PrivateDrop: Practical Privacy-Preserving Authentication for Apple AirDrop

Alexander Heinrich, Matthias Hollick, Thomas Schneider,
Milan Stute, and Christian Weinert, *TU Darmstadt*

https://www.usenix.org/conference/usenixsecurity21/presentation/heinrich

This paper is included in the Proceedings of the
30th USENIX Security Symposium.

August 11–13, 2021

978-1-939133-24-3

Open access to the Proceedings of the
30th USENIX Security Symposium
is sponsored by USENIX.

PrivateDrop: Practical Privacy-Preserving Authentication for Apple AirDrop

Alexander Heinrich Matthias Hollick Thomas Schneider
Milan Stute Christian Weinert

Technical University of Darmstadt, Germany

Abstract

Apple's offline file-sharing service AirDrop is integrated into more than 1.5 billion end-user devices worldwide. We discovered two design flaws in the underlying protocol that allow attackers to learn the phone numbers and email addresses of both sender and receiver devices. As a remediation, we study the applicability of private set intersection (PSI) to mutual authentication, which is similar to contact discovery in mobile messengers. We propose a novel optimized PSI-based protocol called *PrivateDrop* that addresses the specific challenges of offline resource-constrained operation and integrates seamlessly into the current AirDrop protocol stack. Using our native PrivateDrop implementation for iOS and macOS, we experimentally demonstrate that PrivateDrop preserves AirDrop's exemplary user experience with an authentication delay well below one second. We responsibly disclosed our findings to Apple and open-sourced our PrivateDrop implementation.

1 Introduction

Apple AirDrop is a file-sharing service integrated into more than 1.5 billion end-user devices worldwide [5], including iPhone, iPad, and Mac systems, and has been in operation since 2011. AirDrop runs fully offline and only uses a direct Wi-Fi connection in combination with Bluetooth Low Energy (BLE) between two devices. We discovered *two* severe privacy vulnerabilities in the underlying authentication protocol. In particular, the flaws allow an adversary to learn contact identifiers (i.e., phone numbers and email addresses) of nearby AirDrop senders *and* receivers. The flaws originate from the exchange of hash values of such contact identifiers during the discovery process, which can be easily reversed using brute-force or dictionary attacks [35, 42, 66].

Challenge. During authentication, two AirDrop devices run a form of *contact discovery* where they determine if they are mutual contacts, i.e., whether or not they have stored each others' contact information in their address book [92]. A connection is only deemed authentic if the result is positive.

Privacy-preserving contact discovery is commonly addressed via private set intersection (PSI) in the literature (e.g., [55, 59]). PSI protocols, in general, are cryptographic protocols that allow two interacting parties to securely compute the intersection of their respective input sets without leaking any additional data. PSI is already deployed in the real world, e.g., for compromised credential checking in Google's browser Chrome [93] in a business-to-consumer (B2C) context and for calculating ad conversion rates with Google in a business-to-business (B2B) context [51]. In a consumer-to-consumer (C2C) context, PSI has been proposed for preventing cheating in online gaming [20] and most recently for contact tracing in light of the COVID-19 pandemic (e.g., [94]). With our work, we aim to facilitate the deployment of PSI in a C2C context for mutual authentication.

However, the AirDrop scenario poses a unique set of challenges: a solution needs to (a) run completely offline without any third-party server support, (b) consider malicious parties that lie about their address book entries or own contact identifiers, (c) run on mobile devices with restricted energy and computational resources, and (d) preserve the user experience by not adding noticeable authentication delays.

Our contributions. We study the applicability of PSI to realize private mutual authentication for AirDrop. For this, we first systematically explore all possible design options and available building blocks from the literature. Our final solution, called *PrivateDrop*, is based on a Diffie-Hellman-style PSI protocol [53], which is even secure in the presence of malicious actors that actively try to extract sensitive information. We apply a two-way variant of [53] and optimize online performance by minimizing the number of communication rounds and by allowing to precompute expensive operations, e.g., when the device charges overnight. To accommodate malicious inputs, especially attackers lying about their contact identifiers, we propose to use signed PSI inputs [21, 31, 33] that complement AirDrop's current validation records and can be issued using Apple's existing certification infrastructure.

Furthermore, we integrate PrivateDrop into the original AirDrop protocol stack, including the BLE-based discov-

ery mechanism as well as the HTTPS-based authentication phase. We implement both the original AirDrop protocol and our PrivateDrop extension in native code for iOS and macOS, which we open-sourced on GitHub [45].

Finally, in an extensive performance evaluation, we demonstrate that PrivateDrop incurs only negligible overhead in practice. In particular, we experimentally show that the authentication delay stays well below 1 s even for large address books with > 10 k entries, which humans perceive as an "immediate response" [22]. In realistic scenarios, the delay even stays below 500 ms—only a $2\times$ increase compared to the authentication delay in the original insecure AirDrop protocol.

We disclosed both vulnerabilities and our proposed mitigation to the Apple Product Security team and are awaiting their feedback. We summarize our contributions as follows:

(a) We discover and disclose two distinct design flaws in the AirDrop authentication protocol that enable an attacker to learn contact identifiers (phone numbers and email addresses) of nearby devices.
(b) We propose PrivateDrop, a new PSI-based mutual authentication protocol that integrates seamlessly into the current AirDrop protocol stack. Our design is based on a Diffie-Hellman-style PSI protocol [53] and protects against malicious adversaries as well as inputs.
(c) We re-implement the original AirDrop protocol stack, integrate our PSI-based protocol for iOS and macOS, and open-source our code [45].
(d) We experimentally show that PrivateDrop provides immediate responses [22] with < 1 s authentication delay.

Outline. Our paper is structured as follows: We first describe the currently deployed AirDrop protocol (§ 2) and discuss the vulnerabilities we discovered (§ 3). Then, we present our novel PSI-based mutual authentication protocol (§ 4). We furthermore describe our implementation (§ 5), followed by our extensive experimental evaluation (§ 6). Finally, we discuss related work (§ 7) before concluding (§ 8).

2 Background: Apple AirDrop

Apple's file-sharing service AirDrop is integrated in all current iOS and macOS devices. It runs completely offline using a proprietary Wi-Fi link-layer called Apple Wireless Direct Link (AWDL) [90] in combination with Bluetooth Low Energy (BLE). As there exists no official documentation of the involved protocol stack, we describe AirDrop based on the reverse engineering of [92]. In particular, we first define *contact identifiers* and discuss the available *discoverability* settings. Then, we describe the complete technical protocol flow and explain the authentication process as presented in [92].

2.1 Contact Identifiers and the Address Book

Each iOS or macOS device has an address book that is accessible through the *Contacts* application. This address book contains several contact entries that in turn consist of multiple objects such as name or contact information. AirDrop leverages the user's own contact identifiers and their address book entries for authentication purposes. In particular, AirDrop uses phone numbers and email addresses to identify a contact. This is possible as every Apple account (often referred to as *Apple ID* or *iCloud account*) has at least one such contact identifier assigned to it. Apple uses verification emails and SMS to verify the ownership of the email address or phone number, respectively, thus assuring the correctness of the identifiers.

Within the context of this paper, we will only deal with contact identifiers, i.e., phone numbers and email addresses, and disregard the notion of "contacts" that might—in turn—consist of multiple identifiers. We assume there exists a device-local unambiguous mapping for contact identifiers to contact list entries. We use the term *address book (AB)* to refer to the set of contact identifiers of all contact entries in the device's contact list. Note that the AB is controlled by the user and not verified by Apple. In addition, the user's own *contact identifiers (IDs)* are the Apple-verified phone numbers and email addresses that are assigned to the user's Apple account. We use the notation c to refer to an address book entry and ID to refer to an Apple-verified contact identifier.

2.2 Device Discoverability

When opening the sharing pane on an iOS device, nearby devices appear in the user interface if they are discoverable [10]. In particular, receiver devices can be discovered by *everybody* or by *contacts only*, which is the default setting. In either case, an AirDrop sender will attempt to perform a mutual authentication handshake with a responding receiver. Note that the issues addressed in our paper (i.e., the leakage of contact identifiers of sender and receiver during the authentication process) affect both settings.

2.3 Full Protocol Workflow

The AirDrop protocol allows a *sender* to transmit a file or link to a *receiver*. It consists of the three phases *discovery*, *authentication*, and *data transfer*, which we explain based on [92] and depict in Fig. 1: (a) When the sender opens the sharing pane, it starts emitting BLE advertisements that contain a truncated hash for each contact identifier. A receiver compares the sender's hashed contact identifiers with entries in their address book. The receiver activates their AWDL interface if at least one contact match was found in contacts-only mode or if it is discoverable by everyone. The sender then proceeds by searching for AirDrop services with DNS service discovery (DNS-SD) via the AWDL interface. (b) For each discovered service, the sender initiates an authentication procedure via an HTTPS *Discover* request that we detail in § 2.4. If the authentication procedure completes successfully, the receiver's identity is displayed in the sender's user interface. (c) Finally, the sender selects the receiver and sends two subsequent requests: The *Ask* request contains metadata

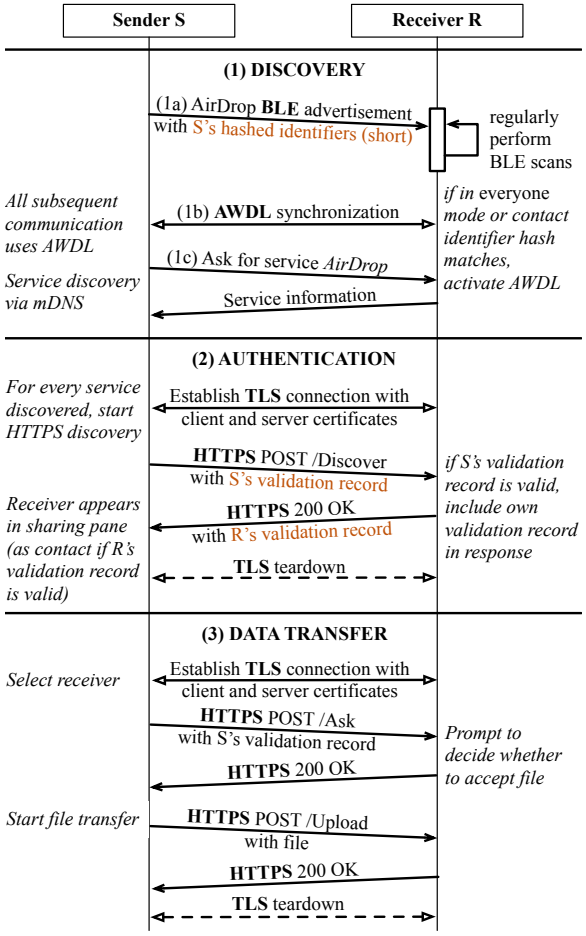

Figure 1: AirDrop protocol (simplified version from [92]). The orange message parts leak the sender's and receiver's contact identifiers, as discussed in § 3.3 and § 3.4, respectively.

about the file, including a thumbnail. The receiver sends their decision on whether to receive the full file. Upon a positive response, the sender continues to transfer the complete file in an *Upload* request or aborts the transaction otherwise.

2.4 Mutual Authentication

An authenticated connection can only be established between users with an Apple ID who are present in each others' address books. In order to authenticate, a device needs to prove that it has registered a certain contact identifier ID_i such as phone number or email address associated with its Apple ID, while the verifying device checks whether ID_i is an address book entry. Authentication involves multiple Apple-signed certificates and a chain of Apple-run certificate authorities (CAs). In particular, AirDrop uses a device-specific certificate σ_{UUID} and a validation record VR_σ, which are both signed by Apple. The devices retrieve them both from Apple once the user logs in to their iCloud account. They can then be used offline in any subsequent AirDrop transaction.

The certificate σ_{UUID} contains an account-specific universally unique identifier (UUID).[1] The certificate is used as a client or server certificate (depending on the role) in the TLS connection. As the UUID in the certificate does not link any contact identifiers, AirDrop uses an Apple-signed *Apple ID validation record* (VR_σ). The validation record contains the UUID from the TLS certificate and all contact identifiers $\mathit{SHA\text{-}256}(ID_1), \ldots, \mathit{SHA\text{-}256}(ID_m)$ that are registered with the user's Apple ID in hashed form. Also, VR_σ includes a signature and the certificate of the signing CA σ_{VR}.[2] Formally, we define VR_σ as follows:

$$VR = (\mathit{UUID}, \mathit{SHA\text{-}256}(ID_1), \ldots, \mathit{SHA\text{-}256}(ID_m)) \quad (1)$$

$$VR_\sigma = (VR, \mathit{sign}(\sigma_{\text{VR}}, VR), \sigma_{\text{VR}}), \quad (2)$$

where $\mathit{sign}(\sigma_{\text{VR}}, VR)$ is the signature of VR for certificate σ_{VR}. During authentication, AirDrop (a) verifies the signature on the received validation record, (b) verifies that the UUID in the certificate matches the one in the validation record, and (c) computes the SHA-256 hash over each normalized[3] address book entry and compares them with the hashes contained in the validation record. Authentication succeeds if all checks pass. If authentication fails on the receiver side, the receiver aborts the connection. However, if authentication fails on the sender side, AirDrop continues the transaction but treats the connection as unauthenticated and the peer as a non-contact. AirDrop shows contacts with their name and picture from the address book in the user interface. Non-contacts are displayed using the device name without a picture instead.

3 Contact Identifier Leakage in AirDrop

We discovered two design flaws in the AirDrop protocol that allow an adversary to learn the contact identifiers (both phone numbers and email addresses) of nearby Apple devices. The two flaws originate from AirDrop's authentication handshake, where hashed contact identifiers are exchanged as part of Apple's validation record. First, we define the threat model and discuss that cryptographic hash functions cannot hide their inputs (called preimages) when the input space is small or predictable, such as for phone numbers or email addresses. Second, we explain where and to what extent AirDrop devices are vulnerable to contact identifier leakage. We responsibly disclosed our findings to Apple (cf. § 8). A subset of the issues presented in the following was independently reported in [25]. However, that report does not address hashed email addresses and receiver leakage (cf. § 3.4), and was published one month *after* our disclosure with Apple. Moreover, there are no signs that [25] followed responsible disclosure.

[1] As an addition to [92], we found that the UUID is not device-specific but equal for all devices using the same Apple account.

[2] We hide the fact that VR_σ contains the complete certificate chain up to Apple's root CA [92] to keep our description short and concise.

[3] Phone numbers are hashed in a normalized digit-only form, e.g., the string "+1 (234) 567-8901" is hashed as "12345678901".

3.1 Threat Model

In this paper, we consider an adversary that wants to learn contact identifiers (phone numbers and email addresses) from non-contact AirDrop devices in proximity. They might then use these identifiers for fraudulent activities such as (spear) phishing attacks or making a profit by selling personal data.

Specifically, the adversary must be in physical proximity of its targets (similar to [88]) and have access to a device with an off-the-shelf Wi-Fi card to communicate via AWDL [89]. We assume that the adversary has full control over the wireless channel and can, e.g., mount machine-in-the-middle attacks [92]. The adversary may lie about its address book (AB) entries and arbitrarily deviate from the protocol description, but cannot break Apple's contact identifier ownership verification (cf. § 2.1), i.e., the adversary is unable to forge valid certificates for arbitrary contact identifiers (IDs).

We assume that Apple is trustworthy as it acts as a certificate authority (cf. § 2.4) and learns the contact identifiers, but not the address book entries, from all of its users through the ownership verification process.

3.2 Recovering Hashed Contact Identifiers

Hashing is insufficient to hide phone numbers or email addresses as the input space is small/predictable [35, 42, 66].

Phone numbers. Recovering the preimage of a hashed phone number can be achieved using brute force because the phone number space is relatively small. For example, a US phone number contains an area code followed by 7 digits. Given this small search space (10^7), it is feasible to check all possible phone numbers on a PC within seconds.

More precisely, a recent work [42] studied three different approaches for efficiently reversing phone number hashes: lookups in large-scale key-value stores, brute-force attacks, and optimized rainbow-table constructions. The authors also modeled a worldwide database of valid mobile phone number prefixes that revealed vast differences in terms of phone number structure between countries and, therefore, the size of the search space (e.g., in Austria, the search space is in the order of 10^{10} compared to 10^7 in the US). Each of the investigated reversal methods was able to reverse SHA-1 hashes with an amortized runtime in the order of milliseconds (e.g., 52 ms for the optimized rainbow-table construction). These results are directly applicable to estimate the effort required for an attacker to recover a phone number from the hashes leaked in AirDrop (cf. § 3.3 and § 3.4). However, since AirDrop uses SHA-256 instead of SHA-1, the runtime and storage requirements stated in [42] likely increase by around factors $3\times$ and $1.6\times$, respectively [49].

Email addresses. Recovering the preimage of a hashed email address is less trivial but possible via dictionary attacks that check common email formats such as `first.lastname@{gmail.com,yahoo.com,...}`. Alternatively, an attacker could generate an email lookup table from data breaches [48] or use an online lookup service for hashed email addresses [34].

3.3 Contact Identifier Leakage of Sender

During the AirDrop authentication handshake, the sender always discloses their own contact identifiers as part of the initial HTTPS POST /Discover message (cf. Fig. 1). A malicious receiver can therefore learn all (hashed) contact identifiers of the sender *without requiring any prior knowledge* of their target. To obtain these identifiers, an attacker simply needs to wait (e.g., at a public hot spot) until a target device scans for AirDrop receivers, i.e., *the user opens the AirDrop sharing pane*. The target device will freely send a discover message to any AirDrop receiver found during the previous DNS-SD service lookup. Therefore, an attacker can learn the target's validation record without any authentication by simply announcing an AirDrop service via multicast DNS (mDNS). After collecting the validation record, the attacker can recover the hashed contact identifiers offline.

3.4 Contact Identifier Leakage of Receiver

AirDrop receivers present their contact identifiers in the HTTPS 200 OK response to the discover message if they know any of the sender's contact identifiers included in the validation record (cf. Fig. 1). A malicious sender can thus learn all contact identifiers *without requiring any prior knowledge* of the receiver *if the receiver knows the sender*. Importantly, the malicious sender does not have to know the receiver: A popular person within a certain context (e.g., the manager of a company) can exploit this design flaw to learn all contact identifiers of other people who have the popular person in their address book (e.g., employees of the company).

4 PrivateDrop: PSI-based Mutual Authentication for AirDrop

In the following, we describe how PSI can be applied to realize PrivateDrop, our private mutual authentication protocol for AirDrop that protects against both attacks described in § 3.

In general, given sender S and receiver R with verified contact identifiers and size-constrained address books (IDs^S, AB^S) and (IDs^R, AB^R), respectively, a privacy-preserving mutual authentication protocol must ensure that S and R learn *at most* those contact identifiers of the other party that they already have in their address book, i.e., S learns *at most* $AB^S \cap IDs^R$ and R learns *at most* $AB^R \cap IDs^S$.[4]

Private set intersection (PSI) protocols are cryptographic protocols that securely compute the intersection $A \cap B$ for two parties with respective private input sets A and B. For the remainder of this paper, we denote the party obtaining the intersection result as *PSI receiver* and the respective other party as *PSI sender*.[5] Importantly, with PSI, no elements outside the intersection, i.e., from $(A \cup B) \setminus (A \cap B)$, are leaked.

[4] During AirDrop authentication, S learns IDs^R if $IDs^S \cap AB^R \neq \emptyset$ and R learns IDs^S unconditionally, resulting in the vulnerabilites described in § 3.

[5] There also exist PSI protocols where both parties can be receivers, but this property is not required for our authentication purposes.

To instantiate PrivateDrop, we first fix our requirements for the authentication protocol, explore the different design options when applying PSI, choose a suitable PSI protocol from the literature, adapt and optimize it for our use case, and seamlessly integrate it into AirDrop.

4.1 Requirements

Our primary goal is to prevent both attacks described in § 3 by protecting *contact identifiers* (Apple-verified phone numbers and email addresses assigned to a user's Apple account, cf. § 2.1) and *validation records* (Apple-signed lists of hashed contact identifiers, cf. § 2.4). Concretely, in terms of functionality and *privacy* for the AirDrop authentication, we want to simultaneously achieve the following properties:

(a) Disclose validation records only *if both parties are mutual contacts*. If both parties are mutual contacts, they already know at least one contact identifier of the respective other party. Thus, the hash values enclosed in the validation records do not leak personal information via brute-force or dictionary attacks (cf. § 3.2).

(b) In the validation records, disclose only those *contact identifiers that the other party already knows*. Even though mutual contacts already know at least one contact identifier of the respective other party, the validation records contain hash values of *all* registered identifiers. Thus, the hash values of contact identifiers not known to the respective other party leak additional personal information via brute-force or dictionary attacks (cf. § 3.2).

We use *A knows B* as a shorthand for *A has one of B's verified contact identifiers (IDs^B) in their size-constrained (cf. § 4.5) address book (AB^A)*, or formally: $AB^A \cap IDs^B \neq \emptyset$.

In terms of *performance*, we want to minimize computation as well as communication overhead. This is important to achieve a low energy consumption for battery-driven mobile devices and to deliver a great user experience with immediate responses. Since AirDrop is primarily used on mobile devices, which might be offline from time to time, our solution must be fully *decentralized* and cannot involve an external server. Furthermore, we have to consider that parties might act *maliciously*, i.e., may try to apply arbitrary strategies with the intent to extract personal information.

4.2 Design Options and Final Design

We now describe how to apply PSI to realize private mutual authentication for AirDrop, considering the requirements defined in § 4.1. The main task is to replace the insecure exchange of hash values that happens in the original authentication phase as a result of sending validation records (cf. § 2.4).

Our high-level idea summarized in Fig. 2 is to have two consecutive PSI executions. The first execution ensures the AirDrop sender knows the receiver, the second that the AirDrop receiver knows the sender. Afterward, as each party is assured that it is stored in the respective other party's address book, it is safe for them to reveal their contact identifiers and validation records. In the following, we detail how to configure the PSI executions to achieve the described outcome by systematically analyzing all possible design options.

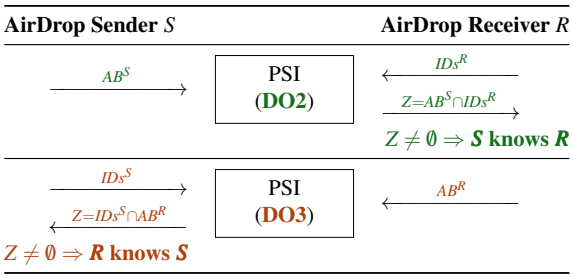

Figure 2: PrivateDrop's PSI-based mutual authentication protocol for AirDrop. The PSI protocols are instantiated using DO2 (green) and DO3 (orange), cf. § 4.2. Inputs are the parties' contact identifiers (IDs) and address books (AB).

Table 1: Available design options (DO) to use PSI for private mutual authentication in AirDrop. Possible inputs are contact identifiers (IDs) and address books (AB). The parties can act as PSI sender (PSI S) or PSI receiver (PSI R).

Design Option	DO1	DO2	DO3	DO4
Role of AirDrop Sender	PSI S	PSI S	PSI R	PSI R
Input of AirDrop Sender	IDs	AB	IDs	AB
Role of AirDrop Receiver	PSI R	PSI R	PSI S	PSI S
Input of AirDrop Receiver	AB	IDs	AB	IDs

The design options (DOs) listed in Tab. 1 differ in (a) the PSI inputs for the AirDrop sender and receiver, i.e., contact identifiers and address books, (b) the roles the parties take in PSI, and (c) the order in which the DOs are executed.

Note that we exclude combinations where both parties input their contact identifiers since the intersection will always be empty. Likewise, we do not consider both parties using their address book as input, since this variant (formalized in [32] as private contact discovery between two users) yields the parties' common contacts (i.e., finds "friends of friends" [12]) but does not determine whether they are mutual contacts.

Regarding the assignment of the PSI roles and the execution order, we can exclude further combinations. As both AirDrop sender and receiver must be assured of being mutual contacts, each must act as PSI receiver once. In the authentication process, the AirDrop sender should be the first to reveal information as otherwise malicious senders could easily extract such information from a large number of innocent receivers by triggering the authentication process. Therefore, the options must be chained such that the AirDrop receiver acts as PSI receiver first (DO1 or DO2) and as sender second (DO3 or DO4). In the following, we discuss the two remaining possibilities.

DO1 → DO4. Here, the PSI sender has their contact identifiers as input, whereas the PSI receiver has their address book as input. As a result, each party is assured that the other party is one of its contacts. This is the exact semantic as

in the original (insecure) authentication protocol. However, since malicious AirDrop receivers do not necessarily abort after receiving an empty result set in the first PSI execution, AirDrop senders have no proof that the receivers know them before revealing their contact identifiers. Since we strictly want to avoid this information leakage (cf. § 3.3), we discard DO1 → DO4.

DO2 → DO3. Here, the PSI sender has their address book as input, whereas the PSI receiver has their contact identifiers as input. At the end of the authentication process, each party can be assured that it is stored in the respective other party's address book. Thus, the AirDrop sender can safely share their contact identifiers that appeared in the outcome of the DO3 execution since the other party already has them stored.

In conclusion, by executing DO2 → DO3 in that particular order (as visualized in Fig. 2), we can fulfill the functional and privacy requirements defined in § 4.1, and prevent our attacks described in § 3.

4.3 Choice of PSI Protocol

Now that we fixed in which order two PSI protocols have to be run, we need to find instantiations. In the literature, many two-party PSI protocols are proposed that could be applied (cf. § 7.2). Especially, a sub-category of PSI protocols specializes in *unbalanced* set sizes, where one party has a much larger input set than the other [26, 27, 55, 59, 82]. The protocols [26, 27] are based on homomorphic encryption with communication linear in the size of the smaller set, but they are computationally expensive. The fastest unbalanced PSI protocols for mobile clients [55, 59, 82] shift most public-key operations to an input-independent precomputation phase and send an encrypted and compressed representation of the larger input set ahead of time to achieve fast online runtimes. Moreover, the protocols of [55] provide security against malicious PSI receivers but only work for semi-honest senders.

However, even though we deal with unbalanced sets, here, the size of the larger input set is determined by the maximum number of address book entries. The size of address books can be reasonably assumed to be well below 100 k and is not in the order of hundreds of millions as considered for unbalanced PSI. Thus, protocols based entirely on public-key encryption (which are extremely inefficient at a large scale) can achieve practical performance. In our setting, *both* parties are not constrained by business incentives or severe legal consequences to behave semi-honestly. Therefore, we must choose a protocol with security against a malicious sender *and* receiver. Furthermore, AirDrop is a protocol that is performed ad-hoc with random communication partners such that distributing encrypted databases in advance is not possible. Finally, we aim at providing industry-grade implementations for integration into Apple's ecosystem. Therefore, we need a simple protocol that does not require complex libraries for oblivious transfer or garbled circuits as needed in the most efficient protocols of [55, 59].

AirDrop/PSI Sender S		AirDrop/PSI Receiver R
Input: $AB = \{c_1, \ldots, c_n\}$		Input: $IDs = \{ID_1, \ldots, ID_m\}$
Output: $\perp$		Output: $AB \cap IDs$
$k, r \xleftarrow{\$} \mathbb{Z}_q$		For $i = 1$ to m:
		$h_i = H(ID_i)$, $\alpha_i \xleftarrow{\$} \mathbb{Z}_q$,
		$y_i = h_i^{\alpha_i}$
	$\xleftarrow{(y_1,\ldots,y_m)}$	
For $i = 1$ to m:		
$z_i = y_i^k$, $a_i = y_i^r$		
$c = H(y_1, z_1, a_1, \ldots, y_m, z_m, a_m)$		
$p = r + k \cdot c$		
	$\xrightarrow{(z_1,\ldots,z_m)}$	
	$\xrightarrow{(a_1,\ldots,a_m),p}$	$c = H(y_1, z_1, a_1, \ldots, y_m, z_m, a_m)$
For $j = 1$ to n:		For $i = 1$ to m:
$u_j = H(H(c_j), H(c_j)^k)$		Abort if $y_i^p \neq a_i \cdot z_i^c$
		$v_i = H(h_i, z_i^{1/\alpha_i})$
	$\xrightarrow{\{u_1,\ldots,u_n\}}$	output $\{ID_i \in IDs \mid \exists j : u_j = v_i\}$

Figure 3: Maliciously secure PSI protocol of [53] applied to DO2 (cf. § 4.2). The non-interactive zero-knowledge AND-proof of knowledge is marked in blue [16, 37, 85].

The PSI Protocol of [53]. Considering all requirements, we resort to a public key-based PSI protocol proposed by Jarecki and Liu [53]. This Diffie-Hellman-style protocol extends the work of Baldi et al. [13] by adding malicious security via zero-knowledge proofs.[6] The required public key operations can be efficiently instantiated with elliptic curve cryptography, for which there exist industry-grade libraries such as MIRACL [68] and built-in operating system capabilities (Apple CryptoKit [7] in iOS and macOS).

In Fig. 3, we summarize the PSI protocol of [53] applied to our use case. Specifically, we show the application to DO2 (cf. § 4.2). The application to DO3 works analogously with the same type of inputs (address book *AB* for PSI sender, identifiers *IDs* for PSI receiver), but the assignment of AirDrop sender/receiver to PSI sender/receiver is swapped.

For simplicity, H in our description denotes a hash function that maps either one or multiple bit strings or group elements to a short bit string of fixed length or an element in a multiplicative group of prime order q. The respective input and output domains are clear from the context. We instantiate H with the SHA-2 family [69] in our implementation (cf. § 5.2).

Informally, the protocol works as follows: (a) the PSI receiver hashes its input elements ID_i with a collision-resistant hash function H to group elements, encrypts the hash values h_i with random keys α_i, and sends the resulting values y_i to the PSI sender; (b) the PSI sender additionally encrypts the received elements with a random secret key k and sends the results z_i to the receiver; (c) the PSI receiver "removes" its own keys α_i such that it obliviously obtains the encryption of its inputs under the sender's key k; and finally (d) the PSI sender sends hashed encryptions u_j of its own input elements c_j in random order to the receiver, who then can compare the

[6] More precisely, malicious security is proven for an *adaptive* PSI functionality, where the receiver makes a series of adaptive queries instead of inputting its set as a whole. However, as the authors argue, any efficient adversary is committed to all its inputs at the execution time, and thus the adaptive functionality can be assumed to be equivalent to regular PSI [13].

values to determine the intersection. Following the PSI protocol of [76], the bitlength l of the values u_j can be reduced to $\lambda + 2\log_2(n)$, where λ is the statistical security parameter (which we set to $\lambda = 40$ in our implementation), and n is an upper bound on the number of address book entries each party has. This yields negligible failure probability $2^{-\lambda}$.

To achieve malicious security, the protocol utilizes a zero-knowledge proof of knowledge that makes sure the PSI sender knows and uses the same key k for computing all values z_i. This requires a so-called AND proof over the individual exponentiations. For an efficient and straight-forward instantiation, we choose Schnorr's DLOG proof [85] and apply the Fiat-Shamir heuristic [16, 37] to turn it into a non-interactive version (in the random oracle model), which does not require additional communication rounds (cf. blue part in Fig. 3).

The protocol in Fig. 3 leaks some information via the number of inputs. For example, one can learn whether an AirDrop sender is popular from the number of address book entries. To prevent such leakage, we pad the input sets with dummy elements to a globally fixed upper bound. For example, it is reasonable to limit the number of address book entries to $n = 10\,\text{k}$ and the number of contact identifiers to $m = 10$. In § 6, we assess the practical performance implications of such limits by conducting experiments with variable m and n.

4.4 Optimizing PSI for PrivateDrop

When integrating the PSI protocol of Fig. 3 into AirDrop, we apply several performance improvements.

Precomputation. First, it is possible for the PSI sender to generate the key k and compute the values u_i ahead of time. This can be done, e.g., overnight when the device is charging. It is only necessary to update the precomputed values as address book entries change. Since AB is the bigger input set, this removes the largest computation bottleneck from the protocol execution. Likewise, the PSI receiver can precompute the values y_i, which change seldomly. Similar precomputation techniques were proposed for passively secure DH-style PSI in [59, 82], and with security against malicious clients in [55]. The security of our protocol follows from the security of the protocol of [53]. Concretely, the simulation-based proof of [53] applies equally, as the parties' views remain identical.

Reusage. Moreover, it is possible to reuse the precomputed values across sessions. In previous works [55, 59, 82] that consider large-scale databases as input sets, the precomputed values are reused by encoding and distributing them in probabilistic data structures like Bloom or Cuckoo filters against which OPRF evaluations are checked.

From a standalone perspective, this allows for user tracking, but in AirDrop, users can already be tracked via the UUID in the TLS certificate used for establishing the protocol communication channel (cf. § 2.4). Avoiding user tracking in the entire AirDrop execution is an important area for future work. However, reusing precomputed encryptions of address book entries over longer periods of time allows tracking changes in the contact composition, i.e., how many contacts were added or removed since the last protocol execution. Even if no changes occur, this leaks some information, e.g., no new person was met or no person was "unfriended". In case this leakage should be avoided, fresh encryptions should be precomputed and never be reused.

Round Complexity. In terms of round complexity, it is possible to bundle the last two messages from the PSI sender to the receiver without changing the receiver's view. Thus, the PSI protocol consists of only one round, and the PSI receiver may ignore the received values u_i in case the zero-knowledge proof verification fails.

Furthermore, we optimize the sequential yet independent execution of DO2 and DO3. For this, we bundle the second message of DO2 with the first message of DO3. In total, both protocol executions require sending three messages, thus two rounds. Importantly, directly including the first DO3 message in the last DO2 message does not negatively impact the AirDrop sender in case of engaging with a malicious receiver. This is because in a sequential execution, the AirDrop sender gets no response at the end of DO2. Also, a malicious AirDrop receiver cannot learn any additional private information from receiving encryptions of hashed contact identifiers. Moreover, since the AirDrop receiver gets no response at the end of DO3 and the sender's inputs can be verified (cf. § 4.5), malicious behavior exploiting the sequential execution of the online phases can only influence correctness, but not input privacy.

Note that instead of our proposed three message protocol, it would be possible to further parallelize computation with a fully symmetric execution of DO2 and DO3. This would require sending four messages but can still be done in two rounds. However, to prevent malicious senders from causing unnecessary work for innocent receivers (denial-of-service attacks), we require the sender to first process the receiver's inputs and reveal its encrypted address book entries before starting the computation (cf. § 4.2). Moreover, the potential gain in overall efficiency via additional parallelization is negligible, since the constant overhead caused by one communication round ($\approx 100\,\text{ms}$, cf. Fig. 8) is larger than the entire online computation ($< 50\,\text{ms}$ even for $m = 10$ IDs, cf. Fig. 7).

4.5 Countering Privacy Attacks

The security properties of the PSI protocol in Fig. 3 prevent malicious parties from learning private information even when arbitrarily deviating from the protocol definition. However, malicious parties might tamper with the protocol inputs, which cannot be prevented by the protocol itself since this is an attack on the ideal functionality of set intersection. We now discuss the impact of such attacks and how to counter them by leveraging Apple's existing certification infrastructure.

Malicious Sender. A malicious AirDrop sender could try to obtain sensitive contact information of, e.g., VIPs by including a VIP's publicly known email address in their address

book. The PSI protocol then yields a match, and the vulnerable hash values of all contact identifiers of the VIP are sent in subsequent steps of the AirDrop protocol (including, e.g., the hashed phone number).

To prevent this attack, we modify the AirDrop protocol flow to release only hashed contact identifiers (in the validation record) for which a match in the PSI protocol was found. This requires a change to the current AirDrop validation record, which contains all contact identifiers, cf. Eqs. (1) and (2) on p. 3. In particular, we create individual validation records for each of the user's m contact identifiers ID_i as follows:

$$VR_i = (UUID, SHA\text{-}256(ID_i)), \quad \forall i \in 1,\ldots,m \tag{3}$$

$$VR_{\sigma,i} = (VR_i, sign(\sigma_{VR}, VR_i), \sigma_{VR}). \tag{4}$$

This yields a scalable solution as creating and distributing the validation records is a one-time cost, and the number of IDs per user m is expected to be small (e.g., $m = 10$).

Malicious Receiver. A malicious AirDrop receiver who knows the sender could try to trick the sender into believing they are mutual contacts by using contact identifiers that are stored in the sender's address book with high probability (e.g., emergency phone numbers). Moreover, with the same approach, a malicious AirDrop receiver can test whether the sender knows a specific person. To prevent such attacks, we propose to have the encrypted contact identifiers signed by Apple. The resulting protocol is then closely related to authorized PSI (APSI) [31, 33] and PSI with certified sets [21].

Similarly to the individual validation records in Eq. (4), we introduce Apple-signed certificates that contain the UUID and the precomputed values y_i for the user's contact identifiers:

$$Y_i = (UUID, y_i), \quad \forall i \in 1,\ldots,m \tag{5}$$

$$Y_{\sigma,i} = (Y_i, sign(\sigma_{VR}, Y_i), \sigma_{VR}) \quad . \tag{6}$$

PrivateDrop verifies that the UUID in Eq. (5) equals the one in the TLS certificate to prevent reuse by another party, thus, mitigating replay and machine-in-the-middle attacks. As with Eq. (4), this is a lightweight addition that does not require major changes in the existing infrastructure. The keys α_i can still be chosen on the client device. Only a simple zero-knowledge protocol must be run with Apple to make sure y_i is actually an encryption of a legitimately hashed contact identifier and the client device is in possession of the keys α_i. This can again be efficiently instantiated with Schnorr's protocol [85] and the Fiat-Shamir heuristic [16, 37] (cf. § 4.3). Alternatively, Apple could choose the keys α_i and hand them to client devices together with signed values $Y_{\sigma,i}$.

Brute-force. Finally, either party could try to guess contact identifiers of the other party by adding a large number of "fake" address book entries (so-called enumeration attacks [42]). However, in contrast to offline brute-force attacks, where up to millions of guesses can be checked per second, the success probability is significantly lower since we strictly limit the size of the input sets to a reasonable upper bound (e.g., $m = 10$ and $n = 10\,\text{k}$, cf. § 4.3).

Table 2: Overhead of PrivateDrop's PSI-based mutual authentication protocol on n address book entries and m contact identifiers, respectively. $|q|$ is the size of group elements, $|sign|$ the size of signatures on encrypted contact identifiers, and l the length of hashes u_i.

Phase	Precomputation		Online	
Computation	Sender S	Receiver R	Sender S	Receiver R
Exp.	$m_S + n_S$	$m_R + n_R$	$3m_S + 3m_R$	$3m_S + 3m_R$
Hash calc.	$m_S + 3n_S$	$m_R + 3n_R$	$2m_S + m_R + 2$	$m_S + 2m_R + 2$
Communication	0		$(3m_S + 3m_R + 2) \cdot \lvert q \rvert + (n_S + n_R) \cdot l + (m_S + m_S) \cdot \lvert sign \rvert$	

4.6 Our PrivateDrop Protocol

In Fig. 4, we show our full PSI-based mutual authentication protocol for AirDrop. Its computation and communication overhead is summarized in Tab. 2. For the computation overhead, we count the required exponentiations and hash operations. We assume that verifying each signature requires one such exponentiation and one hash operation. Obtaining the signature on the values y_i is ignored since the exact overhead depends on the chosen implementation. In case Apple provides keys α_i along with values Y_i and signatures $sign(\sigma_{VR}, Y_i)$, the additional communication overhead in the precomputation phase is only $O(m)$. Otherwise, if the keys are chosen on the client device, a non-interactive zero-knowledge Schnorr proof requires additional computation with $O(m)$ exponentiations and hash operations.

Overhead. Overall, in the precomputation phase, both parties have a computation overhead of $O(m+n)$, which is a one-time cost. In the online phase, the computation overhead is $O(m)$, with $m \ll n$, while the communication overhead is $O(m+n)$. Due to n still being fairly limited in practice (e.g., $n = 10\,\text{k}$) and the availability of a low-latency and high-bandwidth Wi-Fi connection, this communication overhead is very well manageable (cf. our experiments in § 6).

5 Implementation and Integration

We implement both the original AirDrop protocol and our PrivateDrop extension for iOS and macOS to empirically study the overhead caused by PSI. We do not use Apple's closed source AirDrop implementation to provide a fair comparison between non-PSI and PSI. In the following, we discuss our implementation (including mDNS and HTTPS communication) and our integration of PrivateDrop into the original AirDrop protocol stack. Our open-source implementation is available on GitHub [45].

5.1 Implementation of the Base Protocol

Apple does not expose or document a low-level AirDrop API that would allow us to integrate our PrivateDrop extension and conduct a fine-grained performance evaluation. Using an existing open-source implementation of AirDrop [46] is also not an option as it is written in Python, which is not supported on iOS and not optimized for performance.

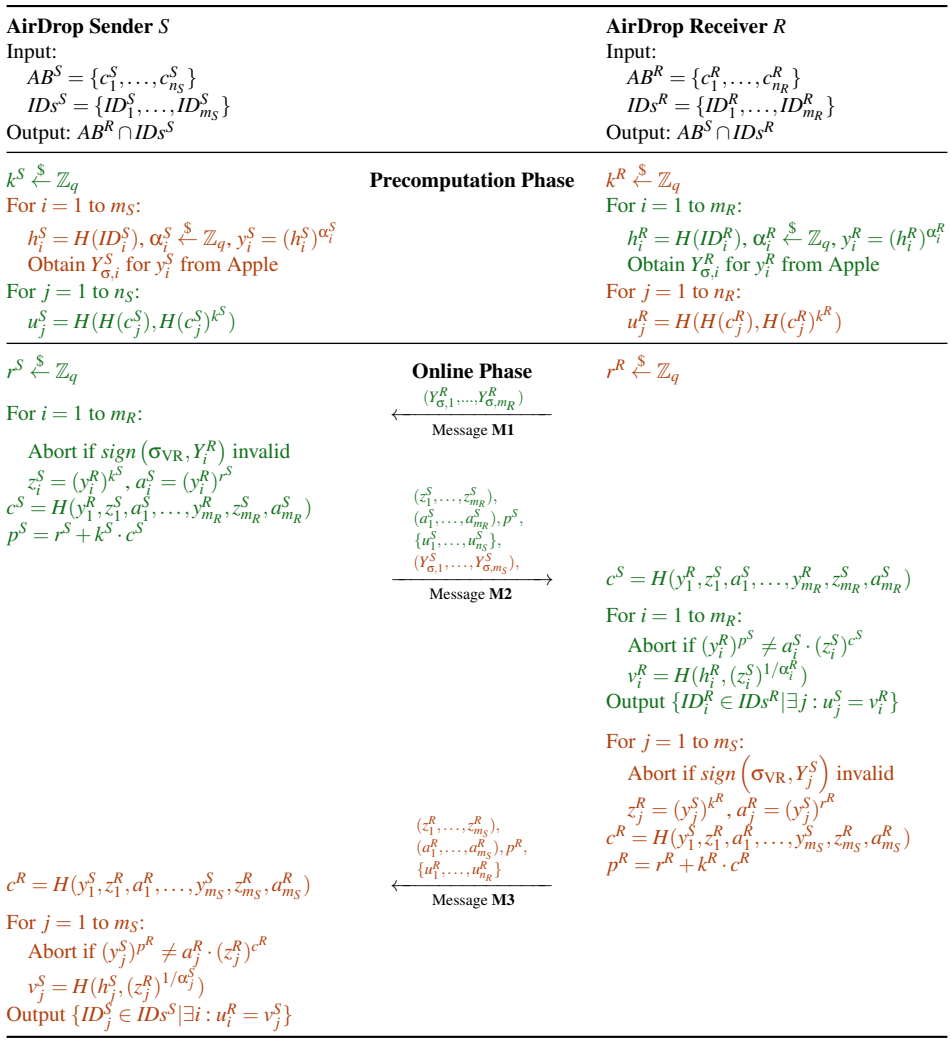

Figure 4: PrivateDrop's full PSI-based mutual authentication protocol for AirDrop. The protocol is based on the optimized and interleaved execution of DO2 (green) and DO3 (orange), cf. Tab. 1 and Fig. 2, divided into a reusable precomputation and an online phase.

Therefore, we re-implement the full AirDrop protocol stack in Swift, Apple's modern programming language that compiles down to assembler code. In particular, we use Apple's public NetService API [8] to announce services via mDNS and bootstrap communication over the AWDL interface. In addition, we use SwiftNIO [9] to achieve high-performance asynchronous network operations and to implement HTTPS communication. In App. C, we show that our AirDrop implementation performs very similar to Apple's.

AirDrop's validation records are implemented using cryptographic message syntax (CMS) [47]. To provide the best integration with Apple's existing certification infrastructure, we also implement the signatures $Y_{\sigma,i}$ in Eq. (6) in CMS. For validation, we use the OpenSSL library [71], as Apple's Security framework provides CMS support only on macOS but not on iOS [6]. The individual validation records $VR_{\sigma,i}$ in Eq. (4) are not part of our implementation.

5.2 Implementation of the PSI Operations

Implementing our PSI protocol requires access to low-level elliptic curve (EC) operations, for which we would have liked to utilize built-in operating system capabilities. Unfortunately, Apple's Swift-based CryptoKit [7] does not expose the required point operations, e.g., addition and scalar multiplication. As an alternative, we use the established open-source library Relic [11]. Compared to other third-party candidates such as MIRACL [68] or libecc [15], Relic is focused on efficiency [73, 81] and portability with support for all relevant architectures, i.e., arm64 (iOS and macOS) and x86_64 (macOS). Also, Relic is written in C, which integrates well with our Swift-based protocol implementation.

We instantiate all primitives to provide a security level of 128 bit. Our Diffie-Hellman-based PSI implementation uses the standardized elliptic curve P-256.

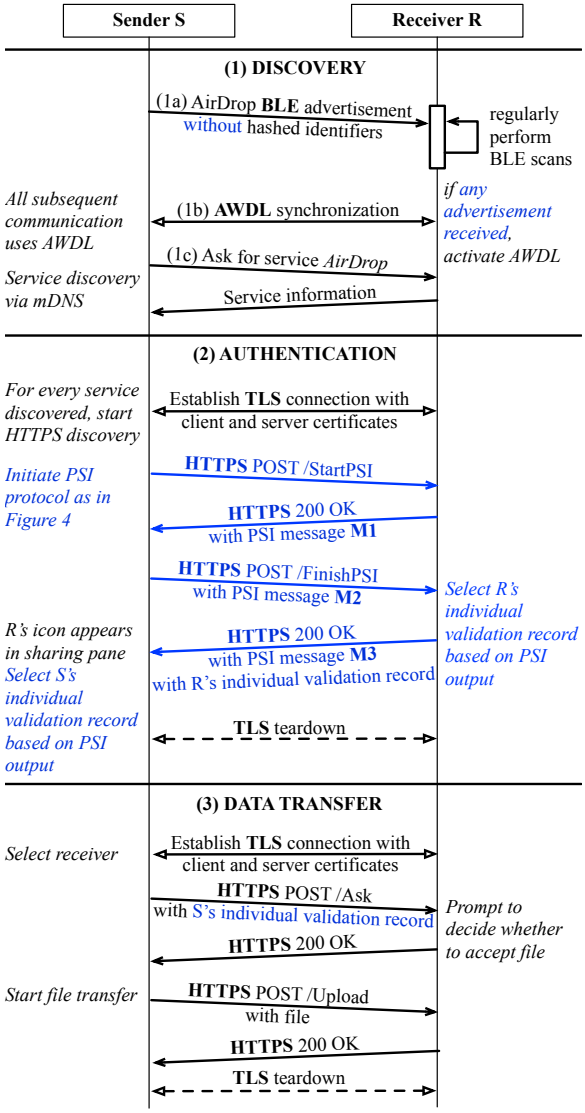

Figure 5: PrivateDrop protocol; changes to the original AirDrop protocol (cf. Fig. 1) highlighted in blue.

5.3 Integration with the HTTPS Handshake

In order to integrate PrivateDrop into AirDrop's HTTPS protocol, we introduce two new HTTPS messages into the authentication phase that we depict in Fig. 5. In particular, we introduce *StartPSI* and *FinishPSI* that include the three messages M1, M2, and M3 from our optimized PSI protocol (cf. Fig. 4) as payload. The protocol is performed immediately after the mDNS discovery is completed and replaces the original HTTPS *Discover* exchange. Since the AirDrop sender acts as the HTTPS client in the protocol, the initial HTTPS request contains no payload and simply signals the receiver to initiate the PSI protocol.

Selecting Individual Validation Records. The output of the PSI protocol determines which individual validation records $VR_{\sigma,i}$ are included in the follow-up requests. If the PSI protocol yields no matches, no validation records are included. If the PSI protocol yields one or more matches, one randomly chosen individual validation record that corresponds to one of the matches is included in the request. Note that, in principle, we could include the validation records for all matches. However, this would yield no benefit as one contact identifier is sufficient to uniquely identify the other party based on the user's address book.[7] On the contrary, transmitting multiple validation records would increase communication overhead and require the receiver to verify multiple signatures.

Communication Rounds. Note that after processing M2, the receiver has already selected the appropriate individual validation record and can send it back to the sender with M3. The sender will include its individual validation record in the *Ask* request when initiating a file transfer. By piggybacking the receiver's validation record to M3, we avoid one additional communication round that would be necessary to exchange $VR_{\sigma,i}$ after the PSI protocol has completed. In total, our PSI-based protocol only incurs one additional communication round compared to the original authentication.

5.4 Integration with the BLE Advertisements

AirDrop's BLE advertisements contain the first two bytes of the sender's hashed contact identifiers, which are also part of the validation record. Receivers use these hashes to check if the sender is a potential contact match and whether they should turn on their AWDL interface to conduct the full authentication handshake. As shown in [92], this mechanism provides no additional security as it can easily be circumvented with brute force. Therefore, the short hashes appear to be an optimization to prevent wakeups of the receiver's Wi-Fi radio that unnecessarily drain the device's battery.

As the purpose of our work is to prevent any leakage of personal information, we propose to not include any (even shortened) contact identifiers and simply set the fields to a fixed value, e.g., `0x0000`. Then, whenever AirDrop receivers overhear such an advertisement, they activate their AWDL interface unconditionally. Coincidentally, this behavior is already implemented by AirDrop receivers that are discoverable by everyone (cf. § 2.2), so we do not expect that this change will incur any practical hurdles.

5.5 Towards Replacing AirDrop

We implemented a fully-functional PrivateDrop prototype. The following changes have to be made by Apple for turning PrivateDrop into a drop-in replacement for AirDrop, which can be deployed with iOS and macOS updates, and

[7] We assume an unambiguous mapping of contact identifiers to contact entries in a user's address book. If a user assigned the same identifier to multiple contacts, then having multiple validation records could help to resolve the ambiguity. In any case, if AirDrop is unable to uniquely identify the other party, it should inform the user, e.g., by displaying an appropriate message. Note that Apple validates ownership of contact identifiers via verification emails or SMS (cf. § 2.1), which prevents multiple registrations, e.g., when users share an office phone number.

Table 3: Experiment parameters.

Protocols	AirDrop, PrivateDrop	
Set sizes	# identifiers m	1, 10, 20
	# address book entries n	100, 1 000, 5 000, 10 000, 15 000
Hardware	Sender (macOS 11) Receiver (iOS 14)	MacBook Pro 15" 2019 iPhone 12 mini
Network connection	Apple Wireless Direct Link (AWDL) [90], USB cable	

requires no hardware modifications: (a) To ensure limited backward compatibility with the orignal AirDrop protocol, PrivateDrop-enabled devices should support AirDrop's *Discover* request but never include AirDrop's validation record to protect themselves against identifier leakage (cf. § 3). PrivateDrop devices would then always appear as non-contacts to AirDrop devices. Note that *downgrade* attacks, i.e., forcing two PrivateDrop devices to use the legacy AirDrop protocol, will hence merely result in unauthenticated connections as PrivateDrop devices will never exchange their validation records with AirDrop devices. (b) Apple's CA infrastructure must be extended to issue $VR_{\sigma,i}$ and $Y_{\sigma,i}$ values. (c) PrivateDrop should use the system's Contact API to provide input for the contact discovery. For evaluation purposes, we use randomly generated contacts. (d) Our implementation currently does not integrate BLE discovery, because iOS hides Apple-specific advertisements in the scan responses and prohibits emitting them for third-party applications. (e) Finally, PrivateDrop currently does not implement individual validation records but uses the AirDrop validation records VR_σ to match the Apple-signed TLS certificates.

6 Experimental Evaluation

We evaluate the performance of PrivateDrop based on our implementation for AirDrop (cf. § 5). To this end, we conduct an extensive experimental evaluation using different Apple devices and variable input sizes over the devices' AWDL interface. We show that the median discovery delay is *well below one second* in any practical setting. In the following, we explain our evaluation metrics and experimental setup. We then present and discuss the evaluation results.

6.1 Evaluation Metrics

We assess the protocol's performance in terms of runtime or *delay*. In particular, we time the protocol flow at several reference points to measure (a) computational overhead, i.e., time spent for calculating cryptographic operations, (b) network overhead, i.e., time spent for transmitting data over the data channel, and (c) overall runtime, i.e., time spent for executing the complete discovery process.

6.2 Experimental Setup

We conduct all experiments using our PrivateDrop and AirDrop implementations (cf. § 5) and summarize all other experiment parameters such as set sizes, hardware, and network environments in Tab. 3.

Set Sizes. Our complexity analysis in § 4.6 shows that the online PSI overhead depends on the number of identifiers m and address book entries n. A previous online study found that Apple users have $n = 136$ contacts on average [92]. Therefore, we select values for n in this order of magnitude but also include values up to $n = 15\,000$ to assess potential scalability limits. Similarly, we select m to cover moderate and extreme limits (1 to 20). For simplicity of presentation, the input sizes are the same for both sender and receiver in all our experiments, i.e., $m = m_S = m_R$ and $n = n_S = n_R$.

Hardware and Network Connection. We use up-to-date Apple devices for the evaluation, in particular, an iPhone 12 mini and a MacBook Pro (2019). A mix of iOS and macOS devices allows us to conduct experiments via a cable network connection (USB) in addition to AWDL, thereby measuring the impact of network-induced delays. In all experiments, the MacBook acts as the sender and the iPhone as the receiver to ensure comparable results.

Environment. We conduct all experiments in a home office environment,[8] where we cannot control interfering Bluetooth and Wi-Fi transmissions. This interference might contribute to the high variance of our AWDL experiments (cf. Fig. 9), which was not observed in previous experiments that used a Faraday tent [90]. We run cable-based experiments to isolate the impact of PrivateDrop, while the AWDL experiments help us to understand performance under real-world conditions.

Test Suite. We implemented a benchmark application for iOS and macOS based on PrivateDrop (cf. § 5) that allows us to define a *scenario*. A scenario is comprised of a fixed set of experimental parameters such as the set sizes and the choice of sender and receiver devices (cf. Tab. 3). For each scenario, we run 100 experiments (Monte Carlo) that each consist of a complete protocol execution. To avoid systematic errors introduced by temporal disturbances, we schedule the individual runs for each scenario in a round-robin fashion. The bar plots indicate the median delay over all runs, and the error bars indicate the 0.05 and 0.95 quantiles. Unless otherwise stated, we measure the delays on the sender side. Each experiment consists of a full protocol run as well as a preparation and cleanup phase: (a) Preparation: we generate the address book at random, precompute the values u_i, and wait until both sender and receiver are ready. (b) Execution: we run a complete protocol execution starting from the DNS-SD discovery to the upload of a file. (c) Cleanup: we shut down the HTTPS and DNS-SD server to close all connections.

[8] Our institution mandated home office due to the COVID-19 pandemic.

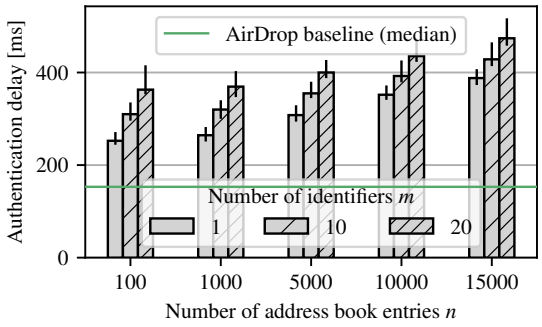

Figure 6: Overall authentication delay for AirDrop (baseline) and PrivateDrop with different set sizes (m,n).

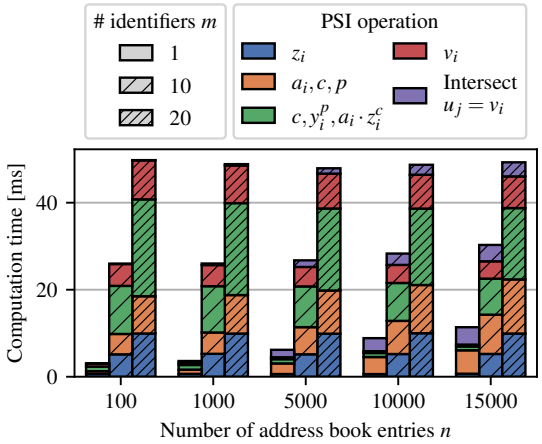

Figure 7: Computation time for the PSI operations on an iPhone 12 with different set sizes (m,n).

6.3 Authentication Delay

We first empirically measure the performance of PrivateDrop's online phase for variable set sizes n and m (cf. Tab. 3). For this, we run a set of experiments between the MacBook Pro 2019 (sender) and iPhone 12 (receiver). In order to minimize noise introduced by the wireless channel, we conduct this experiment via a USB cable connection between sender and receiver. We later evaluate the impact of the wireless channel in § 6.4.

Overall Delay. In Fig. 6, we show the delay of the complete authentication phase (phase (2) in Figs. 1 and 5), for PrivateDrop and AirDrop. AirDrop authentication is independent of m and n, and, therefore, we include the median delay as a baseline. In contrast, the PrivateDrop runtime increases with both m and n as expected. Our results for PrivateDrop show that for moderate settings ($m = 10$, $n = 1\,000$), the median authentication delay is increased by $2\times$ compared to AirDrop. Even for extreme scenarios ($m = 20$, $n = 15\,000$), the overall delay stays below 500 ms. This satisfies our user experience requirement as humans perceive any delay below 1 000 ms as an "immediate response" [22].

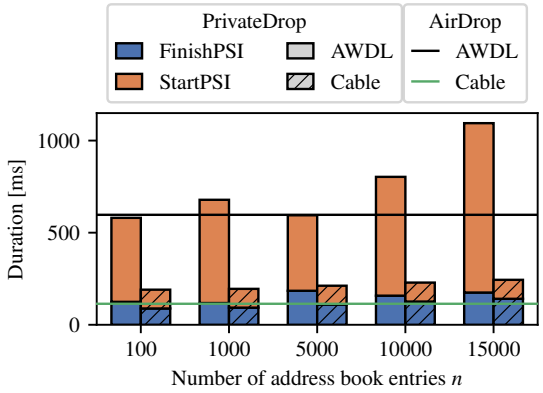

Figure 8: Transmission delay of AWDL and cable connections for the AirDrop (*Discover*) and PrivateDrop (*StartPSI* and *FinishPSI*) requests for a fixed number of identifiers $m = 10$.

PSI Delay. We closer investigate the impact of the PSI online phase on the overall authentication delay. Fig. 7 shows the computation time of the individual operations on an iPhone 12. In fact, only computing the actual intersection depends on the number of address book entries n (cf. violet parts in Fig. 7) and is at most 5 % of the total time for $n = 15\,000$. All other arithmetic operations increase linearly with m, which validates our complexity analysis in § 4.6. In absolute terms, the median computational overhead is less than 12 ms for $m = 1$ and stays below 50 ms for $m = 20$. Note that a complete protocol execution requires identical operations on both sides. To get the total PSI overhead, we can double these numbers if assuming identical hardware for sender and receiver. Still, the PSI operations alone make up less than half of the total authentication delay (cf. Fig. 6). The other major component is networking delay, which we explore next.

6.4 Networking Delay

AirDrop originally uses a wireless connection between sender and receiver. We want to understand the impact of the networking delay and provide a comparison between AWDL and the cable connection (cf. § 6.3). To this end, we repeat the previous experiment over AWDL and measure the transmission delay of the HTTPS requests and replies. In particular, we record timestamps $T_{1..4}$ for each request-response pair, i.e.,

$$T_1 \xrightarrow{\text{Request}} T_2$$
$$T_4 \xleftarrow{\text{Response}} T_3$$

and calculate the delay as $t = T_4 - T_1 - (T_3 - T_2)$ to exclude the receiver-side processing delay. Fig. 8 shows the median transmission delays t incurred by *StartPSI* and *FinishPSI* exchanges for both wireless and cable connections. We add the median transmission delay of AirDrop's *Discover* request for reference. Qualitatively, we can observe that the number of address book entries n has a stronger impact on transmission

delay for AWDL than for the cable connection and that the transmission delay constitutes about *half of the overall authentication delay*. Interestingly, the transmission delay for both PSI requests is similar over the cable, while the first request takes up significantly more time over AWDL. The reason is that the first request includes the time required for connection setup, which generally takes longer over AWDL and has a higher variance, as we discuss next.

High Variance of AWDL Transmission Delays. We noticed a high variance of the transmission delays over AWDL compared to the cable connection (cf. App. A). This effect can be explained by AWDL's channel allocation mechanism. AWDL initially allocates few time slots for transmissions, i.e., little bandwidth is available, and then dynamically allocates more if there is load on the Wi-Fi interface [90]. Thus, initial Wi-Fi transmissions are deferred to the next available time slot, resulting in uncontrollable delays in the order of one second, which is the length of an AWDL period. The increase of available bandwidth over time also explains why the median transmission delay of the first message (*StartPSI*) is significantly larger than the second one (*FinishPSI*).

6.5 Precomputation

While online performance is most crucial for user experience, the precomputation of the encrypted address book entries u_j must also be manageable on mobile devices. Therefore, we evaluate the runtime of calculating the values u_j during the precomputation phase (cf. Fig. 4). As the runtime linearly depends on n (cf. § 4.6), we run a linear regression on the results from an iPhone 12 to approximate the runtime as $n \times 0.331\,\text{ms}$. We provide the raw results in App. B. We see that even for large address books ($n = 10\,\text{k}$), the single-threaded precomputation takes only 3.31 s. To save battery, mobile devices could defer the precomputations to times when they are charging, e.g., overnight.

7 Related Work

We survey closely related works for private mutual authentication, complete our overview of available PSI protocols in addition to our selection process described in § 4.3, review further secure computation techniques, and discuss other privacy leaks in Apple's wireless ecosystem.

7.1 Private Mutual Authentication

The most closely related work to ours is [96]. The authors devise a mutual authentication protocol similar to [3, 4, 54], but geared towards various discovery services, including the contacts-only mode of Apple AirDrop. Utilizing identity-based encryption (IBE) [19], the AirDrop sender distributes encryptions of its identity under a certain "authorization policy". This policy states that only the contacts of this party can decrypt the identity. The authors also implement and benchmark their approach. On a Nexus 5X smartphone, the private authentication takes 360.4 ms.

First of all, the work of [96] mainly targets a different privacy issue in AirDrop, namely the information leakage caused by exchanging the certificates for establishing the TLS connection, which leaks information even to nearby passive adversaries. However, the authors operate under the assumption that these certificates contain the device owner's identity in the clear and are actually used for verifying that sender and receiver are mutual contacts. As recently shown in [92], this is not how AirDrop is currently implemented: the certificates contain only an account-specific UUID while the contact check takes place after the TLS connection is established by exchanging hash values of contact identifiers.

Another conceptual disadvantage of [96] is that Apple, as the IBE root, must provision secret keys to all AirDrop devices, whereas we only require Apple to sign encryptions of hashed contact identifiers where the key can be chosen by the client. Moreover, the system proposed in [96] does not consider subtle issues related to everyday use cases, e.g., how to handle transfers of phone numbers. This would require additional effort to extend the employed IBE scheme with efficient revocation capabilities [18].

In terms of implementation and evaluation, we provide an actual integration into the AirDrop protocol with prototypes on various state-of-the-art Apple devices and demonstrate practical performance under real-world conditions.

7.2 Private Set Intersection

The study of PSI protocols is a very active field of research with various optimizations for different use cases. The "standard" scenario is two-party PSI with balanced input sets and security against semi-honest adversaries, who honestly follow the protocol but try to learn additional information from the transcript. Here, works based on oblivious transfer [60, 76, 79, 80] define the state-of-the-art in terms of concrete performance, while others consider the cost-efficiency in cloud deployment as the most relevant metric [75]. There have been attempts to translate these works to the malicious model [83, 84] with a recent efficiency break-through [74].

PSI was also studied in the multi-party case [43, 50, 61] and extended to generic protocols that can compute an arbitrary symmetric function on top of the intersection [29, 77, 78].

As discussed in § 4.3, most closely related to the problem studied in our work are so-called unbalanced PSI protocols that work particularly well when one of the input sets is much larger than the other [26, 27, 55, 59, 82]. Chen et al. [26, 27] present protocols based on fully homomorphic encryption that are very computation intensive and thus not suitable to be run between two mobile devices. Kiss et al. [59] and Kales et al. [55] optimize protocols based on oblivious pseudorandom function evaluations for the mobile use case, especially so-called mobile contact discovery to privately synchronize address books with user databases in messaging applications. However, these protocols, in the best case, only consider security against malicious PSI receivers but not senders.

There also exist approaches that efficiently outsource PSI computations to a third-party server [1, 2, 56, 57, 99]. However, such protocols are not suitable for our use case since the input parties might be both offline.

Finally, we observe that purely public key-based Diffie-Hellman-style protocols [13, 31, 33, 52, 53, 82], as have been around since the 80's [67, 87], are viable alternatives given the requirements and specified input sizes. Specifically, [31] and [53] are suitable candidates as they are secure against malicious adversaries. We base our work on [53] as it requires fewer exponentiations than the RSA-based protocol of [31] and can be instantiated more efficiently with elliptic curve cryptography. As described in § 4.5, we augment this protocol with signed inputs to prevent certain attacks on the ideal functionality of PSI, similar to the notion of authorized PSI (APSI) [31, 33] and PSI with certified sets [21].

7.3 Secure Computation Protocols

There exist further generic and specialized cryptographic protocols to securely perform the operations necessary for mutual authentication. We efficiently achieve this via PSI in two rounds with $O(m+n)$ complexity (cf. § 4).

Secure two-party computation protocols proposed by Yao [97] and Goldreich, Micali, and Wigderson [40] can obliviously evaluate arbitrary Boolean or arithmetic circuits over private inputs. However, a naive circuit for performing equality tests on m contact identifiers and n address book entries has complexity $O(m \cdot n)$. This complexity can be reduced to be linear with hashing techniques known from so-called circuit-based PSI [29, 76, 77, 78]. Unfortunately, such hashing techniques are incompatible with malicious security [74], which otherwise can be guaranteed with generic approaches [62, 72, 95] at the cost of additional overhead. Furthermore, it is unclear how to efficiently authenticate the contact identifiers used as inputs. There also exist specialized protocols for securely performing comparisons/equality checks (e.g., [30, 64, 98]).

The task of computing the intersection between two sets can be equivalently formulated as the receiver querying/searching the sender's database on its inputs to test for set membership. This can be done while hiding the query and without transferring the entire database via private information retrieval (PIR). While there exists efficient multi-server PIR [28, 36], we consider a two-party setting and hence a single server. State-of-the-art single-server PIR is based on homomorphic encryption [39, 58, 63], which is computationally too demanding for mobile devices. Moreover, PIR does not necessarily protect unrelated database entries, which in our case should remain private. This setting is addressed by works that allow (complex) search queries on encrypted data [38]. Unfortunately, such systems inherently suffer from a certain leakage and have been prone to attacks [17, 23, 70].

7.4 Privacy of Apple's Wireless Ecosystem

AirDrop is part of Apple's larger wireless ecosystem, which has been analyzed for privacy leaks before. AWDL was found to leak personally identifiable information such as the user's real name [92]. Several works [14, 24, 65] have analyzed Apple's Bluetooth implementation and found various ways of tracking devices via static identifiers in Bluetooth advertisements. Finally, [88] discovered that Apple devices can be tracked via identifiers that are randomized asynchronously.

8 Conclusion

In this paper, we solved the problem of privacy-preserving authentication between offline peers, based on the notion of being mutual contacts. We demonstrated the practicability of our approach via a comprehensive experimental performance evaluation, which attests negligible overhead under real-world conditions. We motivated our work with *two* distinct design flaws in AirDrop that allow attackers to learn the phone numbers and email addresses of both sender and receiver devices. However, our proposed protocol can support other applications, even outside of Apple's ecosystem. For example, Google recently launched a similar platform called "Nearby" for Android [41, 86], where device visibility can be restricted to the user's contacts and thus would benefit from our protocol for privacy-preserving authentication.

Our proposed solution PrivateDrop prevents users from disclosing personal information to non-contacts. Still, users remain trackable via their account-specific UUID in the TLS certificate, which gives room for future work. Nevertheless, our results demonstrate that PSI with malicious security is ready for practical deployment, even in offline scenarios between resource-constrained mobile devices. We would be glad to see our open-source implementation being adopted in end-user systems such as AirDrop.

Responsible Disclosure

We informed the Apple Product Security team about our findings (follow-up ID 705937802): We disclosed the sender identifier leakage (cf. § 3.3) in May 2019 and the receiver identifier leakage (cf. § 3.4) as well as our proposed PSI-based protocol (cf. § 4) in October 2020. Apple has not yet commented if they plan to address these AirDrop issues.

Availability

We open-source our PrivateDrop implementation [45] and the code to reproduce our figures [44] as part of the Open Wireless Link project [91].

Acknowledgments

We thank the anonymous reviewers and our shepherd Wouter Lueks for their valuable comments, Benny Pinkas and Gowri R Chandran for insightful discussions, Oliver Schick for help with Relic, and Nanako Honda for explorative work.

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 850990 PSOTI). It was co-funded by the Deutsche Forschungsgemeinschaft (DFG) – SFB 1119 CROSSING/236615297 and GRK 2050 Privacy & Trust/251805230, by the LOEWE initiative (Hesse, Germany) within the emergenCITY center, by the German Federal Ministry of Education and Research and the Hessian State Ministry for Higher Education, Research and the Arts within ATHENE.

References

[1] Aydin Abadi, Sotirios Terzis, and Changyu Dong. "VD-PSI: Verifiable Delegated Private Set Intersection on Outsourced Private Datasets". In: *FC*. Springer, 2016, pp. 149–168.

[2] Aydin Abadi, Sotirios Terzis, Roberto Metere, and Changyu Dong. "Efficient Delegated Private Set Intersection on Outsourced Private Datasets". In: *TDSC* 16.4 (2019), pp. 608–624.

[3] Martín Abadi. "Private Authentication". In: *Privacy Enhancing Technologies*. Springer, 2002, pp. 27–40.

[4] Martín Abadi and Cédric Fournet. "Private Authentication". In: *Theor. Comput. Sci.* 322.3 (2004), pp. 427–476.

[5] Apple Inc. *Apple Reports Record First Quarter Results*. Jan. 28, 2020. URL: https://www.apple.com/newsroom/2020/01/apple-reports-record-first-quarter-results/ (visited on 10/15/2020).

[6] Apple Inc. *Cryptographic Message Syntax Services*. 2020. URL: https://developer.apple.com/documentation/security/cryptographic_message_syntax_services (visited on 10/15/2020).

[7] Apple Inc. *CryptoKit*. 2020. URL: https://developer.apple.com/documentation/cryptokit (visited on 10/15/2020).

[8] Apple Inc. *NetService*. 2020. URL: https://developer.apple.com/documentation/foundation/netservice (visited on 10/15/2020).

[9] Apple Inc. *SwiftNIO*. 2020. URL: https://github.com/apple/swift-nio (visited on 10/15/2020).

[10] Apple Inc. *Use AirDrop on your iPhone, iPad, or iPod touch*. Oct. 2019. URL: https://support.apple.com/en-us/HT204144 (visited on 10/15/2020).

[11] Diego F. Aranha, Conrado P. L. Gouvêa, Tobias Markmann, Riad S. Wahby, and Kevin Liao. *RELIC is an Efficient LIbrary for Cryptography*. URL: https://github.com/relic-toolkit/relic (visited on 10/15/2020).

[12] N. Asokan, Alexandra Dmitrienko, Marcin Nagy, Elena Reshetova, Ahmad-Reza Sadeghi, Thomas Schneider, and Stanislaus Stelle. "CrowdShare: Secure Mobile Resource Sharing". In: *ACNS*. Springer, 2013, pp. 432–440.

[13] Pierre Baldi, Roberta Baronio, Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. "Countering GATTACA: Efficient and Secure Testing of Fully-Sequenced Human Genomes". In: *CCS*. ACM, 2011, pp. 691–702.

[14] Johannes K. Becker, David Li, and David Starobinski. "Tracking Anonymized Bluetooth Devices". In: *PoPETs* 2019.3 (2019), pp. 50–65.

[15] Ryad Benadjila, Arnaud Ebalard, and Jean-Pierre Flori. *libecc Project*. URL: https://github.com/ANSSI-FR/libecc (visited on 10/15/2020).

[16] David Bernhard, Olivier Pereira, and Bogdan Warinschi. "How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios". In: *ASIACRYPT*. Springer, 2012, pp. 626–643.

[17] Laura Blackstone, Seny Kamara, and Tarik Moataz. "Revisiting Leakage Abuse Attacks". In: *NDSS*. Internet Society, 2020.

[18] Alexandra Boldyreva, Vipul Goyal, and Virendra Kumar. "Identity-based Encryption with Efficient Revocation". In: *CCS*. ACM, 2008, pp. 417–426.

[19] Dan Boneh and Matthew K. Franklin. "Identity-Based Encryption from the Weil Pairing". In: *CRYPTO*. Springer, 2001, pp. 213–229.

[20] Elie Bursztein, Mike Hamburg, Jocelyn Lagarenne, and Dan Boneh. "OpenConflict: Preventing Real Time Map Hacks in Online Games". In: *S&P*. IEEE, 2011, pp. 506–520.

[21] Jan Camenisch and Gregory M. Zaverucha. "Private Intersection of Certified Sets". In: *FC*. Springer, 2009, pp. 108–127.

[22] Stuart K. Card, George G. Robertson, and Jock D. Mackinlay. "The Information Visualizer, an Information Workspace". In: *CHI*. ACM, 1991, pp. 181–186.

[23] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. "Leakage-Abuse Attacks Against Searchable Encryption". In: *CCS*. ACM, 2015, pp. 668–679.

[24] Guillaume Celosia and Mathieu Cunche. "Discontinued Privacy: Personal Data Leaks in Apple Bluetooth-Low-Energy Continuity Protocols". In: *PoPETs* 2020.1 (2020), pp. 26–46.

[25] Dmitry Chastuhin. *Apple Bleee: Everyone Knows What Happens on Your iPhone*. July 25, 2019. URL: https://hexway.io/research/apple-bleee/ (visited on 10/15/2020).

[26] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. "Labeled PSI from Fully Homomorphic Encryption with Malicious Security". In: *CCS*. ACM, 2018, pp. 1223–1237.

[27] Hao Chen, Kim Laine, and Peter Rindal. "Fast Private Set Intersection from Homomorphic Encryption". In: *CCS*. ACM, 2017, pp. 1243–1255.

[28] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. "Private Information Retrieval". In: *FOCS*. IEEE, 1995, pp. 41–50.

[29] Michele Ciampi and Claudio Orlandi. "Combining Private Set-Intersection with Secure Two-Party Computation". In: *SCN*. Springer, 2018, pp. 464–482.

[30] Geoffroy Couteau. "New Protocols for Secure Equality Test and Comparison". In: *ACNS*. Springer, 2018, pp. 303–320.

[31] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. "Linear-Complexity Private Set Intersection Protocols Secure in Malicious Model". In: *ASIACRYPT*. Springer, 2010, pp. 213–231.

[32] Emiliano De Cristofaro, Mark Manulis, and Bertram Poettering. "Private Discovery of Common Social Contacts". In: *ACNS*. Springer, 2011, pp. 147–165.

[33] Emiliano De Cristofaro and Gene Tsudik. "Practical Private Set Intersection Protocols with Linear Complexity". In: *FC*. Springer, 2010, pp. 143–159.

[34] Datafinder. *Recover Encrypted Email Addresses*. 2020. URL: https://web.archive.org/web/20191211152224/https://datafinder.com/products/email-recovery (visited on 10/15/2020).

[35] Levent Demir, Amrit Kumar, Mathieu Cunche, and Cédric Lauradoux. "The Pitfalls of Hashing for Privacy". In: *Commun. Surv. Tutorials* 20.1 (2018), pp. 551–565.

[36] Daniel Demmler, Amir Herzberg, and Thomas Schneider. "RAID-PIR: Practical Multi-Server PIR". In: *CCSW*. ACM, 2014, pp. 45–56.

[37] Amos Fiat and Adi Shamir. "How to Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *CRYPTO*. Springer, 1986, pp. 186–194.

[38] Benjamin Fuller, Mayank Varia, Arkady Yerand Emily Shen, Ariel Hamlin, Vijay Gadepally, Richard Shay, John Darby Mitchell, and Robert K. Cunningham. "SoK: Cryptographically Protected Database Search". In: *S&P*. IEEE, 2017, pp. 172–191.

[39] Craig Gentry and Shai Halevi. "Compressible FHE with Applications to PIR". In: *TCC*. Springer, 2019, pp. 438–464.

[40] Oded Goldreich, Silvio Micali, and Avi Wigderson. "How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority". In: *STOC*. ACM, 1987, pp. 218–229.

[41] Google Developers. *Nearby - A platform for discovering and communicating with nearby devices*. URL: https://developers.google.com/nearby (visited on 10/15/2020).

[42] Christoph Hagen, Christian Weinert, Christoph Sendner, Alexandra Dmitrienko, and Thomas Schneider. "All the Numbers are US: Large-scale Abuse of Contact Discovery in Mobile Messengers". In: *NDSS*. Internet Society, 2021.

[43] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. "Scalable Multi-party Private Set-Intersection". In: *PKC*. Springer, 2017, pp. 175–203.

[44] Alexander Heinrich, Matthias Hollick, Thomas Schneider, Milan Stute, and Christian Weinert. *PrivateDrop Evaluation*. URL: https://github.com/seemoo-lab/privatedrop-evaluation.

[45] Alexander Heinrich, Matthias Hollick, Thomas Schneider, Milan Stute, and Christian Weinert. *PrivateDrop Implementation*. URL: https://github.com/seemoo-lab/privatedrop.

[46] Alexander Heinrich and Milan Stute. *OpenDrop: an Open Source AirDrop Implementation*. URL: https://github.com/seemoo-lab/opendrop (visited on 10/15/2020).

[47] Russell Housley. "Cryptographic Message Syntax (CMS)". In: *RFC* 5652 (Sept. 2009). DOI: 10.17487/RFC5652.

[48] Troy Hunt. *Have I Been Pwned*. URL: https://haveibeenpwned.com (visited on 10/15/2020).

[49] Kent Ickler. *Hashcat Benchmarks for Nvidia GTX 1080TI*. June 20, 2017. URL: https://www.blackhillsinfosec.com/hashcat-benchmarks-nvidia-gtx-1080ti-gtx-1070-hashcat-benchmarks/ (visited on 10/15/2020).

[50] Roi Inbar, Eran Omri, and Benny Pinkas. "Efficient Scalable Multiparty Private Set-Intersection via Garbled Bloom Filters". In: *SCN*. Springer, 2018, pp. 235–252.

[51] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Mariana Raykova, Shobhit Saxena, Karn Seth, David Shanahan, and Moti Yung. "On Deploying Secure Computing: Private Intersection-Sum-with-Cardinality Protocols". In: *EuroS&P*. IEEE, 2020.

[52] Stanislaw Jarecki and Xiaomin Liu. "Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection". In: *TCC*. Springer, 2009, pp. 577–594.

[53] Stanislaw Jarecki and Xiaomin Liu. "Fast Secure Computation of Set Intersection". In: *SCN*. Springer, 2010, pp. 418–435.

[54] Stanislaw Jarecki and Xiaomin Liu. "Private Mutual Authentication and Conditional Oblivious Transfer". In: *CRYPTO*. Springer, 2009, pp. 90–107.

[55] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. "Mobile Private Contact Discovery at Scale". In: *USENIX Security*. USENIX Association, 2019, pp. 1447–1464.

[56] Seny Kamara, Payman Mohassel, Mariana Raykova, and Seyed Saeed Sadeghian. "Scaling Private Set Intersection to Billion-Element Sets". In: *FC*. Springer, 2014, pp. 195–215.

[57] Florian Kerschbaum. "Outsourced Private Set Intersection Using Homomorphic Encryption". In: *AsiaCCS*. ACM, 2012, pp. 85–86.

[58] Aggelos Kiayias, Nikos Leonardos, Helger Lipmaa, Kateryna Pavlyk, and Qiang Tang. "Optimal Rate Private Information Retrieval from Homomorphic Encryption". In: *PoPETs* 2015.2 (2015), pp. 222–243.

[59] Ágnes Kiss, Jian Liu, Thomas Schneider, N. Asokan, and Benny Pinkas. "Private Set Intersection for Unequal Set Sizes with Mobile Applications". In: *PoPETs* 2017.4 (2017), pp. 177–197.

[60] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. "Efficient Batched Oblivious PRF with Applications to Private Set Intersection". In: *CCS*. ACM, 2016, pp. 818–829.

[61] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. "Practical Multi-party Private Set Intersection from Symmetric-Key Techniques". In: *CCS*. ACM, 2017, pp. 1257–1272.

[62] Yehuda Lindell. "Fast Cut-and-Choose Based Protocols for Malicious and Covert Adversaries". In: *CRYPTO*. Springer, 2013, pp. 1–17.

[63] Helger Lipmaa and Kateryna Pavlyk. "A Simpler Rate-Optimal CPIR Protocol". In: *FC*. Springer, 2017, pp. 621–638.

[64] Helger Lipmaa and Tomas Toft. "Secure Equality and Greater-Than Tests with Sublinear Online Complexity". In: *ICALP*. Springer, 2013, pp. 645–656.

[65] Jeremy Martin, Douglas Alpuche, Kristina Bodeman, Lamont Brown, Ellis Fenske, Lucas Foppe, Travis Mayberry, Erik C. Rye, Brandon Sipes, and Sam Teplov. "Handoff All Your Privacy - A Review of Apple's Bluetooth Low Energy Continuity Protocol". In: *PoPETs* 2019.4 (2019), pp. 34–53.

[66] Matthias Marx, Ephraim Zimmer, Tobias Mueller, Maximilian Blochberger, and Hannes Federrath. "Hashing of Personally Identifiable Information is not Sufficient". In: *Sicherheit*. Vol. P-281. LNI. GI e.V., 2018, pp. 55–68.

[67] Catherine A. Meadows. "A More Efficient Cryptographic Matchmaking Protocol for Use in the Absence of a Continuously Available Third Party". In: *S&P*. IEEE, 1986, pp. 134–137.

[68] MIRACL UK Ltd. *MIRACL – Multiprecision Integer and Rational Arithmetic Cryptographic Library*. URL: https://github.com/miracl/MIRACL (visited on 10/15/2020).

[69] National Institute of Standards and Technology. *Secure Hash Standard (SHS)*. Tech. rep. Aug. 2015.

[70] Muhammad Naveed, Seny Kamara, and Charles V. Wright. "Inference Attacks on Property-Preserving Encrypted Databases". In: *CCS*. ACM, 2015, pp. 644–655.

[71] OpenSSL Software Foundation. *OpenSSL: Cryptography and SSL/TLS Toolkit*. URL: https://www.openssl.org (visited on 10/15/2020).

[72] Emmanuela Orsini, Nigel P. Smart, and Frederik Vercauteren. "Overdrive2k: Efficient Secure MPC over $\mathbb{Z}_{2^k}$ from Somewhat Homomorphic Encryption". In: *CT-RSA*. Springer, 2020, pp. 254–283.

[73] Daniel Pigatto, Natassya Silva, and Kalinka Castelo Branco. "Performance Evaluation and Comparison of Algorithms for Elliptic Curve Cryptography with El-Gamal based on MIRACL and RELIC Libraries". In: *Journal of Applied Computing Research* 112 (2011).

[74] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. "PSI from PaXoS: Fast, Malicious Private Set Intersection". In: *EUROCRYPT*. Springer, 2020, pp. 739–767.

[75] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. "SpOT-Light: Lightweight Private Set Intersection from Sparse OT Extension". In: *CRYPTO*. Springer, 2019, pp. 401–431.

[76] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. "Phasing: Private Set Intersection Using Permutation-based Hashing". In: *USENIX Security*. USENIX Association, 2015, pp. 515–530.

[77] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. "Efficient Circuit-Based PSI with Linear Communication". In: *EUROCRYPT*. Springer, 2019, pp. 122–153.

[78] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. "Efficient Circuit-Based PSI via Cuckoo Hashing". In: *EUROCRYPT*. Springer, 2018, pp. 125–157.

[79] Benny Pinkas, Thomas Schneider, and Michael Zohner. "Faster Private Set Intersection Based on OT Extension". In: *USENIX Security*. USENIX Association, 2014, pp. 797–812.

[80] Benny Pinkas, Thomas Schneider, and Michael Zohner. "Scalable Private Set Intersection Based on OT Extension". In: *TOPS* 21.2 (2018), 7:1–7:35.

[81] Lucian Popa, Bogdan Groza, and Pal-Stefan Murvay. "Performance Evaluation of Elliptic Curve Libraries on Automotive-Grade Microcontrollers". In: *ARES*. ACM, 2019, 100:1–100:7.

[82] Amanda C. Davi Resende and Diego F. Aranha. "Faster Unbalanced Private Set Intersection". In: *FC*. Springer, 2018, pp. 203–221.

[83] Peter Rindal and Mike Rosulek. "Improved Private Set Intersection Against Malicious Adversaries". In: *EUROCRYPT*. Springer, 2017, pp. 235–259.

[84] Peter Rindal and Mike Rosulek. "Malicious-Secure Private Set Intersection via Dual Execution". In: *CCS*. ACM, 2017, pp. 1229–1242.

[85] Claus-Peter Schnorr. "Efficient Identification and Signatures for Smart Cards". In: *CRYPTO*. Springer, 1989, pp. 239–252.

[86] Daniel Marcos Schwaycer. *Instantly share files with people around you with Nearby Share*. Aug. 2020. URL: https://blog.google/products/android/nearby-share/ (visited on 10/15/2020).

[87] Adi Shamir. "On the Power of Commutativity in Cryptography". In: *ICALP*. Springer, 1980, pp. 582–595.

[88] Milan Stute, Alexander Heinrich, Jannik Lorenz, and Matthias Hollick. "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". In: *USENIX Security*. To appear. USENIX Association, 2021.

[89] Milan Stute, David Kreitschmann, and Matthias Hollick. "Linux Goes Apple Picking: Cross-Platform Ad hoc Communication with Apple Wireless Direct Link". In: *MobiCom*. ACM, 2018, pp. 820–822.

[90] Milan Stute, David Kreitschmann, and Matthias Hollick. "One Billion Apples' Secret Sauce: Recipe for the *Apple Wireless Direct Link* Ad hoc Protocol". In: *MobiCom*. ACM, 2018, pp. 529–543.

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

[92] 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*. USENIX Association, 2019, pp. 37–54.

[93] Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan Boneh, and Elie Bursztein. "Protecting accounts from credential stuffing with password breach alerting". In: *USENIX Security*. USENIX Association, 2019, pp. 1556–1571.

[94] Ni Trieu, Kareem Shehata, Prateek Saxena, Reza Shokri, and Dawn Song. "Epione: Lightweight Contact Tracing with Strong Privacy". In: *Data Eng. Bull.* 43.2 (2020), pp. 95–107.

[95] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. "Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation". In: *CCS*. ACM, 2017, pp. 21–37.

[96] David J. Wu, Ankur Taly, Asim Shankar, and Dan Boneh. "Privacy, Discovery, and Authentication for the Internet of Things". In: *ESORICS*. Springer, 2016, pp. 301–319.

[97] Andrew Chi-Chih Yao. "How to Generate and Exchange Secrets". In: *FOCS*. IEEE, 1986, pp. 162–167.

[98] Ching-Hua Yu and Bo-Yin Yang. "Probabilistically Correct Secure Arithmetic Computation for Modular Conversion, Zero Test, Comparison, MOD and Exponentiation". In: *SCN*. Springer, 2012, pp. 426–444.

[99] Qingji Zheng and Shouhuai Xu. "Verifiable Delegated Set Intersection Operations on Outsourced Encrypted Data". In: *IC2E*. IEEE, 2015, pp. 175–184.

A Authentication Delay over AWDL

Fig. 9 shows the high variance of the authentication delay of PrivateDrop over the AWDL interface. The lower and upper error bars indicate the 0.05 and 0.95 quantiles, respectively. Still, the median authentication delay for PrivateDrop lies within 500 ms and 1 500 ms, depending on (m,n).

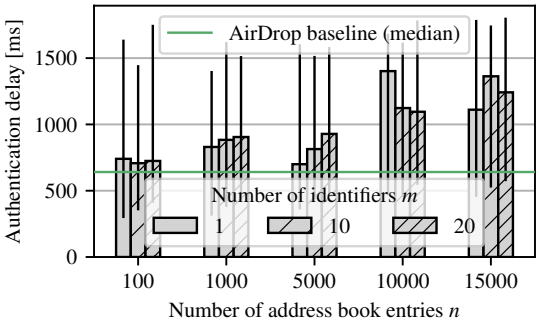

Figure 9: Overall authentication delay for AirDrop (baseline) and PrivateDrop with different set sizes (m,n) (MacBook Pro 2019 → iPhone 12 via AWDL).

B PSI Precomputation

Fig. 10 shows the runtime of the PSI precomputation required for calculating u_i (cf. precomputation phase in Fig. 4) on an iPhone 12. Even with a large address book ($n = 15\,000$), the computation time does not exceed 5 s, which is very manageable for a mobile device that charges overnight.

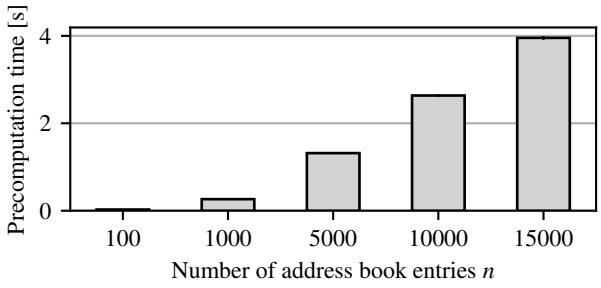

Figure 10: Runtime of PSI precomputation on an iPhone 12.

C Performance Comparison with Apple's AirDrop Implementation

We benchmark our base AirDrop implementation against Apple's original one. To evaluate Apple's implementation, we leverage the system logging facility of macOS (cf. [88]) that produces debug output for AirDrop and provides logs verbose enough to distinguish the authentication phase. We calculate the authentication delay as the timestamp difference of the entries indicating the start and end of the authentication phase. We provide the details in our evaluation repository [44]. We use the same hardware configuration and environment as described in § 6.2. We open the sharing pane on the MacBook Pro and manually wake up the iPhone 12 by tapping on the screen. We repeat this process 100 times and report on the results in Fig. 11 as an empirical cumulative distribution function. The results show that the best-case performance of our implementation is similar to the original one. The high variance of the delay can be attributed to the initialization behavior of AWDL (cf. § 6.4).

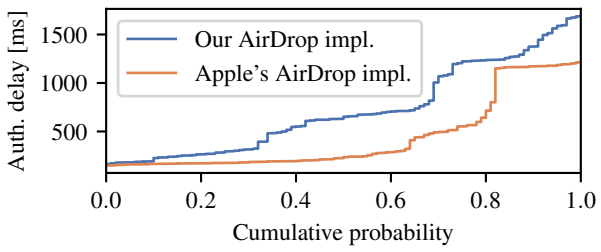

Figure 11: Authentication delay of our AirDrop implementation and Apple's (MacBook Pro 2019 → iPhone 12 via AWDL).