More Than One Million E-Scooters In China. https://insideevs.com/news/613420/segway-ninebot-one-million-sold-china/, 2022.

[148] Xiaomi. Xiaomi Bug Bounty Program On HackerOne. https://hackerone.com/xiaomi?type=team, 2022.

[149] Reuters. Xiaomi-backed Chinese firm acquires iconic scooter maker Segway. https://www.reuters.com/article/us-ninebot-xiaomi-investment-idUSKBN0N60GN20150415, 2015.

[150] Sandeep Mistry. Noble NodeJS BLE Central Module (Abandonware). https://www.npmjs.com/package/@abandonware/noble, 2022.

[151] Dariusz Seweryn. Rxandroidble library. https://github.com/dariuszseweryn/RxAndroidBle, 2022.

[152] NSA. Ghidra Reverse Engineering Suite. https://ghidra-sre.org/, 2022.

[153] MiEcosystem. Mijia BLE Libraries. https://github.com/MiEcosystem/mijia_ble, 2019.

[154] VirusTotal. Yara. https://virustotal.github.io/yara/, 2022.

[155] Nordic SemiConductor. nRF51822 System-on-Chip. https://www.nordicsemi.com/Products/nRF51822, 2022.

[156] ScooterHacking. ScooterHacking Github. https://github.com/orgs/scooterhacking/repositories, 2022.

[157] STM Electronics. ST-LINK Debugger V2. https://www.st.com/en/development-tools/st-link-v2.html, 2022.

[158] OPENOCD. OPENOCD. https://openocd.org/, 2022.

[159] Pavel Revak. PySWD. https://github.com/cortexm/pyswd, 2019.

[160] VooDooShamane from Rollerplausch.com. MiDu Flasher. https://rollerplausch.com/threads/midu-flasher-st-link-downgrade-unbrick.5399/, 2022.

[161] Buildxyz. nRFSec. https://github.com/buildxyz-git/nrfsec, 2020.

[162] BotoX. Xiaomi M365 Firmware Patcher. https://github.com/BotoX/xiaomi-m365-firmware-patcher, 2022.

[163] CamiAlfa. M365Downg. https://play.google.com/store/apps/details?id=com.m365downgrade, 2022.

[164] Brian Benchoff. Security Engineering: Inside the Scooter Startups. https://hackaday.com/2019/02/12/security-engineering-inside-the-scooter-startups/, 2019.

[165] ScooterHacking. ScooterHacking Website. https://scooterhacking.org/, 2022.

[166] ScooterHacking. ScooterHacking Utility App. https://play.google.com/store/apps/details?id=sh.cfw.utility, 2022.

[167] ScooterHacking. ScooterHacking Firmware Toolkit. https://mi.cfw.sh/, 2022.

[168] ScooterHacking. ScooterHacking Ninebot EC Protocol. https://wiki.scooterhacking.org/doku.php?id=nbdocs, 2022.

[169] Hossein Fereidooni, Tommaso Frassetto, Markus Miettinen, Ahmad-Reza Sadeghi, and Mauro Conti. Fitness trackers: Fit for health but unfit for security and privacy. In *2017 IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, 2017.

[170] Vincent Toubiana and Mathieu Cunche. No need to ask the android: Bluetooth-low-energy scanning without the location permission. In *ACM WiSec*, page 147–152, New York, NY, USA, 2021. Association for Computing Machinery.

[171] Mike Ryan. Bluetooth: With low energy comes low security. In *Proceedings of the 7th USENIX Conference on Offensive Technologies*, page 4, USA, 2013. USENIX Association.

[172] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper B. Rasmussen. The KNOB is Broken: Exploiting Low Entropy in the Encryption Key Negotiation Of Bluetooth BR/EDR. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1047–1061, Santa Clara, CA, 2019. USENIX Association.

[173] Statista. Battery electric vehicles - worldwide. https://www.statista.com/outlook/mmo/electric-vehicles/battery-electric-vehicles/worldwide, 2024.

[174] Grand View Research. Electric Scooters Market Size, Share & Trends Analysis Report (2024- 2030). https://www.grandviewresearch.com/industry-analysis/electric-scooters-market, 2024.

[175] Grand View Research. Micro-mobility market size, share & trends analysis report by vehicle type (electric kick scooters, electric skateboards, electric bicycles), by battery, by voltage, by region, and segment forecasts, 2021 - 2028. https://www.grandviewresearch.com/industry-analysis/micro-mobility-market-report, 2024.

[176] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *20th USENIX security symposium (USENIX Security 11)*, 2011.

[177] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. Experimental security analysis of a modern automobile. In *2010 IEEE symposium on security and privacy*, pages 447–462. IEEE, 2010.

[178] Marco Casagrande, Riccardo Cestaro, Eleonora Losiouk, Mauro Conti, and Daniele Antonioli. E-Spoofer: Attacking and Defending Xiaomi Electric Scooter Ecosystem. In *Proceedings of the 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '23, New York, NY, USA, 2023. Association for Computing Machinery.

[179] Aaron Luo. Multidimensional Attack Vectors and Countermeasures. https://media.defcon.org/DEF%20CON%2024/DEF%20CON%2024%20presentations/DEF%20CON%2024%20-%20Aaron-Luo-Drones-Hijacking-Multi-Dimensional-Attack-Vectors-And-Countermeasures-UPDAT pdf, 2016.

[180] Moritz Schloegel and Nico Schiller. Unchained Skies: A Deep Dive into Reverse Engineering and Exploitation of Drones. https://cfp.recon.cx/2023/talk/HLHH89/, 2023.

[181] Pedro Cabrera. Parrot Drones Hijacking. https://www.rsaconference.com/Library/presentation/USA/2018/parrot-drones-hijacking, 2018.

[182] Univ Datos. Folding Electric Scooter Market: Current Analysis and Forecast (2023-2030). https://univdatos.com/report/folding-electric-scooter-market/, 2022.

[183] Ma Si. Segway-Ninebot Eyes Bigger Market Share in Country. https://global.chinadaily.com.cn/a/202204/28/WS6269ee14a310fd2b29e59d1f.html, 2022.

[184] Tech We Want. Who Makes Bird and Lime Scooters? https://techwewant.com/this-is-who-makes-bird-lime-and-jump-scooters-review-b3f6be32221e, 2018.

[185] STMicroelectronics. Migration of Applications from the STM8L and STM8S Series to the STM32C0 Series Microcontrollers. https://www.st.com/resource/en/application_note/an5775-migration-of-applications-from-the-stm8l-and-stm8s-series-to-the-stm32c0-series- pdf, 2022.

[186] Patrick Kiley. Reverse Engineering the Tesla Battery Management System to Increase Power Available. https://www.youtube.com/watch?v=UV2zvgyIF0I, 2020.

[187] Charlie Miller. Battery firmware hacking. *Black Hat USA*, pages 3–4, 2011.

[188] Daljeet Nandha. Exploring Xiaomi's New Firmware Security Measures. https://robocoffee.de/?p=193, 2022.

[189] BotoX. Xiaomi M365 Firmware Patcher (GitHub). https://github.com/BotoX/xiaomi-m365-firmware-patcher/blob/master/xiaotea/xiaotea.py, 2024.

[190] ScooterHacking. ScooterHacking Utility App. https://play.google.com/store/apps/details?id=sh.cfw.utility, 2022.

[191] ScooterHacking. ScooterHacking Utility Homepage. https://utility.cfw.sh/, 2023.

[192] Scootered. Xiaomi M365 Full Specs. https://www.scootered.co.uk/electric-scooter-specs/xiaomi-m365-electric-scooter-full-specification.html, 2024.

[193] Scootered. Xiaomi Mi 3 Full Specs. https://www.scootered.co.uk/electric-scooter-specs/xiaomi-mi-3-electric-scooter-full-specification.html, 2024.

[194] STMicroelectronics. Ultra-low-power 8-bit MCU with 32 Kbytes Flash, 16 MHz CPU, integrated EEPROM. https://www.st.com/en/microcontrollers-microprocessors/stm8l151k6.html, 2018.

[195] STMicroelectronics. STM8L151K6 Datasheet. https://www.st.com/resource/en/datasheet/stm8l151r6.pdf, 2018.

[196] Texas Instrument. 6 to 10-Series Cell Li-Ion and Li-Phosphate Battery Monitor. https://www.ti.com/product/BQ76930, 2022.

[197] Texas Instrument. BQ769x0 Datasheet. https://www.ti.com/lit/ds/symlink/bq76930.pdf, 2022.

[198] Texas Instruments Yevgen Barsukov. Battery Cell Balancing: What to Balance and How. https://www.ti.com/download/trng/docs/seminar/Topic%202%20-%20Battery%20Cell%20Balancing%20-%20What%20to%20Balance%20and%20How.pdf, 2009.

[199] Marco Casagrande. E-Spoofer (GitHub). https://github.com/Skiti/ESpoofer, 2024.

[200] Shashank Sripad, Sekar Kulandaivel, Vikram Pande, Vyas Sekar, and Venkatasubramanian Viswanathan. Vulnerabilities of Electric Vehicle Battery Packs to Cyberattacks on Auxiliary Components. *arXiv preprint arXiv:1711.04822*, 2017.

[201] Texas Instruments. How To Protect 48-V Batteries from Overcurrent and Undervoltage. https://www.ti.com/lit/an/snoaa65/snoaa65.pdf, 2020.

[202] Rui Guo, Minggao Ouyang, Languang Lu, and Xuning Feng. Mechanism of the Entire Overdischarge Process and Overdischarge-induced Internal Short Circuit in Lithium-ion Batteries. *Scientific Reports*, page 30248, 2016.

[203] Xiaomi. Query Estimated Spare Parts Price. https://www.mi.com/uk/support/spare-parts-price/?name=xiaomi-electric-scooter-4-pro-2nd-gen, 2024.

[204] Jason Uher, Ryan G Mennecke, and Bassam S Farroha. Denial of Sleep Attacks in Bluetooth Low Energy Wireless Sensor Networks. In *MILCOM 2016-2016 IEEE Military Communications Conference*, pages 1231–1236. IEEE, 2016.

[205] Philipp Markert, Daniel V Bailey, Maximilian Golla, Markus Dürmuth, and Adam J Aviv. This pin can be easily guessed: Analyzing the security of smartphone unlock pins. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 286–303. IEEE, 2020.

[206] Ding Wang, Qianchen Gu, Xinyi Huang, and Ping Wang. Understanding human-chosen pins: Characteristics, distribution and security. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, New York, NY, USA, 2017. Association for Computing Machinery.

[207] Charles Lu and Jialin Ding. Rainbow Table Attack on 6-Digit PINs (GitHub). https://github.com/clu8/RainbowTable, 2015.

[208] ScooterHacking. ScooterHacking Firmware (GitHub). https://github.com/scooterhacking/firmware/, 2023.

[209] David J Wheeler and Roger M Needham. TEA, a tiny encryption algorithm. In *Fast Software Encryption: Second International Workshop Leuven, Belgium, December 14–16, 1994 Proceedings 2*, pages 363–366. Springer, 1995.

[210] NIST. FIPS 186-5 – Digital Signature Standard (DSS). https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5.pdf, 2023.

[211] Global Platform. Secure Channel Protocol 03. https://globalplatform.org/wp-content/uploads/2014/07/GPC_2.3_D_SCP03_v1.1.2_PublicRelease.pdf, 2019.

[212] NXP. CardLogix SCP03. https://www.cardlogix.com/glossary/scp03-secure-channel-protocol-3/, 2019.

[213] Qingyu Ma, Hong Yang, Alan Mayhue, Yunlong Sun, Zhitong Huang, and Yifang Ma. E-scooter safety: The riding risk analysis based on mobile sensing data. *Accident Analysis and Prevention*, 2021.

[214] Yan Cui, Jianghong Liu, Xin Han, Shaohua Sun, and Beihua Cong. Full-scale experimental study on suppressing lithium-ion battery pack fires from electric vehicles. *Fire Safety Journal*, page 103562, 2022.

[215] Binghe Liu, Yikai Jia, Chunhao Yuan, Lubing Wang, Xiang Gao, Sha Yin, and Jun Xu. Safety issues and mechanisms of lithium-ion battery cell upon mechanical abusive loading: A review. *Energy Storage Materials*, pages 85–112, 2020.

[216] Kyong-Tak Cho, Yuseung Kim, and Kang G Shin. Who killed my parked car? *arXiv preprint arXiv:1801.07741*, 2018.

[217] Marcell Szakály, Sebastian Köhler, Martin Strohmeier, and Ivan Martinovic. Assault and Battery: Evaluating the Security of Power Conversion Systems Against Electromagnetic Injection Attacks. *arXiv preprint arXiv:2305.06901*, 2023.

[218] Luis Garcia, Ferdinand Brasser, Mehmet Hazar Cintuglu, Ahmad-Reza Sadeghi, Osama A Mohammed, and Saman A Zonouz. Hey, My Malware Knows Physics! Attacking PLCs with Physical Model Aware Rootkit. In *NDSS*, pages 1–15, 2017.

[219] Thomas Roth, Fabian Freyer, Matthias Hollick, and Jiska Classen. Airtag of the clones: shenanigans with liberated item finders. In *2022 IEEE Security and Privacy Workshops (SPW)*, pages 301–311. IEEE, 2022.

[220] Ang Cui, Michael Costello, and Salvatore Stolfo. When firmware modifications attack: A case study of embedded exploitation. In *NDSS*, 2013.

[221] Jacob Maskiewicz, Benjamin Ellis, James Mouradian, and Hovav Shacham. Mouse trap: Exploiting firmware updates in USB peripherals. In *8th USENIX Workshop on Offensive Technologies (WOOT 14)*, 2014.

[222] Duha Ibdah, Nada Lachtar, Abdulrahman Abu Elkhail, Anys Bacha, and Hafiz Malik. Dark firmware: a systematic approach to exploring application security risks in the presence of untrusted firmware. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, pages 413–426, 2020.

[223] Nilo Redini, Aravind Machiry, Dipanjan Das, Yanick Fratantonio, Antonio Bianchi, Eric Gustafson, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. Bootstomp: on the security of bootloaders in mobile devices. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 781–798, 2017.

[224] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti. A Large-scale analysis of the security of embedded firmwares. In *23rd USENIX security symposium (USENIX Security 14)*, pages 95–110, 2014.

[225] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the Mirai botnet. In *26th USENIX Security Symposium (SEC)*, pages 1093–1110, 2017.

[226] PCmag. Network-Connected Torque Wrench Used in Factories Is Vulnerable to Ransomware. https://www.pcmag.com/news/network-connected-torque-wrench-used-in-factories-is-vulnerable-to-ransomware, 2024.

[227] Juan Lang, Alexei Czeskis, Dirk Balfanz, Marius Schilder, and Sampath Srinivas. Security keys: practical cryptographic second factors for the modern web. In *International Conference on Financial Cryptography and Data Security*, pages 422–440. Springer, 2016.

[228] Joseph Bonneau, Cormac Herley, Paul C Van Oorschot, and Frank Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *2012 IEEE symposium on security and privacy*, pages 553–567. IEEE, 2012.

[229] Transparency Market Research. FIDO Authentication Market Forecast. https://www.transparencymarketresearch.com/fido-authentication-market.html, 2023.

[230] Yubico. Q3 Interim Report. https://investors.yubico.com/en/wp-content/uploads/sites/2/2023/03/Q3-investor-morning-presentation-231110.pdf, 2023.

[231] FIDO Alliance. U.S. General Services Administration's Rollout of FIDO2 on login.gov. https://fidoalliance.org/u-s-general-services-administrations-rollout-of-fido2-on-login-gov/, 2023.

[232] Charlie Jacomme and Steve Kremer. An Extensive Formal Analysis of Multi-Factor Authentication Protocols. *ACM Transactions on Privacy and Security (TOPS)*, 24(2):1–34, 2021.

[233] Jingjing Guan, Hui Li, Haisong Ye, and Ziming Zhao. A Formal Analysis of the FIDO2 Protocols. In *European Symposium on Research in Computer Security (ESORICS)*, pages 3–21, 2022.

[234] Dhruv Kuchhal, Muhammad Saad, Adam Oest, and Frank Li. Evaluating the Security Posture of Real-World FIDO2 Deployments. In *Proceedings of the ACM conference on computer and communications security (CCS)*, 2023.

[235] Tarun Kumar Yadav and Kent Seamons. A Security and Usability Analysis of Local Attacks Against FIDO2. In *31th Annual Network & Distributed System Security Symposium (NDSS'24)*, 2024.

[236] FIDO Alliance. FIDO Security Reference, Review Draft 23 May 2022. https://fidoalliance.org/specs/common-specs/fido-security-ref-v2.1-ps-20220523.pdf, 2022.

[237] FIDO Alliance. FIDO Security Secretariat. https://fidoalliance.org/certification/secretariat/, 2024.

[238] Yubico. Security Advisory YSA-2024-02 FIDO Relying Party Enumeration. https://www.yubico.com/support/security-advisories/ysa-2024-02, 2024.

[239] Yubico. CVE-2024-35311. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2024-35311, 2024.

[240] FIDO Alliance. FIDO Alliance Specifications Overview. https://fidoalliance.org/specifications, 2024.

[241] W3C. Web Authentication: An API for accessing Public Key Credentials - Level 2. https://www.w3.org/TR/webauthn-2, 2021.

[242] FIDO Alliance. CTAP 2.1 Proposed Standard with Errata. https://fidoalliance.org/specs/fido-v2.1-ps-20210615/fido-client-to-authenticator-protocol-v2.1-ps-errata-20220621.html, 2022.

[243] FIDO Alliance. CTAP 2.2 Review Draft 01. https://fidoalliance.org/specs/fido-v2.2-rd-20230321/fido-client-to-authenticator-protocol-v2.2-rd-20230321.html, 2023.

[244] FIDO Alliance. FIDO Certified Products. https://fidoalliance.org/certification/fido-certified-products/, 2024.

[245] National Institute of Standards and U.S. Department of Commerce Technology. NIST Special Publication 800-63B, Digital Identity Guidelines, Authentication and Lifecycle Management. https://pages.nist.gov/800-63-3/sp800-63b.html#-5112-memorized-secret-verifiers, 2017.

[246] Feitian. Feitian Android App. https://play.google.com/store/apps/details?id=com.ft.entersafe.iepassmanager, 2022.

[247] Feitian. Feitian iOS App. https://apps.apple.com/us/app/iepassmanager/id1504200260, 2022.

[248] Yubico. Yubikey Manager CLI. https://github.com/Yubico/yubikey-manager, 2023.

[249] Ilan Kirschenbaum and Avishai Wool. How to Build a Low-Cost, Extended-Range RFID Skimmer. In *USENIX security symposium*, volume 4, 2006.

[250] Yuyi Sun, Swarun Kumar, Shibo He, Jiming Chen, and Zhiguo Shi. You Foot the Bill! Attacking NFC With Passive Relays. *IEEE Internet of Things Journal*, 8(2):1197–1210, 2021.

[251] Sajeda Akter, Sriram Chellappan, Tusher Chakraborty, Taslim Arefin Khan, Ashikur Rahman, and A. B. M. Alim Al Islam. Man-in-the-Middle Attack on Contactless Payment over NFC Communications: Design, Implementation, Experiments and Detection. *IEEE Transactions on Dependable and Secure Computing*, pages 3012–3023, 2021.

[252] Steffen Klee, Alexandros Roussos, Max Maass, and Matthias Hollick. NFCGate: Opening the Door for NFC Security Research with a Smartphone-Based Toolkit. In *14th USENIX Workshop on Offensive Technologies (WOOT 20)*, 2020.

[253] Thanh Bui, Siddharth Prakash Rao, Markku Antikainen, Viswanathan Manihatty Bojan, and Tuomas Aura. Man-in-the-Machine: Exploiting Ill-Secured Communication Inside the Computer. In *27th USENIX security symposium (USENIX Security 18)*, pages 1511–1525, 2018.

[254] Yubico. Yubico FIDO2 Python Library. https://github.com/Yubico/python-fido2, 2023.

[255] node-hid package developers. node-hid - Access USB HID devices from Node.js. https://www.npmjs.com/package/node-hid, 2023.

[256] Node.js developers. Node.js is an open-source, cross-platform JavaScript runtime environment. https://nodejs.org/en, 2024.

[257] CBOR package developers. CBOR - Encode and parse data in the CBOR data format. https://www.npmjs.com/package/cbor, 2023.

[258] Proxmark. Proxmark RFID Tool. https://proxmark.com, 2024.

[259] ISO. ISO/IEC 7816-4:2013. https://www.iso.org/standard/54550.html, 2013.

[260] Android developers. Android NFC basics. https://developer.android.com/develop/connectivity/nfc/nfc, 2023.

[261] z4yx. FIDO Wireshark protocol dissectors over USB HID. https://gist.github.com/z4yx/218116240e2759759b239d16fed787ca, 2019.

[262] Android. Near Field Communication (NFC) Overview. https://developer.android.com/develop/connectivity/nfc, 2024.

[263] Ziv Kfir and Avishai Wool. Picking Virtual Pockets using Relay Attacks on Contactless Smartcard. In *First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM'05)*, pages 47–58, 2005.

[264] FIDO Alliance. FIDO Functional Certification. https://fidoalliance.org/certification/functional-certification/, 2024.

[265] Tsai Chwei-Shyong, Lee Cheng-Chi, and Min-Shiang Hwang. Password Authentication Schemes: Current Status and Key Issues. *International Journal of Network Security*, 2006.

[266] Joseph Bonneau, Cormac Herley, Paul C. van Oorschot, and Frank Stajano. The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. In *2012 IEEE Symposium on Security and Privacy*, 2012.

[267] Manuel Barbosa, André Cirne, and Luís Esquível. Rogue key and impersonation attacks on FIDO2: From theory to practice. In *Proceedings of the 18th International Conference on Availability, Reliability and Security*. Association for Computing Machinery, 2023.

[268] Kexin Hu and Zhenfeng Zhang. Security analysis of an attractive online authentication standard: FIDO UAF protocol. *China Communications*, 13(12):189–198, 2016.

[269] Hui Li, Xuesong Pan, Xinluo Wang, Haonan Feng, and Chengjie Shi. Authenticator rebinding attack of the UAF protocol on mobile devices. *Wireless Communications and Mobile Computing*, 2020:1–14, 2020.

[270] Christiaan Brand. Advisory: Security Issue with Bluetooth Low Energy (BLE) Titan Security Keys. https://security.googleblog.com/2019/05/titan-keys-update.html, 2019.

[271] Mohammed Jubur, Prakash Shrestha, Nitesh Saxena, and Jay Prakash. Bypassing push-based second factor and passwordless authentication with human-indistinguishable notifications. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, pages 447–461, 2021.

[272] Enis Ulqinaku, Hala Assal, AbdelRahman Abdou, Sonia Chiasson, and Srdjan Capkun. Is Real-time Phishing Eliminated with FIDO? Social Engineering Downgrade Attacks against FIDO Protocols. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 3811–3828, 2021.

[273] Haonan Feng, Hui Li, Xuesong Pan, Ziming Zhao, and T Cactilab. A Formal Analysis of the FIDO UAF Protocol. In *Network & Distributed System Security Symposium (NDSS'21)*, 2021.

[274] Nina Bindel, Cas Cremers, and Mang Zhao. FIDO2, CTAP 2.1, and WebAuthn 2: Provable security and post-quantum instantiation. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1471–1490. IEEE, 2023.

[275] Fabian Schwarz, Khue Do, Gunnar Heide, Lucjan Hanzlik, and Christian Rossow. FeIDo: Recoverable FIDO2 Tokens Using Electronic IDs. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2581–2594, 2022.

[276] Andre Büttner and Nils Gruschka. Protecting FIDO Extensions Against Man-in-the-Middle Attacks. In *International Workshop on Emerging Technologies for Authorization and Authentication*, pages 70–87. Springer, 2022.

[277] Diana Ghinea, Fabian Kaczmarczyck, Jennifer Pullman, Julien Cretin, Rafael Misoczki, Stefan Kölbl, Luca Invernizzi, Elie Bursztein, and Jean-Michel Picod. Hybrid post-quantum signatures in hardware security keys. In *4th ACNS Workshop on Secure Crytographic Implmentation*, 2023.

[278] Google. OpenSK: a Rust Implementation of a FIDO2 Authenticator. https://github.com/google/OpenSK, 2024.

[279] Lucjan Hanzlik, Julian Loss, and Benedikt Wagner. Token meets Wallet: Formalizing Privacy and Revocation for FIDO2. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1491–1508, 2023.

[280] Jessica Colnago, Summer Devlin, Maggie Oates, Chelse Swoopes, Lujo Bauer, Lorrie Cranor, and Nicolas Christin. "It's not actually that horrible" Exploring Adoption of Two-Factor Authentication at a University. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–11, 2018.

[281] Sanchari Das, Andrew Dingman, and L Jean Camp. Why Johnny doesn't use two factor a two-phase usability study of the FIDO U2F security key. In *International Conference on Financial Cryptography and Data Security*, pages 160–179. Springer, 2018.

[282] Stéphane Ciolino, Simon Parkin, and Paul Dunphy. Of Two Minds about Two-Factor: Understanding Everyday FIDO/U2F Usability through Device Comparison and Experience Sampling. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, pages 339–356, 2019.

[283] Leona Lassak, Annika Hildebrandt, Maximilian Golla, and Blase Ur. It's Stored, Hopefully, on an Encrypted Server: Mitigating Users' Misconceptions About FIDO2 Biometric WebAuthn. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 91–108, 2021.

[284] Florian M Farke, Lennart Lorenz, Theodor Schnitzler, Philipp Markert, and Markus Dürmuth. You still use the password after all–Exploring FIDO2 Security Keys in a Small Company. In *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*, pages 19–35, 2020.

[285] Kentrell Owens, Olabode Anise, Amanda Krauss, and Blase Ur. User Perceptions of the Usability and Security of Smartphones as FIDO2 Roaming Authenticators. In *Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021)*, pages 57–76, 2021.

[286] Michal Kepkowski, Maciej Machulak, Ian Wood, and Dali Kaafar. Challenges with Passwordless FIDO2 in an Enterprise Setting: A Usability Study. *arXiv preprint arXiv:2308.08096*, 2023.

[287] Sanam Ghorbani Lyastani, Michael Backes, and Sven Bugiel. A systematic study of the consistency of two-factor authentication user journeys on top-ranked websites. In *30th Annual Network & Distributed System Security Symposium (NDSS'23)*, 2023.

[288] Anna Angelogianni, Ilias Politis, and Christos Xenakis. How many FIDO protocols are needed? Surveying the design, security and market perspectives. *arXiv preprint arXiv:2107.00577*, 2021.

[289] Sanam Ghorbani Lyastani, Michael Schilling, Michaela Neumayr, Michael Backes, and Sven Bugiel. Is FIDO2 the Kingslayer of User Authentication? A Comparative Usability Study of FIDO2 Passwordless Authentication. In *IEEE Symposium on Security and Privacy*, pages 268–285, 2020.

[290] Mike Hanley (GitHub CSO). Securing millions of developers through 2FA. https://github.blog/2024-04-24-securing-millions-of-developers-through-2fa/, 2024.

[291] Google Safety and Security. The beginning of the end of the password. https://blog.google/technology/safety-security/the-beginning-of-the-end-of-the-password/, 2023.