Disrupting *Continuity* of Apple's Wireless Ecosystem Security: New Tracking, DoS, and MitM Attacks on iOS and macOS Through Bluetooth Low Energy, AWDL, and Wi-Fi

Milan Stute Alexander Heinrich Jannik Lorenz Matthias Hollick

Secure Mobile Networking Lab, Technical University of Darmstadt, Germany

Abstract

Apple controls one of the largest mobile ecosystems, with 1.5 billion active devices worldwide, and offers twelve proprietary wireless *Continuity* services. Previous works have unveiled several security and privacy issues in the involved protocols. These works extensively studied AirDrop while the coverage of the remaining vast Continuity service space is still low. To facilitate the cumbersome reverse-engineering process, we describe the first guide on how to approach a structured analysis of the involved protocols using several vantage points available on macOS. Also, we develop a toolkit to automate parts of this otherwise manual process. Based on this guide, we analyze the full protocol stacks involved in three Continuity services, in particular, Handoff (HO), Universal Clipboard (UC), and Wi-Fi Password Sharing (PWS). We discover several vulnerabilities spanning from Bluetooth Low Energy (BLE) advertisements to Apple's proprietary authentication protocols. These flaws allow for device tracking via HO's mDNS responses, a denial-of-service (DoS) attack on HO and UC, a DoS attack on PWS that prevents Wi-Fi password entry, and a machine-in-the-middle (MitM) attack on PWS that connects a target to an attacker-controlled Wi-Fi network. Our PoC implementations demonstrate that the attacks can be mounted using affordable off-the-shelf hardware ($20 micro:bit and a Wi-Fi card). Finally, we suggest practical mitigations and share our findings with Apple, who have started to release fixes through iOS and macOS updates.

1 Introduction

With 1.5 billion active devices, Apple controls one of the largest mobile ecosystems worldwide [5]. Also, Apple is in the unique position of controlling both hard- and software and, therefore, can push new services to all of their platforms (iOS, iPadOS, macOS, tvOS, and watchOS) quickly. As a result, there are currently twelve different wireless services, such as AirDrop and Handoff, that Apple markets under the umbrella term *Continuity* [9]. While these services improve the user experience, wireless protocol designs and implementations offer a large surface for attacks. This has been demonstrated via numerous attacks against standardized protocols, e. g., Bluetooth [1], WEP [39], WPA2 [47], WPA3 [48], GSM [12], UMTS [35], and LTE [29]. Recently, several works have found severe vulnerabilities in Apple's proprietary wireless protocols [11, 18, 34, 44]. In particular, they have demonstrated the trackability of Apple devices that continuously transmit custom Bluetooth Low Energy (BLE) advertisements [18, 34], user identification and denial-of-service (DoS) attacks on Apple's proprietary Apple Wireless Direct Link (AWDL) protocol [44], and machine-in-the-middle (MitM) attacks on AirDrop [11, 44]. While these works have already discovered several vulnerabilities, they have only analyzed a fraction (one out of twelve services) of the potential attack surface. The most costly part of such analyses is the initial investment in reverse-engineering the complex software architecture [42] that implements the various proprietary protocols involved in offering Apple's services. However, the previous works lack an elaborate discussion on the actual process.

This paper provides the first structured guide to reverse engineer these proprietary protocols, which combines insights of previous works with our own experience. To make our guide more accessible and sustainable, we release a toolkit for semi-automated reverse-engineering of Apple's wireless ecosystem. Following this guide, we analyze three previously undocumented protocols used by the Handoff (HO), Universal Clipboard (UC), and Wi-Fi Password Sharing (PWS) services. Using the recovered specifications and our own open-sourced re-implementations, we discover four novel security and privacy vulnerabilities spanning from design errors to implementation issues, attesting—again—the inferiority of *security by obscurity*. The attacks enable new device tracking, DoS, and MitM attacks. We provide proof-of-concept (PoC) implementations for all attacks using only standard hardware such as a regular Wi-Fi card and a low-cost ($20) micro:bit [36] for BLE communication.

In particular, we make the following five contributions. *First*, we make security analysis of Apple's wireless ecosystem more affordable by presenting a structured and semi-automated reverse-engineering method. Our practical guide covers different vantage points and helps to navigate the complex system architecture of iOS and macOS. *Second*, we provide a complete specification of the protocols involved in the HO and UC services. We open-source a parser for Apple's proprietary OPACK serialization format and a sample implementation of the authentication protocol. *Third*, we provide a complete specification of the protocols involved in the PWS service. We accompany the specification with open-source implementations of both requestor and grantor roles. *Fourth*, we discover several security and privacy vulnerabilities and present four novel wireless network-based attacks. These are:

(1) A protocol-level DoS attack on HO and UC that exploits a low-entropy authentication tag in the BLE advertisements and a replay protection mechanism.
(2) A device tracking attack that exploits the asynchronous randomization interval of several AWDL device identifiers, such as MAC address and mDNS records.
(3) A MitM attack that exploits the one-sided authentication in PWS to automatically distribute and fill-in Wi-Fi passwords, which causes the victims to connect to an attacker-controlled Wi-Fi network.
(4) A DoS attack against the PWS protocol that exploits a parsing bug and allows for crashing the Settings app on iOS and, thus, could prevent a user from connecting to a new Wi-Fi network.

And *fifth*, we propose practical mitigations for all discovered vulnerabilities and a previously discovered [34] BLE device tracking attack. We have responsibly disclosed our findings to Apple, who have, so far, fixed two issues through iOS and macOS updates.

The rest of this paper is structured as follows. Section 2 discusses background and related work. Section 3 contains our reverse engineering guide. Section 4 presents the protocol specifications of three Apple services. Section 5 analyses security and privacy aspects of these protocols, presents our attacks, and proposes mitigations. Finally, Section 6 concludes this work.

2 Background and Related Work

In this section, we give an overview of Apple's current list of *Continuity* services, the link-layer protocols they rely on, and finally discuss previous security and privacy analyses in this ecosystem.

2.1 Apple's Continuity Services

Apple's current Continuity portfolio [9] consists of twelve different services that we list in Table 1. They are all used to transfer potentially sensitive user data such as clipboard content, phone calls, photos, and passwords. While Apple provides some high-level security descriptions for some of these services [4], the actual protocol designs and implementations remain closed-source. Previous works, so far, have analyzed one service in depth, i. e., AirDrop [11, 44]. Other works have also analyzed the BLE advertisements for several other services [18, 34]. However, the involved upper-layer protocols remain unknown. In this work, we demonstrate our reverse engineering methodology and use it to analyze the protocols involved in three services that have not been scrutinized before. We briefly describe the purpose of the three services.

Service	AWDL	BLE	Wi-Fi
Handoff (HO)	✓	✓	✓
Universal Clipboard (UC)	✓	✓	✓
Phone	✗	✗	✓
SMS	✗	✗	✓*
Instant Hotspot	✗	✓	✗
Auto Unlock	✓	✓	✗
Continuity Camera	✓	✓	✓
AirDrop	✓	✓	✗
Apple Pay	✗	✓	✗
Wi-Fi Password Sharing (PWS)	✗	✓	✗
Sidecar	✓	?	✗
Continuity Markup and Sketch	✓	?	✗

Table 1: Overview of Apple *Continuity* services and used link-layer protocols. Only one requires online iCloud access (✓*). All others communicate via local networks only.

Handoff (HO) HO allows users with multiple Apple devices to switch between devices while staying in the same application context. An example is Apple's Mail app: Users can start typing an email on their iPhone, switch to their Mac, and click an icon in the Mac's dock to continue writing the email. Third-party developers can add similar functionality to their apps via a public API [3].

Universal Clipboard (UC) UC shares clipboard content across nearby devices of one owner. For example, it allows for copying text on a Mac and pasting the content on an iPhone. Apple's UC and HO implementations use the same protocol as described in Section 4.1.

Wi-Fi Password Sharing (PWS) The PWS service allows a *requestor* device to request a password to a Wi-Fi network while it tries to connect to it. A *grantor* device that knows the password can decide whether it wants to share the password with the requestor. As a use-case, it allows us to share one's home Wi-Fi password with a house guest.

2.2 Wireless Link-Layer Protocols

We briefly introduce the two critical link-layer protocols involved in Apple's *Continuity* services, particularly AWDL and BLE. We have compiled the mapping of service to link-layer technologies in Table 1 by monitoring the interfaces (see Section 3) that become active when using each service.

Apple Wireless Direct Link (AWDL) AWDL is a proprietary Wi-Fi-based link-layer protocol that can co-exist with regular Wi-Fi operations. It offers a high-throughput direct connection between neighboring devices and has previously been reverse-engineered [41, 42]. Apple uses AWDL as a message transport for several Continuity services such as UC and HO.

Bluetooth Low Energy (BLE) BLE [15] operates in the same 2.4 GHz band as Wi-Fi. It is designed for small battery-powered devices such as smartwatches and fitness trackers and, thus, is not suitable for large data transfers. The BLE advertisement packets are a broadcast mechanism that can contain arbitrary data. Advertisements are used when devices set up a connection or share their current activity to nearby devices. Apple relies heavily on custom BLE advertisements to announce their Continuity services and bootstrap the various protocols over Wi-Fi or AWDL [18, 34, 44]. Generic Attribute Profile (GATT) is a BLE protocol that is used for discovering services and for communicating with a peer device. A UUID identifies a single service, and each service can contain several *characteristic* values. A client connects to a server device and accesses the characteristics of a service. The client can write data to, read data from, or receive notifications from the characteristics. Apple uses GATT as a message transport, e. g., to exchange Wi-Fi passwords via PWS as explained in Section 4.2.

2.3 Previous Security and Privacy Analyses of Apple's Wireless Ecosystem

Previous works have analyzed part of the Continuity services. Bai et al. [11] have looked at the risks of using insecure multicast DNS (mDNS) service advertisements and show that they can spoof an AirDrop receiver identity to get unauthorized access to personal files. Stute et al. [44] have reverse engineered the complete AWDL and AirDrop protocols and demonstrate several attacks, including user tracking via AWDL hostname announcements, a DoS attack via desynchronization on AWDL, and a MitM attack on AirDrop. Heinrich et al. [23] have discovered that AirDrop leaks contact identifiers and present a new privacy-preserving protocol for mutual authentication. Martin et al. [34] have extensively analyzed the content of the BLE advertisements transmitted for several Continuity services. They found several privacy-compromising issues, including device fingerprinting and long-term device and activity tracking. Celosia and Cunche [18] have extended this work and discovered new ways of tracking BLE devices such as Apple AirPods, as well as demonstrated how to recover a user's email addresses and phone numbers from the PWS BLE advertisements.

Unfortunately, these works provide no or only a limited discussion of the methods applied to receive their results, in particular, the process of reconstructing the frame format and protocol specifications. In Section 3, we provide a structured guide on how to approach this process. Also, the related work has only covered one Continuity service in full depth (i. e., AirDrop) and discussed the BLE advertisements for several others. In Section 4, we analyze the complete protocol stacks of three previously disregarded services.

3 A Hacker's Guide to Apple's Wireless Ecosystem

This section aims to provide a structured way to conduct reverse engineering[1] of Apple wireless protocols while using practical examples from our analysis of *Continuity* services. First, we show useful vantage points. We explain the binary analysis methodology and share our insights on dynamic analysis. Then, we explain how to access the security key material of Apple services and discuss our methodology's applicability to other protocols in Apple's ecosystem. In the end, we present several tools and scripts that we have developed to facilitate reverse engineering. All services that we analyzed in this paper are available on both macOS 10.15 and iOS 13. iOS and macOS share large parts of their code, and since we found macOS to be much more open and accessible than iOS, we used macOS as the platform that we analyzed. Most of the methods presented in this section can be applied to iOS as well. For some of them (e. g., full keychain access), the researcher requires a *jailbroken* iPhone. Since the discovery of a BootROM exploit called checkm8 and the introduction of checkra1n, jailbreaks became widely available and supported all iOS versions [20]. Finally, all vulnerabilities and attacks presented in Section 5 apply to both macOS and iOS. This section is a revised Ph.D. thesis chapter [40, Chapter 4].

3.1 Vantage Points

We approach protocol analysis from different *vantage points* that we depict in Fig. 1. (1) Static *binary* analysis is tough to conduct as each protocol is implemented across multiple components (frameworks and daemons). Therefore, during the initial stages, it is useful to monitor (2) the *system* as a whole to identify core components that can thoroughly be examined subsequently. Also, data transmitted via (3) *network*

[1] We define a *hacker* as a curious individual who wants to understand the technical details of a (potentially proprietary and closed-source) system to achieve interoperability or conduct a security analysis.

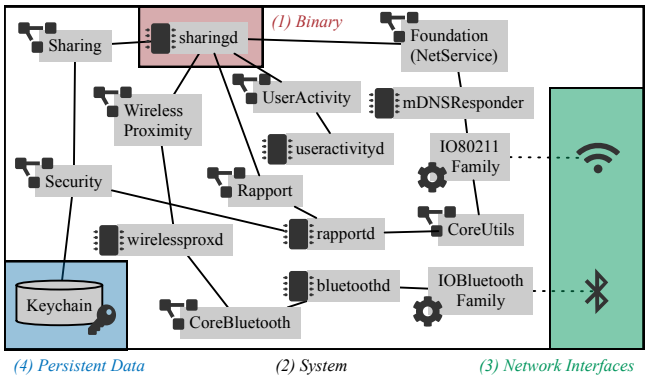


Figure 1: Vantage points that we used during our analysis. We provide a simplified view of components and their interactions, such as daemons (), frameworks (), and drivers () that are used by Handoff and Universal Clipboard.

interfaces is easily accessible using monitoring tools and is tremendously useful for dynamic analysis. We found that the ability to retrieve and use (4) *persistent data*, especially from the system's keychain, is essential for building prototypes and, thus, for validating findings. Finally, any available (5) *documentation* (not shown in Fig. 1) such as patents [45, 46] or Apple's platform security white paper [4] can be helpful for an initial assessment and understanding of some design elements of the service. Having those multiple vantage points at hand enables us to gather more information, to change perspective if we get stuck (e. g., when encountering encrypted traffic), and to resume analysis at a later point (e. g., after extracting the decryption keys). We elaborate on the four vantage points in Fig. 1 in the following.

3.2 Binary Analysis

We analyzed many binaries related to the *Continuity* services to find those parts that finally implement the protocol. We first illustrate our selection process and then discuss the two-part Wi-Fi driver, which implements most of the AWDL protocol stack. We focus our analysis on macOS and assume that the architecture is, in principle, similar to that of iOS as the two operating systems (OSs) share a large common codebase [8].

3.2.1 Binary Landscape

Understanding and navigating the binary landscape of macOS is essential to find and relate components of interest.

Frameworks and Daemons Apple excessively uses *frameworks* and *daemons* in its OSs. Consequently, numerous dependencies result in a complex binary selection process.

Frameworks offer an API to their corresponding singleton daemons and can be used by other daemons and processes. Daemons and their respective frameworks typically have a similar name (e. g., *sharingd* and *Sharing*) or share a derived prefix (e. g., *searchpartyd*, *SPFinder*, and *SPOwner*). We list the locations in the file system in the following. */System/Library/Frameworks* contains frameworks with public documentation[2] such as Security. */System/Library/PrivateFrameworks* contains other frameworks such as Sharing. */usr/libexec* and */usr/sbin* contain most daemons such as sharingd. However, some are also shipped in their respective framework. */usr/lib* and */usr/lib/system* contain low-level libraries such as CoreCrypto.

Drivers The Wi-Fi driver is a kernel extension and, therefore, resides in */System/Library/Extensions*. The driver is split up into a generic component (`IO80211Family`) and chip-specific plugins (such as `AirportBrcmNIC`).

3.2.2 Binary Selection

The purpose of the initial selection process is to identify binaries that may contain relevant code and, thus, sets the scope for the analysis project. To start this process, we can use the system's logging facility (see Section 3.3) to identify processes that become active when starting a particular system function (e. g., AirDrop). If we identify at least one daemon process, we can crawl through its dependencies recursively by running `otool -L` to find related frameworks and libraries. We show part of the discovered dependencies and interactions found for HO in Fig. 1.

3.2.3 Interesting Functions and Code Segments

Due to the size of most binaries that we analyzed, such as the `sharingd` daemon, it is infeasible to analyze the entire program. Instead, it makes sense to identify functions of interest, e. g., those that implement frame handling. Fortunately, Apple does not strip symbol names from (most of) their binaries, such that the symbol table provides useful information and, e. g., lists function names including `-[RPConnection _receivedObject:ctx:]` in the Rapport framework. This function handles received messages shared over AWDL after they have been decrypted. Furthermore, debug log statements give hints about the purpose of a code segment inside a function. Therefore, we can search for debugging strings (using `strings`) and their cross-references to find additional details.

3.3 System Logging

The complete protocol operation is difficult to comprehend with binary analysis alone. We complemented our static analysis with a dynamic approach. In this section, we discuss dedicated macOS logging and debugging facilities that helped during our analyses. In particular, we explain the *Console* application. However, previous work [42] has also used the

[2] https://developer.apple.com/documentation

`ioctl` interface, Broadcom's leaked `wl` utility, and Apple's undocumented *CoreCapture* framework to analyze the Wi-Fi driver. The *Console* aggregates all system and application logs since macOS 10.12 and includes debug messages from the kernel. Alternatively, one can use the `log` command-line tool to access the same information.

Filtering for Interesting Output It is possible to filter logging output, e. g., by process or subsystem. The *predicate-based filtering* is described in detail on the man page of `log`. For example, to get information about HO, we can use

```
log stream --predicate "process == \
  'rapportd' OR process == 'useractivityd'"
```

One of our tools, as described in Section 3.6, uses this ability to identify processes and frameworks that log information about a specific system service, like AirDrop.

Increasing Log Level The `--level debug` flag will increase the log verbosity of processes that make use of `os_log`. In addition, some processes log private data such as keys. To enable this, we can set

```
sudo log config --mode "private_data:on"
```

Since macOS 10.15, the command is no longer available, and we need to disable SIP [25].

3.4 Network Interfaces

Monitoring the Wi-Fi and Bluetooth network interfaces are a quick way to gather information about a particular service. For example, we can identify known protocols, whether encryption is used, or determine whether we are dealing with an undocumented protocol. Besides, we can learn the active wireless communication channels, the timings of packet transmissions, generally monitor the dynamics of a protocol. In the following, we discuss those tools that we have found to be particularly useful for this purpose.

3.4.1 Wireshark

Wireshark [49] is an open-source network protocol analyzer and supports many standardized but also proprietary protocols. While Wireshark identifies known protocols from network traces, it is also possible to implement custom dissectors. We found that writing such a custom dissector in parallel to the reverse engineering process serves multiple purposes: (1) We iteratively document and validate our findings. (2) It helps to deduce the semantics of individual fields, e. g., a random nonce would change in every handshake, while a static key or certificate would remain constant (Section 3.5). And (3) it can be used to evaluate experiments such as those in Section 5.4 by exporting time series data via `tshark`.

3.4.2 Bluetooth Explorer and Packet Logger

Apple ships two Bluetooth debugging tools in the *Additional Tools for Xcode* package.[3] The *Bluetooth Explorer* displays nearby BLE devices and their advertisements in real-time. Apple devices excessively use these advertisements to announce the availability of services such as AirDrop [34]. *BTLEmap* [24] implements a dissector for most of these advertisements. *PacketLogger*, on the other hand, creates network traces for Bluetooth HCI commands and, therefore, provides some of the functionality of *InternalBlue* [33]. Wireshark supports PacketLogger-recorded `.pklg` files, which allow for convenient analysis of Bluetooth traces.

3.4.3 Machine-in-the-Middle Proxy

Encrypted traffic can prohibit us from examining the interesting parts of the protocols. While we could instrument the daemon process and extract packets before transmission (which requires identifying functions that perform those operations), it can be easier to employ MitM proxy tools to open the end-to-end encryption, e. g., for HTTPS [19]. Unfortunately, a MitM proxy is not always successful in intercepting a connection with self-signed certificates, e. g., when certificate pinning is used, so it can be helpful to extract private keys and certificates from the system's keychain.

3.4.4 Custom Prototypes

In an advanced stage of the process, we have collected sufficient information to re-implement (part of) the protocol and, thus, can interact with the target devices actively. In particular, a custom prototype enables us (1) to validate our findings' correctness, e. g., if other devices start interacting with our prototype, we can conclude that the frame format is correct, (2) to find out more details about the protocol, e. g., we could determine which protocol fields mandatory or optional, and (3) to conduct protocol fuzzing as part of the security analysis, e. g., we found parsing-related vulnerabilities in PWS. We list the links to our prototypes the "Availability" section at the end of this paper.

3.5 Keychains

Access to private keys and other secure data used by a particular service or protocol is highly useful in making educated assumptions about what security mechanisms might be employed. Also, extracting key material is essential to build and test prototypes that prove or disprove working hypotheses, e. g., verifying the requirements for an authenticated PWS connection.

[3] https://developer.apple.com/download/more/?=additional%20tools%20xcode

3.5.1 macOS Keychains

In macOS 10.15, there are two types of keychains known as *login* and *iCloud* keychain, respectively. The former is only stored locally on the computer. The iCloud keychain was first introduced in iOS and has since been ported to macOS as well. This keychain provides more features such as protection classes, optional synchronization between devices, and improved access control [4]. As Apple has moved more keychain items from the *login* keychain to the *iCloud* keychain, we believe that Apple will merge them in the future. The *Keychain Access* application is a GUI for displaying and working with either keychain. However, we have found that not all keychain items (e. g., those used by some system services) are displayed.

3.5.2 Security Framework

Fortunately, Apple provides a documented API for accessing keychains via the *Security* framework, which additionally is open-source.[4] For our purposes, the `SecItemCopyMatching` function[5] is particularly interesting as it allows retrieving items such as keys from the keychain. The function requires some query parameters to narrow down the items it should return. To get the relevant query parameters of a target program, we can either statically analyze the binary by searching for references to `SecItemCopyMatching` or monitor the process and extract the parameters at runtime using a debugger. In the case of PWS, the query consists of three keys: `kSecClass`, `kSecReturnRef`, and `kSecValuePersistentRef`. The value of the latter is a serialized object containing all information required to locate a particular item in the keychain.

3.5.3 Accessing Keys of Apple Services

As a security measure, programs not signed by Apple will not get any results even when using the correct query parameters as Apple uses code signing to implement access control to keychain items. To circumvent this measure, we (1) need to set the correct `keychain-access-group` entitlement (`com.apple.rapport` in case of HO or simply the `*` wildcard) during code signing and (2) disable Apple Mobile File Integrity (AMFI), which prevents program with restricted entitlements from starting by setting the following as a boot argument:[6] `amfi_get_out_of_my_way=1`. An automated solution to this is introduced in Section 3.6.

[4] https://opensource.apple.com/source/Security/

[5] https://developer.apple.com/documentation/security/1398306-secitemcopymatching

[6] https://www.theiphonewiki.com/wiki/AppleMobileFileIntegrity

3.6 Automated Reverse Engineering Toolkit

Automated reverse engineering for generic protocols is a hard problem. However, we have identified several possibilities for automating parts of the process on Apple's platforms to make our work more sustainable. We release a toolkit that covers all vantage points mentioned in this section with the publication of the paper (see the "Availability" section at the end of this paper). In particular, the toolkit allows to (1) discover interesting daemons/frameworks and functions based on a keyword, (2) extract the plaintext messages used by `rapportd` that are exchanged by Continuity services, and (3) print any secrets stored in the system keychain that are used by a particular daemon. We elaborate on the individual tools in the following.

3.6.1 Identifying Interesting Binaries

Our toolkit contains a Python script that scans system log messages (Section 3.3) for specified keywords and lists the emitting daemons, frameworks, and subsystems. The tool can then search those binaries and their dependencies (frameworks and libraries) recursively for the same or additional strings and symbols. Finally, the user receives an initial candidate list of binaries and functions to analyze further.

3.6.2 Extracting Plaintext Continuity Messages

Our analysis has shown that many Continuity services use a secure transport service offered by `rapportd`. In analogy to an HTTP MitM proxy, our toolkit allows us to extract exchanged plaintext messages before they are encrypted (outgoing) and after they are decrypted (incoming). Internally, the tool attaches the `lldb` debugger to `rapportd` and uses breakpoints at the respective send and receive functions to print all exchanged messages.

3.6.3 Printing Keychain Items

Continuity services use different security mechanisms to protect their communication, such as TLS in AirDrop or the custom encryption described in Section 4.1.4, which all require one or more secret inputs, such as private keys, certificates, or tokens. Our toolkit provides a way to automatically identify and extract these inputs to facilitate building custom prototypes and, thus, automating the method described in Section 3.5.3. The tool is based on the FRIDA framework [38] to inject code into the *Security* framework to log secrets any time a specific process accesses the keychain.

4 Continuity Protocols

In this section, we present the protocols involved in offering three Continuity services, i. e., Handoff (HO) and Universal Clipboard (UC) in Section 4.1, and Wi-Fi Password Sharing (PWS) in Section 4.2. In particular, we present the operational details of the protocols that we gathered using the methodology in Section 3.

4.1 Handoff and Universal Clipboard

We analyze the protocols involved in the HO and UC services. HO allows a user to continue their current activity in an application on another of their Apple devices. UC allows a user to copy clipboard content (e. g., text) on one device and (seamlessly) paste it on another. For HO or UC, all involved devices have to be logged into the same iCloud account and have Bluetooth and Wi-Fi turned on. We have found that HO's and UC's protocols are identical. In the following, we present the service requirements and the protocols involved in the different phases: (1) the *discovery* phase using BLE advertisements (Section 4.1.2) and mDNS-over-AWDL (Section 4.1.3), (2) the *authentication* phase for deriving a session key (Section 4.1.4), and (3) the *payload transfer* phase that transports the application data (Section 4.1.5). We provide an overview of the entire protocol stack in Fig. 4. In this paper, we discuss the core components of the protocols. The full specification is included in [22].

4.1.1 Requirements

Apple designed HO and UC to work between devices of the same user, i. e., devices that are signed in to the same Apple account. We have found that the iCloud keychain synchronizes the long-term device-specific public keys P^L that can be found under the name *RPIdentity-SameAccountDevice*. These keys are used for an authenticated session key exchange, as shown in Section 4.1.4.

4.1.2 Discovery with BLE

Both HO and UC announce user activities, such as a clipboard copy event, on the host system via BLE advertisements. Receiving devices use the embedded information to, for example, display the icon of the active HO-enabled app in the system dock, as shown in Fig. 2. A click on the icon (HO) or a paste event (UC) triggers the rest of the protocol stack.

Figure 2: iPad dock showing a Handoff icon on the right.

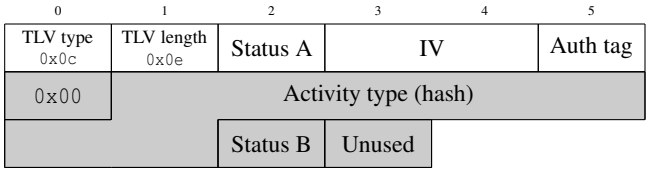


Figure 3: Handoff and Universal Clipboard BLE advertisement payload. Encrypted content is shown in grey.

LSB	Meaning	In A	In B
1	Activity has URL option key		✓
2	Activity contains file provider URL key		✓
3	Activity contains Cloud Docs key		✓
4	Clipboard data available	✓	✓
5	Clipboard version bit		✓
6	Activity auto pull on receivers key		✓

Table 2: Definition of individual status flag bits and whether they are included in status byte A or B (Fig. 3).

The BLE advertisement uses Apple's custom frame structure that has already been described [34] and makes use of manufacturer data to add custom fields. The fields are encoded as TLV8 structures[7] such that a single frame can include multiple fields. Apple uses different field types for its Continuity services. Figure 3 shows the payload of an HO and UC advertisement with type `0x0c`. It contains a plaintext status flag, an IV, an authentication tag, followed by an encrypted payload (shown in grey). Apple uses AES-GCM for encryption and authentication with dedicated BLE encryption key K^{BLE}. For every new advertisement, i. e., new HO or UC activity, the initialization vector (IV) is incremented by one. Upon depleting its IV space (2^{16}), a device triggers a re-keying protocol via the companion link service (Section 4.1.4) to re-new K^{BLE}. The re-keying protocol uses the long-term key P^L for authentication.

The encrypted payload primarily contains an activity type and other status flags. The activity type indicates the application or activity that was triggered and is encoded as a truncated SHA-512 hash of an application-specific string, such as `com.apple.notes.activity.edit-note` for Apple's Note app. Unsupported application activities are ignored. The status B flags are similar to the cleartext status A. Martin et al. [34] discovered that status A is set to `0x08` after the user has copied data on their device. Apparently, Apple has deprecated status A in favor of status B. We found that status B can encode more information, as shown in Table 2. We assume that status A was part of an earlier protocol version, and Apple has kept it for backward compatibility but started

[7] *TLV8* is a type-length-value (TLV) structure where the length field has a length of 8 bits (1 byte).

to encrypt new fields that include more sensitive information (activity type).

To facilitate dynamic analysis of the advertisements, we implemented a macOS application that decrypts and parses all advertisements sent by devices linked to the user's iCloud account (see "Availability" section).

4.1.3 Discovery with mDNS-over-AWDL

The device that broadcasts BLE advertisements can be depicted as a server that can respond to requests from a client device. Upon engaging in an activity, the *client* device that received the *server*'s BLE advertisement enables its AWDL to start service discovery via mDNS and DNS service discovery (DNS-SD), also known as *Bonjour*.

The queried service type is called `_companion-link._tcp.local`. The DNS responses from the server device include an instance name in the pointer (PTR) record, its hostname in the service (SRV) record, IPv6 address (AAAA), and a text (TXT) record. It is noteworthy that Apple implements hostname randomization (similar to medium access control (MAC) address randomization) for the SRV records transmitted via AWDL.

The TXT record is typically used to transfer additional information about the service. The HO TXT record contains the information shown in the following example:

```
rpBA=2E:6D:C1:B7:08:1F,
rpFl=0x800,
rpAD=88d428438a3b,
rpVr=192.1
```

We found that the values `rpBA` and `rpAD` are used to identify if both devices are linked to the same iCloud account and filter out potentially other devices that might respond via the open AWDL interface. In particular, we found that `rpBA` (encoded as a MAC address string) is chosen at random and changes at least every 17 minutes. `rpAD` is an authentication tag generated from the random `rpBA` and the device's Bluetooth Identity Resolving Key (IRK) (used to resolve random BLE addresses [15]) as arguments for a `SipHash` function [10]. Since the IRKs are synced via the iCloud keychain, devices logged into the same iCloud account can try all available IRK in the keychain to find other devices.

4.1.4 Authentication via Pair–Verify

The companion link service, used for HO and UC, implements an authenticated Elliptic-curve Diffie–Hellman (ECDH) key exchange using the long-term keys P^L for mutual authentication. The new session key is used to encrypt follow-up messages. The so-called *Pair–Verify* protocol is based on Apple's Homekit Accessory Protocol (HAP) protocol [6].

The handshake is depicted in Fig. 4. It mainly performs ECDH [28] to exchange a session key K with the ephemeral key pairs (P_s, S_s) and (P_c, S_c). The public keys P_s and P_c are

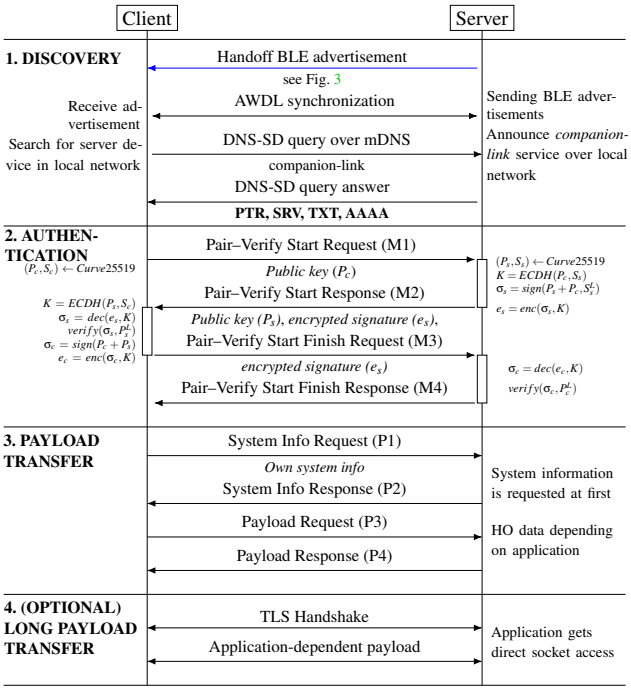


Figure 4: Handoff and Universal Clipboard protocol overview.

authenticated using Ed25519 [14] signatures that use the long-term server (s) and client (c) key pairs (P_s^L, S_s^L) and (P_c^L, S_c^L) for generation and verification. The verification keys P_s^L and P_c^L are synchronized using the iCloud keychain. Then, both devices derive the server and client keys K_s and K_c from the new session key K by using HKDF [27]. The keys are used to protect the follow-up payload transfer with the ChaCha20-Poly1305 cipher [37]. In Section 4.2, we elaborate on the protocol, including an extension that allows authentication between devices that do not have a pre-shared key P^L.

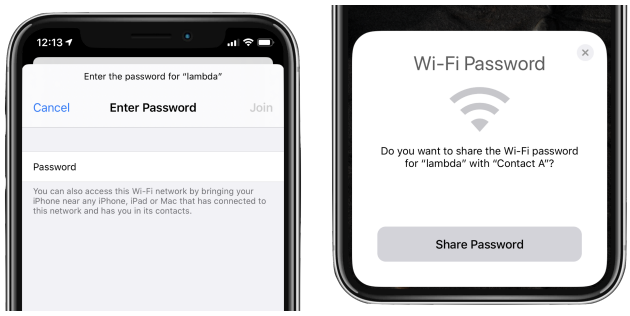
The message format consists of a TLV24[8] encoding that, in turn, contains an *OPACK* dictionary with a single value under the key `_pd`. The value contains TLV8 structures that encode the individual fields used for the key exchange. OPACK is a proprietary undocumented serialization format, and we publish its specification together with a sample implementation in Python (see "Availability" section).

4.1.5 Payload Transfer

To transfer the actual application payload, i. e., clipboard content (UC) or user activity (HO), the companion link service implements another four-way communication protocol that is protected by ChaCha20-Poly1305 [37] using the K_s and K_c keys from the authentication protocol.

The protocol first exchanges the devices' system information (P1 and P2 in Fig. 4) that includes the device model, e. g., `MacBook11,5`, the device name, and several flags. After-

[8] TLV structure with a 24-bit (3-byte) length field.



(a) Requestor. (b) Grantor.

Figure 5: The password view on a requestor and the password sharing dialog on a grantor.

ward, the client requests and receives the application-specific payload (P3 and P4).

The HO developer API offers the ability to transfer additional data by setting up a direct socket connection from the server application to the client application.[9] If specified by the developer, `sharingd` opens a TLS connection (*Long Payload Transfer* in Fig. 4) and passes the open socket to the requesting application. The TLS connection authenticates both sides by using the same Apple ID certificates and validation records used for AirDrop [44] and PWS (Section 4.2). We have found that the same protocol is also used by UC to transfer clipboard contents that are larger than 10 240 bytes. In that case, UC uses the P3 and P4 messages to bootstrap the TLS connection.

4.2 Wi-Fi Password Sharing

Apple also uses BLE to implement a service called PWS, which enables users to share known Wi-Fi password with guests and friends. This service aims to solve the usual hassle of manually entering the password, which can sometimes be challenging if the password is complex or not at hand.

In the following, we call the device that searches for a Wi-Fi password *requestor* and the device that shares the password *grantor*.

PWS is initiated automatically when the password view (in Fig. 5a) is open after selecting an SSID to connect to. No further user interaction is necessary from the user of the requestor. Surrounding devices are notified about the PWS as long as the password view is open. If a grantor is in range, the password sharing dialog (in Fig. 5b) pops up, asking the user to share the password. If the grantor accepts, it sends the encrypted password to the grantor. Potentiality already entered characters in the password text field are overwritten, the shared password is inserted, and the device automatically tries to connect to the Wi-Fi network.

[9] https://developer.apple.com/documentation/foundation/nsuseractivity/1409195-supportscontinuationstreams

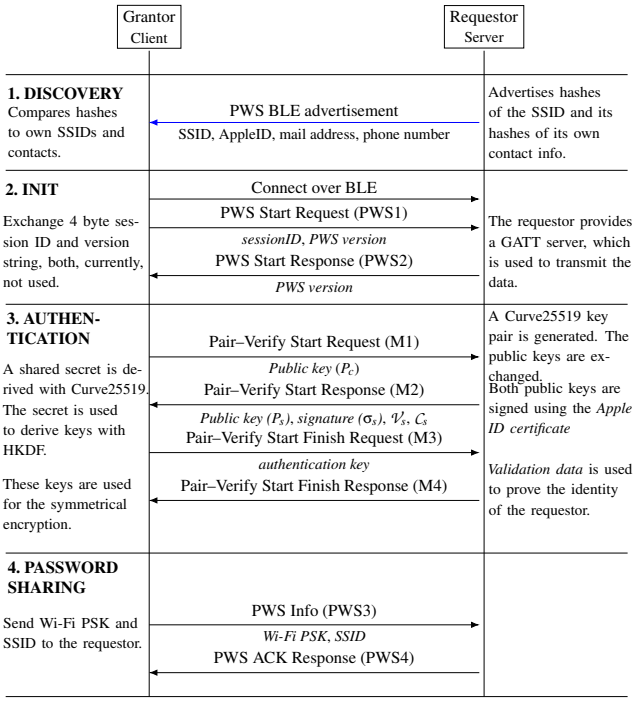


Figure 6: PWS protocol overview.

The PWS protocol consists of four phases that we depict in Fig. 6: (1) the *discovery* phase that uses BLE advertisement to bootstrap the protocol (Section 4.2.3), (2) the *initialization* phase transmits protocol metadata (Section 4.2.4), (3) the *authentication* phase where the requestor proves its identity to the grantor and one symmetrical key is derived (Section 4.2.5), and, finally, (4) the *sharing* phase that transfers the pre-shared key (PSK) for the requested Wi-Fi network (Section 4.2.4). In the following, we first describe the protocol requirements and discuss the basic BLE data transport. We then discuss the four main protocol phases in detail.

4.2.1 Requirements

We believe that Apple aimed to solve the problem of Wi-Fi password sharing with minimal user interaction. Their design has the following requirements [7]: (1) The grantor needs to have the contact information (phone number or email address) of the requestor stored in its address book. (2) The grantor needs to be unlocked. (3) The requestor needs to be signed in with an Apple ID. (4) Both devices need to have Bluetooth enabled.

4.2.2 BLE Data Transport and Frame Format

All messages sent and received are transmitted over BLE using the *value* property of a GATT characteristic. The requestor acts as a GATT server to which the grantor connects to. The grantor sends messages to the requestor by writing to

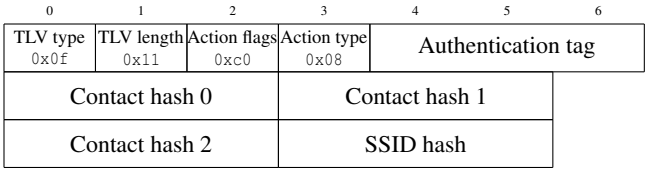


Figure 7: PWS advertisement frame format.

this GATT characteristic. The characteristic also supports the notify flag, which is used by the requestor to respond. Even though the maximum payload length of the GATT characteristic is set to 512 bytes, the payload is split into packets of 101 bytes at the most. To be able to reassemble the complete payload on the other end, the length of the payload is included in the first 2 bytes of the first packet.

The GATT characteristic supports multiple services. To support this, every payload is wrapped is wrapped in a *SFSession*[10] frame. This frame consists of *service type* and a *frame type*, followed by the actual payload. The service type is constant for a specific service. For example, PWS uses the service type `0x07`. The frame type is used to differentiate between different frames of the same service.

4.2.3 Discovery with BLE Advertisements

The requestor sends out BLE advertisements to inform surrounding devices. The frame format follows the same base structure as for HO/UC in Section 4.1.2 but uses a separate type. Figure 7 shows the frame format for the PWS advertisement with TLV8 type `0x0f`. The payload includes the first 3 bytes of the SHA-256 hash of the owner's Apple ID, email address, phone number, and the SSID for which the requestor requests a password.

Surrounding devices check whether any of their contacts match one of the hashed contact identifiers and whether they have a password for the provided SSID hash. If both checks succeed, the grantor prompts its user with the password sharing dialog (Fig. 5b).

4.2.4 Initialization and Wi-Fi Password Sharing

In the initialization phase, two messages are exchanged; both are OPACK encoded dictionaries. The grantor sends the first packet (PWS1) that contains an unused random 4-byte session ID and a protocol version. The requestor responds (PWS2) with its protocol version. After receiving the PWS2 message, the grantor starts the authentication phase as described in Section 4.2.5. Once the handshake is complete, both devices have computed the same shared secret, from which, in the final phase, two keys are derived using HKDF [27], one for each direction. These keys are then used to encrypt both messages with ChaCha20-Poly1305 [37]. The encrypted content is in both messages an OPACK encoded dictionary.

[10] We found the name during the binary analysis.

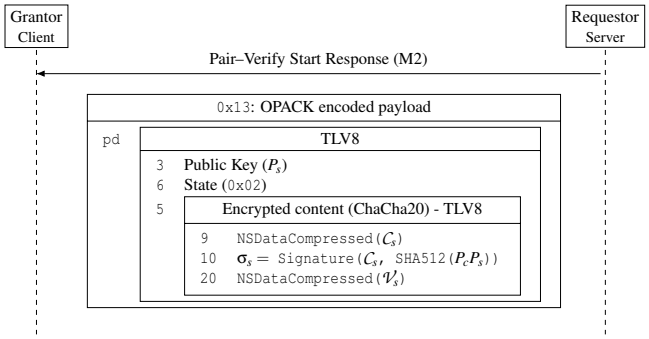


Figure 8: Start Response (M2) in Pair–Verify authentication showing the multi-level encapsulation.

The first message (PWS3) is sent by the grantor and contains the Wi-Fi PSK, the SSID, and the hashed contact identifiers of the grantor. The requestor responds (PWS4) to inform the grantor that the sharing was successful.

Note that it is unclear to us why the grantor sends its contact identifiers as the requestor never uses them. We discuss this issue in Section 5.

4.2.5 Authentication via Extended Pair–Verify

To authenticate and encrypt the actual Wi-Fi password, a Pair–Verify handshake is performed, which derives a shared secret and proves the identity of the requestor to the grantor. A similar version of the Pair–Verify protocol is used in Apple's HAP [6]. However, we have found that Apple uses a custom variation that enables authentication via a user's Apple ID. The Pair–Verify protocol consists of 4 messages, shown in Fig. 6. All messages are encoded using OPACK and contain a dictionary with one key-value pair, the key `pd`, and a TLV8 structure as the value. This TLV8 contains the values we now describe for each message.

First, the grantor generates an ephemeral Curve25519 key pair for the new session and sends a start request (M1) containing the public key P_c. Upon reception, the requestor generates another key pair. The start response (M2) contains the requestor's generated public key P_s, an Apple ID certificate $\mathcal{C}_s$, an Apple ID validation record $\mathcal{V}_s$, and a signature σ_s, as shown in Fig. 8. All fields except the public key are encrypted using ChaCha20 [37] with a key derived from the shared secret and HKDF [27]. The encrypted fields are packed in another TLV8. Both, Apple ID certificate and validation record, are signed by Apple and are also used in the AirDrop protocol [44]. The validation record is tied to the Apple ID certificate with a universally unique identifier (UUID). In particular, the UUID is included in the validation record and the common name of the certificate. The validation record also contains Apple-validated contact identifiers and is used by the grantor to validate the identity of the requestor. The Apple ID certificate is used to sign both public keys, i. e.,

Vulnerability and attack	Sec.	Impact and severity	Mitigation
DoS via IV desynchronization	5.2	+ User is unable to use the HO/UC services	—
Tracking via linear IV	5.3	++ Attacker can track devices over a long period, even across the MAC address randomization interval	—
Tracking via async. randomization	5.4	++ *same as above*	iOS 13.4, macOS 10.15.4 (no CVE)
MitM via Wi-Fi password auto-fill	5.5	+++ Attacker (1) has full control over client network traffic allowing for, e. g., DNS spoofing, and (2) can compromise the device by exploiting vulnerabilities in the Safari web browser	—
DoS via settings app crash	5.6	++ User is unable to connect to a new password-protected Wi-Fi network	iOS 13.5, macOS 10.15.5 (CVE-2020-9827)

Table 3: Overview of discovered vulnerabilities and their real-world impact for iOS and macOS. We rate the severity from low (+) to high (+++). Under mitigation, '—' means that Apple has not yet confirmed or provided a fix for the vulnerability. We provide details on the responsible disclosure process at the end of this paper.

$\sigma_s = \text{sign}(P_c + P_s, k_s)$, which proves to the grantor that the device sending this data, in fact, owns the private key k_s certified by $\mathcal{C}_s$. This signature is also included in the encrypted TLV8. In the finish request (M3), the grantor encrypts an empty string and sends the cipher, which includes a 16-byte Poly1305 authentication tag, to the requestor. Finally, the finish response (M4) contains a fixed state byte (`0x4`) and completes the handshake.

5 Security and Privacy Analysis

Based on our results from reverse-engineering several Continuity protocols, we conduct a comprehensive security and privacy analysis of the iOS and macOS platforms. In particular, we discover a protocol-level DoS attack on HO and UC (Section 5.2), a device tracking attack that exploits the asynchronous randomization interval of several device identifiers (Section 5.4), a MitM attack on PWS that causes a victim to connect to an attacker-controlled Wi-Fi network (Section 5.5), and a DoS attack against PWS that prevents a user from connecting to a new Wi-Fi network (Section 5.6). We a provide a mitigation to a previously [34] discovered device tracking vulnerability (Section 5.3). We provide an overview of the vulnerabilities in Table 3. In the following, we first describe the common attacker model and then discuss in detail the individual vulnerabilities, the attack implementations, and propose practical mitigations for the identified issues.

5.1 Attacker Model

For the following attacks, we consider adversaries that:

- have access to a Bluetooth Low Energy radio and, for the attack presented in Section 5.5, a Wi-Fi radio that can act as an access point,
- are in physical proximity (more precisely, within wireless communication range) of the target device, and
- are otherwise in a non-privileged position, in particular, they (1) do not require any contact information about their target, (2) do not require an existing Bluetooth pairing with the target, and (3) do not require access to the same Wi-Fi network.

5.2 DoS via IV Desynchronization

We exploit the short AES-GCM authentication tag in the HO and UC BLE advertisements to force an IV desynchronization between client and server such that HO and UC become unusable. Apple's deployed replay protection mechanism is unable to defend against this attack and requires the user to reboot their devices.

5.2.1 The Vulnerabilities: Low-Entropy Authentication Tag and IV-based Replay Protection

The HO BLE advertisements are encrypted using AES-GCM with a one-byte authentication tag and a two-byte IV (see Section 4.1). The IV used in the advertisements is a linearly increasing counter to avoid IV reuse with the same key [21]. Whenever a successfully authenticated advertisement is received, the receiver will update the last valid IV with the current one. From there on, any authenticated advertisement that has an IV lower or equal to the current one is discarded.

In addition to the replay protection, we observed, that HO triggers a re-keying protocol whenever the authentication fails. In that case, HO assumes that the sending device has updated its HO key K^{BLE} and queries the sending device for its current key and IV. This re-keying protocol runs over AWDL and uses the same procedure as HO and UC to protect the communication. However, we observed that if the returned key–IV pair match the currently stored pair, no new keys will be exchanged.

5.2.2 The Attack: Trigger Continuous Rekeying

In the following, we denote C as the client device that stores a key–IV pair for a linked server device S. The goal of the attack is to change the IV counter of the key–IV pair at C so that the IV-based replay protection mechanism will drop future valid advertisements of S and, thus, C is no longer able to receive new UC clipboard data or HO activities from S. To achieve this goal, the attacker

(1) generates a valid HO advertisement as shown in Fig. 3,
(2) spoofs S's BLE MAC address by setting it as the source address of the advertisement,
(3) sets the IV in the payload to the maximum value, and
(4) sends out 256 copies of the advertisement to brute-force all authentication tag values.

The attack works because Apple devices use the shared key and the IV in the BLE advertisement to verify the authentication tag. In our attack, we send 255 advertisements with an invalid tag that are all discarded and trigger a re-keying event that has no effect (see Section 5.2.1). One advertisement will, however, have a seemingly valid authentication tag. If the included IV is greater than the currently stored one, C updates the IV and then processes the decrypted payload. At this point, the adversaries have already achieved their goal, and it does not matter that they are unable to forge a valid payload. Since the IV at C has been updated, C will discard any subsequent advertisements from S as all subsequent advertisements contain an IV less or equal to `0xffff`.

To mount the attack on all device pairings in proximity, we repeat this attack with all BLE MAC addresses that we observe. Since we only need to send a BLE advertisement, a $20 micro:bit [36] is sufficient to mount the attack. We used the BLESSED open-source BLE stack [16] to build our PoC.

5.2.3 The Mitigation: Longer Authentication Tag

As a mitigation to the attack, we suggest increasing the length of the authentication tag. While National Institute of Standards and Technology (NIST) recommends using 128 bits [21], the manufacturer data in the BLE advertisements can only carry 24 bytes [34]. As the current HO advertisement already uses 16 bytes (see Fig. 3), Apple could add a new 64-bit authentication tag and keep the current one for backward compatibility. Increasing the search space to 2^{64} would effectively prevent our network-based brute-force attack. Note that limiting "the number of unsuccessful verification attempts for each key" [21] is not a suitable mitigation as it would open up a new DoS attack where the attacker could push the limit and prevent legitimate verification attempts.

5.3 Device Tracking via Linear IV

Martin et al. [34] have discovered that the linearly increasing IV in the HO advertisements can be used for long-term device tracking even though Apple employs MAC address randomization in BLE. The problem is that while the BLE address changes, the IV remains stable. In the following, we propose a practical mitigation that replaces the linear counter with an unguessable pseudorandom sequence.

5.3.1 The Mitigation: Changing the IV sequence

To prevent tracking via the linear IV, we propose to use a shuffled IV sequence with the following properties:

(1) The sequence has a length of 2^{16} and contains all integer values from 0 to $2^{16}-1$ exactly once.
(2) A sender can select the next value in the sequence in constant time.
(3) A receiver can tell if value x is positioned before or after y in the sequence in constant time.
(4) The sender and receiver only need to share a secret.
(5) Given any value in the sequence, an adversary is not able to guess the next or previous item of the sequence.

Figure 9 shows our candidate algorithm for generating a randomized sequence on the Knuth shuffle [26]. It uses a pseudorandom number generator (PRNG) with a seed derived from the shared BLE encryption key K^{BLE} and generates a counter-to-IV mapping. Internally, each HO device now keeps an internal incrementing counter c and uses $\text{fMap}(c)$ as the IV for the next advertisement. Note that c should also be increased on the sending device whenever the MAC changes to synchronize identifier randomization (see Section 5.4). The algorithm also generates the reverse IV-to-counter mapping to identify in constant time whether a received IV x comes before or after the current counter c, which can be done by comparing c with $\text{rMap}(x)$.

While the mitigation is practical from an overhead perspective (constant-time lookup), it is not backward-compatible as it would break the replay-protection mechanism currently employed in Apple's devices (see Section 5.2). Also, note that as the sequence is based on the HO key, the algorithm needs to re-run every time a re-keying event occurs.

```
() function genIVSequence(K^BLE) {
  fMap = [0..2^16-1] /* forward mapping */
  rMap = []          /* reverse mapping */
  seed = HKDF(K^BLE, "IV-sequence")
  prng = PseudoRandomNumberGenerator(seed)
  for (i = len(fMap) - 1; i > 0; i--) {
      j = prng.next(i)
      fMap.swap(i,j)
      rMap[fMap[i]] = i
  }
  return (fMap, rMap)
}
```

Figure 9: Generating a pseudo-random IV sequence.

5.4 Device Tracking via Asynchronous Identifier Randomization

When using a Continuity service such as HO or UC, AWDL emits several device identifiers such as MAC address and hostname in the clear. While Apple has implemented randomization schemes for these identifiers, we found that the intervals are sometimes not in sync and allow for continuous device tracking. AWDL uses Wi-Fi and does, by itself, not offer authentication or encryption. Instead, Apple defers protection to the upper-layer protocol. Therefore, an attacker can monitor all packets sent over the air.

5.4.1 The Vulnerability: Asynchronous Identifier Randomization

Apple has implemented MAC address randomization for AWDL. In 2019, Apple also introduced hostname randomization [44] in the Bonjour service announcements that are sent via AWDL. In this paper, we discovered that Apple introduced the new device identifier `rpBA` in the TXT record of the DNS service announcements (see Section 4.1.3). Apple devices regenerate (or randomize) each identifier after some time; however, this does not happen synchronously.

5.4.2 The Attack: Merging Identifiers

Consequently, identifiers may overlap and, thus, trivially enable device tracking for longer than the randomization interval. To practically mount such an attack, the attacker only needs to be within Wi-Fi communication range of their target(s). In particular, the attacker needs a Wi-Fi card and tune it to channel 44 or 149 (depending on the country [42]) and monitor AWDL frames. Using a simple matching algorithm that stores current identifiers and updates them upon receiving new frames, the attacker can continuously track their targets.

We conduct an experiment in an office environment to demonstrate the problem and the attack and show the exemplary result of tracking an iOS 13 device in Fig. 10. The figure depicts the times when the device emits AWDL frames (top bar). The following bars show when a particular randomized identifier was recorded for the first and last time and, thus, clearly indicate the times at which the overlap occurs. For example, in Fig. 10, the `rpBA` overlaps with the other identifiers for $35 \leq t \leq 38$ min. We note that the intervals for the IPv6 and MAC addresses are perfectly in sync because the link-local IPv6 address is derived from the current MAC address [42]. It is also noteworthy that the randomization intervals of the individual identifiers differ strongly and range from less than one minute (hostname) to more than 35 minutes (`rpBA`).

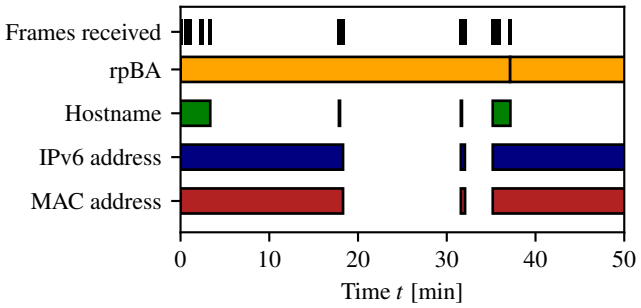


Figure 10: Tracking an iOS device using its randomized identifiers. We show the frame reception time in the top bar. The other bars indicate when the randomized identifiers were recorded for the first and last time. Each bar segment represents a new random identifier.

5.4.3 The Mitigation: Synchronous Randomization

To understand why the overlap with `rpBA` and the long intervals occur, we analyze the `-[CUSystemMonitorImp _rotatingIdentifierMonitorStart]` function in the `CoreUtils` framework. We found that the function sets a timer to 17 minutes to randomize the `rpBA` value but uses a low-level API[11] that allows the system to defer the call to conserve energy. This timer value is neither synchronized with others nor does it update in regular intervals, which results in the analyzed overlaps.

To mitigate this issue, we suggest that the randomization intervals of the identifiers should be synchronized or—at least—not overlap (e. g., hostname and MAC address). In addition, we suggest that the randomization interval for any identifier should not be longer than 15 minutes. We propose to introduce a system-wide randomization API to prevent regression and accommodate future identifiers.

5.5 MitM via Wi-Fi Password Auto-Fill

We exploit the one-sided authentication in the PWS protocol to automatically fill the Wi-Fi password field for requestors, causing the iOS or macOS target to connect to an attacker-controlled Wi-Fi network and raise the attacker to a privileged MitM position. This position allows for mounting secondary attacks such as DNS spoofing or traffic analysis. In addition, the attacker can compromise the target device by triggering Safari exploits.

5.5.1 The Vulnerability: One-Sided Authentication

The MitM attack exploits the *asymmetry* of information that the parties in PWS need to provide: the requestor must provide certified contact information, while the grantor does not,

[11] https://developer.apple.com/documentation/dispatch/1385606-dispatch_source_set_timer

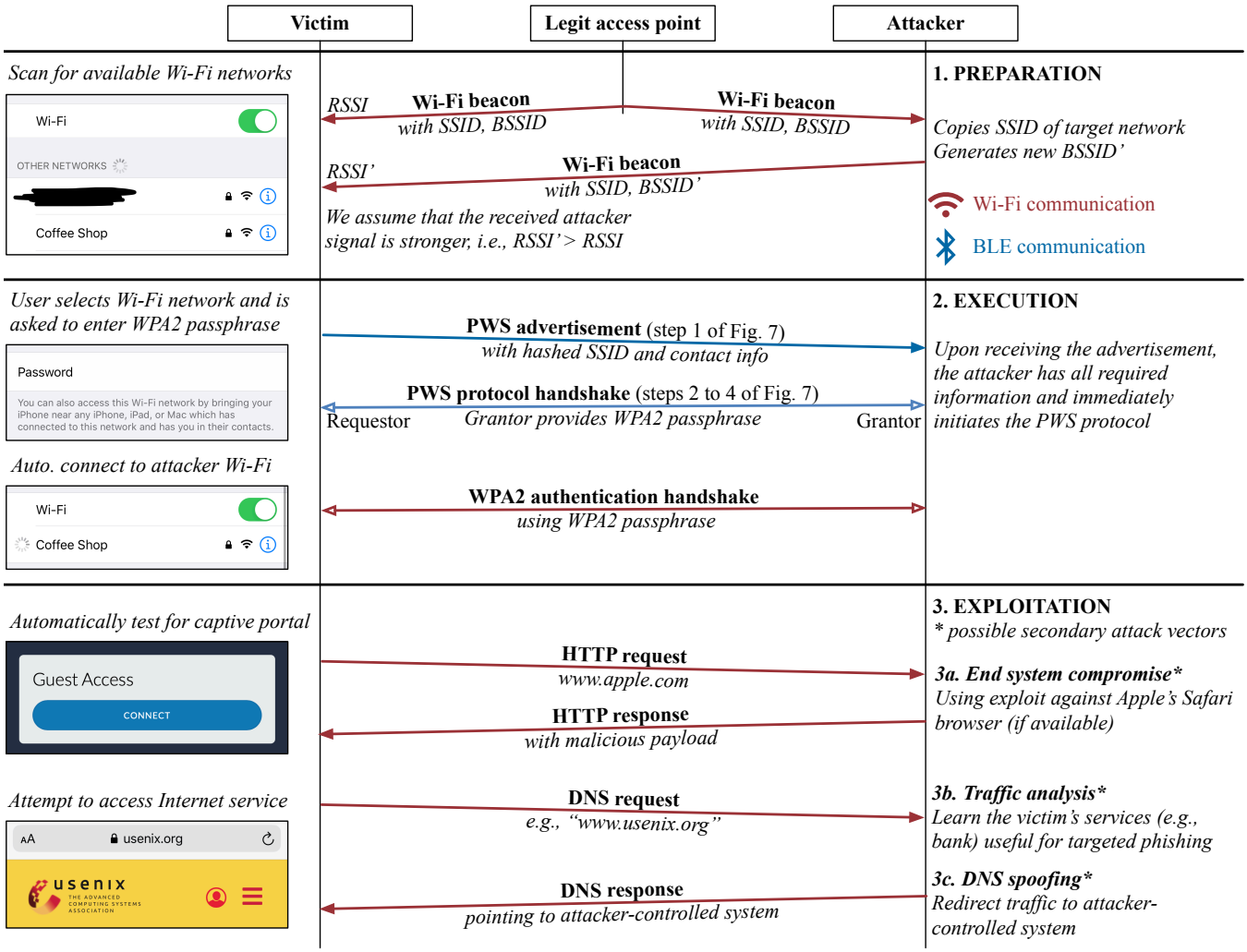


Figure 11: Protocol flow and user interaction of our Wi-Fi password auto-fill attack.

as per Apple's design [4]. In our case, the attacker acts as the grantor and, therefore, does not need to possess any information about their target. We elaborate on this problem in the following.

In Section 4.2.5, we describe that the requestor proves its identity to the grantor using the validation record signed by Apple and the Apple ID certificate. Therefore, the grantor can verify that the requestor owns the contact identifiers in its advertisement. In contrast, the requestor does not check the identity of the grantor. Even though the hashed contact identifiers of the grantor are included in the PWS3 packet, they are never used on the requestor. Also, the PWS3 message does not contain the validation record and Apple ID certificate of the grantor. The mandatory SSID in PWS3 can be easily obtained by scanning the surrounding Wi-Fi networks and comparing the hashed names to the field in the BLE advertisement. We use the missing validation of the grantor, combined with the fact that no user interaction is necessary on the requestor to perform an attack against the requestor.

5.5.2 The Attack: SSID Spoofing and Wi-Fi Password Auto-Fill

This attack targets iOS and macOS devices while they are connecting to a new Wi-Fi network. The goal is to get the target device to connect to a password-protected Wi-Fi network with the same SSID, but controlled by the attacker, further named *spoofed network*. We show the complete protocol flow and user interaction in Fig. 11. The attacker could then use its MitM position to analyze the victim's traffic or mount secondary attacks such as DNS or NTP spoofing [32]. Besides, the attacker could use the automatically loaded captive portal web page [17] to exploit vulnerabilities in the Safari web browser [13], thereby extracting sensitive user data or accessing the user's camera [2].

Our experiments with different setups showed that while opening the password dialog, the requestor saves the BSSID with the strongest signal and only tries to connect to this

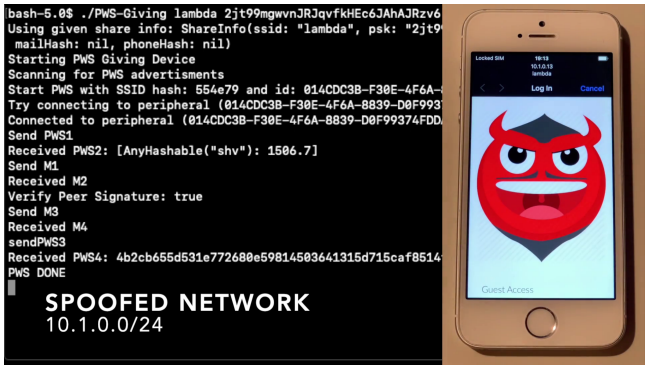


Figure 12: Screen capture of our video PoC [31] for the Wi-Fi password auto-fill attack.

BSSID. For a successful attack, the *spoofed network* needs to be the one with the strongest signal at that moment. The attacker can increase the transmit power of its access point or use directional antennas to increase their chances. The attacker continues by running our PWS client with the original SSID and the PSK of its *spoofed network*. Without any further user interaction required by the victim, once PWS is complete, the target device connects to the *spoofed network*. One problem with the presented attack is that a careful user might notice that they are automatically connected to a Wi-Fi network without having to type any password. We discovered that the grantor could hold the session open after receiving the Pair–Verify M2 packet, wait until the victim entered a password, and continue the attack, i. e., send M3, just before the victim hits connect. If continued in the right moment, e. g., by observing the victim, the attack is more likely to remain unnoticed. We provide a video PoC [31] to demonstrate the practical feasibility of the attack in Fig. 12. In the video, the attacker presents a crafted captive portal web page to its victim upon success.

5.5.3 The Mitigations: Mutual Authentication and Explicit Consent

The SSID duplication attack works due to the interactionless user interface on the requestor and the missing identity validation of the grantor. Therefore, we propose a two-step mitigation. First, we propose to introduce mutual authentication to the Pair–Verify handshake. It is unclear why Apple did not implement this is in the first place, given that AirDrop's authentication protocol is designed in this way [44]. With mutual authentication, the attack would be more difficult to carry out since the attacker would have to be in the contact list of the victim. Second, we propose to change the UI such that the user of the requestor can decide whether to accept a password from a grantor. Again, Apple has already implemented a similar mechanism in AirDrop, where a user is asked to accept an incoming file.

5.6 Preventing Wi-Fi Password Entry via Settings App Crash

We discover a parsing vulnerability in the PWS protocol that allows us to prevent Wi-Fi password entry of nearby devices.

5.6.1 The Vulnerability: Parsing Bug in PWS

While implementing our own PWS client, we discovered that when removing the mandatory SSID or the PSK key-value pair from the dictionary, which is sent in the PWS3 message shown in Fig. 13, the requestor fails to parse the packet and crashes the current application.

```
{
  dn: <Grantor name>,
  gr: 1,
  op: 5,
  eh: [base64(SHA(<email>)), ...],
  ph: [base64(SHA(<phone>)), ...],
  nw: <SSID>,
  psk: <Wi-Fi PSK>
}
```

Figure 13: PWS3 message highlighting the problematic fields.

5.6.2 The Attack: Preventing Password Entry for New Wi-Fi Networks

In this attack, we crash the Settings app on iOS or close the Wi-Fi password window on macOS of every device within Bluetooth range that is currently entering a password for a Wi-Fi network. Every device logged in with an Apple ID and has Bluetooth enabled sends out PWS advertisements once the user enters the Wi-Fi password view. We demonstrate the effectiveness of the attack in a video PoC [30].

5.6.3 The Mitigation: Check for Missing Fields

Apple should be able to fix the vulnerability by checking for empty or missing fields and graciously fail if an unexpected packet is encountered. Until a fix is provided, users can disable Bluetooth on their devices to thwart the attack.

6 Conclusion

Undocumented proprietary protocols are hard to analyze due to the costly initial investment in reverse-engineering, though severe vulnerabilities have been found in the past [18, 23, 34, 44]. Our method to conduct structured reverse engineering of Apple's *Continuity* wireless ecosystem is a crucial cornerstone that enables independent third-party security audits, which, in effect, help to protect the users of 1.5 billion devices worldwide. Using this method, we investigate the protocols involved in the Handoff (HO), Universal Clipboard (UC), and Wi-Fi Password Sharing (PWS) services and discover several vulnerabilities that enable denial-of-service (DoS) attacks, device tracking, and machine-in-the-middle (MitM) attacks. All of the attacks can be practically mounted from an attacker in proximity and only require low-cost hardware. To facilitate similar research in the future, we appeal to the manufacturers to document their proprietary protocols as Apple has already done with their Homekit Accessory Protocol (HAP) stack. In the meantime, we believe that our detailed findings can bootstrap the analysis of other Continuity services as certain protocol components (e. g., OPACK, Pair–Verify) seem to be shared across services such that follow-up work does not have to start from scratch.

Responsible Disclosure

We have shared our findings with Apple as we discovered them. Therefore, the disclosure timeline and progress differ by vulnerability (different follow-up IDs with Apple's product security team). In particular, we disclosed the DoS attack on HO in Section 5.2 on November 27, 2019, the linear IV tracking mitigation in Section 5.3 November 20, 2019, the asynchronous hostname randomization in Section 5.4 on November 27, 2019, the SSID spoofing and Wi-Fi password auto-fill attack in Section 5.5 on February 10, 2020, and the Settings app crash in Section 5.6 on January 13, 2020. So far, Apple has published security updates for two vulnerabilities as detailed in Table 3.

Availability

We release the following open-source software artifacts as part of the Open Wireless Link project [43]:

(1) a reverse-engineering toolkit for Continuity services (`apple-continuity-tools`),
(2) a decryption utility for HO and UC BLE advertisements (`handoff-ble-viewer`),
(3) an implementation of the HO and UC authentication protocol (`handoff-authentication-swift`),
(4) an implementation of a PWS requestor including an OPACK (de)serializer written in Python (`openwifipass`), and
(5) implementations of a PWS grantor and requestor written in Swift (`wifi-password-sharing`).

If the links do not work, prefix the respective project name with `https://github.com/seemoo-lab/`.

Acknowledgments

This work has been funded by the LOEWE initiative (Hesse, Germany) within the emergenCITY center and by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

References

[1] 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: *USENIX Security Symposium*. 2019. URL: https://www.usenix.org/conference/usenixsecurity19/presentation/antonioli.

[2] Apple Inc. *About the Security Content of Safari 13.0.5*. 2020. URL: https://support.apple.com/en-us/HT210922.

[3] Apple Inc. *Adopting Handoff*. 2016. URL: https://developer.apple.com/library/archive/documentation/UserExperience/Conceptual/Handoff/AdoptingHandoff/AdoptingHandoff.html.

[4] Apple Inc. *Apple Platform Security*. 2020. URL: https://support.apple.com/guide/security.

[5] Apple Inc. *Apple Reports Record First Quarter Results*. 2020. URL: https://www.apple.com/newsroom/2020/01/apple-reports-record-first-quarter-results/.

[6] Apple Inc. *HomeKit Accessory Protocol Specification*. 2017. URL: https://developer.apple.com/support/homekit-accessory-protocol/.

[7] Apple Inc. *How to Share Your Wi-Fi password from Your iPhone, iPad, or iPod Touch*. 2019. URL: https://support.apple.com/en-us/HT209368.

[8] Apple Inc. "Introducing iPad Apps for Mac". In: *Apple Worldwide Developers Conference (WWDC)*. 2019. URL: https://developer.apple.com/videos/play/wwdc2019/205/.

[9] Apple Inc. *Use Continuity to Connect Your Mac, iPhone, iPad, iPod Touch, and Apple Watch*. 2020. URL: https://support.apple.com/en-us/HT204681.

[10] Jean-Philippe Aumasson and Daniel J. Bernstein. "SipHash: A Fast Short-Input PRF". In: *INDOCRYPT*. Springer, 2012.

[11] Xiaolong Bai, Luyi Xing, Nan Zhang, Xiaofeng Wang, Xiaojing Liao, Tongxin Li, and Shi-Min Hu. "Staying Secure and Unprepared: Understanding and Mitigating the Security Risks of Apple ZeroConf". In: *IEEE Symposium on Security and Privacy (S&P)*. 2016. DOI: 10.1109/SP.2016.45.

[12] Elad Barkan, Eli Biham, and Nathan Keller. "Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication". In: *Advances in Cryptology (CRYPTO)*. 2003. DOI: 10.1007/978-3-540-45146-4_35.

[13] Ian Beer. *A Very Deep Dive into iOS Exploit Chains Found in the Wild*. 2019. URL: https://googleprojectzero.blogspot.com/2019/08/a-very-deep-dive-into-ios-exploit.html.

[14] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. "High-Speed High-Security Signatures". In: *Journal of Cryptographic Engineering* (2012). DOI: 10.1007/s13389-012-0027-1.

[15] Bluetooth SIG. *Bluetooth Core Specification v5.1*. 2019. URL: https://www.bluetooth.com/specifications/bluetooth-core-specification/.

[16] Paulo Borges. *BLESSED*. URL: https://github.com/pauloborges/blessed.

[17] Solving the Captive Portal Problem on iOS. *Butler, Ross*. 2018. URL: https://medium.com/@rwbutler/solving-the-captive-portal-problem-on-ios-9a53ba2b381e.

[18] Guillaume Celosia and Mathieu Cunche. "Discontinued Privacy: Personal Data Leaks in Apple Bluetooth-Low-Energy Continuity Protocols". In: *Proceedings on Privacy Enhancing Technologies* (2020). DOI: 10.2478/popets-2020-0003.

[19] Aldo Cortesi, Maximilian Hils, and Thomas Kriechbaumer. *mitmproxy: a Free and Open Source Interactive HTTPS Proxy*. URL: https://mitmproxy.org.

[20] Kim Jong Cracks. *checkra1n: Jailbreak for iPhone 5s Through iPhone X, iOS 12.3 and Up*. URL: https://checkra.in.

[21] Morris Dworkin. *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. Special Publication 800-38D. NIST, 2007.

[22] Alexander Heinrich. "Analyzing Apple's Private Wireless Communication Protocols with a Focus on Security and Privacy". MA thesis. Technical University of Darmstadt, 2019.

[23] Alexander Heinrich, Matthias Hollick, Thomas Schneider, Milan Stute, and Christian Weinert. "PrivateDrop: Practical Privacy-Preserving Authentication for Apple AirDrop". In: *USENIX Security Symposium*. To appear. 2021.

[24] Alexander Heinrich, Milan Stute, and Matthias Hollick. "BTLEmap: Nmap for Bluetooth Low Energy". In: *ACM Conference on Security and Privacy in Wireless and Mobile Networks*. WiSec '20. 2020. DOI: 10.1145/3395351.3401796.

[25] Saagar Jha. *Making os_log Public on macOS Catalina*. 2019. URL: https://saagarjha.com/blog/2019/09/29/making-os-log-public-on-macos-catalina/.

[26] Donald Knuth. *The Art of Computer Programming*. Vol. 2. Addison-Wesley, 1969.

[27] H. Krawczyk and P. Eronen. *HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*. RFC 5869. IETF, 2010. DOI: 10.17487/RFC5869.

[28] A. Langley, M. Hamburg, and S. Turner. *Elliptic Curves for Security*. RFC 7748. IETF, 2016. DOI: 10.17487/RFC2016.

[29] Chi-Yu Li, Guan-Hua Tu, Chunyi Peng, Zengwen Yuan, Yuanjie Li, Songwu Lu, and Xinbing Wang. "Insecurity of Voice Solution VoLTE in LTE Mobile Networks". In: *ACM Conference on Computer and Communications Security (CCS)*. 2015. DOI: 10.1145/2810103.2813618.

[30] Jannik Lorenz. *Video PoC: iOS Settings Crash via Apple Wi-Fi Password Sharing*. 2020. URL: https://youtu.be/MrPG5A1vSyE.

[31] Jannik Lorenz. *Video PoC: Man-in-the-Middle Attack via Wi-Fi Password Sharing (Auto-Fill Password)*. 2020. URL: https://youtu.be/a9OE2uZTWow.

[32] Aanchal Malhotra. "Attacking the Network Time Protocol". In: *Network and Distributed System Security Symposium (NDSS)*. 2016. DOI: 10.14722/ndss.2016.23090.

[33] Dennis Mantz, Jiska Classen, Matthias Schulz, and Matthias Hollick. "InternalBlue – Bluetooth Binary Patching and Experimentation Framework". In: *ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*. 2019. DOI: 10.1145/3307334.3326089.

[34] Jeremy Martin, Douglas Alpuche, Kristina Bodeman, Lamont Brown, Ellis Fenske, Lucas Foppe, Travis Mayberry, Erik Rye, Brandon Sipes, and Sam Teplov. "Handoff All Your Privacy: A Review of Apple's Bluetooth Low Energy Implementation". In: (2019). DOI: 10.2478/popets-2019-0057.

[35] Ulrike Meyer and Susanne Wetzel. "A Man-in-the-Middle Attack on UMTS". In: *ACM Workshop on Wireless Security (WiSe)*. 2004. DOI: 10.1145/1023646.1023662.

[36] Micro:bit Educational Foundation. *Micro:bit website*. URL: https://microbit.org.

[37] Yoav Nir and Adam Langley. *ChaCha20 and Poly1305 for IETF Protocols*. RFC 7539. IETF, 2015. DOI: 10.17487/RFC7539.

[38] Ole André V. Ravnås. *Frida: A World-Class Dynamic Instrumentation Framework*. URL: https://frida.re.

[39] Adam Stubblefield, John Ioannidis, and Aviel D. Rubin. "Using the Fluhrer, Mantin, and Shamir Attack to Break WEP". In: *Network and Distributed System Security Symposium (NDSS)*. 2002.

[40] Milan Stute. "Availability by Design: Practical Denial-of-Service-Resilient Distributed Wireless Networks". Ph.D. thesis. Technical University of Darmstadt, 2020. DOI: 10.25534/tuprints-00011457.

[41] Milan Stute, David Kreitschmann, and Matthias Hollick. "Demo: Linux Goes Apple Picking: Cross-Platform Ad hoc Communication with Apple Wireless Direct Link". In: *ACM Conference on Mobile Computing and Networking (MobiCom)*. 2018. DOI: 10.1145/3241539.3267716.

[42] Milan Stute, David Kreitschmann, and Matthias Hollick. "One Billion Apples' Secret Sauce: Recipe for the Apple Wireless Direct Link Ad hoc Protocol". In: *ACM Conference on Mobile Computing and Networking (MobiCom)*. 2018. DOI: 10.1145/3241539.3241566.

[43] Milan Stute, David Kreitschmann, and Matthias Hollick. *The Open Wireless Link Project*. 2018. URL: https://owlink.org.

[44] Milan Stute, Sashank Narain, Alex Mariotto, Alexander Heinrich, David Kreitschmann, Guevara Noubir, and Matthias Hollick. "A Billion Open Interfaces for Eve and Mallory: MitM, DoS, and Tracking Attacks on iOS and macOS Through Apple Wireless Direct Link". In: *USENIX Security Symposium*. 2019. URL: https://www.usenix.org/conference/usenixsecurity19/presentation/stute.

[45] Pierre B. Vandwalle, Tashbeeb Haque, Andreas Wolf, and Saravanan Balasubramaniyan. *Method and Apparatus for Cooperative Channel Switching*. U.S. Patent 9491593. 2016.

[46] Pierre B. Vandwalle, Christiaan A. Hartman, Robert Stacey, Peter N. Heerboth, and Tito Thomas. *Synchronization of Devices in a Peer-to-Peer Network Environment*. U.S. Patent 9473574. 2016.

[47] Mathy Vanhoef and Frank Piessens. "Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2". In: *ACM Conference on Computer and Communications Security (CCS)*. 2017. DOI: 10.1145/3133956.3134027.

[48] Mathy Vanhoef and Eyal Ronen. "Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd". In: *IEEE Symposium on Security & Privacy (S&P)*. 2020. DOI: 10.1109/SP40000.2020.00031.

[49] Wireshark Foundation and contributors. *Wireshark*. URL: https://www.wireshark.org.