ORSHIN

D5.1

Report about Essential and Beyond Essential S&P Guarantees for Inter-device Communication in Restricted Environments

Project number:	101070008
Project acronym:	**ORSHIN**
Project title:	Open-source ReSilient Hardware and software for Internet of thiNgs
Project Start Date:	1st October, 2022
Duration:	36 months
Programme:	HORIZON-CL3-2021-CS-01
Deliverable Type:	Report
Reference Number:	CL3-2021-CS-01 / DD5.1 / 1.0
Workpackage:	WP5
Due Date:	Jun 2025 – M33
Actual Submission Date:	30th June 2025
Responsible Organisation:	ECM
Editor:	Daniele Antonioli
Dissemination Level:	PU
Revision:	1.0
Abstract:	**D5.1** addresses **WP5 T5.1**, **T5.2** and results in *nine research papers*. The contributions in the first four Chapters informed the creation of *BlueBrothers*: three new inter-device communication protocols providing essential and beyond-essential S&P guarantees, presented in Chapter 4.
Keywords:	IoT, IIoT, Constrained, ORSHIN, Confidentiality, Integrity, Authenticity, Protocols, Forward Secrecy, Future Secrecy, Bluetooth, FIDO2

Funded by the European Union under grant agreement no. 101070008. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

Editor

Daniele Antonioli (ECM)

Contributors (ordered according to beneficiary numbers)

Tommaso Sacchetti (ECM)
Marco Casagrande (ECM)

Reviewers

Volodymyr Bezsmertnyi (NXP)
Alberto Battistello (SEC)

Disclaimer

Table of Content

List of Figures

List of Tables

Listings

Executive Summary

Deliverable 5.1 (**D5.1**) focuses on *essential and beyond essential security and privacy (S&P) guarantees for inter-device communication in restricted environments*. It is part of ORSHIN **WP5** and tackles two tasks related to the communications of constrained devices using open-source hardware (OSH):

T5.1: develop inter-device protocols for constrained OSH devices providing *essential* S&P guarantees. These guarantees include confidentiality, integrity, authenticity, and anonymity.

T5.2: develop inter-device protocols for constrained OSH devices assuring *beyond-essential* S&P guarantees, like forward and future secrecy and post-quantum key agreement.

D5.1 successfully addresses **T5.1** and **T5.2** and the related ORSHIN WP5 objectives with novel and practical contributions. Its first four Chapters focus on works where we researched state of the art communication technologies and assessed if and how they provide essential and beyond-essential S&P guarantees. We target pervasive technologies like Bluetooth, NFC, and FIDO2. We discover new vulnerabilities and attacks and develop novel toolkits to reproduce them and effective countermeasures. Thanks to our contributions billions of connected devices now provide stronger security and privacy guarantees. Then, we take advantage of the findings of the four Chapter to design three next-generation security and privacy protocols for Bluetooth. The protocols provide essential and beyond-essential S&P properties and can be shipped to protect the entire Bluetooth ecosystem. Next, we summarize the technical content of each Chapter.

In Chapter 1 we present a security evaluation of pervasive and proprietary wireless communication protocols used by Xiaomi for its e-scooters (**T5.1**). We uncover that Xiaomi protocols fail to provide even basic S&P guarantees as they rely on security through obscurity or ad-hoc but vulnerable mechanisms, including obfuscation and unauthenticated key agreement.

In Chapter 2 we discuss forward and future secrecy guarantees of Bluetooth, a pervasive protocol used by IoT devices, including OSH ones (**T5.2**). We find two new vulnerabilities in the Bluetooth specification related to improper session key derivation. We develop six attacks exploiting these vulnerabilities to impersonate and machine-in-the-middle (MitM) any Bluetooth device, regardless of its hardware and software details. We fix the attacks with an enhanced session key derivation protocol.

In Chapter 3 we analyze FIDO2 (Fast IDentity Online v2), a popular authentication protocol for single-factor and multi-factor authentication. FIDO2 involves constrained IoT devices, like USB dongles used to authenticate. We check if FIDO2 provides the essential and beyond-essential security properties promised in it specification (**T5.1**, **T5.2**), including authenticity, and availability. We uncover two new classes of attacks for FIDO2 we call client impersonation (CI) and API confusion (AC). The CI and AC attacks can be used to delete FIDO2 credentials or track users with FIDO2 credentials. We successfully evaluate the attacks on six popular FIDO2 authenticators including ones from Yubico, Feitian, Solokey, and Google and propose effective countermeasures.

In Chapter 4 we propose three new protocols for inter-device communication providing essential and beyond-essential security and privacy guarantees (**T5.1**, **T5.2**). We design, implement, and evaluate them for Bluetooth, to showcase that they can be deployed on a pervasive IoT technology used by billions of devices, including constrained and OSH ones.

As shown Chapter 5, **D5.1** results in *nine open-access research papers and related research artifacts*. The contributions in the first four Chapters informed the creation of *BlueBrothers*: three new inter-device communication protocols providing essential and beyond-essential S&P guarantees. The BlueBrothers protocols are described in Chapter 4 and are used in the ORSHIN demonstrator to secure Bluetooth communication in constrained environments. The full list of **D5.1** contributions is in the Contribution section in the Conclusion.

List of Abbreviations

Abbr.	Meaning
A2DP	Advanced Audio Distribution Profile
AC	API Confusion or Authentication Challenge
AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
API	Application Programming Interface
ARM	Advanced RISC Machines
AS	Association
BA	Bluetooth Address
BC	Bluetooth Classic
BIAS	Bluetooth Impersonation Attacks
BLE	Bluetooth Low Energy
BLUFFS	Bluetooth Forward and Future Secrecy
BMS	Battery Management System
BPF	Berkeley Packet Filter
CCM	Counter with Cipher block chaining Message auth code
CI	Client Impersonation
CLI	Command Line Interface
COF	Ciphering Offset Number
CR	Challenge Response
CTAP	Client to Authenticator Protocol
CTAP CM	CTAP CredMgmt
CTAP CP	CTAP ClientPin
CTAP GA	CTAP GetAssertion
CTAP GI	CTAP GetInfo
CTAP MC	CTAP MakeCred
CTAP Re	CTAP Reset
CTAP Se	CTAP Selection
CTAP UP	CTAP User Presence
CTAP UV	CTAP User Verification
CTKD	Cross Transport Key Derivation
CVE	Common Vulnerabilities and Exposures
CredId	Credential Identifier
DES	Data Encryption Standard
DH	Diffie-Hellman
DRV	Electric Motor Driver
DoS	Denial of Service
ECB	Electronic Code Book

ECDH	Elliptic-curve Diffie–Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
EKD	Extended Key Derivation
FIDO	Fast IDentity Online
FIPS	Federal Information Processing Standards
FN	Feature Negotiation
GAP	Generic Access Profile
GATT	Generic Attribute Profile
GDB	GNU Debugger
HCI	Host Controller Interface
HID	Human Interface Device
HMAC	Hash-based Message Authentication Code
HOTP	HMAC-based one-time password
IDS	Intrusion Detection System
IM	Instant Messenger
ISK	Intermediate Session Key
ISM	Industrial Scientific Medical
JTAG	Joint Test Action Group
JW	JustWorks
KD	Key Derivation
KDF	Key Derivation Function
KNOB	Key Negotiation Of Bluetooth
L2CAP	Logical Link Control and Adaptation Protocol
LM	Link Manager
LMP	Link Manager Protocol
LSC	Legacy Secure Connections
MAC	Message Authentication Code or Media Access Protocol
MP	Malicious Pairing
MSC	Message Sequence Chart
MitM	Machine-in-the-Middle
NC	Numeric Comparison
NFC	Near Field Communication
OOB	Out of Band
OS	Operating System
OSH	Open-Source Hardware
OSI	Open Systems Interconnections
PE	Passkey Entry
PK	Pairing Key
POC	Proof of Concept
PQ	Post-Quantum
PQDH	Post-Quantum Diffie-Hellman
QR Code	quick-response code
RAM	Random access memory
RC	Root Cause
RCE	Remote Code Execution
RE	Reverse Engineering
RFCOMM	Radio Frequency Communication
RM	Rekey Manager

ROM	Read-only memory
RTT	Round Trip Time
RpId	Relying Party Identifier
SC	Secure Connections
SCO	Secure Connections Only
SD	Session Diversifier or Session Downgrade
SE	Session Entropy
SIG	Special Interest Group
SK	Session Key
SMP	Security Manager Protocol
SS	Shared Secret
SSP	Secure Simple Pairing
STO	Security Through Obscurity
SWD	Serial Wire Debug
S&P	Security and Privacy
SoC	System-on-Chip
TEE	Trusted Execution Environment
TLS	Transport Layer Security
TOFU	Trust on First Use
UART	Universal Asynchronous Receiver-Transmitter
US	United States
USB	Universal Serial Bus
UserId	User Identifier
WebAuthn	Web Authentication

Introduction

Our society depends on *connected devices* to provide essential and even safety-critical services including telecommunication, energy distribution, smart mobility, e-commerce, and so on. A device is a collection of software, like an operating system, and hardware, such as a central processing unit, that are working together towards a common goal. Billions of devices are daily connected with many communication protocols, including wireless ones like Bluetooth and wired ones like USB. Connected devices can operate in a *restricted* environment, including scenarios without the presence of a trusted authority, like a public key infrastructure, or with limited computational and memory capabilities, such as battery-powered IoT devices.

Connected devices, including smartphones, smart wearables, and smart vehicles, manage sensitive and safety-critical data and should provide *security and privacy (S&P) guarantees*. For example, data confidentiality contributes to security and privacy as it allows the secure exchange of data even in an attacker's presence and protects sensitive data from unauthorized access. S&P breaches result in large scale and critical threats to connected devices, including loss of sensitive data, user tracking, and command injection. Our goal with the ORSHIN project is to create methodologies and tools to build secure and privacy-preserving devices taking advantage of *open-source hardware (OSH)* and software (OSS).

Deliverable 5.1 (**D5.1**) contributes to ORSHIN's **WP5 (Secure Auth and Comms)** and focuses on *S&P guarantees for inter-device communication in restricted environments*. Inter-device communication includes wireless protocols like Bluetooth, Wi-Fi, and NFC. A restricted environment comprises mobile, IoT, and OT devices such as smartphones, wearables, industrial controllers, and (electric) vehicles. These devices can use OSS and OSH components such as a Linux OS and a RISC-V CPU. The deliverable has two tasks:

T5.1: develop inter-device protocols for constrained OSH devices providing *essential* S&P guarantees. These guarantees include confidentiality, integrity, authenticity, and anonymity.

T5.2: develop inter-device protocols for constrained OSH devices assuring *beyond-essential* S&P guarantees, like forward and future secrecy and post-quantum key agreement.

D5.1 successfully addresses **T5.1** and **T5.2** and the related **WP5** objectives with novel and practical contributions. Its first four Chapters focus on works where we researched state of the art communication technologies and assessed if and how they provide essential and beyond-essential S&P guarantees. We target pervasive technologies like Bluetooth, NFC, and FIDO2. We discover new vulnerabilities and attacks and develop novel toolkits to reproduce them and effective countermeasures. Thanks to our contributions billions of connected devices now provide stronger security and privacy guarantees.

We pivot on the findings of Chapters 1 to 4 to design three next-generation security and privacy protocols for Bluetooth, we call BlueBrothers. The protocols provide essential and beyond-essential S&P properties, including confidentiality, authenticity, post-quantum key agreement, and

forward and future secrecy. They can be employed to protect the entire Bluetooth ecosystem either at the application layer or below. The protocols are secure by design, simple to understand and implement, and open. We formally model and verify them and provide a proof of concept implementation to the research community. We experimentally evaluate them in constrained scenarios and observe minimal latency and power overheads or even speedups compared to their counterparts in the Bluetooth standard.

Next, we summarize the technical content of each Chapter. In Chapter 1 we present a security evaluation of pervasive and proprietary wireless communication protocols used by Xiaomi for its e-scooters (**T5.1**). We focus on Xiaomi because is the market leading with millions of e-scooters sold and used every day worldwide. We uncover that Xiaomi protocols fail to provide even basic S&P guarantees as they rely on security through obscurity or ad-hoc but vulnerable mechanisms, including obfuscation and unauthenticated key agreement.

In Chapter 2 we discuss forward and future secrecy guarantees of Bluetooth, a pervasive protocol used by IoT devices, including OSH ones (**T5.2**). We find two new vulnerabilities in the Bluetooth specification related to improper session key derivation. We develop six attacks exploiting these vulnerabilities to impersonate and machine-in-the-middle (MitM) any Bluetooth device, regardless of its hardware and software details. We fix the attacks with an enhanced session key derivation protocol.

In Chapter 3 we analyze FIDO2 (Fast IDentity Online v2), a popular authentication protocol for single-factor and multi-factor authentication. FIDO2 involves constrained IoT devices, like USB dongles used to authenticate. We check if FIDO2 provides the essential and beyond-essential security properties promised in it specification (**T5.1**, **T5.2**), including authenticity, and availability. We uncover two new classes of attacks for FIDO2 we call client impersonation (CI) and API confusion (AC). The attacks focus on CTAP (Client to Authenticator Protocol) protocol used by a FIDO2 authenticator and client. We isolate the attack root causes affecting the design of CTAP and develop a toolkit to test the attacks. We successfully evaluate the attacks on six popular FIDO2 authenticators including ones from Yubico, Feitian, Solokey, and Google.

In Chapter 4 we propose three new protocols for inter-device communication providing essential and beyond-essential security and privacy guarantees (**T5.1**, **T5.2**). We design, implement, and evaluate them for Bluetooth, to showcase that they can be deployed on a pervasive IoT technology used by billions of devices, including constrained and OSH ones. The protocols guarantees confidentiality, integrity, and authenticity using state-of-the-art cryptographic protocols and primitives. Moreover, they provide forward and future secrecy within and across Bluetooth session and hybrid post-quantum key agreement.

In Chapter 5, we summarize the results of **D5.1** and its contributions.

Extensions and changes compared to iD5.1

- Rewritten and extended the Executive Summary
- Rewritten and extended the Introduction
- Added Chapter 3 (CTRAPS)
- Rewrote Chapter 4 (It was called BLESS, now it is called BlueBrothers)
- Rewritten and extended the Conclusion, including a list of our 9 research contributions
- Grammar pass of the whole document

- Updated list of abbreviations
- Extended the bibliography

Chapter 1

E-Spoofer: Attacking and Defending Xiaomi Electric Scooter Ecosystem

1.1 Abstract

Xiaomi is the market leader in the electric scooter (e-scooter) segment, with millions of active users. It provides several e-scooter models and Mi Home, a mobile application for Android and iOS to manage and control an e-scooter. Mi Home and the e-scooter interact via Bluetooth Low Energy (BLE). No prior research evaluated the security of this communication channel, as it employs security protocols proprietary to Xiaomi. Exploiting these protocols results in severe security, privacy, and safety issues, e.g., an attacker could steal an e-scooter or prevent the owner from controlling it. In this work, we fill this research gap by performing the first security evaluation on all proprietary wireless protocols deployed to Xiaomi e-scooters from 2016 to 2021. We identify and reverse-engineer *four* of them, each having ad-hoc Pairing and Session phases. We develop *six* attacks exploiting these protocols at the architectural level, and we call them Malicious Pairing (MP) and Session Downgrade (SD). Both attacks can be performed from proximity, if the attacker's machine is within BLE range of the target e-scooter, or remotely, via a malicious application co-located with Mi Home.

An adversary can utilize MP and SD to steal a password-protected and software-locked e-scooter, or to prevent a victim from accessing it via Mi Home. We isolate *six* attack root causes, including the lack of authentication while pairing, and the improper enforcement of the e-scooter password. We open-source the *E-Spoofer* toolkit. Our toolkit automates the MP and SD attacks, and includes a reverse-engineering module for future research. We empirically confirm the effectiveness of our attacks by exploiting eighteen e-scooters (i.e., M365, Essential, and Mi 3), embedding seventeen BLE subsystem boards and eight BLE firmware versions that support all four Xiaomi protocols. We design and evaluate *two* practical countermeasures that address our impactful attacks and their root causes, and we release them as part of BLUFFS. We responsibly disclosed our findings to Xiaomi.

1.2 Introduction

Xiaomi is leading the electric scooter (e-scooter) market [110]. 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 [24] and iOS [25]. 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) [195] and on the security of Xiaomi's fitness tracking ecosystem [51]. 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 [89] 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 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 [211]. 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.

1.3 Xiaomi E-Scooter Ecosystem

Xiaomi is the electric scooter (e-scooter) market leader, sporting the highest number of active users and shipped devices [110]. 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 [24] and iOS [25] that manages Xiaomi's smart home devices, including any e-scooter. Xiaomi's cloud-based backend service manages the e-scooters and their active Mi Home users.

Figure 1.1 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.

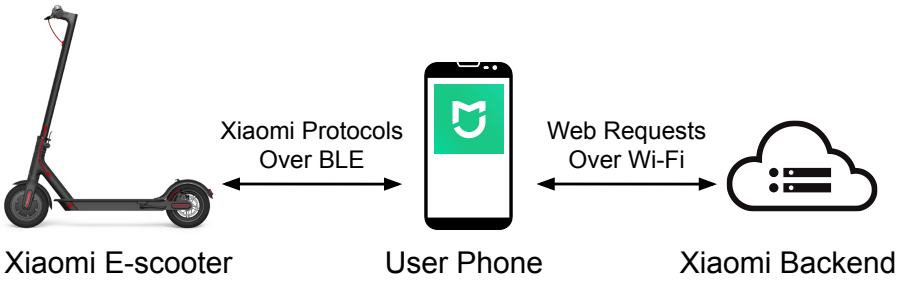

Figure 1.1: Xiaomi e-scooter ecosystem. Xiaomi e-scooter (left), the user smartphone running the Mi Home app (middle), and the Xiaomi backend (right). The e-scooter and the app are paired and connected over BLE. The app associates the e-scooter with the Xiaomi backend over Wi-Fi. We focus on the BLE traffic between the app and the e-scooter.

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 [158].

1.4 Threat Model

Now we present our system model and our proximity-based and remote attacker models. Please refer to Section 1.3 for their related background material.

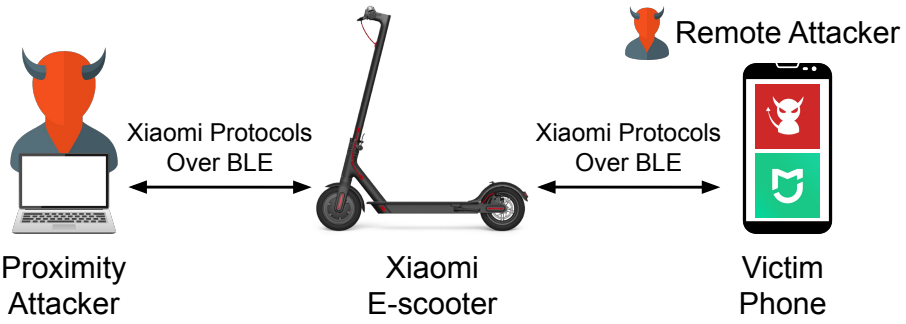

Figure 1.2: Proximity-based (left) and remote (right) attacker models investigated in this work. In the proximity-based threat model, the attacker is within BLE range of a target e-scooter. In the remote threat model, the adversary first installs a malicious Android app on the victim's smartphone. Then, she uses the malicious app (in red) to remotely target an e-scooter within BLE range of the victim's smartphone.

1.4.1 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 1.1. 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* knows the application-layer protocols' details and their security guarantees (e.g., confidentiality, integrity, and authenticity).

1.4.2 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 1.2, 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.

1.5 Reversed Xiaomi Security Protocols

We describe the *four* proprietary Xiaomi protocols that we reverse-engineered (RE). Please see the Appendix for an explanation of our RE methodology. 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 1.1 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*

Table 1.1: 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	Session
P1	No security	None	None
P2	XOR obfuscation	Public XOR mask, no auth	XOR mask obfuscation, no auth, no integrity
P3	AES-ECB and XOR obf	Weak AES-ECB key agr, no auth	XOR obfuscation, implicit auth, no integrity
P4	ECDH and AES-CCM	ECDH, AES-CCM unil auth	v1: HKDF, HMAC, AES-CCM, mutual auth v2: v1 with downgrade protection

(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.

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

1.5.2 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. The mask can be read without requiring authorization before pairing. 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.

1.5.3 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.

1.5.4 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)`. The e-scooter verifies the challenge and if the verification is successful pairing is completed.

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.

1.6 Attacks

We present *four novel attacks* targeting the four Xiaomi custom protocols discussed in Section 1.5 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 1.4. 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)*.

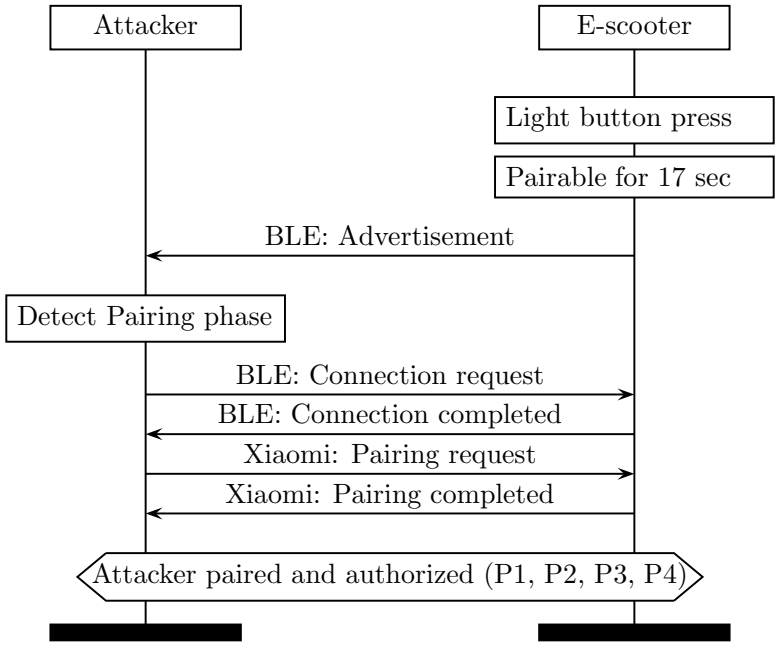

Figure 1.3: Malicious Pairing (MP) attack strategy. The user presses the headlight button. The e-scooter goes into pairable mode for seventeen seconds and advertises it via BLE. The attacker detects the Pairing phase supported by the e-scooter. Then, she establishes a BLE connection without impersonating 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.

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.

1.6.1 Malicious Pairing (MP)

Figure 1.3 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

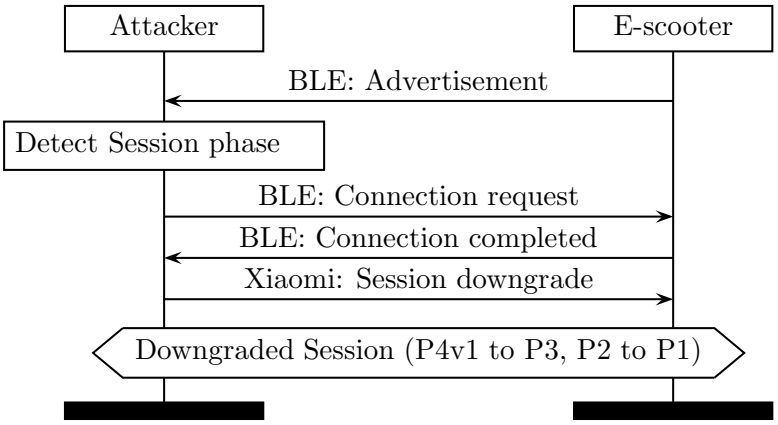

Figure 1.4: Session Downgrade (SD) attack strategy. The app detects a nearby e-scooter vulnerable to SD (i.e., running P4v1 or P2). The attacker skips Pairing and sends the session downgrade command to the e-scooter. The Session is downgraded from P4v1 to P3, or from P2 to P1.

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.

1.6.2 Session Downgrade (SD)

Figure 1.4 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.

1.6.3 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).

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 1.2 in the Appendix 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.

Table 1.2: Mapping between the vulnerabilities (rows) and the presented attacks (columns). We put a ✓ if an attack exploits a vulnerability. 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	✗	✓	✗	✓

1.7 Implementation

Here, we present `E-Spoofer`, a new toolkit to carry out the four attacks presented in Section 1.6, facilitate further reverse-engineering of Xiaomi protocols and help in future security evaluations in the Xiaomi e-scooter ecosystem.

1.7.1 Proximity Attack Module

The `E-Spoofer` proximity attack module performs proximity-based MP and SD *over-the-air*, using BLE. We use Noble [140], 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 back. 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.

1.7.2 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 vulner-

able 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 [178], 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 ACC ESS_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 BL UETOOTH_SCAN permissions.

1.7.3 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* [146] to statically RE portions of the e-scooter's BLE firmware. We used the open-source mijia library [138] 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 [196] 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 1.6.3.

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 [157] 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.

1.8 Evaluation

In this section, we evaluate the four attacks presented in Section 1.6 against *eight* attack scenarios. We cover P1, P2, P3, and P4 – the proprietary Xiaomi application-layer protocols reversed in Section 1.5, 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.

1.8.1 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 [174] 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 [166] 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* [76], 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 [148], PySWD [159], MiDu Flasher [90], and nRFSec [42]. 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 1.7, 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.

1.8.2 Results

Table 1.3 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 1.9. 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 1.7.

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. In the Appendix, we present our reverse-engineering methodology, which is generalizable to any BLE e-scooter and utilizes state-of-the-art tools and techniques. 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 1.8.1. Since the password is only checked by Mi Home, due to the `improper e-scooter password enforcement` (V3) root cause we discuss in Section 1.6.3, 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.

1.9 Countermeasures

To address the four impactful attacks described in Section 1.6, we design and evaluate two *usable*, *backward-compliant*, and *low-cost* countermeasures. The first countermeasure stops the 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

Table 1.3: 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	**Prot**	**E-Sco**	**BLE Board**	**SoC**	**MP**	**SD**	**MP**	**SD**
BLE072	P1	M365	M365 (Orig)	nRF51822 QFAA	✓	-	✓	-
BLE081	P2	M365	M365 (Orig)	nRF51822 QFAA	✓	✓	✓	✓
BLE090	P3	M365	Pro 1 (Clone)	nRF51822 QFAA	✓	✗	✓	✗
BLE122	P4v1	M365	M365 (Orig)	nRF51822 QFAA	✓	✓	✓	✓
BLE129	P4v1	M365	Pro 2 (Clone)	nRF51822 QFAC	✓	✓	✓	✓
BLE152	P4v1	Ess.	Essential (Orig)	nRF51822 QFAC	✓	✓	✓	✓
BLE153	P4v1	Mi 3	Mi 3 (Orig)	nRF51822 QFAC	✓	✓	✓	✓
BLE157	P4v2	Mi 3	Mi 3 (Orig)	nRF51822 QFAC	✓	✗	✓	✗

downgrade command from the vulnerable e-scooter BLE firmware. We now describe them in detail and release them as part of `E-Spoofer`.

1.9.1 Authorized and Authenticated Pairing

The MP attacks presented in Section 1.6.1 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 1.6.3. 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 fac-

tory 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.

1.9.2 Anti-Downgrade BLE Firmware Patching

The SD attacks presented in Section 1.6.2 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 1.6.3.

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 [39] to encrypt the patched firmware with the Tiny Encryption Algorithm (TEA). We reuse the third-party M365DownG app [47] 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.

1.10 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 [221]. The hacker Lanrat evaluated M365 authentication, discovering that it is not enforced by the e-scooter [88]. 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 [8], the authors identify some vulnerabilities in the APIs exposed by the Bird e-scooter sharing platform, which utilizes M356 e-scooters [26]. Public e-scooters from the Lime sharing company are weak to a man-in-the-middle attack that allows for arbitrarily swapping audio files [144]. N. Vinayaga-Sureshkanth et al. [195] 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 [169] is the largest e-scooter hacking community with around 20.000 members. ScooterHacking releases hacking tools [166] and offers a third-party companion app [168] 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 [165] and the BotoX Xiaomi M365 Firmware Patcher [39]. 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 [46, 69, 142, 167]. 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 [67], the authors root a Xiaomi vacuum cleaning robot, inspect its internals, assess data privacy, and flash the robot with custom firmware. Another previous work [190] also finds several security issues with Xiaomi vacuum cleaners.

Several researchers [51, 200, 82, 103] 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. [73] find 95 vulnerabilities in the R1D Xiaomi router. Other Xiaomi IoT devices evaluated in the academic literature are Xiaomi smart speakers [128] and Xiaomi security cameras [194, 187].

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 [143, 182], allowing malicious apps to communicate illegitimately with paired devices. In [186], 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 [220]. 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 [162] broke the Just Works and Passkey modes of BLE Legacy pairing by brute-forcing their temporary key. In 2019, the KNOB [17] 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 [197] 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.

1.11 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). These 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.

RE Methodology

We present the RE methodology that we employed to reconstruct the protocols described In Section 1.5. This methodology can be *reused* by other researchers to tackle similar RE efforts (e.g., reversing a proprietary and unknown application-layer protocol implemented on top of an insecure BLE link-layer). Specifically, we explain how we analyzed the Xiaomi and BLE traffic, the Mi Home app, and the scooter's BLE firmware.

Xiaomi and BLE Traffic

The BLE e-scooter exposes a GATT server with unknown characteristics used to exchange the proprietary P1, P2, P3, P4 payloads. We enumerate these characteristics with a GATT client program (e.g., `gatttool`). The e-scooter utilizes a Xiaomi custom GATT service (`0xFE95`), and its UPNP (`0x0010`) and AVDTP (`0x0019`) characteristics for Pairing and Session, and the Nordic UART service for the encrypted communication during Session. Then, we run multiple Pairing and Session phases using different combinations of e-scooter models, BLE subsystems, and firmware. We capture the generated BLE traffic (HCI-layer, including GATT) in dedicated pcap files. The pcap files are produced by our Android smartphone running Mi Home with Developer Options, and HCI Snooplog turned on. We open the pcaps in Wireshark with custom display filters to focus only on the proprietary application-layer payloads. We use Pyshark and Scapy to reverse the Xiaomi payloads and develop custom dissection classes capable of decoding and re-encoding the packets. For example, we developed the `PairChal` and `SessRand` classes to dissect P4 Pairing and Session security mechanisms. Table tab:appendix-opcodes lists BLe packets from P3 and P4 Pairing and Session phases.

Table 1.4: BLE packets for the Xiaomi protocol P3 and P4. The first and second columns indicate the name we assigned to the packet and the protocol which uses them. The third and fourth columns specify who sends the packet and its content. The value ”00X0” stands for an increasing counter (i.e., 0010, 0020, 0030, 0040) placed in a fragmented packet.

Packet	P	From	Content
SessReq	P3	App	`5AA5 0E 3D21 5D00 Serial`
SessReqOk	P3	Esc	`5AA5 0E 213D 5D00 Serial`
Comms	P3	App, Esc	`5AA5 Len From To Cmd Pay`
PairReq	P4	App	`A2000000`
PairReqOk	P4	Esc	`000000000200`
PairChal1	P4	Esc	`0010 01000000 Chal2Part`
PairChal2	P4	Esc	`0020 Chal2Part`
PairECStart	P4	App, Esc	`000000030400`
PairPubKey	P4	App, Esc	`00X0 PubKey4Part`
PairECEnd	P4	App	`000000000200`
PairSol	P4	App	`00X0 PairSol2Part`
PairSolAck	P4	Esc	`00000100`
PairOk	P4	App	`13000000`
PairOkAck	P4	Esc	`11000000`
SessReq1	P4	App	`24000000`
SessReq2	P4	App	`0000000B0100`
SessRand	P4	App, Esc	`0100 AuthChal`
SessAskRand	P4	Esc	`0000000C0200`
SessSol	P4	App, Esc	`00X0 AuthSol2Part`
SessOk	P4	Esc	`21000000`
Comms	P4	App, Esc	`55AB Len Count Encr Cksm`

Mi Home for Android

For Mi Home, we focused on its Android version because it is much easier to inspect and reverse than its iOS counterpart. We locate the Mi Home APK with `adbshellpmpathcom.xiaomi.smarthome`. We pulled It with `adb pull` and extracted its content, including the decompiled Smali and Java code, with `apktool` and `jadx`. We perform static analysis of the decompiled Java code, looking for API calls to cryptographic primitives and Android's BLE framework. In parallel, we use Frida and Objection for dynamic binary instrumentation of Mi Home. Frida requires a rooted phone and a Frida server application running on the phone. With our dynamic tests, we can list all the Classes involved with Mi Home, isolate the ones related to P1, P2, P3, and P4, hook them to observe their runtime execution, and reimplement some of their behaviors. For example, we found that Mi Home, during Pairing, calls `_m_j.fyp.OOOOOOOo` to perform ECDH, and, during Session, calls `_m_j.fys.OOOOOOOo` to perform the HMAC-based authentication and `_m_j.fyl.OOOOOOOo` to encrypt communications.

E-Scooter BLE Firmware

Reversing the e-scooter BLE firmware is challenging, as is a stripped binary with no debugging symbols. We obtain multiple firmware samples from different sources, i.e., ScooterHacking repositories, the Mi Home app storage, the Xiaomi backend, and by reading the BLE SoC memory at runtime via the ST-Link debugger. We statically analyze the firmware using Ghidra configured for ARM Cortex M0 little-endian. We also configure the Ghidra memory layout using the nRF51822 Product Specification 3.4 [173] from Nordic Semiconductors. In particular, we consult the instruction table to retrieve the memory addresses for instantiating the peripherals such as the FICR_-UICR, POWER, CLOCK, and GPIO. We also use the ST-Link debugger to inspect the firmware at runtime using gdb, dumping its memory and flashing it with different BLE firmware versions.

Chapter 2

BLUFFS: Bluetooth Forward and Future Secrecy Attacks and Defenses

2.1 Abstract

Bluetooth is a pervasive technology for wireless communication. Billions of devices use it in sensitive applications and to exchange private data. The security of Bluetooth depends on the Bluetooth standard and its two security mechanisms: pairing and session establishment. No prior work, including the standard itself, analyzed the *future and forward secrecy* guarantees of these mechanisms, e.g., if Bluetooth pairing and session establishment defend past and future sessions when the adversary compromises the current. To address this gap, we present *six* novel attacks, defined as the *BLUFFS attacks*, breaking Bluetooth sessions' forward and future secrecy. Our attacks enable device impersonation and machine-in-the-middle *across* sessions by only compromising *one* session key. The attacks exploit *two* novel vulnerabilities that we uncover in the Bluetooth standard related to unilateral and repeatable session key derivation. As the attacks affect Bluetooth at the architectural level, they are effective regardless of the victim's hardware and software details (e.g., chip, stack, version, and security mode).

We also release `BLUFFS`, a low-cost toolkit to perform and automatically check the effectiveness of our attacks. The toolkit employs *seven* original patches to manipulate and monitor Bluetooth session key derivation by dynamically patching a closed-source Bluetooth firmware that we reverse-engineered. We show that our attacks have a *critical* and *large-scale* impact on the Bluetooth ecosystem, by evaluating them on *seventeen* diverse Bluetooth chips (*eighteen* devices) from popular hardware and software vendors and supporting the most popular Bluetooth versions. Motivated by our empirical findings, we develop and successfully test an *enhanced* key derivation function for Bluetooth that stops *by-design* our six attacks and their four root causes. We show how to effectively integrate our fix into the Bluetooth standard and discuss alternative implementation-level mitigations. We *responsibly disclosed* our contributions to the Bluetooth SIG.

2.2 Introduction

Bluetooth is a *pervasive* technology for low-power wireless communication [35, 34, 32]. It provides two transports: Bluetooth Classic for high throughput and connection-oriented use cases

and Bluetooth Low Energy (BLE) for connectionless and low throughput scenarios. This paper focuses on *Bluetooth Classic*, from now indicated as *Bluetooth*. As billions of devices, such as smartphones, laptops, speakers, headsets, and tablets, daily employ Bluetooth to exchange sensitive data and commands, Bluetooth must provide strong security and privacy guarantees, including confidentiality, integrity and authenticity.

Bluetooth's security and privacy depend on *pairing* and *session establishment*, two mechanisms specified in the *Bluetooth standard* (v5.3) [33]. Devices use pairing to agree upon a long-term secret called the pairing key. Pairing involves user interaction, such as pressing a button or confirming a numeric value on the screen. Paired devices use session establishment to create encrypted and integrity-protected connections, each protected by a *fresh* session key derived from the (static) pairing key and runtime parameters (key diversifiers). Session establishment, unlike pairing, does *not* require user interaction. These two mechanisms have two security modes: (i) *Legacy Secure Connections (LSC)* using legacy cryptographic primitives and procedures, (ii) *Secure Connections (SC)* employing FIPS-compliant ones, such as ECDH, AES-CCM. Pairing and session establishment are *critical* attack surfaces as if they are vulnerable, an adversary can exploit such vulnerability on any (standard-compliant) Bluetooth device. This critical risk motivated extensive research on pairing [16, 198, 28, 184, 98, 129, 107] and session establishment [18, 14] (see Section 2.10 for more works).

But, no prior work has investigated Bluetooth's *forward* and *future secrecy* guarantees and their relation with pairing and session establishment. Forward and future secrecy, which enable to defend past and future messages from key compromise attacks, are *not* even discussed by the Bluetooth standard. We extrapolated these properties via a careful analysis of the standard. We inferred that Bluetooth should provide forward and future secrecy among sessions if the pairing key stays secret. Hence, an attacker compromising the current session key should not be able to decrypt data from past (i.e., forward secrecy) and future sessions (i.e., future secrecy). Then we questioned this assumption and uncovered that, instead, sessions' forward and future secrecy *can* be broken by stealthily attacking *session key derivation* at the protocol level, *without* knowing the pairing key or triggering a new (suspicious) pairing event.

Specifically, we present the **BLUFFS attacks**, six novel attacks breaking Bluetooth's forward and future secrecy by targeting session establishment. The attacks exploit an attack strategy forcing LSC session establishment and manipulating in novel ways its key derivation to *reuse* a key known to the attacker across sessions. The attacker first installs a weak session key, then spends some time brute-forcing it, and reuses it to impersonate or machine-in-the-middle (MitM) a victim in subsequent sessions (breaking future secrecy) and decrypt data from past sessions (breaking forward secrecy). We decline the attack strategy in six attack scenarios related to the victim's connection role (i.e., initiator or responder) and Bluetooth security mode (i.e., LSC or SC). Moreover, we detail the *four* attacks' root causes, two of which uncover that the standard allows to *unilaterally* derive session keys *without* relying on nonces.

We develop the `BLUFFS` toolkit to perform and detect the BLUFFS attacks automatically and with low effort. The toolkit provides an *attack device* module requiring open-source software, a Linux laptop, and a Cypress/Infineon CYW20819 board [108]. We provide *seven* new patches for the board's closed-source firmware enabling monitoring and tampering with Bluetooth session key derivation. Moreover, our *attack checker* module cleverly parses and analyzes session establishment messages, aka Link Manager Protocol (LMP) packets from a pcap file to automatically compute session keys and detect our attacks.

We demonstrate that the BLUFFS attacks are *effective on a large scale* by evaluating eigh-

teen devices embedding seventeen unique Bluetooth chips. We successfully exploited a broad set of devices (e.g., laptops, smartphones, headsets, and speakers), operating systems (e.g., iOS, Android, Linux, Windows, and proprietary OSes), Bluetooth stacks (e.g., BlueZ, Gabeldorsche, Bluedroid, and proprietary ones), vendors (e.g., Intel, Broadcom, Cypress, Cambridge Silicon Radio, Infineon, Bestechnic, Apple, Murata, Universal Scientific Industrial, Samsung, Dell, Google, Bose, Logitech, Xiaomi, Lenovo, Jaybird, and Qualcomm), and Bluetooth versions (e.g., 5.2, 5.1, 5.0, 4.2, and 4.1).

Motivated by our evaluation results, we propose an *enhanced* Bluetooth session key derivation function that *stops by-design* our attacks and their root causes. Our countermeasure is *backward compatible* with the Bluetooth standard and adds minimal overheads. Specifically, it reuses standard-compliant crypto primitives (i.e., e_1 and e_3) and link-layer functions (i.e., LMP commands). It requires forty-eight (48) extra bytes over the air and three extra function calls. We successfully test the fix against the BLUFFS attacks at the protocol level and release the fix in our toolkit. We also discuss implementation-specific mitigations that vendors can use to mitigate some BLUFFS attacks.

We summarize our contributions as follows:

- We study Bluetooth's forward and future secrecy guarantees, two essential properties currently not discussed by prior work and the Bluetooth standard. We show six novel attacks, named BLUFFS attacks, breaking these properties by exploiting Bluetooth's session key derivation. The threats enable device impersonation and MitM across sessions by only compromising one session key. They do not require user interaction or compromise Bluetooth pairing (keys). The attacks are *specification-compliant* as they target protocol-level weaknesses in the Bluetooth standard. We discuss the four attacks' root causes, two of which are novel for Bluetooth and affect session key derivation. We also explain how the attacks extend the state-of-the-art, including the KNOB and BIAS attacks [18, 14].
- We release `BLUFFS`, a low-cost and reproducible toolkit to perform and automatically check our attacks. The toolkit's *attack device* enables manipulation and monitoring of Bluetooth session key derivation. The toolkit's *attack checker* uses a novel LMP parsing and analysis strategy to detect our attacks from a pcap file automatically. Our toolkit complements and extends the state of the art of Bluetooth security testing, such as [171, 63, 62].
- We tested the six BLUFFS attacks on *eighteen* devices embedding *seventeen* different Bluetooth chips from popular hardware and software vendors. The attacks are *successful* against all six LSC chips with one exception and against all eleven SC chips when the impersonated victim is an LSC device. If both victims support SC, the attacks are effective on two out of eleven victims. From our empirical result we conclude that the BLUFFS attacks are *practical* and a *critical risk* for the Bluetooth ecosystem, and should be fixed with high priority.
- We design a backward-compliant Bluetooth session key derivation function based on *fresh*, *authenticated*, and *mutual* key derivation. Our function stops the six BLUFFS attacks and addresses their four root causes at the protocol level. We show how to integrate our countermeasure into the Bluetooth standard with minimal overhead (e.g., three LMP packets and three function calls). We also present our successful evaluation of the fix against our attacks at the protocol level and release it as part of our `BLUFFS` toolkit.

Responsible disclosure We responsibly disclosed our findings and toolkit to the Bluetooth Special Interest Group (SIG) [36] in October 2022. The Bluetooth SIG acknowledged our findings, coordinated the disclosure with the affected vendors, and reserved *CVE-2023-24023* for our report. We also reached out to Google, Intel, Apple, Qualcomm, and Logitech. Google scored our report with high severity, awarded us a bounty, and is working on a fix. Intel did the same but scored the report with medium severity. Apple and Logitech acknowledged the report and are working on fixes. Qualcomm has not replied yet.

Availability The toolkit to test the BLUFFS attack is available at https://github.com/francozappa/bluffs.

2.3 Preliminaries

We present the required Bluetooth preliminaries and our extrapolation of Bluetooth's forward and future secrecy guarantees from the Bluetooth standard.

2.3.1 Bluetooth

Bluetooth is the *de-facto standard* technology for low-power and reliable wireless communication and has an open specification (v 5.3) [33]. It was born as a cable-replacement wireless protocol for the unlicensed 2.4 GHz ISM (Industrial, Scientific and Medical) band, and evolved to address various use cases requiring high-throughput and persistent connections. For example, it supports wireless audio streaming, file transfer, hands-free services, peer-to-peer connections, and Internet bridging. Bluetooth packets should be protected against relevant attacks, such as device spoofing and MitM, as it transports sensitive data and commands.

The Bluetooth stack loosely follows the Open Systems Interconnection (OSI) model. At the physical layer, it employs synchronized frequency hopping and time division multiple access. The link layer uses a star topology managed by the link manager protocol (LMP). The link layer connection initiator is known as *Central*, while the responder is called *Peripheral*. These two roles can be switched dynamically during connection establishment or while a connection is ongoing. Bluetooth uses a six-byte, unique, and static address to identify a device at the link layer. A Bluetooth address does not contain secret information and is obtained with an inquiry procedure. At the application layer, Bluetooth provides several *profiles*, such as the advanced audio distribution (A2DP) profile. The Bluetooth *Controller* manages the physical and link layers, while the Bluetooth *Host* takes care of the upper layers. The Host and the Controller communicate via the *Host Controller Interface (HCI)*, a protocol based on commands and events.

The Bluetooth standard specifies *link-layer security mechanisms*, providing confidentiality, integrity, and authenticity to upper layers, including all Bluetooth profiles. *Pairing* allows devices to establish a long-term pairing key (PK). The standard defines such a procedure as Secure Simple Pairing (SSP) [33, p. 268]. *Session establishment* enables paired devices to establish a secure session using a fresh session key (SK). SK is derived from PK and constant and variable inputs. The standard includes two security modes affecting pairing and session establishment: LSC, which employs legacy security mechanisms for backward-compliance reasons (e.g., E_0 and SAFER+), and SC that uses FIPS-compliant ones (e.g., ECDH, AES-CCM, and HMAC).

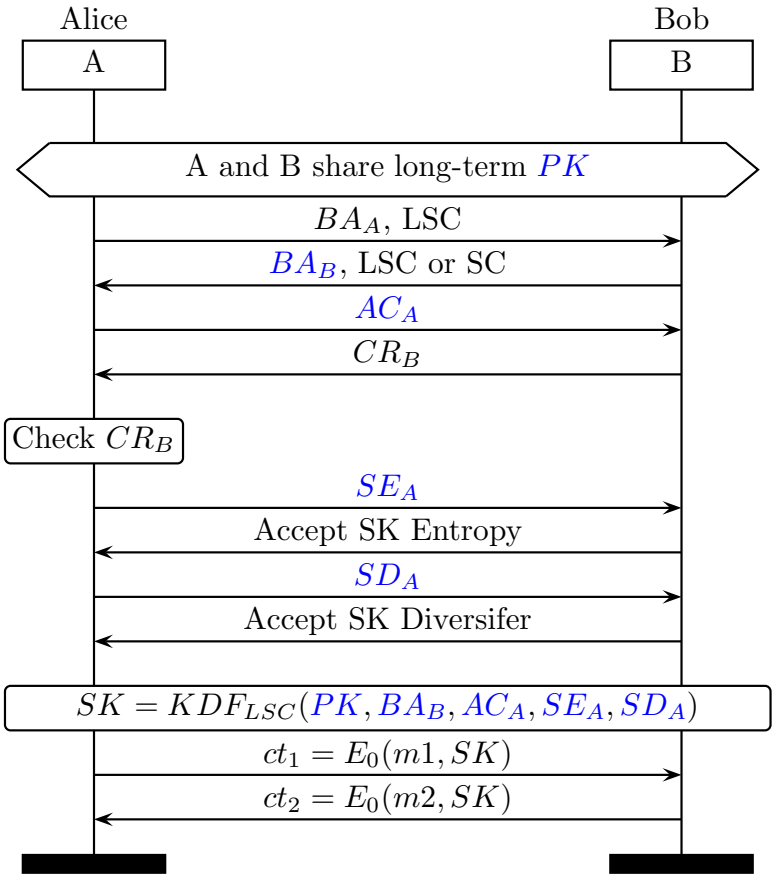

Figure 2.1: **Bluetooth LSC session establishment**. The values in blue are used to compute a fresh SK. PK and BA_B are *constant*, while AC_A, SE_A, and SD_A are *variable*. In this example, Alice (i.e., the Central) controls SK derivation as she provides all the variable SK derivation inputs.

2.3.2 Bluetooth Forward and Future Secrecy

Despite their critical associated risks, Bluetooth's forward and future (i.e., backward) secrecy guarantees are *unexplored*. By compromising forward (future) secrecy, the attacker could break the confidentiality of past (future) sessions. However, we do not know if these attacks and vulnerabilities exist as the Bluetooth standard neither covers nor define forward and future secrecy, and no prior research investigated them. In this work, we address this crucial gap.

We examined pairing and session establishment from the standard and extracted their forward and future secrecy guarantees. Bluetooth *should* provide forward and future secrecy *across sessions* until PK or the SK key derivation function (KDF) are not compromised. Specifically, an attacker compromising the current SK cannot target past and future sessions because each session employs a fresh (i.e., different) SK derived from PK and variable key diversifiers. So it is crucial that PK stays secret and that SK is properly derived. Nevertheless, no prior work evaluated the strength of SK derivation and the existence of related (practical and impactful) attack scenarios.

Now we describe *LSC* session establishment, including its key derivation phase, as is the target of our work. We assume that Alice (Central with address BA_A) and Bob (Peripheral with address BA_B) are paired and share PK. As shown in Figure 2.1, LSC session establishment starts with two messages where Alice and Bob identify themselves and negotiate LSC. Then,

Alice asks Bob to authenticate PK by sending a challenge AC_A. Bob sends back CR_B, a response computed from PK and AC_A, and Alice checks that CR_B equals the response she computed locally. Then, Alice sends SE_A, an SK entropy proposal between 16 and 7 bytes, and Bob can accept it (as in Figure 2.1) or propose a lower value to be accepted by Alice. Once SK entropy negotiation is completed, Alice sends to Bob SD_A, a *session key diversifier*, and Bob acknowledges it. Finally, the devices use KDF_{LSC} to derive SK from *variable* (AC_A, SE_A, SD_A) and *constant* inputs (PK, BA_B).

KDF_{LSC} is the LSC key derivation function specified in the standard and is defined as a system of three equations [33, p. 267]:

$$COF = e_1(PK, AC_A, BA_B) \tag{2.1a}$$
$$ISK = e_3(PK, SD_A, COF) \tag{2.1b}$$
$$SK = e_s(ISK, SE_A) \tag{2.1c}$$

Using Equation 1a, the devices compute a ciphering offset number (COF) from the pairing key, Alice's authentication challenge and Bob's Bluetooth address. The computation uses the e_1 authentication function [33, p. 975], which is based on the SAFER+ block cipher [136]. Then, an intermediate session key (ISK) is computed via Equation 1b, using the pairing key, Alice's session key diversifier, and COF. The second computation employs the e_3 key generation function [33, p. 981]. Finally, Alice and Bob derive SK by reducing the entropy of ISK according to SE_A with the e_s function as shown in Equation 1c. The reduction function relies on on modular arithmetic over polynomials in the finite Galois field [27].

2.4 Threat Model

Here we present the paper's system and attacker models and our notation.

2.4.1 System Model

Our system model considers a scenario where Alice and Bob (i.e., the victims) want to communicate securely using Bluetooth. Alice and Bob represent arbitrary devices (e.g., laptops, headsets, and smartphones) and can employ any Bluetooth profile (e.g., audio, hands-free, and Internet bridge). We assume the victims have already *paired* using their strongest security capabilities (e.g., SSP and SC).

The paired victims establish secure connections using Bluetooth's *session establishment*. Alice is the Central (initiator) and Bob the Peripheral (responder), unless stated otherwise. As discussed in Section 2.3.2, if an attacker compromises the current SK, she should be unable to compromise past and future sessions (i.e., break forward and future session secrecy), as each session employs a fresh (i.e., different) SK.

2.4.2 Attacker Model

Our attacker model considers Charlie, a *proximity-based* attacker in Bluetooth range with the victim(s). The attacker can capture Bluetooth packets in plaintext (e.g., authentication challenges, key diversifiers, and negotiated entropy values) and ciphertext (e.g., encrypted audio, files, or internet traffic). Charlie knows the victim's Bluetooth address, can craft (standard-compliant)

Bluetooth packets, and negotiate arbitrary capabilities. Charlie cannot compromise PK, does not observe the victims while they are pairing, and does not trigger new pairing events. She cannot tamper with the victims' devices, including their hardware and software components. The attacker can downgrade the entropy of SK to the lowest value supported by a victim (e.g., 1 byte for devices not patched against the KNOB attack or 7 bytes) and brute force SK. We do not assume a specific brute-force effort to cover attackers with different capabilities and resources (e.g., motivated and average attackers).

Charlie wants to break the forward and future secrecy of Alice and Bob' sessions. For example, she would like to impersonate Alice to Bob, Bob to Alice, or MitM them *across* sessions to decrypt past messages (i.e., breaking forward secrecy) and decrypt or inject future ones (i.e., compromising future secrecy). These goals are novel as the state-of-the-art assumes an adversary targeting the *current* session (e.g., KNOB [18] and BIAS [14]). Moreover, the attacker would like to exploit *any* Bluetooth device, regardless of its Bluetooth capabilities (e.g., chip, version, software stack, security mode, and supported profiles).

2.4.3 Notation

In the paper, we use the following notation. We indicate a Bluetooth address as BA, an authentication challenge as AC (AU_RAND in the standard), a challenge-response as CR (SRES in the standard), a session key as SK (Kc' in the standard), a pairing key as PK (LK in the standard), a session key entropy proposal as SE and a session key diversifier as SD (EN_RAND in the standard). We abbreviate a key derivation function with KDF. We use A, B, and C subscripts to indicate Alice, Bob, (the victims) and Charlie (the attacker).

2.5 BLUFFS Attacks

In this section, we describe the *BLUFFS attacks*, *six* new threats breaking Bluetooth's forward and future secrecy and enabling impersonation and MitM attacks *across* sessions. We also present the *four* attacks' root causes related to SK derivation during session establishment and explain why our attacks *extend* the state of the art (e.g., KNOB [18] and BIAS [14]). Please refer to Section 2.3 for the attacks' preliminary and Section 2.4 for their threat model.

2.5.1 Attack Description

Strategy The BLUFFS attacks take advantage of a *novel* attack strategy, enabling Charlie to reuse a weak session key (SK_C) across sessions to spoof or MitM arbitrary victims (e.g., LSC and SC Centrals and Peripherals). We now describe such a strategy in an impersonation attack setup with the help of Figure 2.2. Charlie presents to Bob using Alice's Bluetooth address (BA_A) obtained using Bluetooth inquiry procedures or de-anonymization attacks such as [60]. She negotiates LSC mode (LSC) to force LSC session establishment (and key derivation), whether Bob supports LSC or SC. If Charlie is a Peripheral, she switches to the Central role to lead session establishment, including SK derivation. As a consequence, Charlie can target Bob as a Central (initiator) or a Peripheral (responder).

Next, Charlie forces a *fixed* and *weak* session key (SK_C) by cleverly negotiating her session key derivation parameters. Specifically, she sends a constant authentication challenge (AC_C) and ignores Bob's response (CR_C). She proposes the lowest session key entropy value (SE_C) to

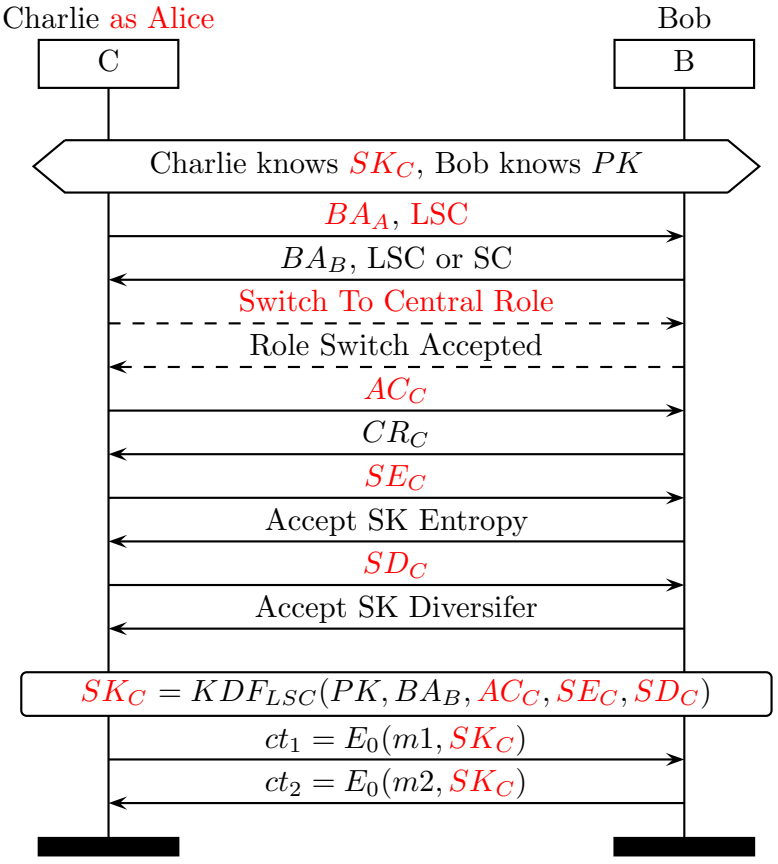

Figure 2.2: **BLUFFS attacks strategy**. Charlie approaches Alice as Bob negotiates LSC regardless of the security mode supported by Bob, and, if she is a Peripheral, switches to the Central role. Then, during LSC key derivation, she proposes *constant* values (AC_C, SE_C, SD_C) to force the derivation of a *fixed* session key (SK_C). Charlie employs this strategy while impersonating (or MitMing) Alice and Bob to *reuse* SK_C across sessions.

(re)establish a weak key and a *constant* session key diversifier SD_C. As a result, Bob (re)derives SK_C by using KDF_{LSC} with constant inputs, i.e., PK, BA_B, AC_C, SE_C, and SD_C. For example, Charlie can set AC_C and SD_C equal to zero, and SE_C equal to one (SK_C has one byte of entropy).

We employ our attack strategy in *six* attacks covering all combinations of impersonation and MitM attacks across sessions (i.e., targeting SC and LSC Centrals and Peripherals). As shown in the following enumeration, the attacker can spoof a LSC Central or Peripheral to a LSC or SC victim (i.e., A1, A2), impersonate a SC Central or Peripheral to a LSC or SC victim (i.e., A4, A5), or MitM a session where one victim supports LSC or both victims support SC (i.e., A3, A6).

A1: Spoofing a LSC Central to a victim Peripheral

A2: Spoofing a LSC Peripheral to a victim Central

A3: MitM session where one victim supports LSC

A4: Spoofing a SC Central to a victim Peripheral

A5: Spoofing a SC Peripheral to a victim Central

A6: MitM session where the victims support SC

The BLUFFS attacks break Bluetooth's session forward and future *without* compromising

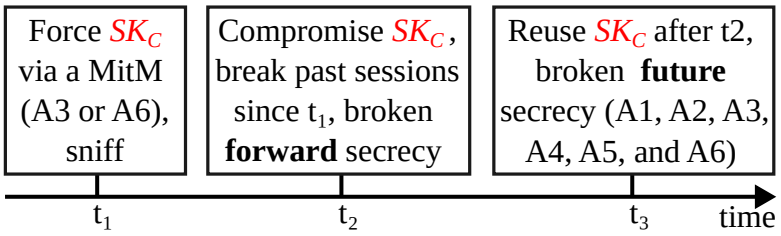

Figure 2.3: BLUFFS attacks timeline. The attacker forces SK_C at time t_1 via a MitM attack (A3 or A6) and sniffs the messages exchanged by the victims. She compromises (brute forces) SK_C at time t_2 and breaks forward secrecy by decrypting past traffic since t_1. She reuses SK_C at time t_3 to impersonate or MitM a victim (A1, A2, A3, A4, A5, or A6) and compromises future secrecy.

prior (strong) SKs negotiated by the victims. We consider forward (future) secrecy broken if Charlie compromises past (future) sessions once SK_C is brute-forced (i.e., compromised). As shown by the timeline in Figure 2.3, the attacker at time t_1 mounts a MitM attack forcing SK_C (A3 or A6), captures the traffic on the current and subsequent sessions, and starts brute forcing SK_C. At $t_2 > t_1$ she brute forces (compromises) SK_C and decrypts all past messages exchanged since t_1 violating forward secrecy. At $t_3 > t_2$ she reuses SK_C to impersonate or MitM Alice and Bob across the next sessions (A1–A6). Hence, she breaks future secrecy by violating the sessions' confidentiality, integrity, and authenticity from t_2 onwards.

Brute force setup and effort Charlie brute forces SK_C employing an *offline* and *parallelizable* setup similar to [18]. She tests offline multiple session keys against one or more sniffed ciphertexts using known Bluetooth packet fields as oracles (e.g., L2CAP and RFCOMM headers decrypting to known constants). The attacker's brute force effort is proportional to SE_C (i.e., SK's entropy). However, it does *not* depend on the number of targeted sessions as it should with proper forward and future secrecy mechanisms. If SE_C is one, the brute force effort is negligible, i.e., 128 trials on average within a key space of 256 elements. Otherwise, if it is seven, the attacker requires 2^{55} trials on average within a key space of 2^{56} elements. Based on prior work breaking symmetric cryptosystems with seven bytes of entropy, such as the data encryption standard (DES) [75, 122], we estimate a moderate effort for a low-cost attacker using commercial equipment (e.g., one to several weeks) and a low effort for a decently funded attacker using distributed computing or optimized hardware (e.g., one to several days).

Impact The BLUFFS attacks have a *severe* impact on Bluetooth's security and privacy. They allow decrypting (sensitive) traffic and injecting authorized messages across sessions by re-using a *single* session key. Prior attacks require leaking PK or brute-forcing one SK per target session to achieve a similar impact. Our attacks can target any Bluetooth device, regardless of its role, security mode, and supported Bluetooth profiles, as they rely on flaws in the standard (detailed next in Section 2.5.2). Moreover, the attacks are *stealthy* since they exploit the Bluetooth firmware (Controller) without requiring user interaction and triggering notifications to the user. Finally, the attacks do *not* require specialized and expensive equipment, as demonstrated by our implementation in Section 2.6, and have a *widespread impact*, as empirically shown in Section 2.7. Motivated by the impact of the attacks, in Section 2.8, we present a practical design-level fix that we recommend integrating into the Bluetooth standard and discuss other implementation-level mitigations.

2.5.2 Attacks Root Causes

The BLUFFS attacks' root causes are *four* architectural vulnerabilities in the specification of Bluetooth session establishment (i.e., RC1, RC2, RC3, and RC4) [33]. RC1 and RC2 are *novel* as they are the first targeting SK derivation and allowing to derive the same SK across sessions (breaking their forward and future secrecy). While RC3 and RC4 have been exploited to attack other session establishment phases. For instance, the BIAS attacks [14] employ RC3 and RC4 to bypass PK authentication, while the KNOB attacks [18] take advantage of them to downgrade the entropy of SK.

RC1: LSC SK diversification is unilateral (new) The LSC SKDF introduced in Section 2.3 and depicted in Figure 2.1 derives a SK using static inputs (i.e., PK, BA) and variable ones (i.e., AC, SE, SD). The variable inputs diversify SKs across sessions. One would expect that both the Central and the Peripheral would contribute to SK diversification. However, the standard allows the *Central* to set all the SK diversification values. Hence, an attacker impersonating a Central (or role switching to a Central when impersonating a Peripheral) can *unilaterally drive* SK diversification (across sessions). We note that the Peripheral's Bluetooth address is unusable as a variable input because Bluetooth (Classic) does *not* support randomized link-layer addresses.

RC2: LSC SK diversification does not use nonces (new) SK is diversified using random numbers (AC [33, p. 625] and SD [33, p. 637]) and a positive integer (SE in [33, p. 962]). As none of them is a nonce, they can be *reused* in past, present, and future sessions without violating the standard. Hence, an attacker who knows a triplet (AC_C, SE_C, SD_C) and the corresponding SK_C, can force the victims to derive the same *attacker-controlled* session key across sessions.

RC3: LSC SK diversifiers are not integrity protected The variable inputs exchanged during SK derivation are sent without integrity protection. As a result, an attacker who is spoofing a device or performing MitM on a session can manipulate AC, SE, and SD, without being detected.

RC4: Downgrading SC to LSC does not require authentication The negotiation of SC or LSC is not integrity protected. Hence, an attacker can always downgrade a session to LSC, regardless of SC support from the victim, and trigger LSC key negotiation and KDF_{LSC} (presented in Figure 2.1).

Root causes and attacks Table 2.1 shows how the six BLUFFS attack presented in Section 2.5.1 map to RC1, RC2, RC3, and RC4. All attacks take advantage of RC1, RC2, and RC3 as they unilaterally derive a constant session key without using a nonce and manipulating the integrity of the session key diversifiers. RC4 is exploited by the three BLUFFS attacks targeting SC to downgrade a session to LSC. We also note that no prior research (and attack) discovered RC1 and RC2.

2.5.3 Comparison with KNOB and BIAS

The KNOB+BIAS attack chain is considered the most effective way to impersonate Bluetooth devices during session establishment. The attacker employs BIAS to bypass PK's authentication,

Table 2.1: Mapping the six BLUFFS attacks to their four root causes. CI and PI stands for Central Impersonation and Peripheral Impersonation.

BLUFFS attack	RC1	RC2	RC3	RC4
A1: Spoofing a LSC Central	✓	✓	✓	×
A2: Spoofing a LSC Peripheral	✓	✓	✓	×
A3: MitM LSC victims	✓	✓	✓	×
A4: Spoofing a SC Central	✓	✓	✓	✓
A5: Spoofing a SC Peripheral	✓	✓	✓	✓
A6: MitM SC victims	✓	✓	✓	✓

then KNOB to downgrade the entropy of SK. The BLUFFS attacks share the same goals but employ *different* steps (e.g., attacking SK derivation) that are *chainable* with the BIAS and KNOB ones.

However, unlike the BLUFFS attacks, the KNOB+BIAS chain does *not* compromise forward and future secrecy as it is effective within the current session. More generally, no prior research investigated the existence of vulnerabilities and attacks on session establishment persisting *across* sessions (i.e., no research on Bluetooth sessions' forward and future secrecy). Our work fills this research gap by presenting the first key-reuse attacks for Bluetooth.

The BLUFFS attacks are successful even if we fix the role-switching and SC session downgrade vulnerabilities discussed in the BIAS paper. The attacker can reuse SK_C against any LSC device while impersonating an LSC Central (A1). In particular, the attacker legitimately negotiates LSC, AC_C, SE_C, SD_C and is not required to authenticate PK. Moreover, devices patched against the KNOB attacks are still vulnerable to the BLUFFS attacks, as they accept SE_C equal to seven.

We enable attack scenarios, which are too costly for KNOB+BIAS. For instance, if we target N_s sessions, our attacks' cost does *not* increase with N_s as we brute force one session key. While the KNOB+BIAS cost is significantly higher as it *linearly* increases with N_s. The cost difference is even more compelling if a victim supports entropy values (SE) higher than seven bytes. To give an intuition about the cost difference, if we assume that brute forcing a SK with seven bytes of entropy takes one week (keyspace is 2^{56}), and a SK with sixteen bytes of entropy takes one thousand years (keyspace is 2^{128}); then our attacks cost one week against seven bytes of entropy and one thousand years against sixteen bytes of entropy, while KNOB+BIAS costs N_s weeks and N_s thousand years.

As a result of our investigation, we formulate and empirically answer new and valuable research questions not addressed by KNOB and BIAS (and any other prior work). For example, we reveal the forward and future secrecy guarantees provided by the Bluetooth standard, their architectural vulnerabilities, how to exploit these vulnerabilities with practical and low-cost attacks, the attacks' effectiveness on actual devices from different hardware and software providers, and how to fix or mitigate the attacks (and their root causes).

2.6 Implementation

We now describe the implementation of our `BLUFFS` toolkit to perform and check the BLUFFS attacks presented in Section 2.5. The toolkit has two modules: an *attack device* and an *attack checker* and extends state-of-the-art tools for Bluetooth security research, such as `internalblue` [171] and the `BIAS` and `KNOB` toolkits [63, 62], with novel and useful features. For example, we unlock the possibility to dynamically manipulate Bluetooth's key derivation parameters and monitor SK across sessions, and automatically detect our attacks from a pcap file.

Our toolkit is *low-cost* as it uses open-source software (e.g., Python and Wireshark) and cheap hardware (e.g., a Linux laptop and a Cypress CYW20819 development board). Its technical details are relevant for reproducing, checking, and extending our experimental setup and results (shown in Section 2.7).

2.6.1 Attack device module

Architecture Our attack device consists of a *Linux laptop* connected via USB to a *CYW20819 board* from Cypress/Infineon. Its initialization setup is the same as the one described in the BIAS repository [63]. In summary, to access link-layer traffic from the laptop's HCI interface, we activate LMP redirection from the board with a vendor-specific command and patch the laptop's Linux kernel to parse the LMP packets. Moreover, we patch the board's firmware using a proprietary binary instrumentation feature from Cypress. Patching the firmware (Bluetooth Controller) is essential to manipulate Bluetooth key derivation. The board's patching is facilitated by *Internalblue* [135], which provides high-level Python APIs to patch the board (i.e., `patchRom`) and read and write its RAM (i.e., `readMem` and `writeMem`).

The CYW20819's vendor-specific patching mechanism is quite complex but clever. First, the unpatched firmware, stored in read-only memory (ROM), receives the `Download_Minidriver` command from our laptop (Bluetooth Host) and stops its execution. Then, the laptop sends a `Write_RAM` command to write in RAM the addresses to be modified in ROM. Finally, the laptop runs the `Launch_RAM` command to register the patches in RAM and resume execution. Hence, anytime the firmware CPU fetches an address in ROM that should be patched, the control flow is redirected to the patch in RAM. For more information about this mechanism, refer to [113].

Firmware patches We developed *seven new patches* for the attack device Bluetooth firmware. The patches, summarized in Table 2.2, allow performing the six BLUFFS attacks presented in Section 2.5. The table's first and second columns indicate the patch name and description, while the last two show the patched firmware function and its ROM address.

Our patches unlock useful security testing capabilities for Bluetooth. The three *man_** patches manipulate AC, CR, and SD, and enable negotiating constant SK diversifiers as in Figure 2.2, and failing session establishment when the attacker has to authenticate a PK. The three *rea_** patches monitor SK's value which is otherwise *hidden* to the HCI and LMP layers. The *rs_nop* patch allows to successfully attack devices asking to role switch to Central regardless of the attacker's role switch strategy. This patch is valuable as it extends the effectiveness of our attacks (and the BIAS+KNOB chain) to a new class of devices. We reuse the patches from the `BIAS` toolkit [63] to negotiate $SE = 7$ for the KNOB attack and avoid PK authentication. We also coded a high-level patching function to ease the development of new patches (see `device/patch.py` in our anonymized repository).

Table 2.2: Seven novel patches for the CYW20819 Bluetooth firmware to perform the BLUFFS attacks. The third and fourth columns indicate the patched firmware function and its address in ROM.

Name	Description	Patched function	Addr
man_ac	Manip. AC	`txAuRand`	`AEB8C`
man_cr	Manip. LSC CR	`txSres`	`AEDC8`
man_sd	Manip. SD	`txStartEncryptReq`	`AE4B4`
rea_sk	Read SK value	`txStartEncryptReq`	`AE5B4`
rea_skec	Read Central SE	`txStartEncryptReq`	`AE5B4`
rea_skep	Read Perip. SE	`procStartEncryptReq`	`AE70C`
rs_nop	No Perip. role sw.	`handleLmpSwitchReq`	`A643C`

We developed the patches in Table 2.2 by *reverse-engineering (RE)* unknown portions of the CYW20819 Bluetooth firmware. In particular, we used Ghidra [192] loaded with the firmware symbols leaked from a Cypress SDK as described in [135]. As we wrote the patches in ARM Thumb-2 assembly, they contain 2-byte and 4-byte instructions aligned to 4-byte boundaries, and the code branches to odd addresses [20]. Currently, to comply with responsible disclosure, we are releasing `man_cr.s`, `rea_sk.s`, and `rs_nop.s`.

Listing 2.1 shows our *rs_nop* patch to refuse Peripheral's role switch requests. Whenever the firmware program counter hits `0xA643C` inside `handleLmpSwitchReq` in ROM, the firmware code jumps to our patch in RAM. The patch passes a zero as `isMssInstantPassed`'s second parameter by zeroing `r1`. Then, it calls (i.e., branch and link) `isMssInstantPassed` and overwrites the routine's return value to `True` by setting `r0` to one. As a side effect, the attack device firmware thinks that the MSS (Minimum Subevent Space) interval has passed and *rejects* the correspondent role switch request. Notably, such rejection is compliant with the standard. Finally, the patch unconditionally jumps back to the next valid ROM instruction in Thumb-2 mode (i.e., branch to an odd address). This patch enables exploitation of a *new* class of devices, such as victims trying to (defensively) role switch to the Central role during session establishment. For example, we can exploit iPhone 12 and 13 by rejecting their role switch requests during session establishment.

2.6.2 Attack checker module

Our attack checker enables new capabilities for Bluetooth static analysis. In particular, given a pcap file containing LMP packets, it automatically isolates Bluetooth sessions, computes session keys, and detects the BLUFFS attacks. We release it as part of our `BLUFFS` toolkit in the `checker` folder. The checker is written in Python 3 and leverages capable and available tools, such as `wireshark/tshark` [204] and `pyshark` [72]. It requires H4 and LMP dissectors for Wireshark v3.6+ [172] or older versions [66]. We now describe the checker's *parser*, *kdf*, and *analyzer* components.

Parser The parser uses `pyshark` to extract relevant LMP packets from a pcap file. It supports *nine* LMP packet types as shown in Table 2.3. Specifically, it parses `LMP_host_connection_req` and `LMP_detach` packets, which indicate when a session starts and ends. It processes entropy negotiation values (SE) from `LMP_encryption_key_size_req` and related `LMP_accepted` packets.

Table 2.3: Nine LMP packets supported by our parser.

LMP packet	Opcode	Description
`LMP_host_connection_req`	51	Start a session
`LMP_detach`	7	Abort a session
`LMP_encryption_key_size_req`	16	Propose SE
`LMP_accepted` (SE)	3 (16)	Confirm SE
`LMP_au_rand`	11	Send AC
`LMP_sres`	12	Send CR
`LMP_start_encryption_req`	17	Send SD
`LMP_accepted` (SD)	3 (17)	Accept SD
`LMP_not_accepted` (AC)	4 (11)	Reject AC

The parser also manages authentication challenges (AC) and responses (CR) from `LMP_au_rand` and `LMP_sres` packets and detects when AC is not accepted by monitoring the relevant `LMP_not_accepted` packet. Moreover, it deals with session key diversifies (SD) by parsing `LMP_start_encryption_req` and consequent `LMP_accepted` packets.

The parser's implementation is at `device/parser.py` and follows an *object-oriented* design. An `LmpBase` parent class, shown in Listing 2.2, parses relevant fields shared by *all* LMP packets. For example, it stores the LMP packet number (`number`), transaction initiator (`tinit`), and opcodes (`op`, `op_ext`). Specialized classes, extending `LmpBase`, manage specific LMP opcodes. For instance, `LmpAuRand`, presented in Listing 2.3, deals with `LMP_au_rand` packets and extracts AC as an hexstring and a bytearray (`aurand` and `aurand_ba`). We developed other eight specialized LMP classes, see `parser.py` for more details.

Kdf The kdf module implements the LSC key derivation function presented in Section 2.3 as shown in Listing 2.4. This functionality is needed to compute and check SKs across sessions automatically. In particular, `kdf.py` computes SK (as in Equation 2.1) by using `e1.py`, `e3.py` and `es.py` and their related cryptographic primitives (such as `h.py`). We provide the kdf code in the toolkit's `device` folder, and we note that it extends [63, 62] by providing the *full* LSC key derivation chain. Our code is sound as is *tested* against the vectors in the Bluetooth standard [33, p. 921] and actual values extracted during our experiments. The kdf test suite can be run with `make tests`.

Analyzer The analyzer module is implemented in `checker/analyzer.py` and *automatically* detects the BLUFFS attacks presented in Section 2.5. It builds on top of the parser and kdf modules presented earlier. The analyzer employs the `gen_analysis` function, shown in Listing 2.5, that takes as inputs a pcap file, a PK, and the Bluetooth address of the victim (Peripheral). Then it calls `gen_sessions` to extract from the pcap a list of LMP sessions (`sessions`). Then, for each session, it calls the `gen_report` function that computes SK from SE, SD, and AC and stores the reports in a list (`reports`). Finally, for each report `gen_analysis` checks if SK_C is reused across sessions (`assert report["sk"] == EXP_SK`). This automation speeded up our large-scale evaluation reported in Section 2.7.

To demonstrate that our module is practical, we provide the material to reproduce our anal-

ysis of the Pixel Buds A-Series earbuds. In the toolkit's `pcap` folder, there is a file prefixed with `lsc-` with the LMP traffic generated while we performed the PI and CI attacks while spoofing an LSC device. Also, we provide a `sc-` prefixed file for the CI and PI attacks while impersonating a SC device. `analyzer.py` contains two test functions with the needed PK, BA, and target SK_C. By running the script, we observe that the attacker *reuses* SK_C across sessions, regardless of her role (i.e., Central or Peripheral). In particular, in the LSC cases SK_C is `c61da2f42fefab75bb15b7927af0a631`, while in the SC scenarios is `3581f68eecc5d1f295894c6bc9262812` and both SK_C have 7 byte of entropy. Under the hood, the script verifies SK_C (`EXP_SK`) with an `assert` statement at line 175. The first session in each test contains an SK different from SK_C, as that session is *not* under attack, but it is the first legitimate session after pairing completion.

2.7 Evaluation

We now present our evaluation setup and results.

2.7.1 Setup

Our evaluation setup tests the six BLUFFS attacks presented in Section 2.5 (i.e., A1, A2, A3, A4, A5, and A6) on a target LSC or SC device. Testing a device requires less than 15 minutes. Our setup relies on the attack device and checker modules introduced in Section 2.6 to automate its repetitive parts (e.g., compute and check the session keys from a pcap file). The setup has *six* steps:

1. First, we test A4, A5, and A6 which involve spoofing and MitM of SC victims. We pair the attack device (also acting as a spoofed victim) with the target victim, and we disconnect them. While pairing, the attack device declares *SC support*.
2. We patch the attack device's firmware (using the patches presented in Table 2.2) to implement the strategy discussed in Section 2.5.1. The patched attack device declares LSC support, monitors SKs across sessions, and sets $AC_C = SD_C = 0$, and $SE_C = 7$ to renegotiate a constant and weak session key (i.e., SK_C). Also, the attack device tries to role switch to Central before session key derivation when it is a Peripheral and refuses role switch requests when acting as a Central. We also force the attack device to send a wrong CR_C to detect a failure in (rare) attack scenarios where the victim asks the Central to authenticate PK (e.g., PI against the BOOM 3 Bluetooth speaker).
3. We test A4 by establishing multiple sessions from the attack device (Central) and capturing the HCI and LMP packets in a pcap file. We also monitor SK from RAM in each session, but this manual step is optional. Then, we employ our attack checker to automatically recompute and compare the SKs from the pcap file. If the computed keys are the same, the attack is successful, as the adversary is impersonating a SC device while reusing SK_C across sessions.
4. We test A5 by establishing multiple connections from the victim to the attack device (Peripheral). We employ the same strategy described in the previous steps, and the attack is effective if we reuse SK_C across sessions.
5. If either the CI or the PI attack is successful, then the victim is also vulnerable to A6, as the adversary can combine CI and PI in a MitM attack against SC victims.

6. We unpair the attack device and the victim and pair them again, but this time the attack device declares *LSC support*. Then, we repeat steps two, three, four, and five to test A1, A2, and A3.

Our setup uses, *without loss of generality*, the attack device both as a victim and the attacker to speed up the experiments. However, as stated in Section 2.4, we stress that the attacker does require neither to pair with the victim devices nor observe them while they are pairing nor trigger a new pairing session. To prove such a claim, we tested scenarios where before attacking the victim, we unpaired the attack device from the victim by overwriting its PK with a wrong value (via a firmware patch), and we were still able to force SK_C across sessions.

2.7.2 Results

Table 2.4 presents our evaluation results obtained by testing the six BLUFFS attacks on *eighteen* heterogeneous and popular devices (second column) embedding *seventeen* unique Bluetooth chips (first column) and employing the most popular Bluetooth versions (third column). We compiled the table following the six steps in Section 2.7.1. The last six columns contain a ✓ if a device is vulnerable to an attack; otherwise, a $\times$. The fourth, fifth, and sixth columns show CI, PI, and MitM attacks when the spoofed victim supports LSC (i.e., A1, A2, and A3). While the last three columns report CI, PI, and MitM attacks while impersonating a SC device (i.e., A4, A5, and A6).

LSC Victims As shown by the first six rows in Table 2.4, *all* tested LSC chips and devices are vulnerable to the six attacks, with one exception. The Logitech BOOM 3 speaker is *not* vulnerable to the PI attacks (A2, A5), as it requires the Central to authenticate PK, thus preventing the attacker from completing session establishment (despite eventually being able to reuse SK_C). The Bose SoundLink speaker also asks the Central to authenticate but is still vulnerable to A2 and A5 as it does *not* check the challenge response. The Google Pixel Buds A-Series (2021) are still vulnerable to the KNOB downgrade resulting in SK_C with *1 byte* of entropy; we reported this worrisome finding to Google and got a “will not fix” response.

SC Victims The last *eleven* rows in Table 2.4 shows our findings about chips and devices supporting *SC*. If the spoofed victim supports LSC, all chips/devices are vulnerable to the CI, PI, and MitM attacks (A1, A2, A3). Hence, an attacker can impersonate any chip/device from the LSC block of rows to any chip/device in the SC set. If we impersonate a SC device, the CYW20819 and CYW40707 chips are vulnerable to A4, A5, and A6, demonstrating that the attacks are effective against SC. Instead, the other eight chips/devices we tested are *not* vulnerable to A4, A5, and A6, as the chips enforce SC between pairing and session establishment, preventing the attacker from downgrading the session to LSC. But, they are still vulnerable to A1, A2, and A3 because of the vulnerabilities we uncover with LSC.

Evaluation impact Driven by our empirical results shown in Table 2.4, we are convinced that the BLUFFS attacks are *practical* and have a *large-scale* impact on the Bluetooth ecosystem. In particular, they can target *SC and LSC* devices (e.g., laptops, smartphones, headsets, and speakers) supporting a wide range of *operating systems* (e.g., iOS, Android, Linux, Windows, and proprietary OS), *Bluetooth stacks* (e.g., BlueZ, Gabeldorsche, Bluedroid, and proprietary ones), *vendors* (e.g., Intel, Broadcom, Cypress, Cambridge Silicon Radio, Infineon, Bestechnic, Apple, Murata, Universal Scientific Industrial, Samsung, Dell, Google, Bose, Logitech, Xiaomi, Lenovo, Jaybird, and Qualcomm), and *Bluetooth versions* (e.g., 5.2, 5.1, 5.0, 4.2, and 4.1).

Table 2.4: **BLUFFS attacks evaluation results**. We run the six BLUFFS attacks against *eighteen* devices with *seventeen* unique Bluetooth chips. All the six tested LSC victims are vulnerable to all the attacks, with one exception. When we impersonate an LSC device to an SC device, all tested eleven SC targets are vulnerable. Comparatively, when we spoof an SC device to another SC device, the attacks are only effective on *two* out of eleven tested chips (i.e., CYW20819 and CYW40707). Our results empirically demonstrate that the attacks are practical and have a widespread impact on the Bluetooth ecosystem. **Notes**: [1]ask to authenticate as a Central, [2]does not check authentication response (CR), [3]vulnerable SK downgrade with 1 byte of entropy, [4]does not allow LSC session establishment if paired with SC. **Acronyms**: USI stands for Universal Scientific Industrial, CYW for Cypress, BCM for Broadcom, and CSR for Cambridge Silicon Radio. A n/a in the Chip column indicates that the chip SoC model is unavailable from public sources.

Chip	Device(s)	BTv	A1	A2	A3	A4	A5	A6
LSC Victims								
Bestechnic BES2300	Pixel Buds A-Series[3]	5.2	✓	✓	✓	✓	✓	✓
Apple H1	AirPods Pro	5.0	✓	✓	✓	✓	✓	✓
Cypress CYW20721	Jaybird Vista	5.0	✓	✓	✓	✓	✓	✓
CSR/Qualcomm BC57H687C	Bose SoundLink[1,2]	4.2	✓	✓	✓	✓	✓	✓
Intel Wireless 7265 (rev 59)	Thinkpad X1 3rd gen	4.2	✓	✓	✓	✓	✓	✓
CSR n/a	Logitech BOOM 3[1]	4.2	✓	×	✓	✓	×	✓
SC Victims								
Infineon CYW20819	CYW920819EVB-02	5.0	✓	✓	✓	✓	✓	✓
Cypress CYW40707	Logitech MEGABLAST	4.2	✓	✓	✓	✓	✓	✓
Qualcomm Snapdragon 865	Mi 10T[4]	5.2	✓	✓	✓	×	×	×
Apple/USI 339S00761	iPhones 12[4], 13[4]	5.2	✓	✓	✓	×	×	×
Intel AX201	Portege X30-C[4]	5.2	✓	✓	✓	×	×	×
Broadcom BCM4389	Pixel 6[4]	5.2	✓	✓	✓	×	×	×
Intel 9460/9560	Latitude 5400[4]	5.0	✓	✓	✓	×	×	×
Qualcomm Snapdragon 835	Pixel 2[4]	5.0	✓	✓	✓	×	×	×
Murata 339S00199	iPhone 7[4]	4.2	✓	✓	✓	×	×	×
Qualcomm Snapdragon 821	Pixel XL[4]	4.2	✓	✓	✓	×	×	×
Qualcomm Snapdragon 410	Galaxy J5[4]	4.1	✓	✓	✓	×	×	×

Moreover, Table 2.4's list of vulnerable chips and devices represents a *lower bound*. We cannot test all Bluetooth devices in the market. However, we are confident that most of them are flawed, as the BLUFFS attacks exploit architectural issues of Bluetooth session key derivation. We can confidently infer that all untested devices employing an exploitable chip from Table 2.4 are vulnerable. For instance, since the Apple H1 chip is in our list; we can predict that the other devices embedding H1 are also affected, e.g., AirPods gen. 2 and 3, AirPods Max, Beats Solo Pro, Powerbeats (2000), Powerbeats Pro, and Beats Fit Pro [203]. Hence, there is a need for a usable countermeasure to fix the BLUFFS attacks by-design, and we address this challenge in Section 2.8.

2.8 Enhanced LSC KDF

Motivated by the impact of our attacks (e.g., results from Section 2.7.2), we present an *enhanced LSC KDF* addressing the six BLUFFS attacks and their four root causes at the architectural level. Our KDF uses authenticated and mutual key derivation and is backward compliant with KDF_{LSC} (Section 2.8.1). We show how to integrate our countermeasure in the Bluetooth standard while entailing minimal computation, throughput, and latency overheads (Section 2.8.2). The fix also aligns with best practices for symmetric key derivation, such as NIST SP 800–56C–rev2 [23]. We report how we successfully tested our fix at the protocol level. Based on our results, we recommend its introduction in the Bluetooth specification (e.g., via an amendment). We also discuss low-cost *implementation-level* mitigations that vendors can employ until the standard is updated (Section 2.8.4).

2.8.1 Design

Figure 2.4 shows the *message sequence chart* of our enhanced KDF which extends KDF_{LSC}, described in Figure 2.1, in four ways:

1. Adds EKD, a feature flag to negotiate our KDF, as shown by the first two messages in Figure 2.4. This flag provides *backward compatibility* as it accommodates devices supporting and not supporting our protocol. It can also enforce the usage of our protocol across sessions, avoiding (malicious) KDF downgrades. The Bluetooth standard employed the same approach when it introduced SC.
2. Defines SD not as a random number but as a *nonce*, (i,e., number usable once). This definition is valuable as it mandates by design to deny SD's re-usage, regardless of the attacker's strategy.
3. Employs the *mutually authenticated key diversification scheme* presented in Figure 2.4, rather than the unilateral and unauthenticated one from the standard. In particular, Alice sends Bob SD_A (i.e., Central SD nonce) and Bob answers with $Mac(SD_A, PK)$, a message authentication code (MAC) computed from the diversifier and PK to acknowledge and authenticate it. Alice aborts the session if the MAC check fails while Charlie cannot produce such MAC since she does not know PK. Then, the protocol enforces a similar exchange of messages from Bob to Alice involving SD_B (i.e., Peripheral SD nonce) and $Mac(SD_B, PK)$. After exchanging these messages, Alice and Bob mutually set and authenticate the session key diversifiers.
4. Uses the $MKDF_{LSC}$ *mutual key derivation function* to compute mutually diversified SK, unlike KDF_{LSC} that allows a single (malicious) party to diversify SK. In particular, $MKDF_{LSC}$ binds SK to SD_A and SD_B, the authenticated nonces sent by Alice and Bob.

Our enhanced KDF fixes the four attack root causes presented in Section 2.5.2. RC1: The key diversification is *mutual* as SK depends on contributions from the Central and the Peripheral (i.e., SD_A and SD_B). RC2: The diversifiers are defined as *nonces* rather than random numbers. RC3: The negotiation of the diversifiers is *integrity protected* using message authentication codes keyed with PK. RC4: We tolerate (malicious) LSC to SC downgrades by providing a stronger LSC key derivation protocol.

Our scheme stops the six BLUFFS attacks regardless of the attacker's role (CI, PI, or MitM) and target security mode (LSC or SC). In particular, the attack strategy presented in Figure 2.2 becomes ineffective, as the victim asks the other party to authenticate SD with PK and aborts

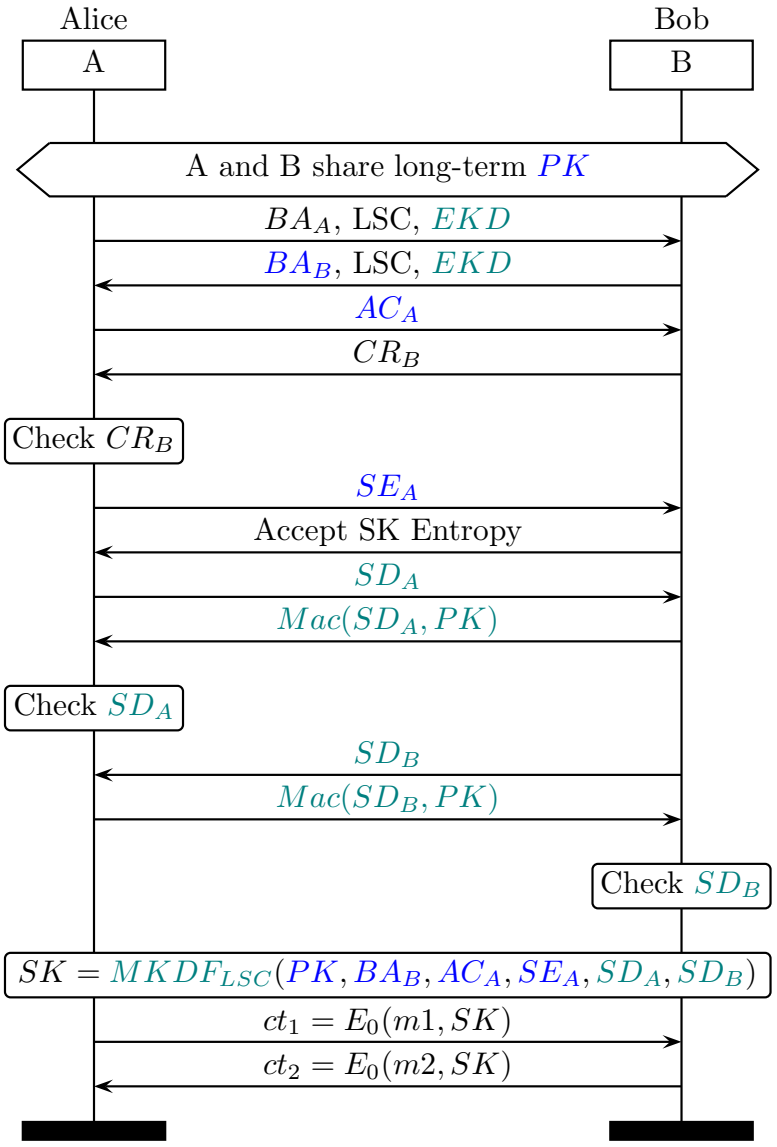

Figure 2.4: **Enhanced LSC session key derivation.** Alice and Bob negotiate the enhanced KDF via EKD to maintain backward compatibility with KDF_{LSC}. They mutually exchange, authenticate, and check session key diversification nonces (i.e., SD_A, SD_B, $Mac(SD_A, PK)$, $Mac(SD_B, PK)$). Then, use the diversifiers in a mutual key derivation function (i.e., $MKDF_{LSC}$) to compute fresh and non-reusable keys across sessions. Our enhanced KDF fixes the BLUFFS attacks and their four root causes by design.

session establishment if authentication fails. Furthermore, the fix prevents the attacks *even if* the attacker successfully authenticates (e.g., by stealing PK), as the attacker cannot control the victim's SD to force a known SK.

Despite being designed to address the BLUFFS vulnerabilities, our KDF mitigates the KNOB attacks and stops the BIAS attacks. The KDF increases the SK brute force effort *exponentially* with the negotiated entropy and *linearly* with the number of target sessions as the attacker must brute force a *new* SK for each session, other than a single SK regardless of the number of target sessions. Hence, our KDF is effective even if the attacker can brute force SK_C as her effort to target n sessions increases from 2^{56} to $n \times 2^{56}$. Moreover, it blocks the BIAS attacks as an adversary who managed to skip PK authentication (e.g., by attacking a victim not patched

against BIAS) cannot bypass our mutually authenticated key derivation protocol without knowing PK.

2.8.2 Integration in the Bluetooth Specification

Our fix requires backward-compliant modifications to the Bluetooth standard (e.g., LSC session establishment) and produces minimal overhead (e.g., one extra negotiation bit, three extra LMP packets carrying in total 48 bytes of extra data to authenticate the diversifiers, three extra function calls to compute the MACs and SK). We now describe these modifications in detail.

EKD requires adding a new LMP feature that should be stored in the firmware and optionally in the OS. For instance, the standard could introduce an EKD flag usable to negotiate our enhanced KDF during LMP feature exchange (as in Figure 2.4). Moreover, a device can enforce EKD usage among sessions and refuse to connect with a device not supporting it.

Mandating nonces rather than random looking SDs requires straightforward textual modification to the standard. For example, instead of defining SD as EN_RAND [33, p. 637], the standard should define it as EN_NONCE. Or when talking about SDs, the document should classify them as “nonces” other than “random numbers”.

Authenticating SD is also easy to implement as during session establishment Alice and Bob already share PK. In particular, we recommend computing the MACs reusing the e_1 authentication function from the standard [33, p. 975] as follows:

$$Mac(SD_A, PK) = e_1(PK, SD_A, BA_B) \tag{2.2a}$$
$$Mac(SD_B, PK) = e_1(PK, SD_B, BA_A) \tag{2.2b}$$

$MKDF_{LSC}$ is a backward compatible extension of KDF_{LSC} presented in Equation 2.1. COF and ISK are computed as in KDF_{LSC} (i.e., Equations 3a and 3b). Then, we add Equation 3c to bind the session key to SD_B by computing a second intermediate session key ISK', reusing the e_3 key generation function [33, p. 981]. In Equation 3d, we reuse e_s to reduce the session key entropy as usual and produce SK. In summary, $MKDF_{LSC}$ is described by the following four equations:

$$COF = e_1(PK, AC_A, BA_B) \tag{2.3a}$$
$$ISK = e_3(PK, SD_A, COF) \tag{2.3b}$$
$$ISK' = e_3(ISK, SD_B, COF) \tag{2.3c}$$
$$SK = e_s(ISK', SE_A) \tag{2.3d}$$

Lastly, we propose two extensions of the LMP protocol to mutually generate and authenticate SD. First, the `LMP_start_encryption_req` command (opcode `17`) which now is used to send SD from the Central [33, p. 638], should be usable also by the *Peripheral* to send its diversifier nonce. Second, we require a *new* LMP command, defined as `LMP_start_encryption_res`, to send a 16 Byte MAC authenticating an SD. Indeed, if Alice is the Central and Bob the Peripheral, we expect the following four LMP messages:

1. Alice: LMP_start_encryption_req(SD_A)
2. Bob: LMP_start_encryption_res($Mac(SD_A, PK)$)
3. Bob: LMP_start_encryption_req(SD_B)
4. Alice: LMP_start_encryption_res($Mac(SD_B, PK)$)

2.8.3 Protocol Level Evaluation

The BLUFFS toolkit includes a Python implementation of our enhanced LSC session key derivation (see `checker/mkdf.py`). We used our implementation to empirically confirm at the *protocol-level* that the BLUFFS attacks are not effective by testing the same attack scenarios exploited in Section 2.5 using the attack strategy in Figure 2.2. Hence, the presented KDF stops the attacks and their root causes (i.e., exploited vulnerabilities) by design.

As shown in `checker/mkdf_tests.py`, the attacker controls AC_C (AU_RAND_C), SD_C (EN_NONCE_C), SE_C (ENTROPY_C). However, she *cannot* authenticate the victim's session key diversifier (MAC_V) as she does not know PK (LK). Even if the adversary manages to bypass SD mutual authentication, she cannot force a known SK_C as she does not control SD_V (EN_NONCE_V). As a result, the attacker cannot conduct the BLUFFS attacks, regardless of her role (Central or Peripheral) and the type of spoofed victim (LSC or SC).

2.8.4 Implementation Level Mitigations

SC-to-SC enforcement Enforcing SC mode between pairing and session establishment stops the attacks when both victims support SC. One can implement this enforcement in the OS (i.e., Bluetooth Host) by storing a SC flag for each paired device and checking that flag during session establishment. As a result, if the attacker impersonates a SC device, the victim can check whether or not the impersonated device supports SC and abort the session when the attacker negotiates LSC. From Table 2.4 – Note 4, we see that ten SC devices already implement this fix. Unfortunately, this mitigation only covers SC-to-SC attack scenarios that currently are less prevalent than LSC-to-SC and LSC-to-LSC ones.

LSC SD cache A device can stop the presented attacks by maintaining a cache of seen SD (i.e., LSC session key diversifiers) and refusing a connection with a (malicious) device proposing a SD in the cache. One can implement this cache in the Bluetooth firmware (i.e., Bluetooth Controller), as SD is not visible by the OS. Unfortunately, this mitigation could be brittle as the cache is unauthenticated. For instance, an adversary can poison the cache with dumb SDs and then negotiate the target SD, which is no more in the cache.

LSC Central authentication A device can stop the PI attacks by requiring an attacker in the Central role to authenticate PK. One can implement this check in the Bluetooth firmware by updating the session establishment code in a backward compliant way. As a result, the attacker cannot complete LSC session establishment, as she cannot authenticate PK. This mitigation is implemented by the Logitech BOOM 3 speaker, as shown in Table 2.4 and detailed in Note 1. Notably, Central authentication only protects against the two PI attacks.

2.9 Listings

Here we present the Listings referenced in the paper.

Listing 2.1: Patch to refuse Peripheral's role switch requests.

```
@ Jumped from 0xA643C (handleLmpSwitchReq)
@ Load second parameter for isMssInstantPassed
ldr r1, [r6, #0x0]
```

```
@ Call isMssInstantPassed
bl #0XA63FE
@ Set return value to True
mov r0, #0x1
@ Jump to ROM at 0xA643C+7 in Thumb-2 mode
b #0xA6443
```

Listing 2.2: Parser's `LmpBase` Class

```
class LmpBase(object):
  """Base Class for LMP Parsing"""
  def __init__(self, pkt):
    self.number = int(pkt.number)
    _tinit = int(pkt.h4bcm.btbrlmp_tid)
    self.tinit = LMP_TRANS_INIT[_tinit]
    self.op = int(pkt.h4bcm.btbrlmp_op)
    if self.op == 127:
      _op_ext = int(pkt.h4bcm.btbrlmp_eop)
      self.op_ext = _op_ext
      self.op_str = LMP_OP_EXT[self.op_ext]
    else:
      self.op_str = LMP_OP[self.op]
```

Listing 2.3: Parser's `LmpAuRand` Class

```
class LmpAuRand(LmpBase):
  """Parse LMP_au_rand"""
  def __init__(self, packet):
    super().__init__(packet)
    self.aurand = packet.h4bcm.btbrlmp_rand
    self.aurand_ba = bytearray.fromhex(
        self.aurand.replace(":", ""))
```

Listing 2.4: Excerpt of Kdf's `kdf` function

```
def kdf(LK, AU_RAND, EN_RAND, BTADDR, ENTROPY):
    """Generate KcPrime"""
    BTADDR.reverse()
    _, COF = e1(LK, AU_RAND, BTADDR)
    log.debug("COF: {}".format(repr(COF)))
    # NOTE: redo reverse as it is passed by reference
    BTADDR.reverse()
    Kc = e3(LK, EN_RAND, COF)
    log.debug("Kc: {}".format(repr(Kc)))
    Kc.reverse()
    KcPrime = Kc_to_Kc_prime(Kc, ENTROPY)
    KcPrime.reverse()
    return KcPrime
```

Listing 2.5: Analyzer's `gen_analysis` function

```
def gen_analysis(PCAP, LK, EXP_SK, BTADD_P):
    """Generate list of sessions and reports"""
    sessions = gen_sessions(PCAP)
    reports = []
    for session in sessions:
        report = gen_report(session, LK, BTADD_P)
        reports.append(report)
    i = 1
    for report in reports:
        print(f"## Begin session: {i}")
        if "keysize" in report:
            print(f"keys: {report['keysize']}")
        if "enrand" in report:
            print(f"enr: {report['enrand'].hex()}")
        if "aurand" in report:
            print(f"aur: {report['aurand'].hex()}")
        if "sk" in report:
            print(f"sk ses: {report['sk'].hex()}")
            # NOTE: check constant SK
            if report["aurand"] == BA_16_ZEROS
              and report["enrand"] == BA_16_ZEROS:
                print(f"sk exp: {EXP_SK.hex()}")
                assert report["sk"] == EXP_SK
        print(f"## End session: {i}\n")
        i += 1
```

2.10 Related Work

Attacks on Bluetooth session establishment Our work is the first presenting attacks breaking Bluetooth's forward and future secrecy and persisting across sessions. Other attacks on session establishment showed that session entropy negotiation is vulnerable to downgrade attacks reducing the strength of SK to 1 byte [18]. The standard now mandates a minimum entropy value of 7 bytes, but recent work showed that some devices classes still accept 1 byte of entropy [13]. Other work uncovered how to bypass session authentication [126, 14]. Recent work analyzed how to employ KNOB+BIAS to exploit different Bluetooth profiles within the same session [1] and the vehicular ecosystem [13].

Attacks on Bluetooth pairing Several works targeted Bluetooth pairing (i.e., the SSP protocol), while in this work, we assume that it is not under attack. In particular, there are SSP's probabilistic invalid curve attacks [28], MitM attacks [184, 98], and cross-transport key derivation attacks [16]. Moreover, related work targeted SSP association [107, 198] and the legacy pairing protocol [129, 179, 111]. We note that attacks on pairing are more challenging to perform and less stealthy than ones on session establishment as pairing is a one-time procedure involving user interaction.

Bluetooth tracking attacks The fact that Bluetooth addresses are not randomizable not only helps to perform the BLUFFS attacks but also enables device tracking threat where an adversary

violates the victim's privacy by tracking his movements using the Bluetooth address as a permanent identifier [205, 101, 105, 60]. Researchers proposed similar attacks for BLE, despite its usage of allowlists and address randomization [219].

Bluetooth firmware research Bluetooth firmware are essential for security research as they implement pairing and session establishment. However, they are proprietary and closed-source, requiring significant reverse-engineering effort to be analyzed and patched. Luckily, researchers have developed tools to inspect and patch popular Bluetooth firmware. For example, Internalblue [135] provides a Python API to interact and patch popular Broadcom and Cypress firmware. Other work focused on Bluetooth firmware's automated extraction of security-related parameters [183], detection of link-layer vulnerabilities [201] and weaknesses in random number generation [185].

Bluetooth fuzzing and implementation bugs We discovered the protocol-level BLUFFS attacks and their root causes by inference from the Bluetooth specification. Then, we automated the repetitive tasks by developing a toolkit. Other research work used directed fuzzing to find crashes, denial of service (DoS), and remote code execution (RCE) *implementation-level* bugs in popular Bluetooth stacks [161, 92, 151]. Other recent work employs differential testing to catch protocol compliance implementation bugs [115] automatically. There are works uncovering implementation-level vulnerabilities resulting in RCE using semi-automated techniques. Notable examples are: BlueBorne [175] impacting Amazon Echo and Google Home, BlueFrag [77] against Android 9, Bleedingbit [176] on Texas Instrument BLE chips, and BleedingTooth [145] targeting BlueZ and the Linux kernel. Unlike the exploits described in this paragraph, the BLUFFS attacks can target a device *regardless of* its implementation details (and bugs).

Forward/future secrecy Forward and future secrecy were extensively studied for Transport Layer Security (TLS) and Instant Messengers (IM). TLS's forward secrecy was evaluated in the wild [106] and TLS 1.3 mandates it using non-static cipher suites, such as ephemeral Diffie-Hellman (DH) key exchange [109]. The double ratchet algorithm [154], used by the most popular IMs (e.g., Signal, and WhatsApp), provides future secrecy with the DH ratchet and forward secrecy with the symmetric ratchets and was analyzed by security researchers [58]. No prior work evaluated Bluetooth's forward and future secrecy properties (not even the Bluetooth standard).

Survey on Bluetooth security There are not so recent survey papers about Bluetooth security [131, 150, 139, 74]. They are an excellent way to get introduced to Bluetooth's security architecture and related threats. However, none of them discusses Bluetooth's forward and future secrecy guarantees.

2.11 Conclusion

This paper presents the first security evaluation of Bluetooth *forward* and *future secrecy* guarantees. It uncovers two new vulnerabilities in Bluetooth's *session establishment*, enabling to reuse of a weak session key across sessions. We show how to exploit these flaws in six attack scenarios to impersonate and MitM arbitrary devices across sessions. Our attacks break Bluetooth's forward and future secrecy as they compromise past and future encrypted messages with novel key reuse attacks. Our findings result from experiments with Bluetooth session establishment on

actual devices and inference from the standard. We focused on SK as, unlike PK, it can be targeted without user interaction, and its entropy can be lowered without violating the standard.

We provide BLUFFS, a low-cost and reproducible toolkit to implement, detect, and fix the attacks. The toolkit includes seven original *patches* to manipulate session key derivation and monitor SKs by patching the attack device's Bluetooth firmware. It also ships *parsing* and *analysis* scripts to detect the attacks from a pcap file. We use our toolkit to evaluate the BLUFFS attacks on a large scale. We exploit *eighteen* devices embedding *seventeen* Bluetooth chips from leading hardware and software vendors and estimate the attacks' impact. For example, our threats are effective in all scenarios where at least one of the victims supports LSC and even in scenarios where the victims support SC. These results translate into millions of exploitable devices.

To address the attacks' critical impact, we develop and test a *protocol-level* countermeasure preventing *by-design* the BLUFFS attacks and their root causes. We design an enhanced KDF for LSC employing *fresh*, *mutual*, and *authenticated* session key derivation. We show how to update the LMP protocol and KDF_{LSC} to integrate our fix in a backward compliant way and with minimal overheads. Specifically, we require one extra LMP command, 48 extra bytes sent over the air, 3 specification-compliant function calls, and minimal textual modifications to the standard. We successfully tested our KDF at the protocol level and released it as part of our BLUFFS toolkit. We hope our fix will soon be added to the standard and implemented by the vendors. Moreover, we recommend to vendors implementation-level mitigations that can be adopted while waiting for an update to the standard.

From this work, we learned three key lessons that we want to share: (i) we should pay more attention to session establishment vulnerabilities, attacks, and fixes effective across sessions, (ii) we should agree on the definitions of Bluetooth's forward and future secrecy and update the standard to discuss these definitions and related risks, (iii) we need open-source Bluetooth firmware (Controllers) and better tooling around them to improve the effectiveness, coverage, and speed of our offensive and defensive evaluations.

Chapter 3

CTRAPS: CTAP Impersonation and API Confusion on FIDO2

3.1 Abstract

FIDO2 is a popular technology for single-factor and second-factor authentication. It is specified in an open standard including the WebAuthn and CTAP application layer protocols. We focus on CTAP which allows the communication between FIDO2 clients and authenticators. No prior work explored the CTAP Authenticator API which is a critical protocol-level attack surface as it deals with credential creation, deletion, and management. We address this gap by presenting the first security and privacy evaluation of the CTAP Authenticator API. We uncover two classes of CTAP protocol-level attacks we call CTRAPS.

The client impersonation (CI) attacks exploit the lack of client authentication to tamper with FIDO2 authenticators. They include zero-click attacks capable of deleting FIDO2 credentials, including passkeys, without user interaction. The API confusion (AC) attacks abuse the lack of protocol API enforcements and confound FIDO2 authenticators, clients, and users into calling unwanted CTAP APIs while thinking they are calling legitimate ones. For example, a victim thinks is authenticating to a website, when they are deleting their credentials. The CTRAPS attacks are conducted either in proximity or remotely and are effective regardless of the underlying CTAP transport (USB, NFC, or BLE).

We detail the eight vulnerabilities in the CTAP specification enabling the CTRAPS attacks. Seven of them are novel and include unauthenticated CTAP clients and trackable FIDO2 credentials. We release `CTRAPS`, an original toolkit to analyze CTAP and conduct the CTRAPS attacks. We confirm the attacks' feasibility by exploiting six popular authenticators, including a FIPS-certified one, from Yubico, Feitian, SoloKeys, and Google, and ten widely used relying parties, such as Microsoft, Apple, GitHub, and Facebook. We discuss eight backward-compliant countermeasures to fix the attacks and their root causes. We responsibly disclosed our findings to the FIDO alliance and the affected vendors.

3.2 Introduction

Fast IDentity Online v2 (FIDO2) is the de-facto standard for single-factor (passwordless) and second-factor (2FA) authentication. Google, Dropbox, and GitHub [124] designed FIDO to offer

a practical and scalable solution for authentication. FIDO has been widely adopted by industries and organizations including Apple, Microsoft, and the US government [83]. Market forecasts predict the FIDO market to rapidly grow from USD 230.6 million in 2022 to USD 598.6 million in 2031 [188]. Yubico, a FIDO authenticator market leader, sold more than 22 million YubiKey authenticators [213]. This growth will continue because of the recent industry-wide push towards single-factor passkey-based authentication [84, 61, 163].

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* who wants to authenticate to the relying party and manages the communication between the authenticator and the relying party. Typically, the authenticator is a dongle, the relying party is a web server, and the client is a web browser or a mobile app.

The authenticator and the client communicate using the *Client to Authenticator Protocol (CTAP)*. CTAP works at the application-layer and is transported over Universal Serial Bus (USB), Near Field Communication (NFC), or Bluetooth Low Energy (BLE). It exposes the client to the *CTAP Authenticator API*, usable to interact with the authenticator, e.g., credential creation, management, and deletion. These API calls might require User Verification (*UV*) and User Presence (*UP*) authorization.

This work focuses on the CTAP protocol and its security and privacy guarantees. There are only a few research studies about CTAP. The authors of [21] performed a provable security analysis on CTAP, highlighting unauthenticated DH key exchange. In a follow-up work [22], they proposed an impersonation attack exploiting CTAP to register an authenticator with an arbitrary relying party. The authors in [96] present a Machine-in-the-Middle (MitM) attack on CTAP resulting in a privacy leak. Other works target the authenticator with fault injection and side channel attacks [123, 160].

No prior work investigated the FIDO2 *CTAP Authenticator API*. This API is a critical protocol-level attack surface as it enables the creation, management, and deletion of credentials and the administration of authenticators. FIDO2 credentials are security and privacy critical as they authorize access to popular online services, including, social media, banking, data sharing, and e-commerce. A protocol-level attack on the CTAP Authenticator API would enable access to and manipulation of any credential stored on any authenticator, regardless of the authenticator's hardware and software details. Hence, it is crucial to assess the API's expected security and privacy properties and if they hold in practice.

We fill this gap by presenting the first security and privacy assessment of the CTAP Authenticator APIs. We uncover two attack classes and eleven related attacks on CTAP that we call **CTRAPS**. The *client impersonation (CI)* attacks exploit the lack of client authentication to tamper with an authenticator. Among others, they allow factory resetting an authenticator without user interaction. The *API confusion (AC)* attacks abuse the lack of protocol API enforcements and confound a FIDO2 authenticator, a client, and a user into calling unwanted CTAP Authenticator APIs while believing they are calling legitimate ones. For instance, a user thinks to be authenticating to a website but they are instead deleting their authenticator credentials.

We consider two attacker models: a CI attacker impersonating a CTAP client and an AC attacker with a MitM position between the client and the authenticator. The adversaries perform the attacks in *proximity* or *remotely*. They do not require physical access to the authenticator, e.g., no side channel or fault injection. Moreover, they do not need to compromise the client or the authenticator, e.g., no client or authenticator malware.

The CTRAPS attacks have a *critical* and *widespread* impact on the FIDO2 ecosystem. They are critical as they violate the security, privacy, and availability of FIDO2 devices. For example, a CI or an AC attacker can factory reset an authenticator deleting all FIDO2 credentials and locking out the victim from the related service. Despite targeting CTAP, the attacks also impact FIDO2 relying parties. For example, they invalidate the non-discoverable credentials stored by the relying party. They are widespread as they exploit protocol-level vulnerabilities in the CTAP application-layer protocol. Hence, they can be conducted against any FIDO2 device regardless of whether CTAP is transported over USB, NFC, or BLE.

We isolate *eight vulnerabilities* in the CTAP specification enabling the CTRAPS attacks. Seven of them are novel within FIDO2. They include unauthenticated CTAP clients, trackable credentials, and weak authorization of (destructive) API calls. The vulnerabilities are *severe* as they affect authenticators and clients implementing CTAP v2.0, v2.1, and v2.2. We also find and disclose an implementation flaw on Yubico's authenticator firmware, allowing an attacker to leak sensitive data and track users. Yubico addressed the problem and assigned it CVE-2024-35311 [216].

We present `CTRAPS`, a new toolkit to experiment with CTAP and conduct the CTRAPS attacks. The toolkit has three modules: CTAP testbed, CTAP clients, and Wireshark dissectors. The testbed provides virtual clients and relying parties, enabling local testing of the attacks without the involvement of actual devices. The CTAP clients module performs the CI and AC attacks. We implemented them to work from proximity and remotely. Our CTAP clients allow the testing of the attacks on real-world authenticators and clients. For example, we release an Android app and a Proxmark3 script to test the CI attacks over NFC. The dissectors module includes an enhanced FIDO2 dissector for Wireshark praising new and useful packet information such as status codes and support for credential management.

We evaluate popular FIDO2 authenticators, clients, and relying parties. We deploy them from proximity and remotely, testing different CTAP transports (USB and NFC). We attack *six authenticators* from Yubico, Feitian, SoloKeys, and Google. One authenticator from Yubico is FIPS-compliant, meaning that it utilizes cryptographic algorithms guaranteeing strict security standards. We also exploit *ten relying parties* offering passkeys and second-factor authentication, including Microsoft, Apple, GitHub, and Facebook.

We discuss eight backward-compliant countermeasures that fix the CTRAPS attacks and their root causes. The fixes include CTAP client authentication, 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. The countermeasures are backward-compliant as they rely on mechanisms already available in the authenticator (e.g., PIN and LED) and do not require extra hardware (e.g., adding a display).

We summarize our contributions as follows:

- We perform the first assessment of the CTAP Authenticator API. We unveil two classes of CTAP protocol-level attacks: CI and AC. The attacks compromise the security, privacy, and availability of the FIDO2 ecosystem. For instance, they (remotely) delete FIDO2 credentials, track users via FIDO2 credentials, and DoS authenticators. They are enabled by eight CTAP protocol level vulnerabilities, seven of which are new.
- We provide a toolkit to evaluate the CTAP Authenticator API and test our attacks in a virtual environment and on actual devices. We successfully conduct our attacks against six authenticators, two transports, and ten relying parties.

- We design eight backward-compliant countermeasures to fix our attacks and their root causes. We also responsibly disclosed our findings to the FIDO2 Alliance and affected vendors.

Responsible disclosure. We responsibly disclosed our findings to the FIDO Alliance in November 2023 [7]. They acknowledged our report and shared it with their members. In May 2024, they provided feedback highlighting that the CI and AC attacks deployed by a proximity-based attacker are less scalable than remote ones. They argued on the effectiveness of CTRAPS attacks against authenticators running on a TEE. They also discussed the possible addition of our attacks to FIDO's threat model.

In December 2023, we reported our findings to the affected authenticator manufacturers (i.e., Yubico, Feitian, SoloKeys, and Google). Google confirmed our findings, assigning them priority P2 and severity S2. They responded that our attacks required a compromised FIDO client and closed the issue without resolution. We argue that Google's assessment is incorrect as our attacks do not require a compromised FIDO client. Yubico confirmed the implementation bug we found, pushed a fix in production, published a security advisory [217], and assigned it CVE-2024-35311 [216]. 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.

Ethics and availability. We conducted our experiments ethically. We evaluated our authenticators and accounts. We did not collect personal data and involved third parties. To advance open science, we open source our contributions, including the `CTRAPS` toolkit, found at https://github.com/Skiti/CTrAPs.

3.3 Background and System Model

We introduce FIDO2, CTAP, and our system model.

3.3.1 FIDO2

FIDO2 [5] is an open and pervasive standard for single-factor and multi-factor authentication. It is managed by the FIDO Alliance. FIDO2 has four entities: an authenticator, a client, a user, and a relying party. In a typical scenario, a user connects their authenticator to the client 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 communication channel to a relying party and a client. Its latest version is WebAuthnL2 [199]. CTAP, the focus of this work, enables a secure connection between a FIDO2 authenticator and a client via the CTAP Authenticator API. For example, the `MakeCred` API registers a new credential while the `GetAssertion` API authenticates a credential.

A FIDO2 *credential* is a key pair used to sign and verify authentication challenges to authenticate a user. The digital signature is computed using standard techniques, like Elliptic Curve Digital Signature Algorithm (ECDSA). Access to the credential private key is guarded by encryption using a credential master key, which is unique to each authenticator and stored in the authenticator.

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 by the relying party and used for multi-factor authentication.

FIDO2 credentials are associated with a credential identifier (CredId), a relying party identifier (RpId), and a user identifier (UserId). The CredId uniquely identifies a FIDO2 credential and is derived from the credential master key stored in the authenticator. When FIDO2 clients authenticate a credential, they must know its associated CredId. The RpId indicates the relying party with which the credential was registered. It is public as it corresponds to the base domain of the relying party (e.g., `login.microsoft.com`).

The UserId represents the user's online account within the relying party's service. The relying party assigns a random UserId to the user during account registration, and it is shared across all FIDO credentials associated with that user. The optional FIDO2 *CredBlob* extension allows a relying party to store additional metadata inside a credential.

3.3.2 CTAP

The Client-to-Authenticator Protocol (CTAP) is a core part of the FIDO2 standard, alongside WebAuthn. It is an application-layer protocol that defines the communication between a FIDO client and an authenticator. CTAP has considerably evolved since its inception. CTAP1, also known as FIDO U2F (Universal 2nd Factor), introduced a second-factor authentication mechanism to combat phishing. CTAP2.0 maintains backward compatibility with CTAP1 while introducing passwordless (single-factor) authentication. CTAP2.1 [2] adds the credential protection policy, discoverable credential management (i.e., the `CredMgmt` API), and biometric authentication. CTAP2.2 [4], the latest CTAP version still considered a draft, supports hybrid authenticators and QR codes.

CTAP relies on two user authorization mechanisms to secure API calls from the client: (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 3.1 shows the seven *CTAP Authenticator APIs* studied in this paper and their *UV* and *UP* requirements:

- `MC`: `MakeCred` registers a new credential bound to an online account with a relying party.
- `GA`: `GetAssertion` authenticates to a relying party by proving possession of a credential.
- `CM`: `CredMgmt` enumerates, modifies, and deletes the authenticator's discoverable credentials.
- `CP`: `ClientPin` handles *UV* based on a user PIN to be submitted via the client's UI.
- `Re`: `Reset` wipes all discoverable and non-discoverable credentials and generates a new master key.
- `Se`: `Selection` selects an authenticator to operate among the available ones.
- `GI`: `GetInfo` returns the authenticator's details, like manufacturer, transports, extensions, and settings.

The `GetAssertion`, `CredMgmt`, and `ClientPin` APIs have API subcommands. For example, `CredMgmt(GetCredsData)` returns the number of stored discoverable credentials and `CredMgmt(DelCreds)` deletes all discoverable credentials. Some API subcommands, compared to their

Table 3.1: CTAP Authenticator APIs with their *UV* and *UP* authorization requirements, and support for subcommands. Yes[1]: depends on the client and relying party configuration, Yes[2]: depends on API subcommand.

CTAP API	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

original API, have more relaxed requirements. For instance, `ClientPin(KeyAgreement)` does not require *UV*.

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 weak privacy protection.

3.3.3 System Model

We adopt the official FIDO2 system model [5]. Figure 3.1 shows the system model's four entities: authenticator, client, relying party, and user. The user connects the authenticator to the client to authenticate on a service hosted by the relying party. The entities support up to CTAP2.2 and WebAuthnL2 (i.e., the latest and supposedly most secure FIDO2 protocol versions). Next, we describe each entity.

Authenticator. The authenticator is a FIDO2 authenticator: a user device that can be connected to the client (e.g., a USB/NFC dongle). The authenticator runs a CTAP server that exposes the CTAP Authenticator API. The API is accessible over USB, NFC, and BLE. The authenticator supports FIDO2's *UP* and *UV* user authorization mechanisms. It stores discoverable credentials and the credential master key.

Client. The client is a FIDO2 client handling the communication between the authenticator and the relying party. It exposes a CTAP client to the authenticator and a WebAuthn client to the relying party. The client could be a web browser, a mobile app for Android [79] or iOS [80], or a command line tool like the Yubico CLI [215].

Relying party. The relying party is an online service that relies on FIDO2 passwordless or multi-factor authentication. It runs a WebAuthn server that responds to FIDO2 registration and authentication requests. The relying party stores non-discoverable credentials, and user and credential identifiers. The relying party communicates with the client using TLS. Offline operations on the authenticator, like deleting discoverable credentials, indirectly affect the relying party by making the user unable to log into their online service.

User. The user owns an authenticator and a device that runs the FIDO2 client, e.g., a YubiKey dongle and a laptop. They utilize their authenticator to register FIDO2 credentials and

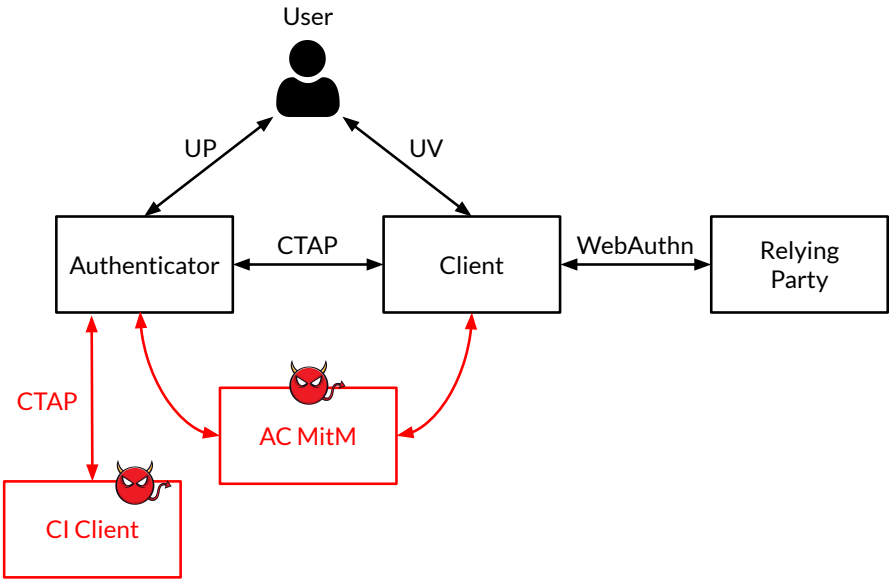

Figure 3.1: **CTRAPS threat model.** The user authenticates to the relying party using a client (e.g., browser) and an authenticator (hardware dongle). The user when needed grants UP by pressing a button on to the authenticator and UV by submitting a PIN to the client. We study two attacker models: (i) a client impersonation attacker targeting the authenticator over CTAP (left), (ii) a MitM attacker in the CTAP channel between the authenticator and the client.

authenticate to the associated relying party. To do so, they connect their authenticator to the client and provide *UV* and *UP*, if necessary. The user manages the authenticator via the client, without connecting to a relying party. For example, they can check their discoverable credentials and change the authenticator's PIN.

3.4 CTRAPS Client Impersonation Attacks

The *CTRAPS CI attacks* target an authenticator while spoofing a client to perform CTAP API calls without user authorization. CI attacks factory reset the authenticator via the `Reset` API, track the user via `GetAssertion`, lock the authenticator via `ClientPin`, and profile the authenticator via `GetInfo`. They exploit five protocol-level CTAP vulnerabilities we found. For instance, the absence of CTAP client authentication facilitates impersonation, the use of NFC transport allows to bypass *UP*, and the lack of *UV* when calling `Reset` enables unauthorized factory resets.

The attacks advance the state of the art in FIDO2's security and privacy by introducing client impersonation. This is a new class of attacks previously unseen in FIDO2. The CI attacks require limited or no user interaction, depending on the CTAP transport. For example, by using NFC, they bypass *UP*, leading to zero-click attacks. The CI attacks also involve *no client compromise*, being deployed from a client owned by the attacker. Next, we introduce the CI attacker model and describe the attacks.

3.4.1 CI Attacker Model

The CI attacker model assumes an attacker impersonating a CTAP client to the victim's authenticator, referenced as CI Client in Figure 3.1. The attacker is in proximity of the victim's authenticator, or can remotely connect to it. They have no physical access to and do not tamper with the

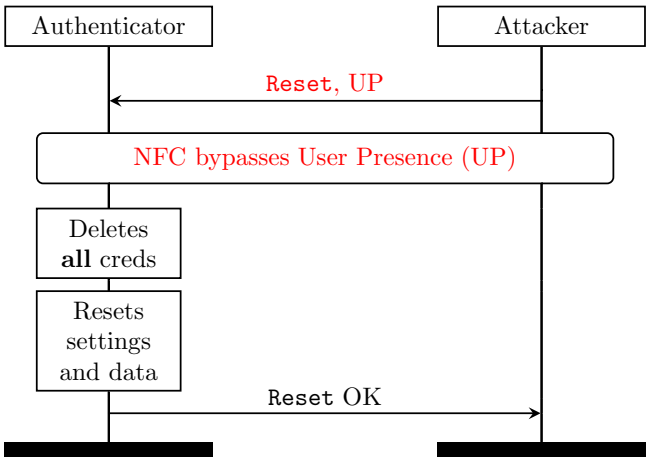

Figure 3.2: **CI_1 attack.** Factory reset authenticator via `Reset`. While in NFC range, the attacker calls the `Reset` API. Over NFC, the authenticator skips *UP* and instantly factory resets, deleting all of its discoverable and non-discoverable credentials.

victim's client and authenticator. They do *not* install malware on the victim's device running the FIDO2 client.

The CI attacker model maps to several relevant attack scenarios. For example, they can approach a target authenticator over NFC while impersonating a client (e.g., via a smartphone or a Proxmark), place a malicious NFC device in a place where a user might touch it with an authenticator (e.g., under a table), They can also communicate with the victim's authenticator using a compromised hardware device, such as a USB hub that connects the user's machine and the authenticator, or virtual USB peripheral, through a setup similar to [43].

3.4.2 CI Attacks Description

We describe the four CI attacks, which we label CI_1, CI_2, CI_3, and CI_4.

CI_1: Factory reset authenticator. In CI_1, the attacker abuses the `Reset` API to factory reset an authenticator, as shown in Figure 3.2. The attacker connects to the authenticator and, without authenticating, issues a factory reset command (which requires *UP*). Over USB, the attack requires one click (*UP*) and the authenticator having been plugged into the USB port within the last ten seconds. Over NFC, the attacker achieves zero-click reset by exploiting a CTAP quirk intended to enhance usability. That is, NFC communication inherently implies user presence, allowing *UP* to be bypassed. The execution of the factory reset wipes out all credentials, even the non-discoverable ones stored by the relying party, as it erases the credential master key necessary for decryption. It also deletes the authenticator's settings, including the PIN, user preferences, and stored data. Then, the authenticator confirms the successful reset.

CI_2: Track user from credentials. In CI_2, instead of using `GetAssertion` for authentication, the attacker exploits it to leak identifying data and track the user, as shown in Figure 3.3. CI_2 requires a pre-determined list of RpId for which the attacker aims to leak credentials. This is straightforward, as this information is publicly accessible. Although the `GetAssertion` API requires both *UV* and *UP*, the attacker can circumvent both authorizations, resulting in a *zero-click* data leak and enabling user tracking. They bypass *UP* by issuing a `GetAssertion` command containing the *up=false* option. They bypass *UV* by only targeting relying parties that register credentials using the weak and default *CredProtect=UVOptional* policy, such as Microsoft and

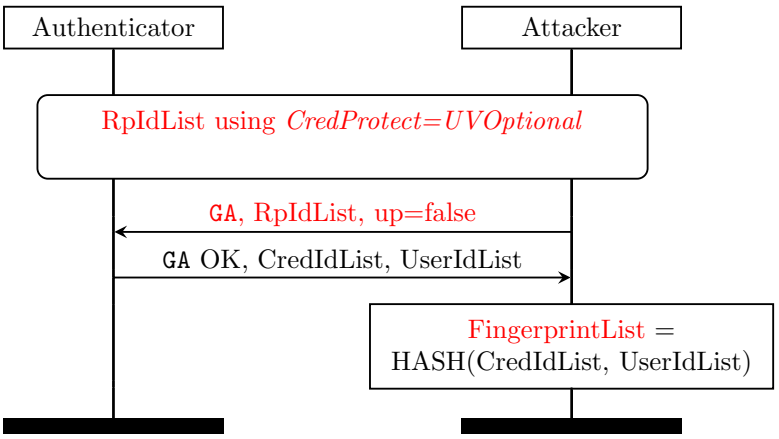

Figure 3.3: **CI_2 attack.** Track user from credentials via `GetAssertion`. The attacker connects to the authenticator and calls the `GetAssertion` API (`GA` in the figure). They skip *UV* by targeting relying parties using the weak and default *CredProtect* default policy and skip *UP* by passing *up=false*. The authenticator returns a list of credential and user identifiers, used by the attacker to fingerprint the authenticator and track the user.

Apple. Executing `GetAssertion` returns a list of credential and user identifiers. These identifiers can be used to fingerprint the user and to track them over multiple connections by performing CI_2 each time and looking for matching fingerprints. CI_2 also works on credentials protected by stronger policies (i.e., *CredProtect=UVRequired* and *CredProtect=UVOptionalWithCredIDList*), but requires *UV* or knowledge of the credential identifiers.

CI_3: Force authenticator lockout. In CI_3, the attacker abuses the `ClientPin` API, protecting the authenticator from PIN brute-forcing, to lock the authenticator or even force a factory reset. They submit to the authenticator several wrong PIN guesses in a row via the `ClientPin(GetPinToken)` subcommand. After three wrong guesses, the authenticator enters a soft lock mode preventing 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 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.

CI_4: Profile authenticator. In CI_4, the attacker calls `GetInfo` to leak the authenticator's technical details. This attack can be used as a stepping stone to more advanced attacks, to profile the user and track them in future connections, and to assess whether the authenticator is vulnerable to an implementation-specific attack like [216]. The leaked details include the manufacturer, model, and FIDO2 version, and the supported algorithms, transports, options, and extensions. The authenticator also discloses user settings, such as FIDO2 being disabled over a specific transport.

3.5 CTRAPS API Confusion Attacks

The *CTRAPS AC attacks* take advantage of a novel attack technique for FIDO, which we refer to as *API confusion*. API confusion tricks a client, an authenticator, and their user into calling a CTAP Authenticator API while they think they are calling a different one. The called API has the same or lower *UV* and *UP* requirements of the intended API. For example, AC attacks can erase FIDO2 credentials, including passkeys, lock the user out of their authenticator, and track them.

Table 3.2: There are 49 ways to perform AC against 7 CTAP Authenticator APIs. The user intends to call `API A`, instead is tricked into calling `API B`. ✓[1]: proximity-based attacker, ✓[2]: default *CredProtect=UVOptional* if credential protection is enabled, n/a: not applicable.

	CM	Re	GA	MC	CP	Se	GI
CM	n/a	✓[1]	✓	✓[1]	✓	✓	✓
Re	n/a	n/a	✓[2]	n/a	✓	✓	✓
GA	✓	✓	n/a	✓	✓	✓	✓
MC	✓	✓	✓	n/a	✓	✓	✓
CP	✓	✓[1]	✓	✓[1]	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

AC is effective as it does not require social engineering [191] or other deception techniques [134] to trick the user into calling an unwanted API. The user cannot detect an API confusion because it requires expected *UV* or *UP* actions. The AC attacks exploit the eight protocol-level vulnerabilities we outline later. For instance, the absence of authenticator feedback during API calls grants stealthiness and the use of static credential and user identifiers enables user tracking.

No prior work considered the AC attack vector for FIDO2. Existing attacks on FIDO include MitM on the Diffie-Hellman key exchange, CTAP traffic eavesdropping, U2F impersonation, physical access, and side channel attacks on the authenticator. Moreover, the AC attacks target the entire CTAP Authenticator API surface, whereas previous research only focused on `ClientPin` and `MakeCred`. Next, we will introduce the AC attacker model and attacks.

3.5.1 AC Attacker Model

The AC attacker model assumes a MitM attacker between the authenticator and the client, referenced as AC MitM in Figure 3.1. The attacker is either in proximity to the authenticator and the client (e.g., an NFC skimmer) or can contact them from remote (e.g., a remotely controllable USB hub). They are unable to modify the authenticator's firmware or compromise a legitimate FIDO2 client and relying party. They have no physical access to the authenticator.

An AC attacker model has several associated real-world attack scenarios. For example, they can get a MitM position over NFC interposing an NFC skimmer between the client and the authenticator. They can achieve a MitM position over USB with setups such as those discussed in [193] and [120]. For instance, the attacker can remotely compromise a USB device connected to the user's device running the FIDO2 client, such as a USB hub that routes traffic between other USB peripherals.

Alternatively, they can gain privileged access via techniques like UACMe [155] and then leverage USBPcap to USB MitM a victim's Windows machine running the FIDO2 client. The attacker can also install on the user's machine a malicious app exploiting libraries that provide access to USB HID traffic.

3.5.2 AC Technique and Combinations

The seven AC attacks rely on the API confusion attack technique. The attacker intercepts a call to `API A` and changes (i.e., confounds) it to `API B`. This action only requires that `API B` has the same or lower *UV* and *UP* authorization requirements than `API A`. The AC technique has six steps:

1. The user calls `API A` through the client. The API might 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 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 calls `API B` rather than `API A` based on the AC combinations in Table 3.2.
4. If required by `API A`, the attacker obtains *UP* from the user, unable to realize they are under attack. The attacker can only obtain *UP* once, as multiple requests would alarm the user. This step is bypassed whenever NFC proximity implies *UP*.
5. The authenticator executes `API B` and returns a success message.
6. The attacker informs the victim via the CTAP client that `API A` was successfully executed.

The AC strategy is effective on *7* CTAP Authenticator APIS and provides *49* ways to confound the victim as shown in Table 3.2. Multiple (`API A`, `API B`) pairs achieve the same goal. The amount of available pairs depends on their *UV* and *UP* requirements and, in the case of AC_3, 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, AC_1 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.

3.5.3 AC Attacks Description

We describe seven AC attacks labeled AC_1, AC_2, AC_3, AC_4, AC_5, AC_6, and AC_7. AC_1 exploits all possible ways to call `CM`, AC_2 does this with `Re`, and so on.

AC_1: Delete discoverable credentials. In AC_1, the attacker abuses the `CredMgmt` API to delete all discoverable credentials stored on the authenticator, as shown in Figure 3.4. 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*. First, they check the existence of discoverable credentials to erase (StoredCredsAmount) via `CredMgmt(GetCredsMetadata)`. Second, they retrieve the list of relying parties stored on the authenticator (RpIdList) via `CredMgmt(EnumRps)`. Third, they use RpIdList to retrieve the list of stored credential identifiers (CredIdList) via `CredMgmt(EnumCreds)`. Fourth, they use CredIdList to delete all discoverable credentials via `CredMgmt(DelCreds)`. Finally, they falsely return `API A` OK to the user.

AC_2: Factory reset authenticator. In AC_2, the attacker exploits the `Reset` API to factory reset the authenticator, similar to CI_1. Since `Reset` over USB requires *UP*, but not *UV*, an attacker can confound `MakeCred`, `GetAssertion`, and `Selection` into a `Reset` call. An attacker over NFC, able to bypass *UP*, can also confound `CredMgmt`, `ClientPin`, and `GetInfo`.

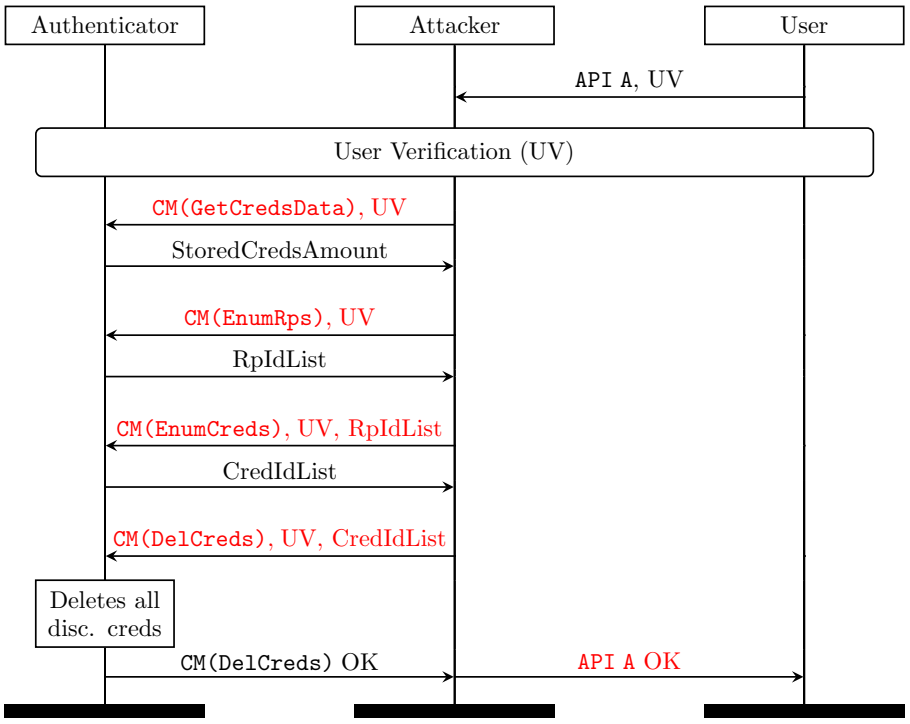

Figure 3.4: **AC_1 attack.** Delete discoverable credentials attack with proximity. The user intends to call `API A`, requiring *UV* but not necessarily *UP*. For example, `GetAssertion`, `ClientPin`, or `MakeCred`. The attacker obtains *UV* from the unsuspecting user. Instead of `API A`, they call `CredMgmt` (`CM` in the figure). They execute four `CredMgmt` subcommands which list and then delete all discoverable credentials on the authenticator.

AC_3: Track user from credentials. In AC_3, the attacker misuses the `GetAssertion` API to leak unique identifiers as fingerprints and track the user, similar to CI_2. They can confound `MakeCred`, *CredMgmt*, and *ClientPin* into a `GetAssertion` call, if they want to access credentials protected by the *CredProtect=UVRequired* or *CredProtect=UVOptionalWithCredIDList* policies. Additionally, the attacker can also confound `Reset`, `Selection`, and `GetInfo` if they only wants to access credentials protected by the weak *CredProtect=UVOptional* default policy.

AC_4: Fill authenticator's credential storage. In AC_4, the attacker repeatedly calls `MakeCred` to register new discoverable credentials, until the authenticator's credential storage is full. They exploit the *rk=true* option to enforce the generation of discoverable credentials over non-discoverable ones. A filled storage compromises the authenticator's availability as the user cannot register new discoverable credentials.

AC_5: Force authenticator lockout. In AC_5, the attacker abuses the `ClientPin` API to lock the authenticator and force a mandatory factory reset, similar to CI_3. Although `ClientPin` requires *UV*, the attacker wants to fail multiple PIN attempts (i.e., they do not need *UV*). Consequently, they can confound any API call into a `ClientPin` call, as they do not need authorization.

AC_6: Authenticator DoS. In AC_6, the attacker calls `Selection` 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), they can prolong the attack.

AC_7: Profile authenticator. In AC_7, the attacker invokes `GetInfo` to retrieve the authenticator's details. Then, similar to CI_4, they use this information as a stepping stone to other attacks,

tracks the user, or checks whether the authenticator is vulnerable to implementation-specific attacks [216]. Not requiring *UV* or *UP*, the attacker can confound any API call into a `GetInfo` call.

3.6 Implementation

In this section, we present `CTRAPS`, a novel toolkit that implements the CTRAPS attacks and enables experimentation with CTAP. The toolkit has *three* modules: a CTAP testbed (Section 3.6.1), four customizable CTAP clients (Section 3.6.2), and an enhanced FIDO2 Wireshark dissector (Section 3.6.3).

The CTAP testbed and the Electron app CTAP client need the user's authorization to connect and communicate with the authenticator. Linux requires adding extra `udev` rules, macOS asks to accept a notification on the screen, and Windows needs admin privileges. This limitation is expected, as it is also present in FIDO2 apps released by authenticator manufacturers, such as the Yubico Authenticator App and the Feitian Authenticator Tool. Now, we will describe the implementation of each module and highlight their novelties.

3.6.1 CTAP Testbed

Our CTAP testbed includes a virtual WebAuthn relying party and a virtual WebAuthn/CTAP client. The testbed can test real authenticators without having to tamper with actual credentials and also launch the CTRAPS attacks. Our relying party and client extend the Yubico open-source Python library for FIDO2 called `python-fido2` [214].

Virtual relying party. The virtual relying party is implemented as a customizable WebAuthn server. It includes standard relying party templates and fast customization of the server's parameters. For example, we implemented a template imitating a Microsoft relying party, including its FIDO2 identifier (i.e., *login.microsoft.com*). The virtual relying party is useful to quickly test real authenticators against CI and AC attacks. For example, we can automatically register credentials with different protection policies on the authenticator.

Virtual client. The virtual client offers a convenient CTAP API, offering low-level access to any CTAP message. It can send CTAP commands in any order, or issue custom and malformed payloads. It can be configured with different CTAP authorization requirements, authentication challenges, and origins.

3.6.2 CTAP Clients

We developed four custom CTAP clients: an Android app performing proximity CI over NFC, an Android app performing remote CI over NFC, a Proxmark3 script that executes proximity CI over NFC, and an Electron app simulating a MitM attacker to test remote AC over USB. We released in the CTRAPS GitHub repository five video demonstrations, showing how to deploy the CTRAPS attacks on real authenticators using our clients.

Android app for proximity CI over NFC. We implemented the proximity CI attacks using an Android app which impersonates a FIDO2 client over NFC. The app runs on a device owned by the attacker and targets any authenticator that comes within the NFC range. For example, it can perform CI_2 to leak identifiers and track the user.

Android app for remote CI over NFC. We utilized an Android app to implement the remote CI attacks over NFC. The app is installed on a device owned by the victim. It spoofs a legitimate NFC app, enticing the user to scan their authenticator (e.g., by asking for FIDO2 authentication). The attacker can connect to the app and manage the CTAP connection with the authenticator. The app does not need root privileges and asks at runtime for the dangerous `android.permission.NFC`, required to gain access to the `android.nfc` [68] API. However, this is not a concern, as the app is not trying to conceal its NFC capabilities. The app also needs the standard install-time `android.permission.INTERNET` to exfiltrate the data collected through CI_2 and CI_4.

Proxmark3 for proximity CI over NFC. We implemented the proximity CI attacks using the Proxmark3 [156], an open-source and programmable development kit for NFC (RFID). We wrote a Lua script using the Proxmark3 ISO14443 Type A module (i.e., `read14a`) to communicate to the authenticator via CTAP-compliant APDUs. By equipping the Proxmark3 with a long-range high-frequency antenna, we were able to extend its reach. The long-range antenna has an indicative range of 100 to 120 millimeters, as opposed to the 40 to 85 millimeters of the built-in antenna.

Electron app to simulate AC over USB. We developed an Electron app that simulates a MitM attacker. The app uses the *node-hid* module to access the USB HID traffic, gaining a MitM position between a FIDO client and an authenticator communicating over USB. The app scans for local HID devices and identifies the authenticators from their properties (e.g., the product and manufacturer fields). Then, it connects and sends binary data over USB to the authenticator. The Electron app is compatible with Windows, macOS, and Linux.

3.6.3 FIDO2 Wireshark Dissector

We extended an unofficial Wireshark FIDO2 dissector found in [218]. We add valuable features, such as support for the `CredMgmt` API. We include parsers for `WAITING` and `PROCESSING` keepalive status codes that identify when authenticators are unavailable. We parse the authenticator's capabilities in the `CTAPHID_INIT` message, which are useful for testing AC_7. 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. The FIDO2 dissector is included in our toolkit as a Lua script (i.e., `fido2-dissectors.lua`).

3.7 Evaluation

We evaluated our eleven attacks against *six* popular and recent authenticators from Yubico, Feitian, SoloKeys, and Google. We also tested *ten* widely used relying parties, including Microsoft, Apple, GitHub, and Facebook. Next, we will present our evaluation setup and results.

3.7.1 Setup

Authenticators. We evaluate *six* popular FIDO2 authenticators. Table 3.3 shows their technical details. 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. The authenticators support USB and NFC, except for OpenSK which has an NFC module but supports only USB. The Solo V1 requires a button press to activate NFC. Unfortunately, we could not find any FIDO2 authenticator supporting BLE.

Table 3.3: 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, DCr: discoverable credentials.

Authenticator	**Manuf**	**Year**	**FVer**	**OSF**	**DCr**
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

Table 3.4: 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**: not applicable as the authenticator does not implement the `Selection` API.

Authenticator	**CI**$_1$	**CI**$_2$	**CI**$_3$	**CI**$_4$	**AC**$_1$	**AC**$_2$	**AC**$_3$	**AC**$_4$	**AC**$_5$	**AC**$_6$	**AC**$_7$
YubiKey 5	✓	✓	✓	✓	✓	✓	✓	✓	✓	n/a	✓
YubiKey 5 FIPS	✓	✓	✓	✓	✓	✓	✓	✓	✓	n/a	✓
Feitian K9	✓	✓	✓	✓	✓	✓	✓	✓	✓	n/a	✓
Solo V1	✓	✓	✓	✓	✓	✓	✓	✓	✓	n/a	✓
Solo V2 Hacker	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
OpenSK	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

CI$_1$: Factory reset authenticator, **CI**$_2$: Track user from credentials, **CI**$_3$: Force authenticator lockout, **CI**$_4$: Profile authenticator, **AC**$_1$: Delete discoverable credentials, **AC**$_2$: Factory reset authenticator, **AC**$_3$: Track user from credentials, **AC**$_4$: Fill authenticator's credential storage, **AC**$_5$: Force authenticator lockout, **AC**$_6$**:** Authenticator DoS, **AC**$_7$: Profile authenticator.

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, and, as such, it should provide high security guarantees. We ran OpenSK on an NRF52840 dongle, but any board supporting OpenSK would have worked.

Relying parties. Our list of relying parties covers pervasive and heterogeneous online services, including software as a service, social, gaming, cryptographic signing, authentication, and cloud storage. We registered our authenticators with *ten* FIDO2 relying parties: Adobe, Apple, DocuSign, Facebook, GitHub, Hancock, Microsoft, NVidia, Synology, and Vault Vision. Some of them offer Single Sign-On (SSO), enabling access to multiple services. For example, a single set of FIDO2 credentials can log into Microsoft, OneDrive, Outlook, and Minecraft. As a consequence, erasing a single credential has a widespread effect on multiple online services.

CTRAPS toolkit. We evaluated the CI and AC attacks using the tools included in the CTRAPS toolkit. We installed our two Android apps, found in the CTAP clients module, on a Google Pixel 2 (OS: Android 11), a RealMe 11 Pro (OS: Android 14), and a Xiaomi Redmi Plus 5 (OS: Android 8.1). We used a Proxmark3 RDV4 with a long-range high-frequency antenna to deploy the proximity CI attack with an extended NFC range. We tested our Electron app 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).

3.7.2 Authenticators Results

Table 3.4 shows the evaluation results for the CI and AC attacks on six FIDO2 authenticators. All six of them were vulnerable to the CTRAPS attacks, even the FIPS-compliant YubiKey. 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. However, AC_6 does not apply to the four authenticators which do not support the `Selection` API.

We also found a `CredMgmt` implementation vulnerability on the YubiKey 5 and YubiKey 5 FIPS, which improperly handles the authenticator's state for `CredMgmt`. They allow the client to call `CredMgmt(EnumRpsGetNextRp)` without invoking `CredMgmt(EnumRpsBegin)` first, which is an illegal state. We exploit this flaw to achieve a zero-click *leak of relying party names*. Our attack calls `CredMgmt(EnumRpsGetNextRp)` to reveal the names of all the relying parties, stored on the authenticator, except one. This attack bypasses *UV* and works regardless of the *CredProtect* policy. We reported it to Yubico, which assigned it CVE-2024-35311 and addressed it in their latest firmware. However, since YubiKeys do not support firmware updates, this fix is only available to newer authenticators, leaving older ones vulnerable.

The CI and AC attacks over NFC have a maximum range of two centimeters on a smartphone. The Proxmark3 built-in antenna also achieved the same range, which we could extend to six and a half centimeters by attaching a long-range antenna. Prior work demonstrated that, with specialized equipment, the NFC range can be extended up to 50 centimeters [119].

We also tested *combinations* of CI and AC attacks, to develop more advanced variants. For example, we found multiple ways to enhance our user tracking attacks (CI_2 and AC_3). The attacker can refine the user's fingerprint using AC_7 or register new credentials, with metadata of their choice, on the authenticator using AC_4.

3.7.3 Relying Parties Results

As shown in Table 3.5, we tested eight relying parties supporting discoverable credentials and two employing non-discoverable credentials. Our evaluation includes relying parties because our attacks affect them, even though we do not utilize WebAuthn. For example, AC_1 deletes discoverable credentials, causing the user to lose access to their online account. Although relying parties using non-discoverable credentials are not vulnerable to AC_1 and AC_4, they remain open to our factory reset, user tracking, and DoS attacks.

CI_1, AC_1, and AC_2 block web authentication to the relying party by deleting the user's FIDO2 credentials. CI_2 and AC_3 utilize the user identifiers generated by the relying party to track users. CI_3, AC_4, AC_5, and AC_6 prevent relying parties from communicating with the authenticator. Among the relying parties supporting discoverable credentials, we found that only Microsoft and Apple employ the weak *CredProtect=UVOptional* policy. This policy allows to bypass *UV* when accessing credentials. As a result, an attacker can deploy a zero-click variant of CI_2 and AC_3 to track users through their Microsoft and Apple credentials.

Table 3.5: 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 (Disc, DiscWeak) or non-discoverable (NonDisc) credentials. We indicate with DiscWeak 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 does not support discoverable credentials.

Rp	RpId	Cred	Delete Creds	Track User	DoS Authenticator
Adobe	`adobe.com`	Disc	CI_1, AC_1, AC_2	CI_2, AC_3	CI_3, AC_4, AC_5, AC_6
Apple	`apple.com`	DiscWeak	CI_1, AC_1, AC_2	CI_2, AC_3	CI_3, AC_4, AC_5, AC_6
DocuSign	`account.docusign.com`	NonDisc	CI_1, AC_2	n/a	CI_3, AC_5, AC_6
Facebook	`facebook.com`	NonDisc	CI_1, AC_2	n/a	CI_3, AC_5, AC_6
GitHub	`github.com`	Disc	CI_1, AC_1, AC_2	CI_2, AC_3	CI_3, AC_4, AC_5, AC_6
Hancock	`hancock.ink`	Disc	CI_1, AC_1, AC_2	CI_2, AC_3	CI_3, AC_4, AC_5, AC_6
Microsoft	`login.microsoft.com`	DiscWeak	CI_1, AC_1, AC_2	CI_2, AC_3	CI_3, AC_4, AC_5, AC_6
NVidia	`login.nvgs.nvidia.com`	Disc	CI_1, AC_1, AC_2	CI_2, AC_3	CI_3, AC_4, AC_5, AC_6
Synology	`account.synology.com`	Disc	CI_1, AC_1, AC_2	CI_2, AC_3	CI_3, AC_4, AC_5, AC_6
Vault Vision	`auth.vaultvision.com`	Disc	CI_1, AC_1, AC_2	CI_2, AC_3	CI_3, AC_4, AC_5, AC_6

3.8 Discussion

We discuss the countermeasures to fix the CTRAPS attacks and the issues we found in the FIDO reference threat model.

3.8.1 Countermeasures

We discuss *eight* backward-compliant countermeasures fixing the eleven CTRAPS attacks and their associated eight vulnerabilities. Each countermeasure addresses a specific vulnerability (e.g., C1 fixes V1) and helps reduce the CTAP attack surface. Although we have not implemented these countermeasures, we designed them to be implementable as amendments to the FIDO2 standard or as FIDO2 extensions. Next, we will describe each countermeasure.

C1: Trusted CTAP clients. We address V1 by recommending that the FIDO Alliance provides a list of trusted CTAP clients. The FIDO ecosystem offers several certifications, including the FIDO Functional Certification [6] which attests to 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 [210].

C2: Authenticator visual feedback. We address V2 by requiring the authenticator to provide the user with visual feedback regarding the API that was called. 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 step.

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 [2]).

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 their 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` must require *UV*. We address V6 by requiring *UV* to call `Reset`. Hence, the user must authorize a factory reset by entering a valid PIN.

C7: `CredMgmt` must require *UP*. We address V7 by requiring *UP* to call `CredMgmt`. Hence, the user must authorize credential deletion one by one, to avoid deleting multiple credentials with a single API call.

C8: Rate limiting `Selection` calls. We address V8 by enforcing temporal rate limiting on `Selection` 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 of the countermeasures. The deployment of C1 and C8 does not affect usability. C2 requires the user to notice the authenticator's visual feedback. Implementing C3 introduces an additional *UP* check each time the client connects to the authenticator over NFC, which is costly. C4 forces the user to remember a second PIN. C5 introduces one *UV* and *UP* check each time the credential and user identifiers are being rotated out, e.g., once per month. The implementation of C6 would require PIN verification for every call to `Reset`. C7 would add a button press each time the `CredMgmt` API is invoked.

Authenticator with a display. We do not consider adding a display to a roaming authenticator in the list of countermeasures as is not backward-compliant and would require to recall all vulnerable authenticators without a display. Moreover, for newer authenticators, it entails significant hardware and software modifications, such as adding a secure display, a secure display controller firmware, and a battery, that would introduce usability, performance, and cost issues.

3.8.2 FIDO Reference Threat Model Issues

The FIDO Alliance released a reference threat model [3] outlining security assumptions, goals, and threats against clients, authenticators, and relying parties. Although non-normative, it is the only official source detailing the FIDO threat model. After studying it and working on the CTRAPS attacks, we identified *three issues* (IS1, IS2, and IS3) with the FIDO reference threat model:

IS1: Unclear security boundaries. The threat model presents six broad security assumptions, but breaks them when discussing threats. For example, SA-4 states that the user device and applications involved in a FIDO2 operation act as trustworthy agents of the user. This implies that the client (e.g., browser or mobile app) must be inherently trusted. However, at the same time, the threat model includes threats that violate SA-4, such as *T-1.2.1: FIDO client corruption*. This leads to unclear security boundaries, making it difficult to differentiate trusted components

Table 3.6: Comparing prior attacks on FIDO with the CTRAPS attacks. We assign each attack a complexity and an impact. For example, the complexity for a MitM is Mid, whereas we consider spoofing a client as Low complexity. Similarly, hijacking a session has a Mid impact, while permanently destroying credentials carries a High impact.

Attack	Class	Protocol	Transp	Surface	Impl	Reqs	Complex	Impact
CTAP MitM [96]	DH MitM	CTAP2.0	All	ClientPin	✗	MitM	Mid	Mid
Privacy leak [96]	Eavesdropping	CTAP2.0	All	MakeCred	✗	n/a	Low	Low
Auth rebind [96]	Auth rebind	WebAuthn	All	Creds.create	✗	n/a	High	High
Parallel session [96]	Session hijack	WebAuthn	All	Creds.get	✗	n/a	Mid	Mid
ECDSA extract [123]	Side channel	n/a	n/a	NXP A7005	✗	Phy access	High	High
Titan sign in [40]	Relay	U2F	BLE	Google acc	✓	Proximity	Mid	Mid
Evil maid [130]	Phy access	n/a	n/a	Auth TEE	✗	Phy access	High	High
Auth MitM [22]	DH MitM	CTAP2.0/2.1	USB	ClientPin	✓	Mal browser	Mid	Mid
Web MitM [22]	Session hijack	WebAuthn	USB	Creds.get	✓	Mal browser	Mid	Mid
Rogue key [22]	Auth rebind	WebAuthn	USB	Creds.create	✓	Mal browser	Mid	High
FIDOLA [134]	Session hijack	WebAuthn	USB	Creds.get	✓	Malware	High	Mid
CTRAPS CI	Impersonation	CTAP2.0/2.1/2.2	All	Auth API	✓	Proximity	Low	High
CTRAPS AC	API confusion	CTAP2.0/2.1/2.2	All	Auth API	✓	MitM	Mid	High

from ones that could be compromised.

IS2: Missing proximity threats. Although FIDO supports proximity transports like NFC and BLE, its threat model groups proximity-based threats together with physical access ones. However, these threats differ in key aspects. For example, proximity threats have a range. Consequently, our proximity CI and AC attacks do not fit within this threat model.

IS3: Security goals are narrow. The security goals of the threat model are based on [54] (2006) and [38] (2012). These two research papers outlined the security goals of an ideal authentication scheme, focusing on password-based schemes and web authentication. As a result, the security goals are too narrow to capture the complexities of the FIDO ecosystem. For example, there are no security goals for the Authenticator API, that could address the AC attacks, or discoverable credentials, that are relevant to AC_1, CI_1, and AC_2.

3.9 Related Work

We present related work on FIDO, covering existing attacks, formal analysis, FIDO extensions and enhancements, usability studies, and surveys.

Attacks on FIDO(2). Researchers found attacks on older FIDO versions (UAF, U2F), such as authenticator rebinding, parallel sessions, and multi-user attacks [104, 127], USB HID man-in-the-middle attacks [41], BLE pairing [40], relying party public key substitution [170], bypassing push-based 2FA [114], real-time phishing [191], and side channel attacks [160, 117]. FIDO2 was also found vulnerable to deception [134], misbinding [212], physical [123, 130], and rogue key or impersonation attacks [121, 22]. Moreover, researchers found issues on lower layers trusted by FIDO2, including an IV reuse on the Samsung Keystore [180]. No prior attack investigated *client impersonation* or *API confusion* on CTAP, including its *latest* version.

Formal analysis. The formal analysis and verification community extensively researched FIDO. The community formally verified FIDO's Universal Authentication Framework (UAF) [152, 81], FIDO2 (including its privacy, revocation, attestation, and post-quantum crypto) [21, 29, 99, 30]. Yubico proposed a key recovery mechanism based on a backup authenticator that was proven secure using the asynchronous remote key generation (ARKG) primitive [91]. Existing

research on formal analysis is *not* covering our CI and AC attacks.

Extensions. FIDO supports extensions to add optional features in a backward-compliant way. For instance, FeIDO [164] 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 [44]. 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 [94], the authors present a hybrid post-quantum signature scheme for FIDO2 and tested it using OpenSK [95] (which we exploit in this work). In [100], the authors propose a global key revocation procedure for WebAuthn that revokes credentials without communicating to each individual relying party WebAuthn server. True2F [65] presented a backdoor-resistant FIDO U2F design, protecting the authenticator from a malicious browser by requiring the authenticator interaction during every authentication, and from fingerprinting by rate limiting credential registration. 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 [59, 64, 55, 125], FIDO2 roaming authenticators [78, 149], passkeys [118], and cross-site 2FA [132]. Our paper is *orthogonal* to usability studies.

Surveys. There are several FIDO survey papers. In [9] the authors describe the evolution of FIDO protocols, security requirements, and adoption factors. In [133], the authors surveyed the adoption of passwordless authentication among a large user base, considering users' perceptions, acceptance, and concern with single-factor authentication without passwords. Our paper is *orthogonal* to surveys.

3.10 Conclusion

No prior work assessed the CTAP Authenticator API, a critical surface exposed by a client to an authenticator to manage, create, and delete credentials. We address this gap by presenting the first security and privacy evaluation of the CTAP Authenticator API. We uncover two classes of protocol-level attacks that abuse it. The CI attacks spoof a CTAP client to a target authenticator. The AC attacks leverage a MitM position to change CTAP API calls made by the user to an API desired by the attacker while stealing their authorizations. They utilize API confusion, a novel attack strategy within FIDO2.

We uncover eleven CI and AC attacks, impacting millions of FIDO2 users. They can be deployed by a proximity-based or a remote attacker. For example, they delete FIDO2 credentials and master keys (security breach) and track users through their credentials (privacy breach). Our attacks are effective on the entire FIDO2 ecosystem as they target eight vulnerabilities we discovered in the CTAP specification. These flaws include the lack of CTAP client authentication and improper API authorizations. The CTRAPS attacks are low-cost, as they do not require specialized equipment, and stealthy, as they do not trigger unexpected user interactions.

We develop the `BLUFFS` toolkit to test our attacks with a cheap setup. It includes a CTAP testbed with a virtual relying party and a virtual client, four CTAP clients that deploy our attacks (e.g., Android apps and Proxmark3 scripts), and an enhanced Wireshark dissector for CTAP. We successfully exploit six authenticators and ten relying parties from leading FIDO2 players such

as Yubico, Feitian, Google, Microsoft, and Apple. We design eight legacy-compliant countermeasures to fix our attacks and their root causes.

We share *three lessons* we learned about FIDO2 *credential storage* and *passwordlessness*, which are valuable for the current transition from single-factor authentication to 2FA and passkeys [61, 163]: (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., CI_2, AC_1, AC_3, and AC_4). (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) The 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).

All experiments in this study were conducted ethically and solely on authenticators and accounts under our control, with no involvement of third-party personal data. To support reproducibility and advance open science, we are releasing our artifacts, including the CTRAPS toolkit. Our toolkit is securely hosted in a repository at https://github.com/Skiti/CTrAPs. We already responsibly disclosed our findings to all affected parties, including the FIDO Alliance, and we respected their timeline.

Chapter 4

BlueBrothers: Three New Protocols to Enhance Bluetooth Security

4.1 Abstract

The Bluetooth standard defines pairing and session establishment to secure the communications of the Bluetooth ecosystem. Despite extensive research, these protocols still suffer from critical design vulnerabilities, such as a lack of message integrity, leading to large-scale and impactful attacks, including impersonation and machine-in-the-middle ones. Moreover, the protocols' specifications are complex, scattered, and informal, making implementing them and verifying their security properties challenging. These issues justify the need for new Bluetooth security protocols that are unaffected by current design issues and have novel security guarantees, such as forward and future secrecy. These protocols should also be simple, easy to update, formally modeled, and verified to prove their security properties.

We present BlueBrothers, three novel Bluetooth security protocols, which are secure by design, simple, and open. BB-Pairing is a pairing and session establishment protocol replacing the one in the standard based on a long-term symmetric key, with one using static and ephemeral key pairs. BB-Session is a better alternative to symmetric session establishment in the standard based on asymmetric key agreement. Unlike standard session establishment, it requires devices to share static public keys exchanged via BB-Pairing and provides inter-session future secrecy. BB-Rekey is a new session key refresh protocol providing intra-session forward and future secrecy, which is also lacking in the standard.

We implement BlueBrothers for Bluetooth Low Energy (BLE) by modifying NimBLE, a production-grade and open-source BLE stack, and for Bluetooth Classic (BC) on top of the Linux BlueZ APIs. We formally model and verify the three protocols using ProVerif. The formal verification proves they benefit from integrity, confidentiality, authentication, and forward and future secrecy against active and passive attackers. We evaluate their performance in worst-case conditions using constrained BLE devices (nRF52). Our results show that BB-Pairing and BB-Rekey have up to 55.6% and 29.3% less latency compared to the standard Bluetooth protocols, while BB-Session introduces some latency overhead. All the protocols have a negligible impact on real-world power consumption, similar to or lower than their standard counterparts.

4.2 Introduction

Bluetooth is a pervasive wireless communication technology defined in the Bluetooth Core Specification document v6.1 [37], maintained by the Bluetooth Special Interest Group (SIG). We refer to the specification document as the *standard*. The standard defines two Bluetooth modes: Bluetooth Classic (BC) for high-throughput applications and Bluetooth Low Energy (BLE) for low-power scenarios, and this work covers both of them.

The security and privacy of Bluetooth rely on two protocols defined in the standard called *pairing* and *session establishment*. Pairing is a key agreement protocol that derives a long-term symmetric key called Pairing Key (PK) and supports user-assisted or out-of-band authentication. Session establishment derives a fresh Session Key (SK) from the PK and random nonces, then encrypts a session with it. These protocols protect the *link layer (stack)* and are employed by billions of heterogeneous devices, including mobile, IoT, and wearable ones. Hence, a *design vulnerability* on pairing or session establishment has critical and widespread consequences on the security of Bluetooth.

We review the security of pairing and session establishment and find that they are still affected by critical design vulnerabilities enabling seventeen attacks, including impersonation and Machine-in-the-Middle (MitM) (see Table 4.1). We isolate *four design vulnerabilities categories* enabling these attacks: no message integrity (C1) [15, 198, 56], no replay and reflection protection (C2) [57], weak or no forward and future secrecy (C3) [10], and weak or no entity authentication (C4) [18, 14, 208]. For example, an attacker can impersonate BC devices by chaining the KNOB, BIAS, and BLUFFS attacks [18, 14, 10] or MitM BC and BLE pairing via method confusion attacks [198].

Another critical issue with Bluetooth security stems from the *complexity* of its specification, which is three-thousand-page-long. The details about pairing and session establishment are scattered throughout multiple sections, and no reference implementation is publicly available. This complexity makes the two security protocols challenging to understand, implement, update, and formally model and analyze, as highlighted by previous research [209, 181, 112]. Moreover, it results in critical *implementation vulnerabilities* adding to the design flaws discussed earlier and leading to severe threats, like Denial of Service (DoS) and Remote Code Execution (RCE) [175, 93].

Motivated by current design and complexity issues with Bluetooth security protocols, we unveil *BlueBrothers*, three new Bluetooth security protocols. They are called *BB-Pairing*, *BB-Session*, and *BB-Rekey* and can replace pairing and session establishment at the link layer, or vendors can use them at the application layer. The protocols require security by design, e.g., address C1–C4 and the seventeen associated attacks from Table 4.1. Moreover, their specification is open and straightforward to simplify formal modeling and verification.

BB-Pairing replaces pairing and its symmetric PK with a pairing and session establishment protocol based on static-ephemeral key pairs. Unlike pairing, it provides message integrity across all phases, strong associations, protection against replay and reflection attacks, and secure session establishment. Moreover, it includes a novel *hybrid post-quantum (PQ) key agreement* to provide a session with post-quantum security properties.

BB-Session takes over symmetric session establishment with an asymmetric protocol, relying on static keys securely exchanged using BB-Pairing and ephemeral keys. Unlike session establishment, it is authenticated, integrity protected, resilient against replay and reflection attacks, and provides a hybrid PQ key agreement. Moreover, it provides *inter-session forward secrecy*, two

new Bluetooth security properties protecting past sessions against static private key compromise.

BB-Rekey is a new protocol to refresh SK within a session, unlike session establishment that requires aborting a session to update SK. It leverages an encrypted and authenticated channel to perform the rekey, which can be symmetric or asymmetric. The symmetric rekey is based on fresh nonces and provides *intra-session forward secrecy*. The asymmetric rekey uses a Diffie-Hellman (DH) key exchange and guarantees *intra-session forward and future secrecy*. These two new Bluetooth security properties protect past and future messages within a session if SK is compromised.

We formally verify the security of the BlueBrothers protocols using ProVerif, an automated protocol verifier [31]. Specifically, we test BB-Pairing and BB-Session for secrecy, integrity, and authentication, while BB-Rekey for future secrecy against short-term key compromise. We confirm these properties hold, showing the security of our protocols.

We implement the BlueBrothers protocols for BLE and BC. The BLE implementation is based on NimBLE [19], an open-source BLE stack. It shows that link layer integration with existing devices is possible. The BC implementation relies on Linux and BlueZ and demonstrates that the protocols can also be used standalone at the application layer. Both implementations are reproducible, using open-source software and low-cost and available hardware. They are Proof of Concept (POC) and should not be used in production.

We evaluate the latency and power consumption performances of the BlueBrothers protocols against their standard counterparts. Our setup consists of constrained BLE devices (nRF52 boards) running pairing, session establishment, and rekeys. Our experimental results show that BB-Pairing and BB-Rekey reduce the average latency by up to 60% and power consumption by up to 66%. While BB-Session introduces a 2x latency overhead due to asymmetric operations, its power consumption is negligible.

We summarize our contribution as follows:

- We review pairing and session establishment from the standard and find that they are affected by design and complexity issues leading to critical attacks, including impersonation and MitM. We extract four vulnerability classes (C1–C4) and seventeen attacks effective on billions of BC and BLE devices.
- We design BlueBrothers, three new Bluetooth security protocols to replace pairing and session establishment. They provide stronger security guarantees and a simple and open design. The protocols address C1–C4 and related attacks and provide new Bluetooth security properties like hybrid post-quantum intra- and inter-session forward and future secrecy.
- We formally model the protocols and verify their security properties using ProVerif. We implement them for BLE at the link layer (NimBLE) and BC at the application layer (Linux and BlueZ). We evaluate their latency and power consumption on constrained BLE devices (nRF52 boards) and confirm that they introduce minimal overheads or even improve upon their counterparts in the standard.

4.3 Bluetooth Preliminaries

Bluetooth (BT) is a pervasive wireless technology supporting many use cases, like audio streaming, data transfer, and device control. It is specified in the Bluetooth standard, whose latest version is v6.1 [37]. There are two BT transports: BC for high-throughput and connection-oriented

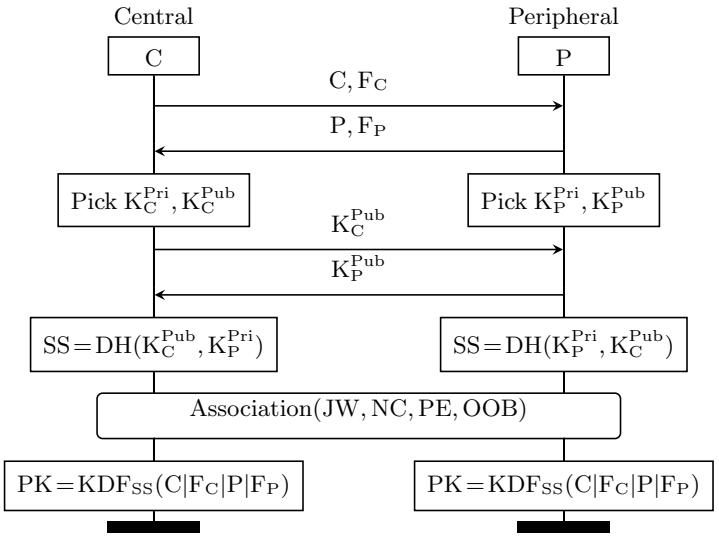

Figure 4.1: BT pairing overview. A Central and a Peripheral negotiate pairing features, agree on a DH shared secret (SS), run one of the four association protocols, and derive a long-term PK. | indicates a concatenation.

wireless services and BLE for ultra-low-power and connection-less scenarios. In a BT connection, the initiator is called Central, while the responder is called Peripheral.

BT has two logical components, the Host and the Controller, which communicate using the Host Controller Interface (HCI). The Host implements high-level protocols and functionalities, such as logical link setup via the Logical Link Control and Adaptation Protocol (L2CAP). The Controller deals with low-level and time-critical operations, such as physical and link layer management. The HCI serves as the communication bridge between Host and Controller, allowing them to exchange commands, events, and data.

The standard defines *pairing* and *session establishment* security protocols to protect Bluetooth communications. The specification of these protocols is complex and differs between BC and BLE. Moreover, it involves (legacy) sub-protocols, modes, features, negotiations, and cryptographic primitives.

Pairing and session establishment have two security modes called *Legacy Secure Connections (LSC)* and *Secure Connections (SC)*. LSC employs legacy cryptographic primitives such as the E0 stream cipher [87]. SC uses FIPS-compliant primitives, including Elliptic Curve Diffie-Hellman (ECDH) and AES-CCM. A device can enforce SC and refuse to pair and connect with LSC devices by setting Secure Connections Only (SCO) mode, also known as FIPS mode.

4.3.1 Bluetooth Pairing

Pairing (also referred to as Secure Simple Pairing (SSP) for BC) is a key establishment protocol that derives a long-term symmetric key called PK. It relies on the Trust on First Use (TOFU) principle and provides optional authentication by requiring user interaction or devices to exchange secrets out-of-band. It does not rely on trusted third parties.

Figure 4.1 shows an overview of BT pairing, which abstracts both BC SSP and BLE SC pair-

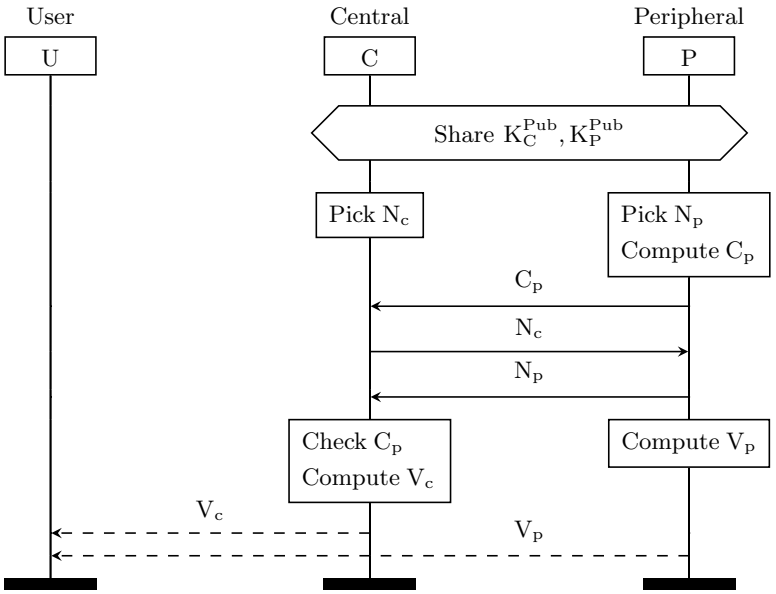

Figure 4.2: Bluetooth NC association. The devices share their public keys $\mathrm{K}^{\mathrm{Pub}}$, pick a random value N, and the Peripheral computes the confirmation value as $\mathrm{C_P}$ equal to $\mathrm{KDF}(\mathrm{K_C^{Pub}}|\mathrm{K_P^{Pub}}|\mathrm{N_P}|0)$. The Central checks $\mathrm{C_P}$, and then both devices compute the values to display to the user as $(\mathrm{V_C})\mathrm{V_P} = \mathrm{KDF}(\mathrm{K_C^{Pub}}|\mathrm{K_P^{Pub}}|\mathrm{N_C}|\mathrm{N_P})$. The user confirms the association if the values match. Otherwise, it cancels it. JW has the same messages but skips the user interaction part. The dashed messages represent user-device interactions.

ing. The devices exchange their identities (C, P) and pairing features ($\mathrm{F_C}$, $\mathrm{F_P}$). The features include the supported security mode, I/O capabilities, and Cross-Transport Key Derivation (CTKD). The latter enables pairing once and deriving two PKs, one for BC and one for BLE, and is available for devices supporting BC and BLE. If pairing is over BLE, the features include an integer between 7 and 16 to negotiate the PK entropy.

Next, the devices pick ECDH key pairs, exchange their public keys (K_C^{Pub}, K_P^{Pub}), and compute a DH shared secret (SS). Depending on their input/output (I/O) capabilities, they engage one of the four association protocols: JustWorks (JW), Numeric Comparison (NC), Passkey Entry (PE), or OOB. JW does not require user interaction but is not authenticated, while the other three provide authentication relying on user assistance of a pre-shared key.

Figure 4.2 shows NC association. The devices share their public keys $\mathrm{K}^{\mathrm{Pub}}$, then they both pick a random value N, and the Peripheral computes the confirmation value as $\mathrm{C_P}$ equal to $\mathrm{KDF}(\mathrm{K_C^{Pub}}|\mathrm{K_P^{Pub}}|\mathrm{N_P}|0)$. The Central checks $\mathrm{C_P}$, and then both devices compute the values to display to the user as $(\mathrm{V_C})\mathrm{V_P} = \mathrm{KDF}(\mathrm{K_C^{Pub}}|\mathrm{K_P^{Pub}}|\mathrm{N_C}|\mathrm{N_P})$. The user confirms the association if the values match. Otherwise, it aborts pairing. We represent the user-device interaction with dashed lines.

JW association uses the same messages of NC, but it skips the interaction with the user.

After association, the devices derive PK using a Key Derivation Function (KDF), taking as inputs the concatenation of their identities, pairing features, and keyed with SS. For BLE, the PK length is adjusted according to the negotiated entropy values.

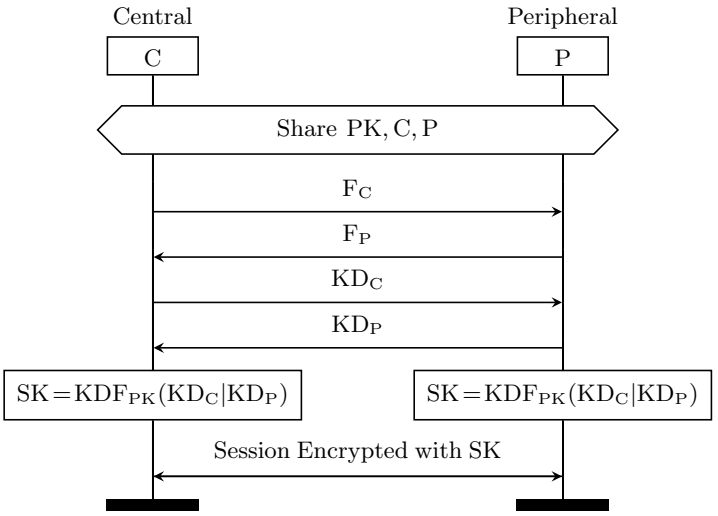

Figure 4.3: BT session establishment overview. A Central and a Peripheral share PK, negotiate their session features, exchange session key diversifiers, derive SK from PK and the diversifiers, and protect the session with SK. BC also includes PK mutual authentication and SK entropy negotiation phases before the exchange of key diversifiers.

4.3.2 Bluetooth Session Establishment

Session establishment enables paired devices to generate a session key (SK) to encrypt their communication. The SK is derived from the PK and fresh random nonces. Typically, devices pair once over BC or BLE and then automatically establish secure sessions once in range.

Figure 4.3 provides an overview of BT session establishment mapping to BC and BLE SC session establishment. The devices share PK and their identities and start the protocol by exchanging their session features (F_C, F_P), including their security mode. For BC, the protocol also includes PK mutual authentication based on challenge-response and SK entropy negotiation not shown in the figure.

Then, the devices exchange key diversifiers (KD_C, KD_P) and derive from them a fresh SK using a KDF keyed with PK. The messages are not encrypted, integrity-protected, or authenticated at this stage. Finally, the devices encrypt their messages with SK using AES-CCM, an Authenticated Encryption with Associated Data (AEAD) cipher, or the E0 stream cipher that does not provide message integrity.

SK provides forward and future secrecy against SK compromise. The attacker can decrypt messages protected by the compromised SK but not ones of prior or future sessions as they use different SKs. The standard allows refreshing a SK by terminating the session and starting a new one but does not allow an intra-session rekey. A PK compromise breaks forward, and future secrecy as the attacker can compute any SK from the PK and the plain text key diversifiers.

4.4 Motivation

We review the state-of-the-art Bluetooth security and isolate two open challenges: 1) pairing and session establishment are affected by critical *design vulnerabilities* leading to impactful and large-

Table 4.1: BC and BLE pairing and session establishment are vulnerable to seventeen design-level attacks. The attacks span from 2019 to the current day and cover different protocol phases, i.e., Association (AS), Feature Negotiation (FN), and Key Derivation (KD). They are effective against any SC and LSC security modes (SM). They are enabled by four vulnerability categories (C1–C4).

Attack	Yr	Phase	SM	Vulns.
Pairing				
BLE Pairing confusion [56]	23	FN, AS	SC	C1
BC Pairing confusion [56]	23	FN, AS	SC	C1
BLE Method confusion 2 [56]	23	FN, AS	SC	C1
BC Method confusion 2 [56]	23	FN, AS	SC	C1
BC BLUR [16]	22	FN	SC	C1, C2
BLE BLUR [16]	22	FN	SC	C1, C2
BC Method confusion [198]	21	FN, AS	SC	C1
BLE Method confusion [198]	21	FN, AS	SC	C1
BC BlueMirror BT-A [57]	21	AS	LSC	C1, C2
BLE BlueMirror BLE-A [57]	21	AS	LSC	C1, C2
BLE BlueMirror PE-A1 [57]	21	AS	SC	C1, C2
BLE BlueMirror PE-A2 [57]	21	AS	SC	C1, C2
BLE KNOB [15]	20	FN	SC	C1
Session Establishment				
BC BLUFFS [10]	23	KD	SC	C3
BC BIAS [14]	20	FN	SC	C2, C4
BLE BLESA [208]	20	KD	SC	C4
BC KNOB [18]	19	FN	SC	C2, C4

scale attacks, like device impersonation, 2) the specification of pairing and session establishment is *complex*, causing problem in maintenance, updates, formal modeling, and security analyses and resulting in further vulnerabilities and attacks, such as RCE ones. These two challenges, which are detailed next, motivate this work.

4.4.1 Bluetooth Security Design Issues

We conducted a Bluetooth security survey and found *seventeen attacks* exploiting design issues on pairing and session establishment. The attacks, listed in Table 4.1, span from 2019 to current days and affect Bluetooth versions up to v6.1, SC and LSC modes, and different protocol phases. Seventeen of them target pairing, and four exploit session establishment. They result in eavesdropping, impersonation, and MitM threats, breaking Bluetooth's confidentiality, integrity, and authenticity.

Pairing Issues

Pairing feature negotiation (FN) and association (AS) phases are vulnerable. Attacks on FN exploit the lack of message integrity to downgrade the entropy of PK with the KNOB attack [15].

Alternatively, they can overwrite a PK across BLE and BC with the BLUR attacks [16]. The BlueMirror attacks [57] target the AS phase and exploit the lack of message integrity and reflection protection to bypass user-assisted authentication. Other attacks target both FN and AS. For instance, the Method and Pairing Confusion attacks [56, 198] exploit the lack of message integrity in both phases to bypass user-assisted authentication.

Session Establishment Issues

Session establishment FN and key derivation (KD) are also vulnerable because they lack message integrity, entity authentication, and protection against message replay and reflection. This happens despite the availability of a shared PK. An attacker impersonates a BC device without knowing PK by chaining the KNOB and BIAS attacks [18, 14]. Alternatively, they spoof a BC device across secure sessions by forcing the reuse of a weak SK using the BLUFFS attacks [10]. For BLE, the adversary spoofs a BLE Peripheral during a secure reconnection by aborting SK derivation via the BLESA attack [208].

Vulnerability Categories

We analyzed the root causes of the seventeen attacks in Table 4.1 and isolated four design vulnerability categories:

C1: No message integrity

C2: No replay and reflection protection

C3: Weak or no forward and future secrecy

C4: Weak or no entity authentication

As shown in the last column of Table 4.1, the seventeen attacks exploit at least one of the four categories. For instance, the pairing confusion attacks [56] abuse lack of integrity protection (C1), the BIAS attacks [14] exploit the lack of reflection protection (C2) and the weak entity authentication (C4), and the BLUFFS attacks take advantage of forward and future secrecy issues among sessions (C3).

4.4.2 Complex Bluetooth Security Specification

Another issue with Bluetooth security is that the pairing and session establishment specification is complex. It includes sub-protocols, features, and cryptographic primitives, changing for BC and BLE according to the association, security mode, and I/O capabilities supported by a device. Moreover, the specification is informal and scattered across a 3500-page core document, errata, and supplementary (paywalled) files. Hence, it is hard to understand, implement, update, and conduct security analyses on pairing and session establishment.

Previous works, like [209, 181, 112, 208], highlight this complexity and provide formal models, covering some of the pairing and session establishment modes and related issues and attacks from Table 4.1. However, no formal model covers all pairing and session establishment combinations for BC and BLE. This is mainly due to the complexity of the standard, which, for instance, increases among Bluetooth version updates.

This complexity also results in numerous implementation mistakes and associated threats, further complicating the situation for a Bluetooth defender. Researchers found critical implementation-

level flaws in BC and BLE stacks, leading to DoS and RCE [175, 145, 92], or security issues such as skipping DH confirmation, installing a zeroed PK [93].

4.5 BlueBrothers Protocols

We introduce *BlueBrothers, three new security protocols* to address current Bluetooth security design flaws and specification complexity as motivated in Section 4.4. The protocols offer security by design and address the seventeen attacks shown in Table 4.1 and the related four vulnerability categories (C1–C4). They provide new security guarantees compared to pairing and session establishment in the standard, like integrity-protected feature negotiation, intra- and inter-session forward and future secrecy, and hybrid post-quantum (PQ) key agreement. They rely on open and standard cryptographic primitives and mechanisms.

Their design is simple, making them easy to implement and integrate on existing Bluetooth devices as replacements for pairing and session establishment at the link layer or as standalone applications. Next, we describe their threat model, requirements, design, and integration on Bluetooth.

4.5.1 BlueBrothers Threat Model

System model. We consider two entities that aim to securely communicate over Bluetooth (BC and BLE) using the BlueBrothers protocols. The entities represent generic Bluetooth devices with any I/O capability and security settings. This assumption is important as our protocols should protect any Bluetooth use case. The entity that initiates a protocol acts as Central, and the responder acts as Peripheral. We do not set requirements such as a trusted Public Key Infrastructure (PKI) or globally pre-shared keys as they are unrealistic for Bluetooth. We also assume cryptographic primitives to be secure, including random number generators.

Attacker model. We assume an attacker in Bluetooth range with the victims aiming at exploiting design issues in the BlueBrothers protocols. They aim to break the BlueBrothers's security guarantees, including confidentiality, integrity, authenticity, inter-session forward secrecy, intra-session forward, and future secrecy.

The attacker has the capabilities of a Dolev-Yao adversary [70], e.g., intercept, inject, modify, drop, replay, reflect, or redirect messages and initiate parallel protocol runs. They can access public information about the victim, such as their Bluetooth name, address, and advertised capabilities. They can attempt to launch any attack listed in Table 4.1 and abuse the related vulnerability categories (C1–C4). Moreover, we assume that 1) an attacker can somehow obtain a SK to attempt breaking intra-session forward and future secrecy and 2) a passive attacker can obtain a long-term static private key to break inter-session forward secrecy.

The attacker cannot compromise standard cryptographic building blocks like AES, ECDH, or HKDF. Physical attacks, like side-channel, fault injection, or implementation issues, are out of scope as the attacker focuses on design flaws.

4.5.2 BlueBrothers Requirements

R1: Secure by Design. The protocols should protect against the four vulnerability categories (C1–C4) and seventeen related attacks including the ones shown in Table 4.1. They should

add new Bluetooth security guarantees to defend against emerging attacks, like PQ ones. The protocols and associated security properties should be formally modeled and verified.

R2: Simple. The protocols' design should be simple and representable with formal notation like message sequence charts (MSC) and symbolic models. This results in protocols that are easy to understand, analyze, verify, implement, and update.

R3: Open. The protocols' specification should rely on open and standard cryptographic primitives and mechanisms. Moreover, it should itself be public. These allow anyone to access, analyze, and use our protocols and speed up their adoption on a large scale.

4.5.3 BB-Pairing Design

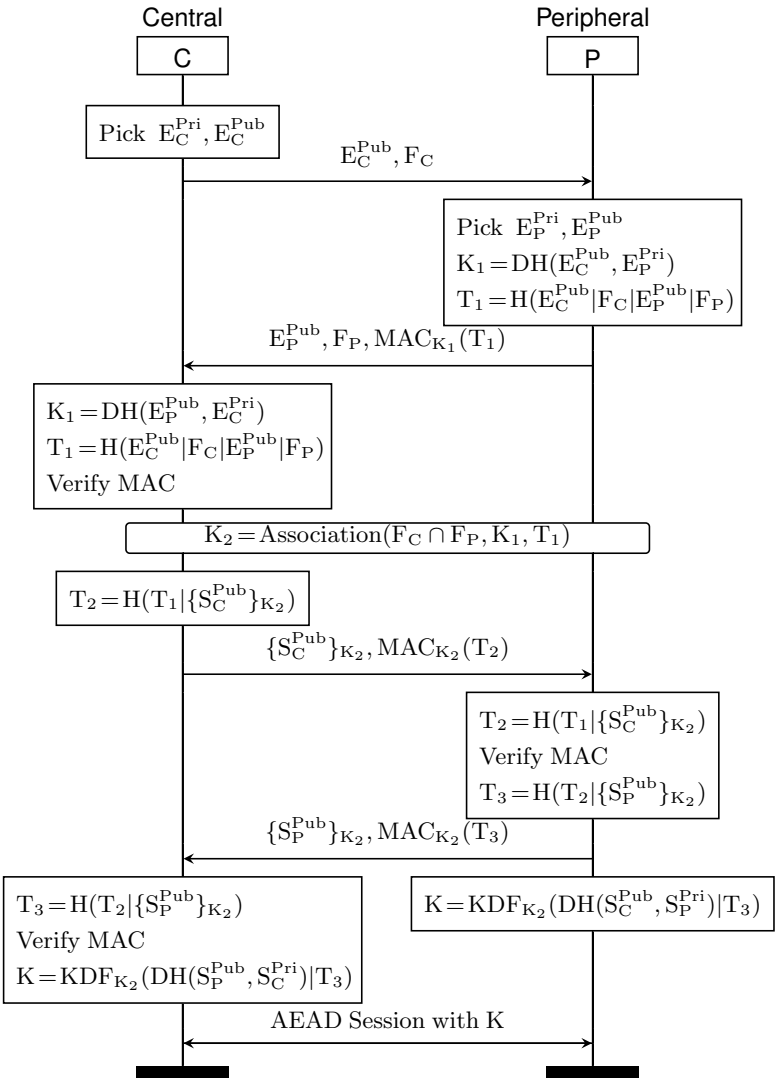

Figure 4.4: BB-Pairing MSC. The devices securely negotiate their feature, authenticate using association, securely exchange their static keys, and establish a secure session protected by a session key (K). The hash transcript (T) is used to authenticate and integrity protect the messages. If PQDH is negotiated, the ephemeral key exchange relies on Post-Quantum Diffie-Hellman (PQDH), thus achieving a hybrid PQ key agreement.

The first BlueBrothers protocol is called BB-Pairing and replaces pairing and its PK with an authenticated key agreement based on *static and ephemeral asymmetric keys*. Unlike pairing, it provides authenticated and integrity-protected feature negotiation, secure associations, protection against message replay and reflection, identity hiding, and hybrid PQ key agreement. Moreover, it includes a mutually authenticated secure session establishment protected against key downgrade and reuse, as well as message replay and reflection, achieving inter-session forward and future secrecy. Hence, the protocol is not affected by C1–C4.

Figure 4.4 presents the BB-Pairing MSC. Next, we describe its *four* phases: feature negotiation, association, static key exchange, and secure session. During a protocol run, the devices maintain and update a message hash transcript (T) that guarantees integrity and authentication in combination with Message Authentication Code (MAC) or AEAD encryption. The protocol is aborted whenever a MAC check or AEAD decryption fails.

Feature negotiation

The Central picks an ephemeral key pair $(\mathrm{E}_{\mathrm{C}}^{\mathrm{Pri}}, \mathrm{E}_{\mathrm{C}}^{\mathrm{Pub}})$ and transmits the public key along with its pairing features $(\mathrm{F_C})$ to Peripheral. The features include the supported associations and hybrid PQDH and do not include key entropy to avoid key downgrade attacks. When the PQ Hybrid mode is negotiated, after the static key exchange, the devices use two extra messages to perform a PQDH key agreement. Using a hybrid PQ key agreement allows for generating a SK with PQ contributions, thus addressing an attacker with post-quantum capabilities.

The Peripheral picks an ephemeral key pair $(\mathrm{E}_{\mathrm{P}}^{\mathrm{Pri}}, \mathrm{E}_{\mathrm{P}}^{\mathrm{Pub}})$, computes K_1 as a DH shared secret. Then, it sends Central its ephemeral public key, pairing features $(\mathrm{F_P})$, and a MAC of the current hash transcript (T_1) keyed with K_1 to authenticate the message. The Central computes K_1, T_1, and verifies the message integrity. The FN phase is integrity protected, but it does not provide entity authentication, as the devices do not authenticate each other yet.

Association

BB-Pairing includes new NC, OOB, PE, and JW association methods that are more secure and have less computational overhead than the standard ones. The devices determine the association method based on their features using the same logic as the standard, e.g., if both devices support NC, they use it.

New NC. Figure 4.5 illustrates BB-Pairing's NC. The devices use the shared K_1 and T_1 to generate $\mathrm{N_C}$ and $\mathrm{N_P}$ as HMAC-based One-time Password (HOTP) [141]. They display the values to the user. If the user confirms that they are the same, the devices derive K_2 using a KDF whose inputs are $\mathrm{N_C}$=$\mathrm{N_P}$ and K_1. Otherwise, BB-Pairing is aborted.

New OOB. BB-Pairing's OOB assumes the existence of a pre-shared secret (psk) distributed to the devices using other mechanisms, e.g., via Near Field Communication (NFC). The devices then compute $\mathrm{K}_2 = \mathrm{KDF}_{\mathrm{K}_1}(\mathrm{psk})$. This association method does not require user interaction and relies on the same assumption of the standard OOB (i.e., the presence of an eternal communication channel).

New PE. PE for BB-Pairing is shown in Figure 4.9. The devices share K_1 and T_1 and compute $\mathrm{N_C}$ and $\mathrm{N_P}$ as for NC. The Central displays $\mathrm{N_C}$ to the user who inputs it on Peripheral. If the user input value is equal to $\mathrm{N_P}$, the devices compute K_2 from N and K_1. Otherwise, BB-Pairing is aborted.

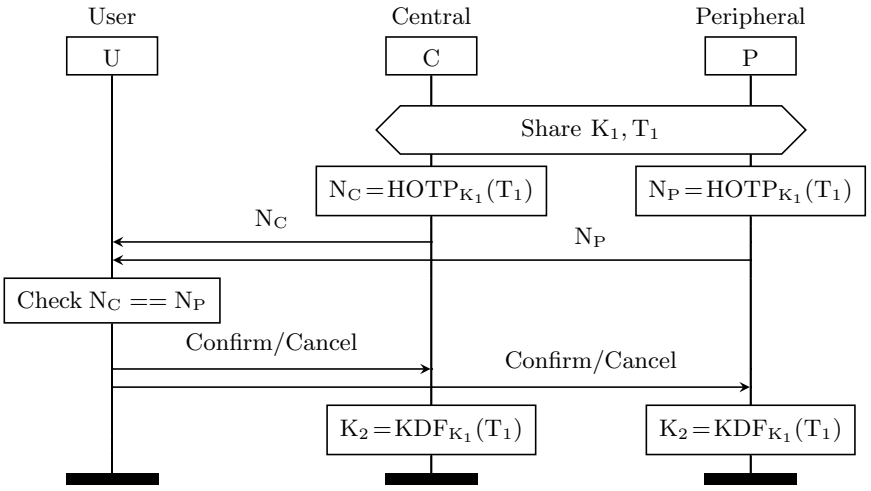

Figure 4.5: BB-Pairing new NC. The devices compute and display $\mathrm{N_C}$ and $\mathrm{N_P}$. The user confirms that they are the same and the devices derive $\mathrm{K_2}$.

New JW. BB-Pairing supports JW to cope with Bluetooth devices with no I/O capabilities. During JW, the devices computes $\mathrm{K_2}$ from $\mathrm{T_1}$ and $\mathrm{K_1}$ without user interaction. There is no authentication as in the standard JW.

BB-Pairing NC and PE associations are more secure than the ones in the standard. The standard association methods rely on plaintext public keys and random nonces to compute the confirmation value and do not provide message authentication or integrity. Moreover, they are not resilient against message reflection. BB-Pairing associations instead rely on $\mathrm{K_1}$ and $\mathrm{T_1}$ to compute the confirmation values and $\mathrm{K_2}$, which is then used to encrypt and authenticate the upcoming messages. This ensures message integrity, and since the phase is bound to the fresh ephemeral keys, it also enables protection against replay and reflection attacks.

Static key exchange

In Figure 4.4, the devices perform a secure static key exchange after the association phase. The Central updates the transcript ($\mathrm{T_2}$) and sends its $\mathrm{S_C^{Pub}}$ encrypted and authenticated using $\mathrm{K_2}$ and $\mathrm{T_1}$. The Peripheral updates the hash transcript ($\mathrm{T_2}$), decrypts and stores the received public key, then updates the hash transcript again ($\mathrm{T_3}$) and sends its $\mathrm{S_P^{Pub}}$ encrypted and authenticated using $\mathrm{K_2}$ and $\mathrm{T_3}$. The Central updates the transcript ($\mathrm{T_3}$) and decrypts and stores the peripheral static public key.

The static keys represent the devices' identity. They replace the PK to mitigate key compromise attacks and strengthen forward and future secrecy. Without the PK, an attacker can only impersonate a trusted device if they compromise the static private key, for which they need to break the DH key exchange.

Using freshly generated ephemeral keys ensures session key separation, as even if an attacker compromises a static private key, they cannot decrypt past or future session messages. Moreover, since the static key exchange is encrypted, this phase provides *identity hiding* and protects against tracking attacks via static credentials. The keys are generated once during the device bootstrap and stored in memory or secure storage if available. They are not changed

unless a device is factory reset or the user explicitly changes them.

Secure session

At the end of BB-Pairing the two devices establish a secure session, unlike the standard pairing protocol. They both compute a session key K using a KDF keyed with K_2 and input with the concatenation of a DH secret computed from the static keys and T_3. Then, they run an encrypted and authenticated session using K. If the devices negotiated PQ security in the feature negotiation, two extra messages are exchanged to perform a PQDH key derivation, and the result is mixed with K to obtain the final session key. The PQDH key exchange is encrypted and authenticated using K.

Since K depends on a fresh DH secret, the session benefits from forward and future secrecy, as even if a static private key is compromised, the attacker cannot decrypt future session messages unless it breaks the ephemeral DH key exchange. Moreover, the session is not vulnerable to session key reuse and device impersonation attacks as the entities and their key diversifiers are mutually authenticated, and the latter cannot be replayed or reflected.

4.5.4 BB-Session Design

The second BlueBrothers protocol is called BB-Session. It replaces the symmetric session establishment in the standard with an asymmetric authenticated key agreement and session protocol. It requires two devices to share their static public keys via BB-Pairing. Unlike session establishment, it provides mutual authentication, protection against key downgrade and reuse, inter-session future secrecy, and protection against replay and reflection attacks. Hence, it avoids C1–C4 by design.

Figure 4.6 shows BB-Session MSC. The protocol has *two phases*: feature negotiation and secure session. Like for BB-Pairing, the devices maintain and update T for messages and protocol integrity.

Feature negotiation

BB-Session's feature negotiation is authenticated, integrity protected, and immune to message replay and reflection. The Central generates an ephemeral key pair (E_C^{Pub}, E_C^{Pri}), derives K_1 and update the hash transcript T_1. Then, it sends to Peripheral its ephemeral public key, session features $\mathrm{F_C}$, and a MAC authenticating T_1 with K_1. The features include support for PQDH and lack key entropy to avoid entropy downgrades.

The Peripheral computes K_1, T_1, and verifies the MAC. It generates an ephemeral key pair $(\mathrm{E_P^{Pub}}, \mathrm{E_P^{Pri}})$, derives a session key (K) using a KDF keyed with K_1 and as input the DH shared secret. Then it updates the hash transcript (T_2). Then, it sends Central its ephemeral public key, session features, and a MAC keyed with K authenticating T_2.

The Central derives K mirroring the Peripheral computations, updates the hash transcript (T_2), and verifies the message integrity and authenticity. If the devices negotiate PQDH, they also derive PQ key pairs, securely exchange their PQ public keys, compute a PQ DH shared secret, and mix it with K using a KDF, achieving a hybrid PQ key agreement. These optional messages and computations are not shown in Figure for readability 4.6.

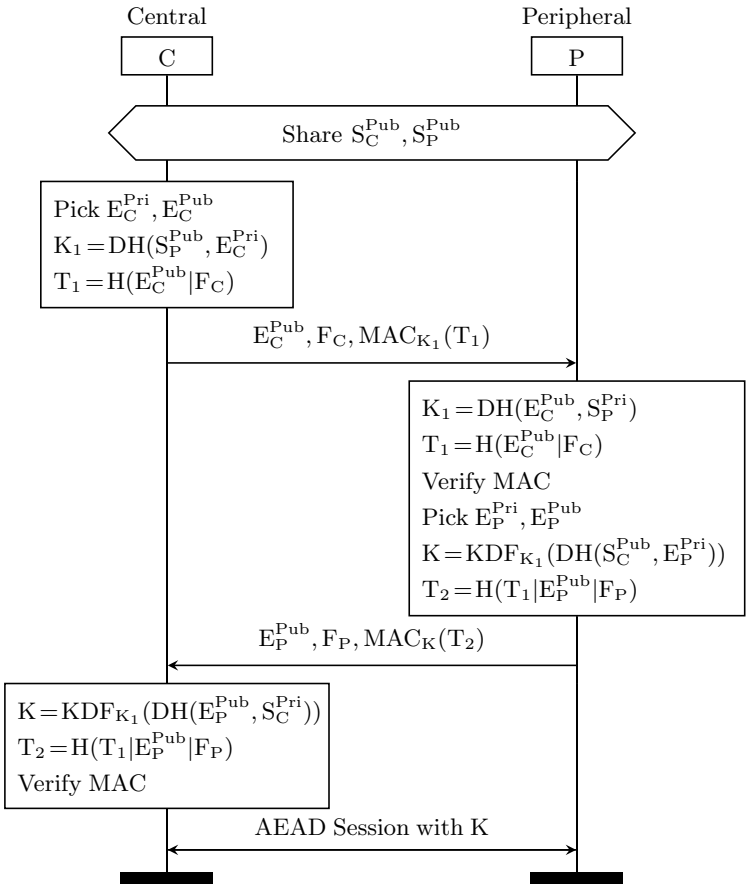

Figure 4.6: BB-Session MSC. The devices use the shared static keys to authenticate the ephemeral key exchange and securely negotiate features. Then they derive a session key (K) and securely communicate.

Secure session

The devices run an AEAD session using K. We note that such a session provides the same security properties as a BB-Pairing session, including strong inter-session forward and future secrecy and hybrid PQ key agreement.

4.5.5 BB-Rekey Design

BB-Rekey is the third BlueBrothers protocol. It runs over an encrypted and authenticated session established with BB-Pairing or BB-Session. It addresses C4 by providing *intra-session forward and future secrecy*, two new Bluetooth security properties to protect past and future messages within a session, even if the current K is compromised. The Central dynamically initiates the rekey procedure and can be *asymmetric* or *symmetric*. The asymmetric rekey provides intra-session forward and future secrecy using a DH key exchange, while the symmetric rekey provides intra-session forward secrecy using nonces.

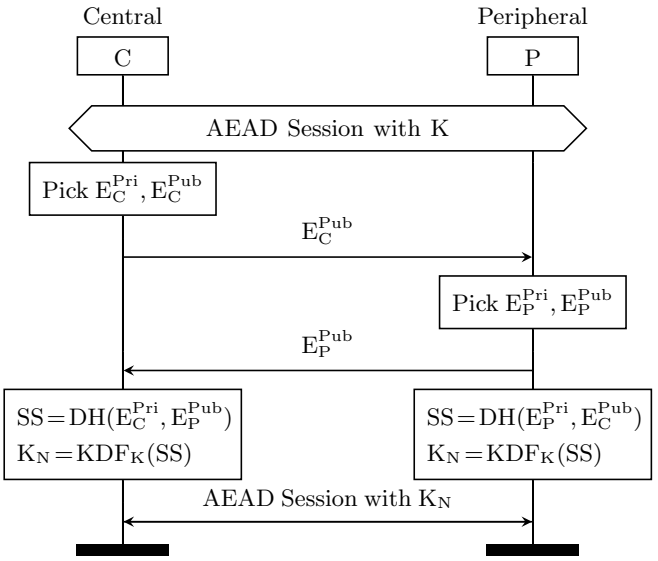

Figure 4.7: BB-Rekey new asymmetric rekey MSC. The devices exchange ephemeral public keys, compute a DH shared secret, and derive a new session key ($\mathrm{K_N}$) without aborting the session.

Asymmetric rekey

Figure 4.7 depicts BB-Rekey's asymmetric rekey MSC. The devices pick ephemeral key pairs and exchange their public keys. They compute SS and use it to derive the new session key ($\mathrm{K_N}$) via a KDF keyed with K. The asymmetric rekey provides intra-session future secrecy because even if the attacker compromises K and eavesdrops on the rekey messages, they cannot know SS as it results from a DH computation. Hence, they cannot compute the new session key ($\mathrm{K_N}$).

Symmetric rekey

BB-Rekey's symmetric rekey is shown in Figure 4.8. The devices pick and exchange two nonces ($\mathrm{R_C}$, $\mathrm{R_P}$). Then, they derive $\mathrm{K_N}$ using a KDF keyed with K and input with the nonces' concatenation. The symmetric rekey provides intra-session forward secrecy as an attacker who knows $\mathrm{K_N}$ cannot compute past messages encrypted with K as they cannot invert the secure one-way function employed by KDF.

4.5.6 Protocols Integration on Bluetooth

The BlueBrothers protocols are designed considering the challenges related to their integration in the Bluetooth ecosystem. They can be integrated at the link layer (stack) with a standard update or at the *application-layer* without updating the standard.

Link layer integration. The protocols can be integrated into the standard as they keep TOFU security model and do not rely on trusted third parties or a PKI. We introduce two feature flags to negotiate them at the link layer. The `BB` flag indicates BlueBrothers support. The `BBO` flag signals that a device only uses BlueBrothers and protects it from protocol downgrade attacks. These flags are similar to the `SC` and `SCO` ones already in the standard. Hence, if a device does not support them, it can maximize backward compatibility and allow usage of current pairing and

session establishment or refuse the secure connection.

Application layer integration. The protocols can also be integrated at the application layer, similarly to Apple's Magic Pairing for Peripherals [102]. A vendor can pick an L2CAP channel and use it as a logical channel to run the protocols. It can authenticate devices not knowing their static public keys by pre-distributing them or securely exchanging them using BB-Pairing authenticated with vendor pre-shared secrets and OOB association.

4.5.7 Extra MSCs

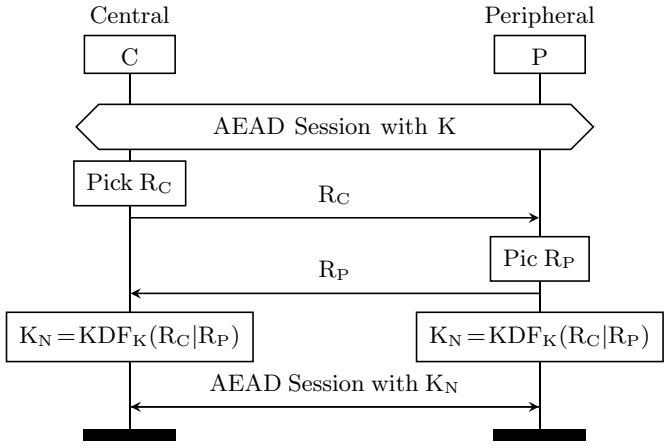

Figure 4.8: BB-Rekey new symmetric rekey MSC. The devices exchange nonces (R_C, R_P), and derive a new session key (K_N) without aborting the session.

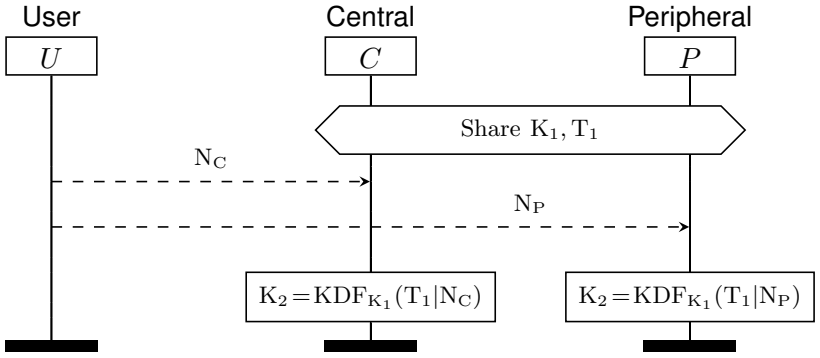

Figure 4.9: BB-Pairing new PE. The user inputs an identical Passkey on both devices. Alternatively, the passkey may be displayed on one device, and the user then inputs it on the other. In such a case, the passkey is computed as in Figure 4.5 using an HOTP.

4.6 BlueBrothers Verification with ProVerif

We formally verified the three BlueBrothers protocols using the automatic cryptographic protocol verifier ProVerif. In this section, we show that the results confirm the security of the protocols,

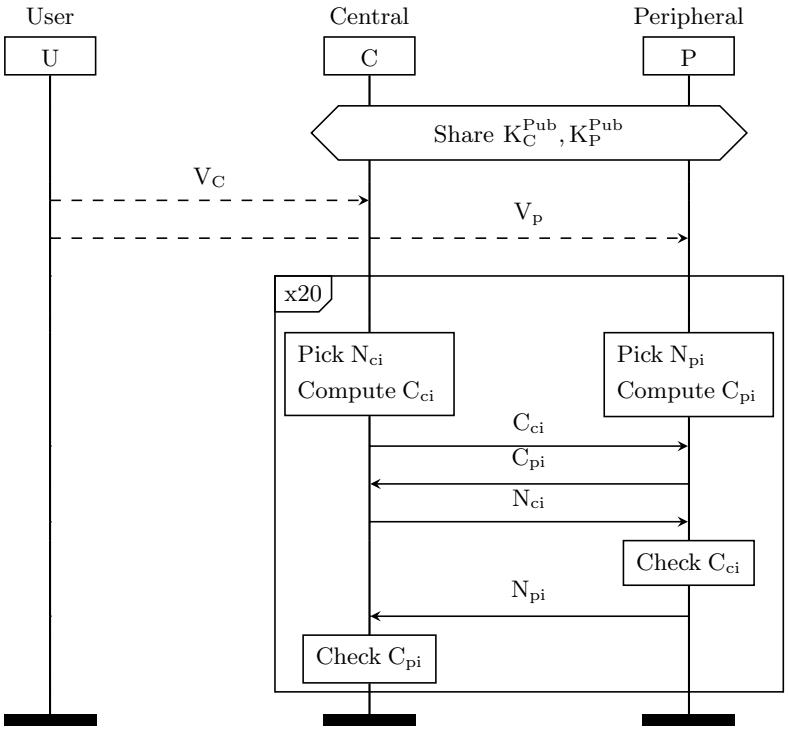

Figure 4.10: Bluetooth Association with PE. The user inputs an identical Passkey on both devices. Alternatively, the passkey may be displayed on one device, and the user then inputs it on the other. The confirmation values are computed as follows: $C_{ci} = KDF(K_C^{Pub}|K_P^{Pub}|N_{ci}|V_C)$, $C_{pi} = KDF(K_P^{Pub}|K_C^{Pub}|N_{pi}|V_P)$.

describe the models' creation, and list the queries we used.

We report the models in Appendix 4.6.4. Here, we briefly explain the details necessary to understand them and the results. In Listing 4.2, we document the cryptographic primitives used. We model a Diffie-Hellman exponentiation using the notation `DH(publickey, privatekey)` to mirror the notation used in the protocol MSCs throughout the paper to improve the models' readability.

4.6.1 BB-Pairing Security Analysis

The BB-Pairing protocol model, shown in Listing 4.3, defines the User, Central, and Peripheral roles, mirroring their descriptions in Figure 4.4. We specifically model the NC association, which is detailed in Figure 4.5. We do not run an unbounded number of instances since the user is assumed to be physically interacting with the Central and Peripheral devices during the association phase and, thus, by assumption, would not do that with more than one set of devices at a time. All public keys and negotiation parameters are assumed to be known to the attacker and are explicitly given to the attacker before the protocol starts. For this protocol, we model four properties: secrecy and integrity of the session key, prevention of user decision bypass, and mutual authentication of Central and Peripheral.

Secrecy is modeled by encrypting a dummy value at the end of the protocol with the session key K and requiring this value to remain secret. If the attacker can obtain K, they can also obtain

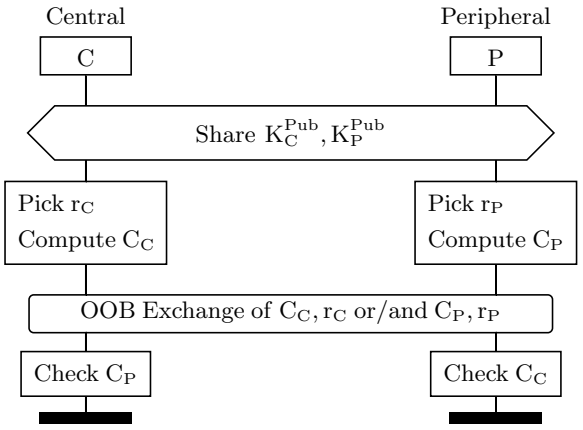

Figure 4.11: Bluetooth Association with OOB. The devices exchange their confirmation values (C_C, C_P) using an out-of-band communication channel. The values are computed similarly to the other association mechanisms.

```
Query(ies):
 - Query not attacker(SKtest[]) is true. (1)
 - Query event(PeripheralTerm(sk1,k,n1,n2)) && event(CentralTerm(sk2,k,n1,n2)) ==> sk1 = sk2
     is true. (2)
 - Query event(CentralTerm(sk,k,n1,n2)) && event(PeripheralTerm(sk,k,n1,n2)) ==> event(
     UserSaysOK(sk)) is true. (3)
 - Query inj-event(CentralTerm(sk,k,n1,n2)) ==> inj-event(PeripheralAccepts(sk,n1,n2)) is true
     . (4)
 - Query inj-event(PeripheralTerm(sk,k,n1,n2)) ==> inj-event(CentralAccepts(sk,n1,n2)) is true
     . (4)
Associated lemma(s):
 - Lemma event(UserSaysOK(ka)) && event(UserSaysOK(kb)) ==> ka = kb is true in process 1.
```

Listing 4.1: BB-Pairing ProVerif results. Queries are provable and true.

the value. If not, K benefits from secrecy.

Integrity is modeled by requiring that if the protocol terminates correctly for both parties, their session key must be the same. This ensures that an adversary cannot manipulate the resulting key and cause one (or both) parties to believe that they have a valid key if they do not.

User decision is modeled by requiring that the user must have accepted the connection during the association phase if the protocol terminates correctly for both parties. This ensures that the adversary cannot bypass the user's decision and complete the protocol despite the user rejecting (or not interacting).

Authentication is modeled by requiring that if either device terminates the protocol correctly, it must be the case that the other device accepted the protocol run. This ensures that the adversary cannot complete a run with one of the devices by impersonating the other device.

Listing 4.1 shows the summary output of ProVerif when run on the model of BB-Pairing. The queries are numbered with ① for secrecy, ② for integrity, ③ for user decision, and ④ for authentication.

4.6.2 BB-Session Security Analysis

The BB-Session protocol model is shown in Listing 4.4. We again model the roles mirroring their description in Figure 4.6 and explicitly give the attacker the long-term public keys corresponding to the two devices and the negotiation parameters. Since the user is not directly involved in the execution of this protocol, we allow an unbounded number of instances from the attacker (e.g., they can start infinite parallel executions of the protocol) and consider a security property not verified if it is violated in at least one instance.

For BB-Session we verify the secrecy and integrity of the resulting session key and authentication of the Peripheral device. This is the same as for BB-Pairing, except for two details: i) the user is not directly involved, so there is no requirement involving the user's decision, and ii) we only verify that Peripheral authentication.

While Central's authentication is not confirmed when the protocol terminates, the resulting key still benefits from the same properties as BB-Pairing key. Even if an adversary starts a session, they do not have the required session key at the end of the protocol. Hence, Central authentication is implicitly confirmed (or not) with the first message after the handshake. This is a deliberate design choice to use one less message and is similar to what is done by Wireguard [71]. If explicit mutual authentication is required, adding a third key confirmation message to the protocol is possible.

The queries' details and ProVerif's output are the same as for BB-Pairing so we do not repeat them.

4.6.3 BB-Rekey Security Analysis

The BB-Rekey protocol model is shown in Listing 4.5. As before, we model the roles mirroring their description in Figure 4.7. This protocol does not use the long-term (static) public keys. Hence, the attacker does not need them. We model the secrecy and integrity of the resulting session key and mutual authentication of the devices, precisely like the other two protocols.

One additional property we require for BB-Rekey is future secrecy, i.e., if an attacker manages to compromise the current K, they should not have access to $\mathrm{K_N}$ after a successful rekey. To model this, we use a copy of the same model with two changes: i) we explicitly give K to the attacker before rekeying starts, and ii) we require the attacker to be passive.

With these assumptions, we verify that an attacker who knows K cannot obtain $\mathrm{K_N}$, even given an unbounded number of instances. We note that requiring a passive attacker is unavoidable unless we let the attacker have the session key but *not* the private key of one of the devices. Since the attacker would have gotten the session key by compromising a device, we find the passive attacker to be the most realistic and honest assumption.

The queries' details and ProVerif's output are the same as for BB-Pairing so we do not repeat them.

4.6.4 ProVerif Models

In this section, we present the ProVerif models of the protocols: BB-Pairing (with NC Association) in Listing 4.3, BB-Session in Listing 4.4, and BB-Rekey in Listings 4.5 and 4.6. Each model lists the entire process and all the queries, which is sufficient to check the validity of the model. We

leave out some type and event definitions to improve readability. Listing 4.2 shows the models of the cryptographic primitives used throughout the other four listings.

```
type key.

fun ENC(key, bitstring): bitstring.
reduc forall m: bitstring, k: key; DEC(k, ENC(k, m)) = m.

fun MAC(key, bitstring): bitstring.
reduc forall k: key, m: bitstring; VERIFY(k, m, MAC(k, m)) = true.

fun HASH(bitstring): bitstring.
fun KDF(key, bitstring): key.

type publickey.
type privatekey.
fun pk(privatekey): publickey.

fun DH(publickey, privatekey): key.
equation forall a,b: privatekey; DH(pk(b), a) = DH(pk(a), b).

(* encoding for keys *)
fun K2B(key): bitstring [typeConverter].
reduc forall m: key; B2K(K2B(m)) = m.

(* encoding for publickeys *)
fun PK2B(publickey): bitstring [typeConverter].
reduc forall m: publickey; B2PK(PK2B(m)) = m.
```

Listing 4.2: Cryptographic primitives.

4.7 BlueBrothers Implementation

We implement BlueBrothers for BLE by patching NimBLE, an open-source BLE stack by the Apache Software Foundation compliant with Bluetooth v5.4. Our implementation supports any device compatible with NimBLE, including nRF51, nRF52, nRF53, and DA1469x chips. We also implement BlueBrothers for BC at the application layer using Linux and BlueZ. We picked the following primitives for our implementations: Ascon and ChaChaPoly for AEAD, Ascon-Hash and Blake2b for secure hashing, key derivation and OTP, Curve 25519 for ECDH, and Kyber768 for PQDH. The BLE implementation also includes the primitives of the standard, like curve P256, AES-CMAC, and AES-CCM.

As our implementation is a POC, we discourage its usage in production. Moreover, it is not tested against side-channel attacks nor optimized for performances. Next, we describe the BlueBrothers protocols' implementation details for BLE and BC.

4.7.1 BLE Implementation for NimBLE

The BLE implementation involves patches to NimBLE's Host and Controller components. We implement BB-Pairing and BB-Session by patching NimBLE's Security Manager (SM) component in the Host. Moreover, we add logic and messages to the NimBLE's Link-Layer (LL) component in the Controller to provide a key confirmation mechanism. BB-Rekey is implemented in a new Host component, which we call the Rekey Manager (RM), and by extending the LL component. Next, we describe the modifications in detail.

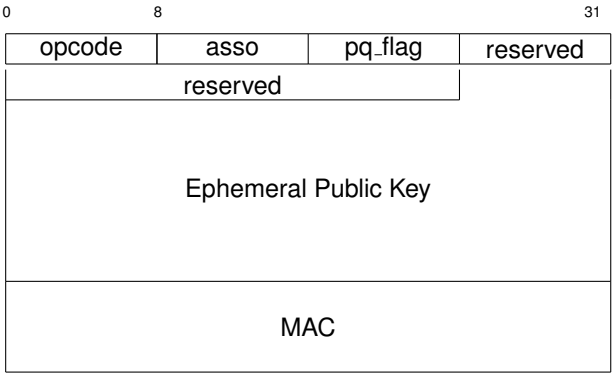

Figure 4.12: BB-Pairing SMP Pairing Request/Response packet. The 1-byte IO capabilities and OOB flags are the same as in the standard. The size of the Ephemeral Public Key is 32 bytes for ECDH and 1184 bytes for PQDH. The MAC size is 16 bytes.

NimBLE Host

Our patched Host allows compile-time configuration to enable different C25519 for ECDH and PQDH via two setting flags, i.e., `ENABLE_SM_25519` and `ENABLE_SM_PQ`. Disabling its related code at compilation time saves memory when not using PQDH. We set the SM L2CAP channel Maximum Transmission Unit (MTU) to 87 bytes for ECDH and to 1208 bytes for PQDH. In the SM component (see `ble_sm_alg.c`), we also implement the following high-level cryptographic functions. Each function calls a reference cryptographic implementation we embed in the stack:

- `update_hash_transcript` to update T.
- `aead_encrypt/decrypt` to perform AEAD encryption/decryption.
- `crypto_auth` and `crypto_auth_verify` to compute and verify the MACs.
- `hkdf` to implement the HMAC Key Derivation Function (HKDF) construct.
- `hotp` to generate HOTP one-time numeric codes during association.

We modify some SMP packets and create new ones to implement BB-Pairing and BB-Session. We also modify and create the respective packet processing functions and insert them in NimBLE's SM state machine `ble_sm_dispatch`, replacing the standard ones.

For example, Figure 4.12 shows the structure of the modified SMP Pairing Request and Response messages used for BB-Pairing. The packet contains an `asso` byte-field that indicates the association mechanism, a `pq` byte-flag that indicates support for PQDH, four reserved bytes, a DH public key up to 1184 bytes, and a 16-byte MAC. The `asso` field uses bitwise flags to efficiently represent combinations of options (JW, NC, PE, OOB) using a single uint8_t (8-bit byte). Each flag is assigned a unique bit position (e.g., $\mathrm{JW} = 1 << 0$, $\mathrm{NC} = 1 << 1$), allowing their combination using bitwise OR $(|)$ and check them using bitwise AND $(\&)$. For example, `flags = FLAG_JW | FLAG_PE` sets both the JW and PE bits. This approach enables compact storage and fast checks for the presence or absence of any combination of the four flags, using values from 0 to 15. The complete mapping is at the beginning of the `ble_sm_priv.h` file. To process the new Pairing Request and Response messages, we modify the `ble_sm_pair_req_rx` and `ble_sm_pair_req_rsp` functions.

When devices perform association, they check their respective supported mechanism and pick the most secure that is common to both, e.g., if they both allow JW and NC, they pick the

latter. If the devices negotiated NC or PE, they use `hotp` to generate the association passcode, otherwise for JW, they use `hkdf` to derive K from T and the DH shared secret. If they opted for OOB association, they use the `hkdf` function to derive K from the OOB shared secret and the DH one. For the static key exchange of BB-Pairing, we modify the Pairing Public Key message by adding a 16-byte MAC field. While for the optional PQDH key exchange, we define a new message called Pairing PQ Key and implement two message handler functions for it `ble_sm_sc_pqdh_exec` and `ble_sm_sc_pqdh_rx`.

Once the session key (K) is computed in the Host, it is sent to the Controller using a new HCI command we call BLE_HCI_LE_F_ENCRYPT with opcode 0x0100. BB-Session implementation relies on similar modifications. Hence, we skip its description.

BB-Rekey relies on a new component we call Rekey Manager (RM) in the `host/src/ble_rm.c` file. The RM uses a dedicated L2CAP channel with seven (7) as channel identifier (CID) and uses two messages `ble_rm_rekey_req` and `ble_rm_rekey_rsp` containing either the public keys for the asymmetric mode, or a 32-byte random nonce for the symmetric mode. The RM exposes a rekey API through BLE's Generic Access Profile (GAP), which allows Central to start a key refresh from the application layer. The devices compute $\mathrm{K_N}$ in the Host as in Figures 4.7 and 4.8 and send it to the Controller using a new HCI command with call BLE_HCI_LE_REKEY with opcode 0x0101.

NimBLE Controller

In the Controller, we add four LL messages:

1. `LL_ENCF_REQ` opcode 0x2A
2. `LL_ENCF_RSP` opcode 0x2B
3. `LL_REKEY_REQ` opcode 0x2C
4. `LL_REKEY_RSP` opcode 0x2D

The first two messages are used for session key agreement by BB-Pairing and BB-Session. The first is in cleartext, while the second is encrypted and authenticated with K. The last two messages have the same purposes for BB-Rekey: the first is encrypted and authenticated with K while the second is with $\mathrm{K_N}$. These messages do not appear in the MSCs as they are specific to the BLE implementation. They are necessary because, in BLE, the Host handles asymmetric cryptography and pairing-related operations, while the Controller handles low-level encryption.

4.7.2 BC Implementation for BlueZ

We implement the BlueBrothers protocols at the application layer for BC as a Linux application using BlueZ. The protocols run over L2CAP channel 0x0235 and are implemented in a C library called `bb-lib`. The library provides a high-level API to build and process the messages. For example, `bb_session_rx` and `bb_session_build` are the functions to process and build BB-Session messages.

4.8 Performance Evaluation for BLE

We evaluate the latency and power consumption of BlueBrothers protocols for BLE. We focus on a worst-case scenario where constrained BLE devices (nRF52 boards) communicate. We compare our protocols in different configurations (e.g., classic vs. PQ key agreement) and with the ones in the standard. We consider the latter as our performance evaluation baseline. Our experimental results show that the BlueBrothers protocols have acceptable time and power overheads or perform even better than the baseline while providing much better security guarantees. Next, we describe our experimental setup and results.

4.8.1 Setup

Our setup includes two Nordic nRF52840 development boards with Mynewt v1.13.0 and our patched version of NimBLE v1.8.0. Mynewt is the reference open-source real-time OS for NimBLE, which Apache also maintains. We position the boards one meter (1m) apart in a room with other BT and Wi-Fi devices sharing the 2.4 GHz band. We run measurements with different configurations for each protocol, e.g., using Hybrid PQ mode and using ECDH with C25519 or P-256. The Hybrid PQ mode uses a combination of Kyber768 and C25519.

The devices run custom Central and Peripheral applications and NimBLE stack. The Central is a modified `btshell` app [19] providing a serial interface to automatically test connecting, pairing, session establishment, and asymmetric and symmetric rekeys. The Peripheral uses the `bleprph` app [19].

We measure latency using the Mynewt native time APIs, which have microseconds (us) resolution, and we convert these values to milliseconds (ms) for readability. We compute power consumption in Coulombs (C) using a Nordic Power Profiler Kit II sampling at 1000 Hz. We convert the values to µAh to compare against actual devices' power consumption data. We reduce experimental noise by running ten iterations for each protocol configuration and computing the average latency value and standard deviation. This gives us latency measurements, which we use to estimate the power consumption from the data registered by the Nordic Power Profiler application. For instance, the nRF52 provides timestamps relative to its boot process, while the power profiler application timestamps are relative to the beginning of the monitoring. Since they are independent, we isolate the protocols run by placing the starting point 100 ms before the first energy consumption spike, representing the first message the Peripheral device received as shown in Figure 4.13.

4.8.2 Results

Table 4.2 shows the latency and power consumption experimental results. It compares three BB-Pairing configs against BLE pairing and session establishment (rows 1–4), three BB-Session configs against session establishment (rows 5–8), and one symmetric and three asymmetric BB-Rekey rekeys against the BLE key refresh procedure (rows 9–13). Latency results are shown in Table 4.2's second column in ms. They indicate a range computed from ten experimental runs' average and standard deviation. The power consumption (in µAH) is computed over the output data of the Nordic Power Profiler application using the average latency to isolate the protocol run as explained in Section 4.8.1.

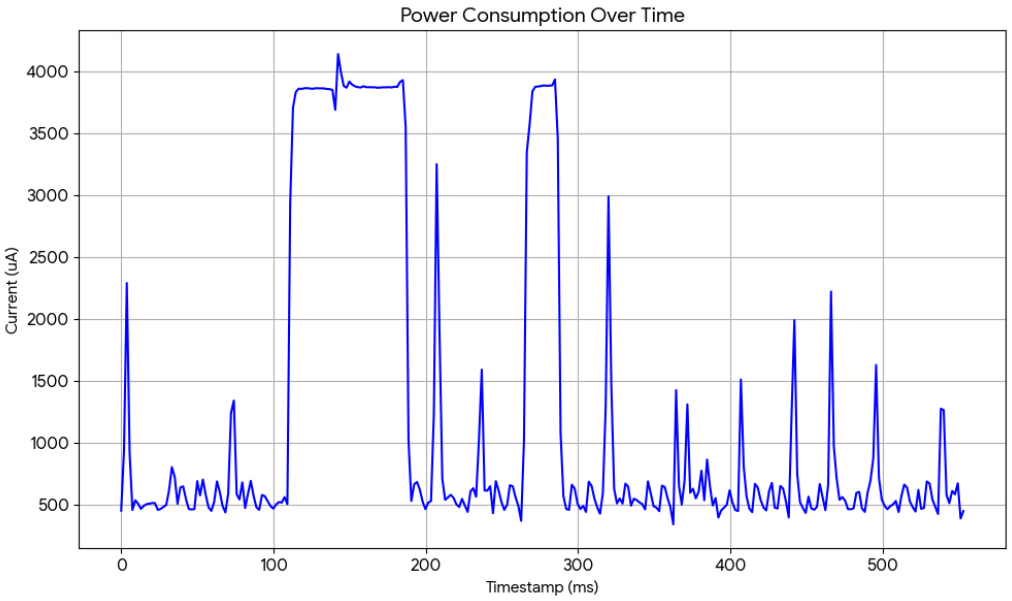

Figure 4.13: BB-Pairing Power Consumption Chart. We consider the start of the protocol as 100ms before the first energy spike, which corresponds to the first DH computation.

Latency Range

BB-Pairing with C25519 and Hybrid PQ is faster than pairing plus session establishment. The latency is reduced from 1225 ms to 553 ms (-54.9%) when using C25519 and to 692 ms (-55.6%) when using Hybrid PQ mode. With P-256, the latency increases to 1464 ms, causing a 19.5% overhead.

BB-Session is slower than the standard session establishment. The baseline latency is 328 ms, increasing to 556 ms (+69.5%) when using C25519 and 656 ms (+100.8%) with the Hybrid PQ configuration. When using P-256, the latency increases to 1830 ms (+468.3%). These results are expected as BB-Session employs asymmetric cryptography while session establishment is symmetric.

BB-Rekey is faster than the standard key refresh except when using P-256 in asymmetric mode. The baseline latency is 456 ms, and it improves to 186 ms (-59.3%) when using the symmetric rekey, to 389 ms (-14.7%) with asymmetric C25519, to 322 ms (-29.3%) with asymmetric Hybrid PQ. Asymmetric P-256 is slower than the baseline as it has a latency of 1072 ms (+134.0%).

From the latency results, we conclude that DH key exchange using P-256 is unfit for the protocols as its latency overhead is too high. However, C25519 and Hybrid PQ are faster than the baseline for BB-Pairing and BB-Rekey. For BB-Session we argue that the latency overhead with C25519 or Hybrid PQ is acceptable in a performance-security trade-off, as forward and future secrecy require using asymmetric cryptography.

Average Power

BB-Pairing has generally lower power consumption than pairing plus session establishment. The power consumption goes from 0.524 µAh to 0.187 µAh (-64.3%) when using C25519 and from 0.524 µAh to 0.176 µAh (-66.5%) when using Hybrid PQ. In contrast, P-256's power consumption is slightly higher at 0.606 µAh (+15.6%).

Table 4.2: BLE latency and power consumption of several BlueBrothers configurations against their counterparts in the standard. BB-Pairing is compared against standard pairing plus session establishment as it provides both features. The latency is shown as a range computed as $\mathrm{avg} \pm \mathrm{stdev}$. The Hybrid PQ mode uses Kyber768 and C25519.

BLE Protocol	**Latency (ms)**	**Power (µAh)**
BB-Pairing C25519	$548 - 558$	0.187
BB-Pairing Hybrid PQ	$680 - 704$	0.244
BB-Pairing P-256	$1463 - 1465$	0.606
Std Pairing+Session Est.	$1218 - 1232$	0.524
BB-Session C25519	$552 - 560$	0.192
BB-Session Hybrid PQ	$650 - 662$	0.239
BB-Session P-256	$1803 - 1857$	0.523
Std Session Est.	$327 - 329$	0.055
BB-Rekey Symmetric	$175 - 198$	0.040
BB-Rekey Asym C25519	$372 - 406$	0.123
BB-Rekey Asym Kyber768	$307 - 337$	0.106
BB-Rekey Asym P-256	$1059 - 1085$	0.336
Std Session Rekey	$440 - 472$	0.076

BB-Session has a higher power consumption than the standard session establishment, going from 0.055 µAh to 0.192 µAh (+249.1%) when using C25519, to 0.239 µAh (+334.5%) when using Hybrid PQ, and to 0.523 µAh (+850.9%) when using P-256. These results are expected due to our use of asymmetric cryptography.

We argue that BB-Session's power overhead is acceptable even for a real-world constrained device when using C25519 or Hybrid PQ. For instance, the average daily power consumption of an Apple AirTag, whose battery is a 240 mAh CR2032, is about 650 µAh. To cause a small 5% (32.5 µAh) increase in the daily power consumption, a device would need 169 sessions per day when using C25519 or 136 when using Hybrid PQ. These amounts of session establishments per day are unrealistic, as session establishment happens when two devices disconnect and reconnect.

BB-Rekey symmetric rekey has a lower power consumption than the baseline, while asymmetric rekey performs slightly worse than the baseline. The power goes from 0.076 µAh to 0.040 µAh (-47.4%) when using the symmetric rekey, to 0.123 µAh (+62.5%) when using the asymmetric rekey with C25519, to 0.106 µAh (+39.5%) when using Hybrid PQ, to 0.336 µAh (+342.1%) when using P-256.

BB-Rekey's power overheads are small and acceptable. For example, to obtain the same 5% increase in the daily power consumption of an Apple AirTag, a device would need to perform 812 symmetric rekeys per day, 264 asymmetric rekey with C25519 or 306 with Hybrid PQ. This means that devices could perform asymmetric rekey every 10 minutes without impacting the battery life. While even more frequent symmetric rekeys.

Overall, the power results confirm that no protocols should be used with P-256 as it introduces a high power consumption overhead. In contrast, C25519 and Hybrid PQ perform better and are closer to the baseline. We note, however, that BB-Pairing power consumption is not as relevant

as BB-Session or BB-Rekey since pairing is executed sporadically. Hence, P-256 could still be suitable for BB-Pairing.

4.9 Related Work

Security Extensions. Researchers developed Bluetooth security extensions to cover specific attacks. For example, [97] presents a lightweight certificate-based alternative to BLE pairing with JW. [86] mitigates attacks against SC pairing in a backward-compliant way using deferred challenge-response authentication. Legacy BT pairing was extended with asymmetric cryptography to address passive and active attacks [206]. We do not focus only on pairing or authentication, but we improve the security of the entire Bluetooth stack without introducing a trusted PKI.

IDS. Researchers explored Bluetooth Intrusion Detection System (IDS). Reconnaissance, DoS, and information theft attacks on legacy BC were addressed by [147]. BlueShield [207] detects spoofing attacks on BLE via cyber-physical fingerprints. OASIS [52] embeds an IDS inside a popular BLE controller for low-level detection. BlueSWAT [53] focuses on detecting session-based BLE attacks via a finite state machine and eBPF. BLEGuard [45] checks for BLE spoofing attacks via pre-detection, reconstruction, and classification models built using a dataset of simulated attacks. IDSs are orthogonal to the protocols proposed in our work.

Testing. Tools for testing the security of Bluetooth implementations are available in the literature. The BLEDiff [116] framework checks BLE protocol noncompliance via differential testing, while the Frankenstein [161] fuzzer relies on firmware emulation. More BC and BLE fuzzers were presented in SweynTooth [93] and BrakTooth [92]. Implementation testing is orthogonal to our work, as we focus on the design of Bluetooth protocol.

Formal Verification. Several Bluetooth security protocol phases have been formally modeled and verified using the Tamarin and ProVerif symbolic protocol verifiers. In [181], the authors modeled BLE SC pairing associations with Tamarin [137], rediscovering two attacks and uncovering a new one. [112] developed a Tamarin model for BLE Passkey Entry and used it to rediscover three attacks and find two new ones. In [56], the authors used Tamarin to analyze the key agreement protocol and discover two new pairing confusion attacks.

In [208, 209], the authors modeled key sharing and data transmission of BC, BLE, and Mesh with ProVerif, finding five known vulnerabilities and two other security issues. Always using ProVerif [177] found a double misbinding attack on BC pairing. Our work relies on ProVerif to formally verify our new protocols, not the existing ones since we aim to provide a simpler and more secure alternative.

BC PE was also computationally analyzed with the CYBORG authenticated key exchange model, which can study attacks on human-assisted association [189]. [85] cryptographically analyzed SC pairing protocols of both BC and BLE against active attacks in the TOFU model.

4.10 Conclusion

This work presents the BlueBrothers protocols to replace the standard Bluetooth pairing and session establishment. The need for new Bluetooth security protocols is motivated by the design vulnerabilities and attacks affecting the current ones and their complex specifications. This leads to implementation-level flaws and makes formal verification and security analysis challenging.

BB-Pairing provides authenticated pairing and session establishment. BB-Session guarantees authenticated key agreement using the static public key distributed via BB-Pairing. It benefits from key compromise protection (i.e., inter-session forward and future secrecy) by leveraging ephemeral DH operations. BB-Rekey is a novel key refresh mechanism enabling intra-session forward and future secrecy.

The BlueBrothers protocols advance the state-of-the-art for Bluetooth security. They provide new Bluetooth security properties, like intra-session forward and future secrecy or hybrid PQ key agreement. They address four classes of design issues (C1–C4) and seventeen related impactful attacks (Table 4.1).

They can be integrated with the Bluetooth standard using a negotiation mechanism or as standalone application-layer protocols. They use open cryptographic mechanisms and primitives. Their design and implementation are open source. For instance, their design follows the ephemeral-static DH model used in the Noise Protocol Framework [153], WireGuard [71], and WhatsApp [202]. While the rekeying logic is inspired by the Double Ratchet algorithm [154].

We successfully model and verify the BlueBrothers protocols and their security properties with ProVerif. We implement the protocols by patching a real-world BLE stack (NimBLE) and as a BlueZ application for BC. We evaluated their latency and power consumption performance with constrained BLE devices (nRF52). The experiments show that BlueBrothers impact on latency and energy consumption is better in some cases and worse in others than the standard Bluetooth security protocols. However, when worse, the impact is negligible in practice, making BlueBrothers suitable for real-world deployment.

```
let User() =
  in(uvisc, (Kc: key));
  in(uvisp, (Kp: key));
  if Kc = Kp then
  ( out(uinp, (ok)); event UserSaysOK(Kc) )
  else
  ( out(uinp, (abort)); event UserSaysAbort(Kc) ).
let Central(Fc: bitstring, Sc: privatekey) =
  new Ec: privatekey;
  out(c, (pk(Ec), Fc));
  in(c, (pkep: publickey, Fp: bitstring, m1: bitstring));
  let K1 = DH(pkep, Ec) in
  let T1 = HASH((pk(Ec), Fc, pkep, Fp)) in
  if VERIFY(K1, T1, m1) then
  out(uvisc, (K1));
  in(uinp, (result: decition));
  if result = ok then
  let K2 = KDF(K1, T1) in
  event CentralAccepts(K1, Fc, Fp);
  let T2 = HASH((T1, ENC(K2, PK2B(pk(Sc))))) in
  out(c, (ENC(K2, PK2B(pk(Sc))), MAC(K2, T2)));
  in(c, (encpksp: bitstring, m2: bitstring));
  let T3 = HASH((T2, encpksp)) in
  if VERIFY(K2, T3, m2) then
  let K = KDF(K2, (T3, DH(B2PK(DEC(K2,encpksp)), Sc))) in
  out(c, (ENC(K, SKtest)));
  event CentralTerm(K1, K, Fc, Fp);
  0.
let Peripheral(Fp: bitstring, Sp: privatekey) =
  in(c, (pkec: publickey, Fc: bitstring));
  new Ep: privatekey;
  let K1 = DH(pkec, Ep) in
  let T1 = HASH((pkec, Fc, pk(Ep), Fp)) in
  out(c, (pk(Ep), Fp, MAC(K1, T1)));
  out(uvisp, (K1));
  in(uinp, (result: decition));
  if result = ok then
  let K2 = KDF(K1, T1) in
  event PeripheralAccepts(K1, Fc, Fp);
  in(c, (encpksc: bitstring, m1: bitstring));
  let T2 = HASH((T1, encpksc)) in
  if VERIFY(K2, T2, m1) then
  let T3 = HASH((T2, ENC(K2, PK2B(pk(Sp))))) in
  out(c, (ENC(K2, PK2B(pk(Sp))), MAC(K2, T3)));
  let K = KDF(K2, (T3, DH(B2PK(DEC(K2,encpksc)), Sp))) in
  out(c, (ENC(K, SKtest)));
  event PeripheralTerm(K1, K, Fc, Fp);
  0.
process
  new Fc: bitstring; out(c, Fc);
  new Fp: bitstring; out(c, Fp);
  new Sc: privatekey; out(c, pk(Sc));
  new Sp: privatekey; out(c, pk(Sp));
  ((User)|(Central(Fc, Sc))|(Peripheral(Fp, Sp)))
(* secrecy *)
query attacker(SKtest).
(* integrity *)
query sk1, sk2, k: key, n1, n2: bitstring; event(PeripheralTerm(sk1, k, n1, n2)) && event(
    CentralTerm(sk2, k, n1, n2)) ==> sk1 = sk2.
(* user decition cannot be bypassed *)
query k, sk: key, n1, n2: bitstring; event(CentralTerm(sk, k, n1, n2)) && event(PeripheralTerm
    (sk, k, n1, n2)) ==> event(UserSaysOK(sk)).
(* authentication *)
query k, sk: key, n1, n2: bitstring; event(CentralTerm(sk, k, n1, n2)) ==> inj-event(
    PeripheralAccepts(sk, n1, n2)).
query k, sk: key, n1, n2: bitstring; event(PeripheralTerm(sk, k, n1, n2)) ==> inj-event(
    CentralAccepts(sk, n1, n2)).
```

Listing 4.3: BB-Pairing model.

```
let Central(Fc: bitstring, pksp: publickey, Sc: privatekey) =
  new Ec: privatekey;
  let K1 = DH(pksp, Ec) in
  let T1 = HASH((pk(Ec), Fc)) in
  out(c, (pk(Ec), Fc, MAC(K1, T1)));
  in(c, (pkep: publickey, Fp: bitstring, m1: bitstring));
  let K = KDF(K1, K2B(DH(pkep, Sc))) in
  let T2 = HASH((T1, pkep, Fp)) in
  if VERIFY(K, T2, m1) then
  event CentralAccepts(K1, Fc, Fp);
  out(c, ENC(K, SKtest));
  event CentralTerm(K1, Fc, Fp);
  0.
let Peripheral(Fp: bitstring, pksc: publickey, Sp: privatekey) =
  in(c, (pkec: publickey, Fc: bitstring, m1: bitstring));
  let K1 = DH(pkec, Sp) in
  let T1 = HASH((pkec, Fc)) in
  if VERIFY(K1, T1, m1) then
  event PeripheralAccepts(K1, Fc, Fp);
  new Ep: privatekey;
  let K = KDF(K1, K2B(DH(pksc, Ep))) in
  let T2 = HASH((T1, pk(Ep), Fp)) in
  out(c, (pk(Ep), Fp, MAC(K, T2)));
  out(c, ENC(K, SKtest));
  event PeripheralTerm(K1, Fc, Fp);
  0.
process
  new Fc: bitstring; out(c, Fc);
  new Fp: bitstring; out(c, Fp);
  new Sc: privatekey; out(c, pk(Sc));
  new Sp: privatekey; out(c, pk(Sp));
  (!Central(Fc, pk(Sp), Sc))|(!Peripheral(Fp, pk(Sc), Sp))
(* secrecy *)
query attacker(SKtest).
(* authentication *)
query sk: key, n1, n2: bitstring; event(CentralTerm(sk, n1, n2)) ==> inj-event(
    PeripheralAccepts(sk, n1, n2)).
query sk: key, n1, n2: bitstring; event(PeripheralTerm(sk, n1, n2)) ==> inj-event(
    CentralAccepts(sk, n1, n2)).
```

Listing 4.4: BB-Session model.

```
let Central() =
  new Ec: privatekey;
  out(c, (pk(Ec), MAC(K, (P, pk(Ec)))));
  in(c, (pkep: publickey, m1: bitstring));
  if VERIFY(K, (C, pkep), m1) then
  event CentralAccepts(K, pk(Ec), pkep);
  let SS = DH(pkep, Ec) in
  let Kn = KDF(K, K2B(SS)) in
  out(c, ENC(Kn, KNtest));
  event CentralTerm(Kn, K, pk(Ec), pkep);
  0.
let Peripheral() =
  in(c, (pkec: publickey, m1: bitstring));
  if VERIFY(K, (P, pkec), m1) then
  new Ep: privatekey;
  event PeripheralAccepts(K, pkec, pk(Ep));
  out(c, (pk(Ep), MAC(K, (C, pk(Ep)))));
  let SS = DH(pkec, Ep) in
  let Kn = KDF(K, K2B(SS)) in
  out(c, (ENC(Kn, KNtest)));
  event PeripheralTerm(Kn, K, pkec, pk(Ep));
  0.
process
  ((Central())|(Peripheral()))
(* secrecy *)
query attacker(K).
query attacker(KNtest).
(* authentication *)
query k, kn: key, pkc, pkp: publickey; event(CentralTerm(kn, k, pkc, pkp)) ==> inj-event(
    PeripheralAccepts(k, pkc, pkp)).
query k, kn: key, pkc, pkp: publickey; event(PeripheralTerm(kn, k, pkc, pkp)) ==> inj-event(
    CentralAccepts(k, pkc, pkp)).
(* integrity *)
query k, k1, k2: key, pkc, pkp: publickey; event(PeripheralTerm(k1, k, pkc, pkp)) && event(
    CentralTerm(k2, k, pkc, pkp)) ==> k1 = k2.
```

Listing 4.5: BB-Rekey model.

```
process
  out(c, K); (* Give old key to the passive attacker *)
  ((!Central())|(!Peripheral()))

(* future secrecy *)
set attacker = passive.
query attacker(KNtest).
```

Listing 4.6: BB-Rekey model for future secrecy.

Chapter 5

Summary and Conclusion

D5.1 successfully addresses its goals and advances the state of the art of essential and beyond essential S&P guarantees for inter-device communication in restricted environments. It addresses **T5.1** and **T5.2** by uncovering vulnerabilities and attacks in state of the art inter-device communication technologies such as Bluetooth and FIDO2. Moreover, it creates original and stronger alternatives like the BlueBrothers protocols for Bluetooth. Next, we recall the tasks from ORSHIN's description of action document, and explain how we complete them with **D5.1**.

T5.1 Description

> "*Essential S&P guarantees for inter-device communication*. Constrained device-to-device communication, such as the ones between IoT and IIoT devices, might not always provide necessary S&P guarantees for several reasons. For example, a device might lack a hardware accelerator block to perform cryptographic operations that are too expensive in software. With the advent of open-source hardware, we will have access to more and better hardware blocks that we can integrate into embedded devices to provide essential security guarantees by design."

T5.1 Completion Chapter 1 addresses **T5.1** by showing how real-world and pervasive proprietary intra-device communication protocols used for e-scooters are not capable of providing essential S&P guarantees, despite promising them when marketing the products. While, Chapter 3 targets **T5.1** by demonstrating that even standard and more open intra-device communication protocols, such as FIDO2, can fail to provide essential S&P guarantees, like confidentiality, integrity and authenticity. Chapter 4 presents new intra-device communication protocols that are not vulnerable to the attacks presented in the other Chapters but provide essential S&P properties by design using a simple and open specification.

T5.2 Description

> "*Beyond essential S&P guarantees for inter-device communication*. Secure messaging applications, such as Signal and WhatsApp, provide attractive security guarantees beyond the essentials, such as forward and backward secrecy. Unfortunately, those guarantees are very rarely considered in constrained communication environments, such as IoT and IIoT, because they are complex and relatively new in the field. We want to fill this gap by pivoting on open-source hardware to develop protocols providing "beyond-essential" security guarantees for constrained devices. Those protocols

will significantly raise the bar for an attacker by providing the best possible defence solutions in the most constrained scenarios. For example, those advanced protocols will protect the confidentiality of past and future data even if the present secret key is compromised."

Addressing T5.2 Chapter 2 addresses **T5.2** by assessing the forward and future secrecy guarantees in the Bluetooth standard. It demonstrates that these guarantees are not holding by uncovering novel vulnerabilities and attacks and recommend effective countermeasures. Moreover, Chapter 3 contributes to **T5.2** by analyzing beyond-essential properties of FIDO2 like resistance against credential deletion or tracking attacks. It uncovers new vulnerabilities and attacks capable of violating these properties and discuss effective fixes. Chapter 4 completes **T5.2** by proposing new inter-device communication protocols providing beyond-essential S&P properties like forward and future secrecy, identity hiding, and forward and future secrecy across and among Bluetooth sessions. The BlueBrothers artifact will be used in **D5.3** and **WP6**.

Research Contributions Overall, **D5.1** generates *nine open access research papers and open source research artifacts*. The papers were presented at prestigious security and privacy scientific conferences and journals (e.g., S&P CORE A* and A), hacking events (e.g., BlackHat US and DEF CON), invited research talks (e.g., Uni of Oxford and ETHZ) or are under submission for such venues. The deliverable, for space reasons, covers four papers but next we list them all:

P1: Chapter 1 appears in a paper titled *E-Spoofer: Attacking and Defending Xiaomi Electric Scooter Ecosystem* at the ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec), 2023 [50] and as a selected talk at the Toulouse Hacking Convention (THCON), 2024 [48].

P2: Chapter 2 appears in a paper titled *BLUFFS: Bluetooth Forward and Future Secrecy Attacks and Defenses* at the ACM Conference on Computer and Communications Security (CCS), 2023 [10] and in two selected talks at the 37th Chaos Communication Congress (37c3), 2023 [11] and Toulouse Hacking Convention (THCON), 2024 [12].

P3: Chapter 3 appears in a paper titled *CTRAPS: CTAP Impersonation and API Confusion on FIDO2* at the IEEE European Symposium on Security and Privacy (Euro S&P), 2025. Moreover, it is accepted at DEF CON 33 (2025). [49].

P4: Chapter 4 is currently under submission in a paper titled *BlueBrothers: Three New Protocols to Enhance Bluetooth Security* at the IEEE Symposium on Security and Privacy (S&P), 2026.

P5: A paper titled *CheckOCPP: Automatic OCPP Packet Dissection and Compliance Check* is accepted at IEEE Automotive Cybersecurity Euro S&P Workshop (ACSW), 2025.

P6: A paper titled *Bluetooth Security Testing with BlueToolkit: a Large-Scale Automotive Case Study* is accepted at USENIX Workshop On Offensive Security (WOOT), 2025

P7: A paper titled *SimProcess: High Fidelity Simulation of Noisy ICS Physical Processes* is accepted at ACM Cyber-Physical System Security Workshop (CPSS), 2025

P8: A paper titled *EmuOCPP: Effective and Scalable OCPP Security and Privacy Testing* is accepted at USENIX Symposium on Vehicle Security and Privacy (VehicleSec), 2025

P9: A paper titled *E-Trojans: Ransomware, Tracking, DoS, and Data Leaks on Battery-powered Embedded Systems* is under submission at the Annual Computer Security Applications

Conference (ACSAC), 2025. Moreover, it is accepted at Black Hat USA (BH US), 2025.

For resources about a paper, like slides, code, demos, see: https://francozappa.github.io/publication/

Bibliography

[1] Mingrui Ai, Kaiping Xue, Bo Luo, Lutong Chen, Nenghai Yu, Qibin Sun, and Feng Wu. Blacktooth: Breaking through the Defense of Bluetooth in Silence. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 55–68, 2022.

[2] 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.

[3] 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.

[4] 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.

[5] FIDO Alliance. FIDO Alliance Specifications Overview. https://fidoalliance.org/specifications, 2024.

[6] FIDO Alliance. FIDO Functional Certification. https://fidoalliance.org/certification/functional-certification/, 2024.

[7] FIDO Alliance. FIDO Security Secretariat. https://fidoalliance.org/certification/secretariat/, 2024.

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

[9] 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.

[10] Daniele Antonioli. BLUFFS: Bluetooth Forward and Future Secrecy Attacks and Defenses. In *ACM Conference on Computer and Communications Security (CCS)*, November 2023.

[11] Daniele Antonioli. BLUFFS: Breaking and fixing the Bluetooth standard. One More Time. https://www.youtube.com/watch?v=2HstGZPZpZY, 2023. 37th Chaos Communication Congress (37c3).

[12] Daniele Antonioli. BLUFFS: Breaking and fixing the Bluetooth standard. One More Time., 2024. Toulouse Hacking Convention (THCON).

[13] Daniele Antonioli and Mathias Payer. On the Insecurity of Vehicles Against Protocol-Level Bluetooth Threats. In *2022 IEEE Security and Privacy Workshops (SPW)*, pages 353–362. IEEE, 2022.

[14] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. BIAS: Bluetooth impersonation attacks. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 549–562. IEEE, 2020.

[15] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. Key Negotiation Downgrade Attacks on Bluetooth and Bluetooth Low Energy. *ACM Transactions on Privacy and Security (TOPS)*, 23(3):1–28, 2020.

[16] Daniele Antonioli, Nils Ole Tippenhauer, Kasper Rasmussen, and Mathias Payer. BLURtooth: Exploiting Cross-Transport Key Derivation in Bluetooth Classic and Bluetooth Low Energy. In *Proceedings of the Asia conference on computer and communications security (ASIACCS)*, May 2022.

[17] 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.

[18] 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, 2019.

[19] Apache. NimBLE is an open-source BLE 5.4 stack. https://github.com/apache/mynewt-nimble, 2024.

[20] ARM Developers. ARM Thumb-2 instruction set. https://developer.arm.com/documentation/ddi0344/k/programmers-model/thumb-2-instruction-set, 2022.

[21] 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, 2021.

[22] 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.

[23] Elaine Barker, Lily Chen, Richard Davis, et al. Recommendation for key-derivation methods in key-establishment schemes. *NIST Special Publication*, 800:56C, 2018.

[24] Ltd Beijing Xiaomi Mobile Software Co. Mi Home (Android). https://play.google.com/store/apps/details?id=com.xiaomi.smarthome, 2022.

[25] Ltd Beijing Xiaomi Mobile Software Co. Mi Home (iOS). https://apps.apple.com/us/app/mi-home-xiaomi-smart-home/id957323480, 2022.

[26] Brian Benchoff. Security Engineering: Inside the Scooter Startups. https://hackaday.com/2019/02/12/security-engineering-inside-the-scooter-startups/, 2019.

[27] Christoforus Juan Benvenuto. Galois field in cryptography. *University of Washington*, 1(1):1–11, 2012.

[28] Eli Biham and Lior Neumann. Breaking the Bluetooth Pairing–Fixed Coordinate Invalid Curve Attack. http://www.cs.technion.ac.il/~biham/BT/bt-fixed-coordinate-invalid-curve-attack.pdf, 2018.

[29] 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.

[30] 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.

[31] Bruno Blanchet et al. Modeling and verifying security protocols with the applied pi calculus and ProVerif. *Foundations and Trends® in Privacy and Security*, 1(1-2):1–135, 2016.

[32] Bluetooth SIG. Bluetooth Market Update 2020. https://www.bluetooth.com/bluetooth-resources/2020-bmu/, 2020.

[33] Bluetooth SIG. Bluetooth Core Specification v5.3. https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=521059, 2021.

[34] Bluetooth SIG. Bluetooth Market Update 2021. https://www.bluetooth.com/bluetooth-resources/2021-bmu/, 2021.

[35] Bluetooth SIG. Bluetooth Market Update 2022. https://www.bluetooth.com/2022-market-update/, 2022.

[36] Bluetooth SIG. Reporting Security Vulnerabilities. https://www.bluetooth.com/learn-about-bluetooth/key-attributes/bluetooth-security/reporting-security/, 2022.

[37] Bluetooth SIG. Bluetooth Core Specification v6.1. https://www.bluetooth.com/specifications/specs/core-specification-6-1/, 2025.

[38] 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.

[39] BotoX. Xiaomi M365 Firmware Patcher. https://github.com/BotoX/xiaomi-m365-firmware-patcher, 2022.

[40] Christiaan Brand. Advisory: Security Issue with Bluetooth Low Energy (BLE) Titan Security Keys. https://security.googleblog.com/2019/05/titan-keys-update.html, 2019.

[41] 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.

[42] Buildxyz. nRFSec. https://github.com/buildxyz-git/nrfsec, 2020.

[43] Bulkwarkid. Virtual FIDO. https://github.com/bulwarkid/virtual-fido, 2025.

[44] 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.

[45] Hanlin Cai. Securing Billion Bluetooth Devices Leveraging Learning-Based Techniques. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 23731–23732, 2024.

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

[47] CamiAlfa. M365Downg. https://play.google.com/store/apps/details?id=com.m365downgrade, 2022.

[48] Marco Casagrande, , and Daniele Antonioli. E-Spoofer: Attacking and Defending Xiaomi Electric Scooter Ecosystem, 2024. Toulouse Hacking Convention (THCON).

[49] Marco Casagrande and Daniele Antonioli. CTRAPS: CTAP Impersonation and API Confusion on FIDO2. In *IEEE European Symposium on Security and Privacy (Euro S&P)*, July 2025.

[50] Marco Casagrande, Riccardo Cestaro, Eleonora Losiouk, Mauro Conti, and Daniele Antonioli. E-Spoofer: Attacking and Defending Xiaomi Electric Scooter Ecosystem. In *ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2023.

[51] 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.

[52] Romain Cayre, Vincent Nicomette, Guillaume Auriol, Mohamed Kaâniche, and Aurélien Francillon. OASIS: An Intrusion Detection System Embedded in Bluetooth Low Energy Controllers. In *Asia Conference on Computer and Communications Security (AsiaCCS)*, 2024.

[53] Xijia Che, Yi He, Xuewei Feng, Kun Sun, Ke Xu, and Qi Li. BlueSWAT: A Lightweight State-Aware Security Framework for Bluetooth Low Energy. In *Conference on Computer and Communications Security (CCS)*, 2024.

[54] Tsai Chwei-Shyong, Lee Cheng-Chi, and Min-Shiang Hwang. Password Authentication Schemes: Current Status and Key Issues. *International Journal of Network Security*, 2006.

[55] 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.

[56] Tristan Claverie, Gildas Avoine, Stéphanie Delaune, and José Lopes Esteves. Tamarin-based Analysis of Bluetooth Uncovers Two Practical Pairing Confusion Attacks. In *European Symposium on Research in Computer Security*, pages 100–119. Springer, 2023.

[57] Tristan Claverie and José Lopes Esteves. Bluemirror: reflections on Bluetooth pairing and provisioning protocols. In *2021 IEEE Security and Privacy Workshops (SPW)*, pages 339–351. IEEE, 2021.

[58] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. *Journal of Cryptology*, 33:1914–1983, 2020.

[59] 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.

[60] Marco Cominelli, Francesco Gringoli, Paul Patras, Margus Lind, and Guevara Noubir. Even black cats cannot stay hidden in the dark: Full-band de-anonymization of Bluetooth Classic devices. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 534–548. IEEE, 2020.

[61] Mike Hanley (GitHub CSO). Securing millions of developers through 2FA. https://github.blog/2024-04-24-securing-millions-of-developers-through-2fa/, 2024.

[62] Daniele Antonioli (francozappa). KNOB attack repository on Github. https://github.com/francozappa/knob, 2020.

[63] Daniele Antonioli (francozappa). BIAS attack repository on Github. https://github.com/francozappa/bias, 2021.

[64] 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.

[65] Emma Dauterman, Henry Corrigan-Gibbs, David Mazières, Dan Boneh, and Dominic Rizzo. True2F: Backdoor-resistant authentication tokens. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 398–416, 2019.

[66] Dennis Mantz (demantz). BTBB Wireshark plugin from the Ubertooth libbtbb project. https://github.com/demantz/lmp_wireshark_dissector, 2022.

[67] 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.

[68] Android developers. Android NFC basics. https://developer.android.com/develop/connectivity/nfc/nfc, 2023.

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

[70] Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208, 1983.

[71] Jason A Donenfeld. WireGuard: Next Generation Kernel Network Tunnel. In *NDSS*, pages 1–12, 2017.

[72] Dor Green. Pyshark Python packet parser using wireshark's tshark. https://kiminewt.github.io/pyshark/, 2022.

[73] 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.

[74] John Dunning. Taming the blue beast: A survey of Bluetooth based threats. *IEEE Security & Privacy*, 8(2):20–27, 2010.

[75] Electronic Frontier Foundation (EFF). Cracking DES. https://archive.org/details/crackingdes00elec, 1998.

[76] STM Electronics. ST-LINK Debugger V2. https://www.st.com/en/development-tools/st-link-v2.html, 2022.

[77] ERNW. CVE-2020-0022 an Android 8.0-9.0 Bluetooth Zero-Click RCE – BlueFrag. https://insinuator.net/2020/04/cve-2020-0022-an-android-8-0-9-0-bluetooth-zero-click-rce-bluefrag/, 2020.

[78] 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.

[79] Feitian. Feitian Android App. https://play.google.com/store/apps/details?id=com.ft.entersafe.iepassmanager, 2022.

[80] Feitian. Feitian iOS App. https://apps.apple.com/us/app/iepassmanager/id1504200260, 2022.

[81] 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.

[82] 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)*, pages 19–24, 2017.

[83] 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.

[84] FIDO Alliance. FIDO Alliance Publishes New Specifications to Promote User Choice and Enhanced UX for Passkeys. https://fidoalliance.org/fido-alliance-publishes-new-specifications-to-promote-user-choice-and-enhanced-ux-for-passkeys, 2024.

[85] Marc Fischlin and Olga Sanina. Cryptographic analysis of the Bluetooth secure connection protocol suite. In *Advances in Cryptology–ASIACRYPT 2021*, pages 696–725. Springer, 2021.

[86] Marc Fischlin and Olga Sanina. Fake It till You Make It: Enhancing Security of Bluetooth Secure Connections via Deferrable Authentication. In *ACM conference on Computer and Communications Security (CCS)*, 2024.

[87] Scott Fluhrer and Stefan Lucks. Analysis of the E0 encryption system. In *Selected Areas in Cryptography: 8th Annual International Workshop, SAC 2001 Toronto, Ontario, Canada, August 16–17, 2001 Revised Papers 8*, pages 38–48. Springer, 2001.

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

[89] 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.

[90] VooDooShamane from Rollerplausch.com. MiDu Flasher. https://rollerplausch.com/threads/midu-flasher-st-link-downgrade-unbrick.5399/, 2022.

[91] 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.

[92] Matheus E Garbelini, Vaibhav Bedi, Sudipta Chattopadhyay, Sumei Sun, and Ernest Kurniawan. BrakTooth: Causing Havoc on Bluetooth Link Manager via Directed Fuzzing. In

31st USENIX Security Symposium (USENIX Security 22), pages 1025–1042, 2022.

[93] Matheus E. Garbelini, Chundong Wang, Sudipta Chattopadhyay, Sun Sumei, and Ernest Kurniawan. SweynTooth: Unleashing Mayhem over Bluetooth Low Energy. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 911–925. USENIX Association, 2020.

[94] 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.

[95] Google. OpenSK: a Rust Implementation of a FIDO2 Authenticator. https://github.com/google/OpenSK, 2024.

[96] 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.

[97] Chandranshu Gupta and Gaurav Varshney. An improved authentication scheme for BLE devices with no I/O capabilities. *Computer Communications*, 200:42–53, 2023.

[98] Keijo Haataja and Pekka Toivanen. Two practical man-in-the-middle attacks on Bluetooth Secure Simple Pairing and countermeasures. *Transactions on Wireless Communications*, 9(1):384–392, 2010.

[99] 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.

[100] 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.

[101] Simon Hay and Robert Harle. Bluetooth tracking without discoverability. In *International Symposium on Location-and Context-Awareness*, pages 120–137. Springer, 2009.

[102] 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.

[103] 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.

[104] Kexin Hu and Zhenfeng Zhang. Security analysis of an attractive online authentication standard: FIDO UAF protocol. *China Communications*, 13(12):189–198, 2016.

[105] Jun Huang, Wahhab Albazrqaoe, and Guoliang Xing. BlueID: A practical system for Bluetooth device identification. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 2849–2857. IEEE, 2014.

[106] Lin-Shung Huang, Shrikant Adhikarla, Dan Boneh, and Collin Jackson. An experimental study of TLS forward secrecy deployments. *IEEE Internet Computing*, 18(6):43–51, 2014.

[107] Konstantin Hypponen and Keijo MJ Haataja. Nino man-in-the-middle attack on Bluetooth Secure Simple Pairing. In *Proceedings of the International Conference in Central Asia on Internet*, pages 1–5. IEEE, 2007.

[108] Infineon. CYW20819. https://www.infineon.com/cms/en/product/wireless-connectivity/airoc-bluetooth-le-bluetooth-multiprotocol/airoc-bluetooth-le-bluetooth/cyw20819/, 2022.

[109] Internet Engineering Task Force (IETF). The Transport Layer Security (TLS) Protocol Version 1.3. https://www.rfc-editor.org/rfc/rfc8446, 2018.

[110] iResearch. iResearch Coverage On Xiaomi. http://www.iresearchchina.com/Upload/201808/20180824143739_3256.pdf, 2018.

[111] Markus Jakobsson and Susanne Wetzel. Security weaknesses in Bluetooth. In *Proceedings of the Cryptographers' Track at the RSA Conference*, pages 176–191. Springer, 2001.

[112] Mohit Kumar Jangid, Yue Zhang, and Zhiqiang Lin. Extrapolating Formal Analysis to Uncover Attacks in Bluetooth Passkey Entry Pairing. In *NDSS*, 2023.

[113] Jiska YouTube Channel. InternalBlue Tutorial - 2021 Edition. https://www.youtube.com/watch?v=UANnKx91vyg, 2021.

[114] 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.

[115] Imtiaz Karim, Abdullah Al Ishtiaq, Syed Rafiul Hussain, and Elisa Bertino. BLEDiff: Scalable and Property-Agnostic Noncompliance Checking for BLE Implementations. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1082–1100. IEEE Computer Society, 2023.

[116] Imtiaz Karim, Abdullah Al Ishtiaq, Syed Rafiul Hussain, and Elisa Bertino. BLEDiff: Scalable and property-agnostic noncompliance checking for ble implementations. In *2023 IEEE Symposium on Security and Privacy (SP)*, 2023.

[117] 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.

[118] 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.

[119] 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.

[120] David Kierznowski. BadUSB 2.0. https://github.com/withdk/badusb2-mitm-poc, 2016.

[121] 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.

[122] Sandeep Kumar, Christof Paar, Jan Pelzl, Gerd Pfeiffer, and Manfred Schimmler. Breaking ciphers with COPACOBANA–a cost-optimized parallel code breaker. In *Cryptographic Hardware and Embedded Systems-CHES 2006: 8th International Workshop, Yokohama, Japan, October 10-13, 2006. Proceedings 8*, pages 101–118. Springer, 2006.

[123] Ninja Lab. A Side Journey to Titan. https://ninjalab.io/a-side-journey-to-titan, 2024.

[124] 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.

[125] 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.

[126] Albert Levi, Erhan Çetintaş, Murat Aydos, Çetin Kaya Koç, and M Ufuk Çağlayan. Relay attacks on Bluetooth authentication and solutions. In *Proceedings International Symposium on Computer and Information Sciences*, pages 278–288. Springer, 2004.

[127] 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.

[128] 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.

[129] Andrew Y Lindell. Attacks on the pairing protocol of Bluetooth v2.1. *Black Hat USA, Las Vegas, Nevada*, 2008.

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

[131] Angela M Lonzetta, Peter Cope, Joseph Campbell, Bassam J Mohd, and Thaier Hayajneh. Security vulnerabilities in Bluetooth technology as used in IoT. *Journal of Sensor and Actuator Networks*, 7(3):28, 2018.

[132] 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.

[133] 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.

[134] Ahmed Tanvir Mahdad, Mohammed Jubur, and Nitesh Saxena. Breaching Security Keys without Root: FIDO2 Deception Attacks via Overlays exploiting Limited Display Authenticators. In *Proceedings of the ACM conference on computer and communications security (CCS)*, 2024.

[135] Dennis Mantz, Jiska Classen, Matthias Schulz, and Matthias Hollick. InternalBlue Bluetooth binary patching and experimentation framework. In *Proceedings of the 17th Annual*

International Conference on Mobile Systems, Applications, and Services, pages 79–90, 2019.

[136] James L Massey, Gurgen H Khachatrian, and Melsik K Kuregian. Nomination of SAFER+ as candidate algorithm for the Advanced Encryption Standard (AES). *NIST AES Proposal*, 1998.

[137] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. The TAMARIN prover for the symbolic analysis of security protocols. In *Computer Aided Verification (CAV) 2013*, pages 696–701. Springer, 2013.

[138] MiEcosystem. Mijia BLE Libraries. https://github.com/MiEcosystem/mijia_ble, 2019.

[139] Nateq Be-Nazir Ibn Minar and Mohammed Tarique. Bluetooth security threats and solutions: a survey. *International Journal of Distributed and Parallel Systems*, 3(1):127, 2012.

[140] Sandeep Mistry. Noble NodeJS BLE Central Module (Abandonware). https://www.npmjs.com/package/@abandonware/noble, 2022.

[141] David M'Raihi, Frank Hoornaert, David Naccache, Mihir Bellare, and Ohad Ranen. HOTP: An HMAC-Based One-Time Password Algorithm. RFC 4226, December 2005.

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

[143] 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.

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

[145] Andy Nguyen. BleedingTooth: Linux Bluetooth Zero-Click Remote Code Execution. https://google.github.io/security-research/pocs/linux/bleedingtooth/writeup, 2020.

[146] NSA. Ghidra Reverse Engineering Suite. https://ghidra-sre.org/, 2022.

[147] Terrence OConnor and Douglas Reeves. Bluetooth network-based misuse detection. In *2008 Annual Computer Security Applications Conference (ACSAC)*, pages 377–391. IEEE, 2008.

[148] OPENOCD. OPENOCD. https://openocd.org/, 2022.

[149] 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.

[150] John Padgette. Guide to Bluetooth security. *NIST Special Publication*, 800:121, 2017.

[151] Haram Park, Carlos Kayembe Nkuba, Seunghoon Woo, and Heejo Lee. L2Fuzz: Discovering Bluetooth L2CAP Vulnerabilities Using Stateful Fuzz Testing. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 343–354. IEEE, 2022.

[152] 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.

[153] Trevor Perrin. The Noise Protocol Framework specification (Revision 34). https://noiseprotocol.org/noise.html, 2018.

[154] Trevor Perrin and Moxie Marlinspike. The double ratchet algorithm. *GitHub wiki*, 2016.

[155] UACMe Project. UACME. https://github.com/hfiref0x/UACME, 2025.

[156] Proxmark. Proxmark RFID Tool. https://proxmark.com, 2024.

[157] Ole André Vadla Ravnås. Frida. https://frida.re/, 2023.

[158] Reuters. Xiaomi-backed Chinese firm acquires iconic scooter maker Segway. https://www.reuters.com/article/us-ninebot-xiaomi-investment-idUSKBN0N60GN20150415, 2015.

[159] Pavel Revak. PySWD. https://github.com/cortexm/pyswd, 2019.

[160] 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.

[161] Jan Ruge, Jiska Classen, Francesco Gringoli, and Matthias Hollick. Frankenstein: Advanced wireless fuzzing to exploit new Bluetooth escalation targets. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 19–36, 2020.

[162] 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.

[163] 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.

[164] 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.

[165] ScooterHacking. ScooterHacking Firmware Toolkit. https://mi.cfw.sh/, 2022.

[166] ScooterHacking. ScooterHacking Github. https://github.com/orgs/scooterhacking/repositories, 2022.

[167] ScooterHacking. ScooterHacking Ninebot EC Protocol. https://wiki.scooterhacking.org/doku.php?id=nbdocs, 2022.

[168] ScooterHacking. ScooterHacking Utility App. https://play.google.com/store/apps/details?id=sh.cfw.utility, 2022.

[169] ScooterHacking. ScooterHacking Website. https://scooterhacking.org/, 2022.

[170] 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.

[171] seemoo-lab. InternalBlue repository on Github. https://github.com/seemoo-lab/internalblue, 2021.

[172] Seemoo-lab. BTBB plugin for Wireshark 3.6. https://github.com/seemoo-lab/h4bcm_wireshark_dissector, 2022.

[173] Nordic Semiconductor. nrf51822 product specification. https://infocenter.nordicsemi.com/pdf/nRF51822_PS_v3.4.pdf, 2021.

[174] Nordic SemiConductor. nRF51822 System-on-Chip. https://www.nordicsemi.com/Products/nRF51822, 2022.

[175] Ben Seri and Gregory Vishnepolsky. The Attack Vector BlueBorne Exposes Almost Every Connected Device. https://armis.com/blueborne/, 2017.

[176] Ben Seri, Gregory Vishnepolsky, and Dor Zusman. BLEEDINGBIT: The hidden attack surface within BLE chips. https://armis.com/bleedingbit/, 2019.

[177] 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*, pages 453–464, 2019.

[178] Dariusz Seweryn. Rxandroidble library. https://github.com/dariuszseweryn/RxAndroidBle, 2022.

[179] Yaniv Shaked and Avishai Wool. Cracking the Bluetooth PIN. In *Proceedings of the conference on Mobile systems, applications, and services (MobiSys)*, pages 39–50. ACM, 2005.

[180] 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.

[181] Min Shi, Jing Chen, Kun He, Haoran Zhao, Meng Jia, and Ruiying Du. Formal analysis and Patching of BLE-SC pairing. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 37–52, 2023.

[182] 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.

[183] Pallavi Sivakumaran and Jorge Blasco. argXtract: Deriving IoT Security Configurations via Automated Static Analysis of Stripped ARM Cortex-M Binaries. In *Annual Computer Security Applications Conference*, pages 861–876, 2021.

[184] Da-Zhi Sun, Yi Mu, and Willy Susilo. Man-in-the-middle attacks on Secure Simple Pairing in Bluetooth standard V5.0 and its countermeasure. *Personal and Ubiquitous Computing*, 22(1):55–67, 2018.

[185] Jörn Tillmanns, Jiska Classen, Felix Rohrbach, and Matthias Hollick. Firmware insider: Bluetooth randomness is mostly random. *arXiv preprint arXiv:2006.16921*, 2020.

[186] 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.

[187] 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.

[188] Transparency Market Research. FIDO Authentication Market Forecast. https://www.transparencymarketresearch.com/fido-authentication-market.html, 2023.

[189] Michael Troncoso and Britta Hale. The Bluetooth CYBORG: Analysis of the full human-machine passkey entry AKE protocol. In *Network and Distributed System Security Symposium (NDSS)*, 2021.

[190] 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.

[191] 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.

[192] US NSA Research Directorate. ghidra: a software reverse engineering (sre) suite of tools. https://ghidra-sre.org/, 2022.

[193] Rijnard van Tonder and Herman Engelbrecht. Lowering the USB Fuzzing Barrier by Transparent Two-Way Emulation. In *8th USENIX Workshop on Offensive Technologies (WOOT 14)*, 2014.

[194] 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.

[195] 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.

[196] VirusTotal. Yara. https://virustotal.github.io/yara/, 2022.

[197] 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.

[198] 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. IEEE, 2021.

[199] W3C. Web Authentication: An API for accessing Public Key Credentials - Level 2. https://www.w3.org/TR/webauthn-2, 2021.

[200] 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*, page 286–298, New York, NY, USA, 2020. Association for Computing Machinery.

[201] Haohuang Wen, Zhiqiang Lin, and Yinqian Zhang. Firmxray: Detecting Bluetooth link layer vulnerabilities from bare-metal firmware. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 167–180, 2020.

[202] WhatsApp. WhatsApp Encryption Overview. https://www.bitsoffreedom.nl/wp-content/uploads/WhatsApp-Security-Whitepaper.pdf, 2016.

[203] Wikipedia Community. Apple silicon. https://en.wikipedia.org/wiki/Apple_silicon, 2022.

[204] Wireshark developers. Wireshark homepage. https://www.wireshark.org/, 2022.

[205] Ford-Long Wong and Frank Stajano. Location privacy in Bluetooth. In *Proceedings of the European Workshop on Security in Ad-hoc and Sensor Networks*, pages 176–188. Springer, 2005.

[206] Ford-Long Wong, Frank Stajano, and Jolyon Clulow. Repairing the Bluetooth pairing protocol. In *Proceedings of International Workshop on Security Protocols*, pages 31–45. Springer, 2005.

[207] Jianliang Wu, Yuhong Nan, Vireshwar Kumar, Mathias Payer, and Dongyan Xu. BlueShield: Detecting spoofing attacks in Bluetooth Low Energy networks. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, pages 397–411, 2020.

[208] Jianliang Wu, Yuhong Nan, Vireshwar Kumar, Dave Jing 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)*, 2020.

[209] Jianliang Wu, Ruoyu Wu, Dongyan Xu, Dave Jing Tian, and Antonio Bianchi. Formal model-driven discovery of Bluetooth protocol design vulnerabilities. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2285–2303. IEEE, 2022.

[210] Wei Wu, Pu Wang, Lei Zhao, and Wei Jiang. An Intelligent Security Detection and Response Scheme Based on SBOM for Securing IoT Terminal devices. In *2023 IEEE 11th International Conference on Information, Communication and Networks (ICICN)*, pages 391–398, 2023.

[211] Xiaomi. Xiaomi Bug Bounty Program On HackerOne. https://hackerone.com/xiaomi?type=team, 2022.

[212] 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.

[213] Yubico. Q3 Interim Report. https://investors.yubico.com/en/wp-content/uploads/sites/2/2023/03/Q3-investor-morning-presentation-231110.pdf, 2023.

[214] Yubico. Yubico FIDO2 Python Library. https://github.com/Yubico/python-fido2, 2023.

[215] Yubico. Yubikey Manager CLI. https://github.com/Yubico/yubikey-manager, 2023.

[216] Yubico. CVE-2024-35311. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2024-35311, 2024.

[217] Yubico. Security Advisory YSA-2024-02 FIDO Relying Party Enumeration. https://www.yubico.com/support/security-advisories/ysa-2024-02, 2024.

[218] z4yx. FIDO Wireshark protocol dissectors over USB HID. https://gist.github.com/z4yx/218116240e2759759b239d16fed787ca, 2019.

[219] Yue Zhang and Zhiqiang Lin. When Good Becomes Evil: Tracking Bluetooth Low Energy Devices via Allowlist-based Side Channel and Its Countermeasure. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 3181–3194, 2022.

[220] 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.

[221] 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-unsuspecting-riders/, 2019.