MASHaBLE: Mobile Applications of Secret Handshakes over Bluetooth LE

Yan Michalevsky
Stanford University
yanm2@cs.stanford.edu

Suman Nath
Microsoft Research
suman.nath@microsoft.com

Jie Liu
Microsoft Research
jie.liu@microsoft.com

ABSTRACT

We present new applications for cryptographic secret handshakes between mobile devices on top of Bluetooth Low-Energy (LE). Secret handshakes enable mutual authentication, with the property that the parties learn nothing about each other unless they have been both issued credentials by a group administrator. This property provides strong privacy guarantees that enable interesting applications. One of them is proximity-based discovery for private communities. We introduce MASHaBLE, a mobile application that enables participants to discover and interact with nearby users if and only if they belong to the same secret community. We use direct peer-to-peer communication over Bluetooth LE, rather than relying on a central server. We discuss the specifics of implementing secret handshakes over Bluetooth LE and present our prototype implementation.

CCS Concepts

•**Security and privacy** → **Mobile and wireless security;** *Privacy protections;*

Keywords

Mobile security; Bluetooth LE; IoT; secret handshakes; mutual authentication.

1. INTRODUCTION

Recent revelations about the extent of surveillance, as well as rising awareness regarding digital privacy, have increased the demand for means to communicate privately and prevent disclosure of user data. Many mobile applications geared to this purpose have been released in the past two years, including Telegram [9], Signal Private Messenger (by Open Whisper Systems) [7], Yik-Yak [10], After School [1], and LegaTalk [5]. They aim to provide different privacy guarantees against different adversaries.

Telegram and Signal rely on cryptographic methods for establishing a shared secret key between two parties. The shared key is used for end-to-end encryption of communication between two users. While messages can be routed through a central server, no one can decipher the content of the messages. However, the origin and destination (which reveal the communicating parties) are known to the server, and by extension, to any adversary that can obtain control of it. Yik-Yak and similar apps provide anonymity, to some extent, as well as message dissemination to nearby peers. The messages are broadcast through the server to unknown users, and are therefore not end-to-end encrypted. Therefore, an adversary that has access to the server can obtain the content, and also de-anonymize the origin.

These applications have a high demand: as of July 2016, Telegram messenger for Android has between 100 to 500 million downloads on the Google Play store. Yik-Yak has between 1 and 5 million downloads, and the After School app has hundreds of thousands of downloads. These applications' iOS versions are also popular, judging by thousands of reviews in the Apple Store. This confirms a substantial and growing interest in private/anonymous messaging, and forming anonymous communities.

Inspired by Yik-Yak and similar applications attempting to provide anonymity and location-based message propagation, we propose a mobile application that enables the creation of secret virtual communities, protects geo-location privacy and provides anonymity.

More specifically, our application enables mobile users to enroll in a community and obtain credentials by which they can present themselves to other members of the secret community. These credentials allow them to perform a mutual authentication procedure called *secret handshake* (SH), which is reminiscent in its properties to a secret handshake in the physical world.

For instance, imagine a fraternity where membership is kept secret. Its members want to identify fellow members as they walk around campus, and to communicate privately with nearby members without being exposed. In a setting where one party is willing to disclose its identity first, it could use a signed certificate to authenticate itself. Unfortunately, in our setting, when two fraternity members encounter each other, neither is willing to provide a traditional signed certificate, as it may expose them to a non-member.

One possible solution is for each member to have a mobile application that reports GPS coordinates to a server maintained by the fraternity. All communication with the server is encrypted. The server identifies two users in proximity of one another and notifies them. In addition, the two users use the server to exchange messages.

MobiCom'16, October 03 - 07, 2016, New York City, NY, USA

ISBN 978-1-4503-4226-1/16/10…$15.00

DOI: http://dx.doi.org/10.1145/2973750.2973778

The above approach has several disadvantages. First, anyone who can eavesdrop on the mobile communication and is able to identify the source and origin learns the affiliation of the user from the fact that it sends encrypted messages to the organizational server. This could be prevented by anonymity solutions such as ToR. However, unless there is a large number of users, a powerful adversary can mount timing attacks.

Second, interaction involving a central server assumes having internet connectivity - an assumption that is often problematic for mobile devices. It also complicates the setup and requires securing the server against attackers. In addition, it requires entrusting location information and message content to a third party.

Finally, there is the power efficiency consideration. Using GPS for high-precision localization is energy consuming, as is using cellular communication for interacting with the server.

Our proposal aims to eliminate unnecessary interaction with a third party, leveraging physical proximity and minimizing energy consumption, while providing strong privacy and anonymity guarantees. To achieve this we use cryptographic secret handshakes over Bluetooth Low Energy (BLE) protocol.

Modern smartphone devices are equipped with BLE transceivers that enable transmission and scanning with very low power cost. These transceivers implement the Bluetooth Smart standard and enable a device to constantly transmit short messages, known as advertising, or scan for such advertisements from other devices. One notable example of using BLE advertisements is iBeacon, standardized and promoted by Apple [11].

Our basic idea is that each mobile device, affiliated with the secret community in our example, constantly transmits and scans for advertisements. The advertisements seem completely random to any eavesdropper whereas, in fact, they are attempts to initiate secret handshakes with nearby devices. Nowadays, we are already surrounded by many devices that use BLE advertisements. It is therefore possible to go unnoticed as yet another device.

There are more interesting applications to secret handshakes by mobile devices over BLE. One of them is headcounting of attendants to a secret event. Some commercial products, like Doubledutch.me [4], support headcounting by putting a BLE beacon transmitter at a venue that asks nearby mobile users to confirm attendance. However, if we want the event to be private, we need to prevent users that are not affiliated with the event from identifying that beacon. We also want to protect the privacy of the attendants by making sure their reply to the broadcast cannot identify them as attending this event. Once again we can use a secret handshake between a device transmitting the event-advertising beacon and nearby mobile users to perform mutual authentication and enable headcounting, attendance confirmation or any other private data exchange related to the event.

Another application is BLE-based car locking and unlocking using a mobile device. Using a BLE beacon, an owner can unlock the car when in proximity to it. In addition, the car can transmit a beacon when parked, to make it easier to find it. The car can also lock automatically once it stops receiving a beacon from the mobile device, making sure the car is locked once the driver walks away. *Complete Keyless* [2] and *Connect2Car* [3] are two examples of vendors addressing this need. However, if the beacons can be associated with the vehicle or the mobile device, they enable tracking. An unlinkable secret handshake[1] between the two eliminates this concern.

These examples make a strong case for enabling secret handshakes over BLE on modern smartphones and mobile platforms. BLE opens a new venue of possible private communication and IoT applications. However, it also comes with certain constraints, most importantly the limited amount of data that can be exchanged over BLE packets. We note that, for some applications of interest, exchanging small amounts of data after the handshake is useful, and for others, like headcounting or car unlocking, the success of a handshake protocol itself may be enough, without any further data transmission.

This work aims to demonstrate the practicality of frequently performing secret handshakes between mobile devices, using BLE technology. By implementing a library for secret handshakes we hope to open the door for building various interesting applications. We determined a suitable cryptographic scheme for implementing secret handshakes under the constraints imposed by BLE. We also address the issue of tracking mobile devices that transmit the same or linkable beacons over time by discussing ways to support *unlinkability*, based on existing cryptographic techniques and on features supported by BLE.

As a proof of concept we developed MASHaBLE, a mobile application for performing secret handshakes between Windows Phone devices. As part of its development we also implemented a .NET wrapper for pairing-based cryptography. This is a useful tool for any pairing-based crypto applications for the Windows Universal platform, including mobile and desktop.

While we design a protocol suitable for BLE, and implement it, we claim no novelty in the underlying cryptography. We base our protocol on proven cryptographic constructions.

To summarize our main contributions:

- We propose novel applications for mobile devices supporting BLE, based on a scheme for cryptographic secret handshakes that has previously been mostly theoretic.
- We provide technical details of and identify constraints imposed by the BLE stack, and pick a suitable SH scheme. We also identify computational optimizations applicable to our setting.
- We provide a prototype implementation of the secret handshake scheme for Windows Phone devices. We make our implementation available as an open-source library that can facilitate development of additional pairing-based cryptographic applications for Windows Universal framework, including desktop and Windows phone devices.
- Evaluate our implementation and show that it is practical for use on mobile devices in terms of battery consumption and performance.

In the following section we provide some necessary technical background. We give an introduction to BLE and

[1] We explain unlinkability in 2.2.3.

explain cryptographic secret handshakes constructed from pairing-based key agreements. We then discuss the specifics of implementing the cryptographic scheme over BLE and our prototype implementation for Windows Phone. Finally, we discuss related work and possible future directions for research.

2. BACKGROUND

2.1 Bluetooth Low Energy

Bluetooth Low Energy (BLE) was introduced as part of the Bluetooth 4.0 specification. This radio standard was designed to minimize power consumption [44] and be suitable for low cost and low bandwidth transmitters. BLE devices can operate for extended periods of time, powered by a tiny energy source such as a coin cell. BLE chips are nowadays present in most modern smartphones, such as Windows phone, Google Nexus, and iPhone. It has been widely adopted by smartphone manufacturers.

Bluetooth 4.1 is an update to the specification and is the current reference for developing BLE applications. It is important to note that BLE is not directly compatible with classic Bluetooth and these are, despite the similarities, two different protocols.

Low power consumption by design comes with a cost. The modulation frequency of 1 Mbps [44] imposes an upper limit on throughput. The specification states minimal and maximal intervals of 7.5 ms and 4 s respectively, before entering an idle state after a connection (to save power). This sets up an upper limit for the number of connections per second. Some implementations, for instance nRF51822 by Nordic Semiconductor, impose additional restrictions: limited number of packets can be transmitted per each connection (6 in the case of nRF51822), and each packet contains only a small number of bytes (20 bytes for nRF51822). The latter is an important constraint that influenced our choice of secret-handshake scheme, among the possible variants.

BLE operates well for short ranges. While it's possible to configure devices to operate on a range of tens of meters (line-of-sight), the typical operation range would be several meters.

All of these are important considerations to take into account when designing a protocol to be executed over BLE.

Two modes of communication are available: *broadcast* and *connected* modes. Broadcast mode enables a device to send data to any other device listening for transmissions. As the name suggests, this is the way to transmit to multiple devices at once. Sending broadcasts is called *advertising* and the broadcast packets *advertisements*. A device that listens to advertisements is called a *scanner*. In our context, using this mode enables attempting a handshake with all scanners in proximity of the advertiser. Figure 1 depicts the structure of a BLE advertisement packet.

If two devices need to exchange data they can use the connected mode. The broadcast indicates that the device can be connected to, and the scanner can initiate a connection following a received broadcast. Once a connection is established the devices can exchange data.

On Windows Phone, it is possible to do both advertising and scanning at the same time. We make use of this property to implement our message exchange.

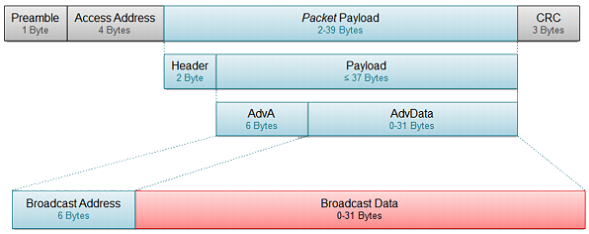


Figure 1: Structure of a BLE advertisement packet. Total size of an advertisement packet is 47 bytes.

2.1.1 Tracking prevention

Bluetooth packets include *Bluetooth device address* fields, similar to an Ethernet MAC address. It is a 6 byte number that uniquely identifies a device. Two types of addresses exist. The *public device address* is fixed for a device. A *random device address*, on the other hand, can be set dynamically and change across different connections.

If the public device address is used as the source Bluetooth device address, it would be possible to track the device around a certain area by scanning for BLE packets and examining the source address field.

In order to prevent tracking of a certain device by its MAC address, BLE supports randomized MAC addresses. The MAC address in the transmitted packets changes over time.

2.2 Cryptographic secret handshakes

Cryptographic secret handshakes, introduced by Balfanz et al. [13], are related to the broader area of Automatic Trust Negotiation (ATN). Secret handshakes enable two parties to establish that they are affiliated with a certain group, and to disclose their respective roles in the group. They guarantee that this information is disclosed only when the handshake succeeds, and no information is obtained by either party when the handshake fails. As such, it is useful when none of the parties are willing to be the first to reveal its affiliation. In this sense, the scheme offers properties similar to a physical secret handshake between two people. However, there is one caveat. One party learns about the success of the handshake before the other, and if it chooses to terminate the communication without responding to the other party's challenge, the other party is left uncertain as to the success of the secret handshake.

2.2.1 Security

The secret handshake has to be secure against the following attacks:

1. Group member impersonation - an adversary who is unaffiliated with the group performs a successful handshake with a group member.
2. Group member detection - an adversary who intercepts a broadcast from a group member identifies the group it is associated with.
3. Tracking - an adversary is able to tell that two different handshake attempts were made by the same party. Resiliency to this attack is called *unlinkability*.

Formal definitions of the security games corresponding to the above properties can be found in [13, 19, 12].

2.2.2 A concrete SH scheme

The scheme proposed in [13] takes a minimum of 3 communication rounds. Two messages are exchanged to establish a shared secret key, after which another is exchanged in order to enable both parties to confirm that they indeed obtained the same shared key. The scheme is based on pairings over elliptic-curves (EC) and its security relies on the Bilinear Diffie-Hellman assumption (BDH). First, let us briefly explain the notion of pairings.
A pairing [21] e is a bilinear map $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$ such that

$$e\left(a \cdot u, b \cdot v\right) = e\left(u, v\right)^{ab}$$

where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t$ are cyclic groups, $u \in \mathbb{G}_1, b \in \mathbb{G}_2$, $a, b \in \mathbb{Z}_n$ and $(\cdot)$ is a group operation, e.g. multiplication over an elliptic-curve. In case $\mathbb{G}_1$ and $\mathbb{G}_2$ are the same group, the pairing is called symmetric. Symmetric pairings are, in fact, slower to compute and we therefore recommend using asymmetric pairings in the implementation. The most known examples are the Weil and Tate pairings, which can be efficiently computed using Miller's algorithm [39].

Using pairings for cryptography has resulted in applications such as identity-based and attribute-based encryption [17], 3-party Diffie-Hellman key exchange [31], BLS short signatures [18], secret handshakes and more.

While other options exist, we use a secret handshake constructed from pairing-based key agreements due to Balfanz et al. [13]. The scheme is simple and elegant, and the exchanged credentials are short, making it a suitable candidate for integration with BLE (on which we elaborate in section 3).

Let us sketch the protocol for performing a secret handshake between two parties. We use a hash-function H modeled as a random oracle that maps arbitrary strings to elliptic-curve points. $\mathbb{Z}_q$ denotes an integer group of large prime order q, where q is 512 bits long.

Two parties, *Alice* and *Bob*, want to perform a handshake. First, there is a setup stage involving interaction of both parties with a credential-authority (CA) service, as depicted in figure 2a.

The CA possesses a master secret key $t \in \mathbb{Z}_q$, called "the group secret". It issues credentials to *Alice* in the form of a pseudonym P_A and a secret elliptic-curve point T_A, i.e. (P_A, T_A) where $T_A = t \cdot H\left(P_A\right)$. Similarly, *Bob* obtains (P_B, T_B) where $T_B = t \cdot H\left(P_B\right)$. The two parties exchange their pseudonyms without disclosing their secret points.
Alice generates a session key by computing

$$\begin{aligned} K_A &= e\left(H\left(P_B\right), T_A\right) = e\left(H\left(P_B\right), t \cdot H\left(P_A\right)\right) \\ &= e\left(H\left(P_B\right), H\left(P_A\right)\right)^t \end{aligned} \tag{1}$$

where e is a bilinear map, i.e. a pairing operation, and *Bob* generates a session key by computing

$$\begin{aligned} K_B &= e\left(T_B, H\left(P_A\right)\right) = e\left(t \cdot H\left(P_B\right), H\left(P_A\right)\right) \\ &= e\left(H\left(P_B\right), H\left(P_A\right)\right)^t \end{aligned} \tag{2}$$

Due to the properties of pairings the two keys are the same, i.e. $K_A = K_B$. By sending a challenge value and getting a response for it, the two can verify that the handshake succeeded. Alternatively, one party can send to the other some content, which will be successfully decrypted if and only if the handshake succeeded. These steps are illustrated in figure 2b.

It is important to include a "direction" bit with the responses to prevent reflection attacks on the challenge–response protocol. A concrete example can be found in [13] (section 4.2).

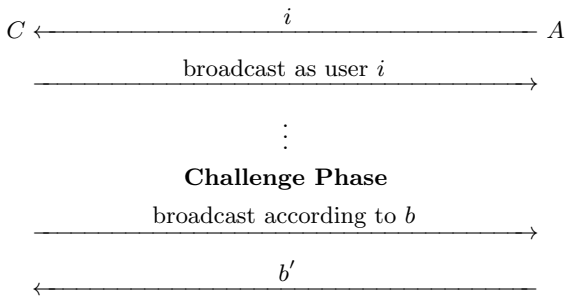
Note that since this construction involves hashing an arbitrary string to an element of $\mathbb{G}_2$, it is important to choose a type of pairing that indeed supports this kind of hashing. Galbraith et al. [23] present a taxonomy of pairings and their respective properties. According to the properties of the different types of pairings, we need to use Type 1 or Type 3 pairings to enable hashing arbitrary strings onto elliptic-curve group elements.

Security of this secret handshake scheme against group member detection and group member impersonation is based on the assumption that Bilinear Diffie-Hellman problem is hard for all probabilistic polynomial-time algorithms. Detailed theorems and proofs of security can be found in section 5.2 of [13]. Security of our protocol implementation against those attacks stems from the security of this underlying cryptographic scheme.

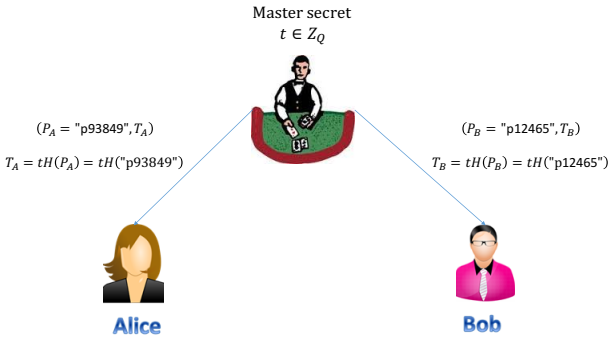
2.2.3 Unlinkable secret handshakes

One problem with the previous scheme is that a powerful eavesdropper who monitors all communication in various locations would be able to track the users. While the eavesdropper doesn't know whether the handshake succeeded or not, it sees the same pseudonym over and over again. We would therefore like to have an untraceable, or unlinkable secret handshake scheme, one that guarantees that an eavesdropper cannot draw a link between two broadcasts, correctly associating them with the same identity. Providing unlinkability addresses the tracking attack, mentioned in section 2.2.1, on the application level, whereas randomized MAC addresses (section 2.1.1) prevent tracking on the datalink level. A combination of both measures has to be used to prevent tracking.
Due to the importance of this property we include a detailed discussion of unlinkability for secret handshakes.

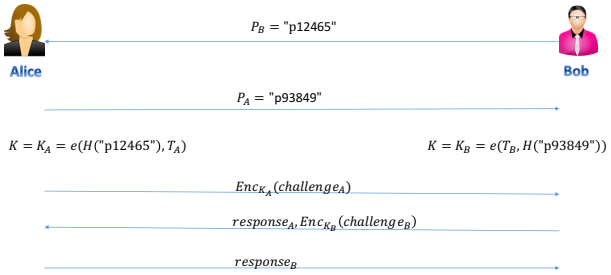
To provide a formal definition of unlinkability, we define the following security game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.



The adversary is allowed to perform a polynomial number of broadcast queries to the challenger. It specifies the identity of the user i for which it requests a broadcast. For each query, $\mathcal{C}$ obtains credentials as described in the secret handshake scheme and simulates a new user in the system by sending over a valid advertisement to $\mathcal{A}$. Finally, $\mathcal{A}$ performs another query. The challenger picks at random a bit $b \in \{0, 1\}$. If $b = 0$, $\mathcal{C}$ responds with a valid broadcast for one of the previously used identities, and if $b = 1$, $\mathcal{C}$ responds



(a) Setup: CA issues credentials to both parties. T_A and T_B are secret and each party keeps it to itself. Actual pseudonym values were added for illustration.



(b) Handshake. Note that we have redundant rounds here: the challenge message could be sent earlier together with the pseudonym. We present it as separate messages for clarity.

Figure 2: Secret handshake protocol.

with credentials corresponding to a new identity, previously unused in the game. Only $\mathcal{C}$ knows b. $\mathcal{A}$ outputs a guess b' for the value b picked by $\mathcal{C}$.

Definition 1. *A secret handshake scheme is unlinkable if an efficient adversary $\mathcal{A}$ has only negligible advantage in winning the game over randomly guessing b, i.e.*

$$Adv(\mathcal{A}) = \left| Pr(b' = b) - \frac{1}{2} \right| < \varepsilon$$

where ε is negligible in the security parameter.

A straightforward approach is generating many pseudonyms and secret points for each client during the enrollment phase. When the client performs handshakes, it uses different credentials each time, which prevents an eavesdropper, or an active attacker, from linking those handshakes to each other, and therefore prevents tracking the client. This naive approach has obvious drawbacks. It increases the amount of storage since the user has to store credentials for all future executions of the protocol. It also limits the number of unlinkable handshakes to the number of pairs of credentials issued at the setup. It might be good enough when a user is unlikely to deplete its pool of credentials, or if it is occasionally possible to communicate with the group administrator to obtain fresh credentials. In addition, this method provides revocation and traitor tracing, which help protect against an adversary that is capable of corrupting users.

We can also achieve unlinkability by a simple modification to the previous scheme, based on an idea introduced by Huang and Cao [28][2] and later used by Gu and Xue [25]. Instead of P_A and P_B being strings that are visible during the initial message exchange between the two parties, we denote by P_A and P_B random points on the elliptic curve. t is once again the master secret, and member secrets are computed as

$$T_A = t \cdot P_A \qquad T_B = t \cdot P_B$$

P_A and P_B are never sent as is in the clear. Instead, for each handshake, Alice picks a random $r \in \mathbb{Z}_q$ and sends rP_A to Bob. Bob picks a random $s \in \mathbb{Z}_q$ and sends sP_B to Alice. Alice computes

$$K_A = e\left(sP_B, T_A\right)^r = e\left(P_B, P_A\right)^{rst} \tag{3}$$

and Bob computes

$$K_B = e\left(T_B, rP_A\right)^s = e\left(P_B, P_A\right)^{rst} \tag{4}$$

so that $K_A = K_B$.

None of the two learns the permanent pseudonym of the other. An adversary that eavesdrops on the messages cannot link different handshake attempts by the same member. And yet, both parties obtain the same key that can now be used for encrypting communication between the two. This scheme provides information theoretic security against tracking. This has been previously argued in [28, 25], but due to the simplicity of this randomization technique, we include a formal claim, and its trivial proof below.

Claim 1. *The handshake scheme presented in 2.2.3 is unlinkable according to Definition 1.*

Proof. The unlinkability of this scheme is information theoretic with a trivial proof. Since each pseudonym picked by the challenger is multiplied by a uniformly drawn random element in $\mathbb{Z}_q$, each broadcast is indistinguishable from a random element to $\mathcal{A}$. □

It is important to note, that this scheme becomes vulnerable to member impersonation in case an adversary corrupts a member of the group, as pointed out by Yoon [46]. Given a member's permanent credentials, the adversary can generate new ephemeral credentials that will pass a handshake. This, however, poses a problem even for newer SH schemes such as [12]. It is therefore important to avoid this method for achieving unlinkability when corruption is likely, and to examine new schemes that provide revocation and traitor tracing along with unlinkability. In the meantime, we can stick to the former simple method of issuing multiple credentials to each member.

2.2.4 Security parameters

Security is quantified as the number of basic operations (e.g AES encryptions) needed for recovering the secret key and breaking the cryptographic scheme.
Breaking a symmetric-key scheme with key size k requires

[2]Their particular scheme, however, has a flaw, as pointed out in [43, 47], due to release of the group public-key. Nevertheless, it doesn't affect our use of the pseudonym randomization idea.

2^k basic operations. Galbraith et al. [23] summarize the equivalence between given symmetric key sizes and the corresponding EC group element sizes, required to provide the same level of security. To enjoy 128-bit security[3] we have to use EC subgroups having size of at least 256 bits, i.e. 32 bytes. It means that we need 32-byte long pseudonyms (for unlinkable handshakes). Compromising for 80-bit security, we can use 160-bit (20 byte) elements[4]. For string pseudonyms (fist scheme), 20 bytes provide 128-bit security.

3. SECRET HANDSHAKES OVER BLE

In this section we discuss ways to integrate secret handshakes into BLE. Our goal is to reuse, as much as possible, the existing BLE message exchange. This part examines the ideal vs. the currently available and practical. We look into ways to reconcile the requirements of a protocol for secret handshakes with BLE constraints and with the current state of supported BLE features on popular mobile devices.

3.1 Challenges

Implementing secret handshakes over BLE poses several challenges. Addressing those challenges at the design stage is one of the main contributions of this paper. Some key issues are the limited amount of data that is possible to transfer between devices over BLE, the fact that we are operating within the restrictions of a given standard, and various constraints imposed by existing BLE implementations both in hardware and software. In addition, we had to pick an underlying cryptographic SH scheme that fits those constraints.

1. **Fitting into BLE packet size:** The restriction on the packet sizes influences our choice of the underlying cryptographic scheme. BLE advertisement packets can fit up to 39 bytes of payload, with further restriction on how many of them we can control (31 bytes on the platform we ended up using). The messages exchanged during a pairing process can fit 16 bytes only. This requires choosing a scheme in which little data is exchanged.

2. **Avoiding Bluetooth pairing:** We do not want the devices to "know" each other, i.e. be paired prior to the handshake attempt, as this contradicts our use-cases. Our proposals for integration avoid this. The first one is a new pairing mode that replaces the existing BLE pairing with a secret-handshake. The second uses BLE advertisements, which do not require prior pairing in order to be published or processed by a scanning party.

3. **Fitting within the standard:** BLE communication is standardized and we want to operate within the capabilities and restrictions of the standard. While the proposition we make in section 3.3 considers a modification of BLE, we also aim for a prototype that works given the currently available functionality, which we present in section 3.4.

4. **Low energy consumption:** Power consumption becomes a major issue since we aim for use-cases where a handshake attempt is frequently initiated. While BLE itself is designed to consume little energy, we also need to make sure that our protocol doesn't add an energy overhead that is not acceptable for a mobile device.

[3]That is the expected level of security nowadays.

[4]This compromise, if affordable, can simplify the proposed implementation, as we will see later.

3.2 Choice of the underlying SH scheme

The scheme presented in section 2.2.2 is a good fit for addressing the small packet size, since the parties only exchange a pseudonym in the form of a single string (or single group element in the case of the modified unlinkable scheme). Some other schemes discussed in section 6, such as [12, 19], require sending more data to initiate a handshake. Moreover, this scheme is simple to understand, and given a library for computing pairings[5], it also results in a relatively simple implementation. Finally, it requires a single pairing computation by each party, compared to [12] which requires three.

While we presented some considerations, surveying and comparing different SH schemes was not the central goal of this work, as much as demonstrating practicality using an existing method. A follow-up work on the subject could greatly benefit from a thorough comparison of the different schemes available, and in particular, comparing their performance and bandwidth requirements.

In the following, we present two methods for integrating the underlying SH scheme into BLE.

3.3 A new pairing mode

Our first proposition for executing the secret handshake procedure over BLE is to reuse the Bluetooth *pairing* procedure[6]. Pairing, in both classic Bluetooth and BLE, establishes a short-term shared key that can be used for encrypted communication between two Bluetooth devices. It can be used for a short message exchange or to establish a long-term key to be used in subsequent sessions.

BLE currently offers several pairing methods:

- Just Works - requires no user interaction[7]. Does not protect from man-in-the-middle (MITM) attacks.
- Numeric comparison - a 6-digit number is displayed on both devices, which confirm the pairing if they both see the same number. Protects from MITM attacks.
- Passkey entry - one device displays a 6 digit number, which has to be entered on the other device to confirm the pairing.
- Out-of-band (OOB) - An external method is used to provide information to both devices that is used to complete the pairing. For instance, NFC can be used to provide each device with an encryption key.

The numeric comparison and passkey entry methods are not very secure, as shown by Mike Ryan [41]. Rather than relying on existing pairing methods, we propose the secret

[5]One of our contributions is providing a library wrapper for our platform of choice.

[6]Not to be confused with the mathematical operation over elliptic curves, mentioned in the explanation of secret handshakes.

[7]Sometimes the user will be asked to confirm the pairing, but not to insert a PIN or to compare passkeys.

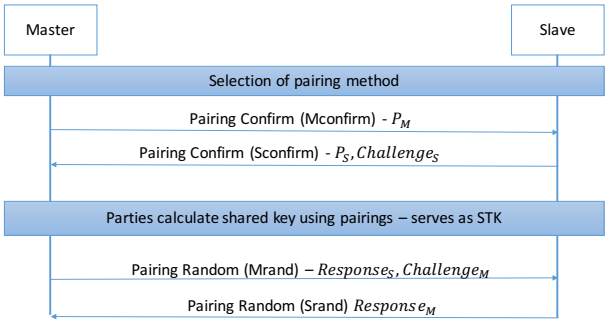


Figure 3: Secret handshake as a new pairing mode. Pseudonyms, challenges and responses are embedded in already existing fields of messages exchanged as part of the pairing procedure.

handshake for establishing trust and obtaining a shared session key. Therefore we can simply use Just Works to establish the Short Term Key (STK), as we do not care about security at this step. The STK establishment process is followed by a pairing confirmation, in the form of *Pairing Confirm* messages sent by both devices.

We use the *Pairing Confirm* messages to exchange pseudonyms between *Alice* and *Bob* by setting the value of the *Mconfirm* field to be the pseudonym. This implies that we have to use 128-bit pseudonyms. At the next step we use the *Pairing Random* message to send an encrypted challenge from *Alice* to *Bob* by setting *Mrand* to be the challenge. *Bob* computes the response and sends it in the *Srand* field in its *Pairing Random* reply message to *A*, which can now verify the success of the handshake[8]. This process is depicted in figure 3.

However, despite using existing messages, this approach requires introducing an additional pairing method and extending the BLE stack with new functionality. And since the pseudonym sizes are limited by the length of the *Mconfirm* messages, it enables less than 80-bits security, which is not acceptable in modern cryptographic implementations. Achieving the desired security levels would require further modifications to the pairing protocol.

3.4 Advertiser-scanner interaction

As mentioned in section 2, BLE supports broadcasting advertisements. Clients can scan and filter advertisements of specific types. An advertisement packet allows only 31 bytes of data to be set. With *passive scanning* the advertised data is limited to the contents of one advertisement packet. With *active scanning*, the scanner can request more data and receive another packet from the advertiser in response.

Windows .NET Bluetooth API currently enables controlling only the Manufacturer Specific Data (AD type 0xFF), which is 20 bytes.

This method of performing the handshake doesn't require extending the BLE specification. It relies on transmitting data between an *advertiser Alice* and a *scanner Bob*[9] in a way that does not require a Bluetooth pairing.

[8]Non-resolvable random Bluetooth addresses are used for all communication mentioned above.

[9]Here, advertiser and scanner are terms taken directly from the BLE specification, referring to a party in advertising state and a party in a scanning state respectively.

The advertiser broadcasts its pseudonym and, if the client that receives the broadcast in a *scanning* state is interested in performing a handshake, it enters the *initiating* state. The client that enters the *connection* state from an initiating state acts as a Master, while the advertiser that enters it from the advertising state acts as a Slave. The connection state serves for verifying the success of the handshake by exchanging a challenge and a response over the encrypted link.

In section 2.2.4 we mentioned that 32-byte pseudonyms are required. Because an advertisement packet cannot contain more than 31 bytes of payload, we have to submit two packets. By using *Active Scanning*, the scanner can initiate a Scan Request once an advertisement packet is received. The advertiser sends another packet in response, which provides enough space to serve a 32-byte string.

After exchanging pseudonyms, the parties derive a 128-bit (16 bytes) long symmetric key. Any subsequent communication for "challenge-response" can therefore fit in one advertisement packet. As previously mentioned, 80-bit security requires only 20-byte long pseudonyms, which now can fit in a single advertisement packet. This size is also enough for 128-bit security with the first (linkable) scheme.

Note that while the scheme in [13] can ultimately require only 3 messages, it would require packing more data into each message. We avoid it due to the limited advertisement length, and instead exchange pseudonyms first, and then proceed with a challenge and response phase, resulting in 4 messages.

To support simultaneous interaction between many devices, we need to instantiate a separate state machine for each handshake attempt, and associate it with the Bluetooth device address of the other party. That way, advertisements received from different parties do not interfere with each other while executing the protocol. Of course, when using a randomized MAC address, we are required to maintain the same device address across all protocol rounds.

While the pairing mode option (3.3) is appealing, it would require adopting our proposal and have it integrated into the BLE standard. Currently, it also restricts us to smaller security parameters. We chose to implement our prototype using advertiser-scanner interaction, as suggested in the last subsection (3.4).

The scanner-advertiser interaction does not require a Bluetooth pairing between the devices, and yet enables exchanging small amounts of data by the same method.

4. IMPLEMENTATION

In this section we discuss various practical considerations that influenced our prototype implementation. We describe the chosen alternative and walk through the implementation.

4.1 Implementation Challenges

Two main factors affect the choice of platform:

- Custom control over BLE communication.
- Convenient implementation of pairings over elliptic curves: pairings are fairly complicated to implement and we would therefore like to begin with an existing, tested implementation.

The current state of BLE deployment on mobile platforms poses challenges to our goals. Only a few smartphone models can broadcast BLE advertisements. In addition, few fields within the advertisement packets can be controlled.

iOS is fairly restrictive in terms of an application's ability to control the data that is transmitted in advertisement packets. While iOS provides support for Apple's own iBeacon protocol, it does not allow customizing the advertisement data, which precludes using iOS as a prototyping platform for testing our protocol.

The Android framework provides the APIs necessary for doing BLE advertising and scanning through *BluetoothLeAdvertiser* and *BluetoothLeScanner* respectively. One particular appeal of Android is the fact that applications are programmed in Java, and there exists a Java library, named JPBC [20], that implements pairing operations over elliptic-curves. It provides the functionality we need for implementing pairing-based secret handshakes.

Unfortunately, most Android phones do not currently support Bluetooth advertisement broadcasting in practice, and we had to defer this option for now. It would be highly useful to enable this capability in future models.

4.2 Windows Phone

Windows phones, running Windows 10 OS, support publishing BLE advertisements and scanning. In addition, they enable advertising and scanning simultaneously. We chose to prototype the handshake scheme on Windows Phone platform since at the moment it was the only widespread mobile framework that enables advertising, provides control of the advertisement packet data, and also enables developing rich user interface using a high-level language.

First, we needed an implementation of elliptic curve operations and pairings that would run on the device. Windows Phone applications are executed on a .NET Runtime and are usually written in C#. It is also possible to integrate native compiled code through *Windows Runtime Components*. One direction was attempting to port the JPBC library, written in Java, to C#. While the task can be partially automated using tools like Sharpen [8], it would still involve a significant effort. Most obstacles arise from the fact that not all Java constructs and syntax are directly translatable to C#.

We adapted the Stanford PBC library [36] to run on ARM and implemented a .NET wrapper to enable using it for Windows Universal applications written in C#. Specifically, we used PBC v0.5.14.

Lynn's PBC library [36] is a C implementation of pairings over elliptic curves. It is easily parametrizable in terms of choosing the algebraic fields to work with, the types of elliptic curves to use, and provides useful functions such as random element generation and hashing arbitrary strings onto elliptic curves.

Its fastest benchmark indicates that a pairing computation can be done in 11 ms [35]. This enables performing about 90 pairing computations per second. Since our handshake protocol requires a single pairing computation (for each party), this is unlikely to become a bottleneck.

Originally, PBC requires the GNU Multi-Precision library (GMP) for its big-integer operations. We used the Multiple-Precision Integers and Rationals library (MPIR) [6], which is compatible with GMP, since it was already adapted for building with Visual Studio. We did the necessary porting to compile it for ARM so that it can be used on a Windows Phone platform.

We replaced the random data generation function originally used in PBC with a modified version to support the new API on Windows 10.
We used *CryptographicBuffer::GenerateRandom* for generating random bytes.

The ported PBC library for ARM is provided in form of a static library that can be linked with a Windows Runtime Component.

4.2.1 PbcProxy - a .NET wrapper for PBC

In order to integrate the ported PBC library with a .NET implementation of the protocol logic, we have written **PbcProxy** - a Windows Runtime Component that links with the library and exposes the interfaces needed for implementing the handshake scheme.

The wrapper is implemented as a Windows Runtime Component, a DLL that exposes some of the functionality of the PBC library by bridging between managed and unmanaged code. All classes reside in the *PbcProxy* namespace. The following interfaces are exposed by the wrapper:

- Element - provides multiplication and power operations over group elements, and serializes an element to its byte representation. The serialization is used to obtain an encryption key.
- Pairing - provides functions for hashing strings to group elements, obtaining a random element in a given group and applying pairings to pairs of elements, one in $\mathbb{G}_1$ and the other in $\mathbb{G}_2$.
- Group - an interface representing an abstract group, and instantiations for representing $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{Z}_n$.

To use *PbcProxy* we have to first initialize it by calling one of the methods *init()* or *init(seed)*. The first method sets PBC to use a random number generation function that uses *CryptographicBuffer::GenerateRandom*, while the second sets it to use a deterministic generator passing it a seed. Initializing two devices with a common seed is useful for testing by simulating the common dealer using a local instance. In our demo application, we simulated two phones getting issued credentials using the same master secret by independently setting their random seed to the same value.

We also provide a test method *PBC.test()* that implements a simple pairing computation, to test that the PBC library works correctly on the Windows phone platform.

PbcProxy exports the classes *Pairing* and *Element* that expose group element operations and the pairing application. A common interface *GroupIface* requires implementing an *initElement* method for each group type we want to work with. This interface serves for decoupling the specific groups used from the implementation of the *Pairing* and *Element* classes. We have currently implemented subclasses for the groups $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{Z}_n$ that allow using $\mathbb{Z}_n$ elements and referring to the respective groups used in a pairing. For example, in the secret handshake scheme, one credential is an element in $\mathbb{G}_1$ and the other is an element in $\mathbb{G}_2$.

The dependencies between the different software components are summarized in figure 4.

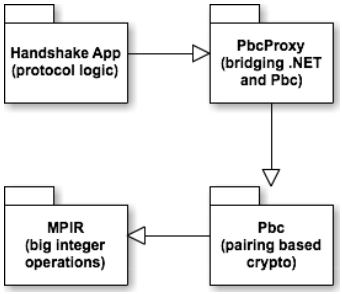


Figure 4: Component dependencies: PbcProxy bridges between the protocol logic implementation and the library providing pairing-based crypto, which uses big integer operations provided by MPIR.

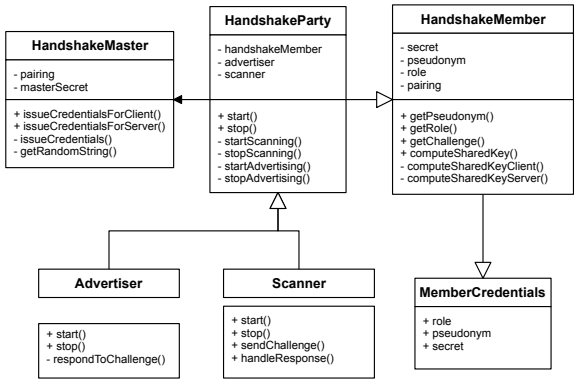


Figure 5: Class diagram of the handshake prototype implementation. Each party participating in a handshake instantiates both Advertiser and Scanner and alternates between them to communicate.

4.2.2 *Handshake scheme prototype*

We provide documentation of our prototype application. It is structured as follows. A party participating in the handshake is represented by one of the classes *Advertiser* or *Scanner*. The first, as the name suggests, publishes advertisements and the latter is scanning for advertisement packets. A mobile application, however, can perform both roles simultaneously. The two classes inherit common functionality from *HandshakeParty*. Advertiser and Scanner subclass HandshakeParty. HandshakeParty implements the communication protocol, and uses an instance of *HandshakeMember* that provides the cryptographic functionality and interacts with *PbcProxy*. *HandshakeMaster* simulates the credentials dealer, providing an instance of *MemberCredentials* upon request. The relationships between the classes are summarized in the UML diagram in figure 5.

Figure 6 illustrates our realization of the secret handshake protocol using interaction between an advertiser and a scanner. It uses the fact that Windows Phone enables performing advertisement and scanning at the same time. Fig. 7 illustrates the state transition for the two parties based on the received broadcast.

The *Advertiser* starts publishing its pseudonym P_A, while at the same time scanning for a reply from the *Scanner*. The Scanner, upon receiving an advertisement packet, pauses the scanning, interprets the data as a pseudonym and com-

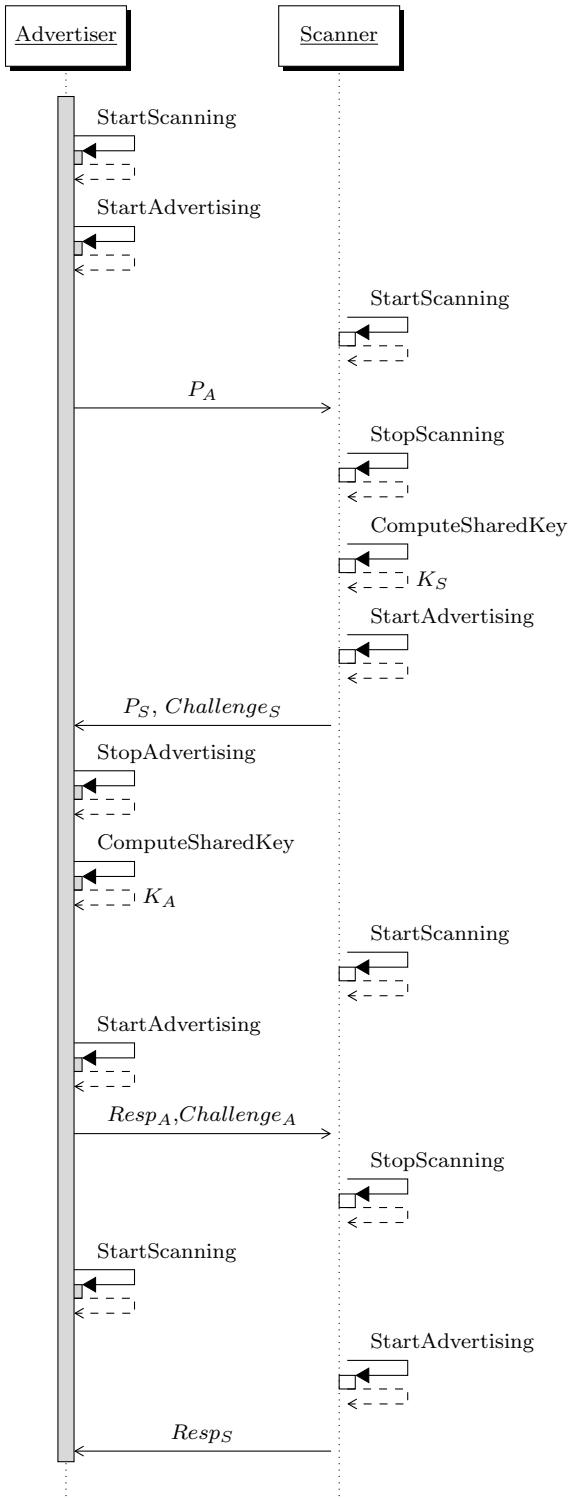
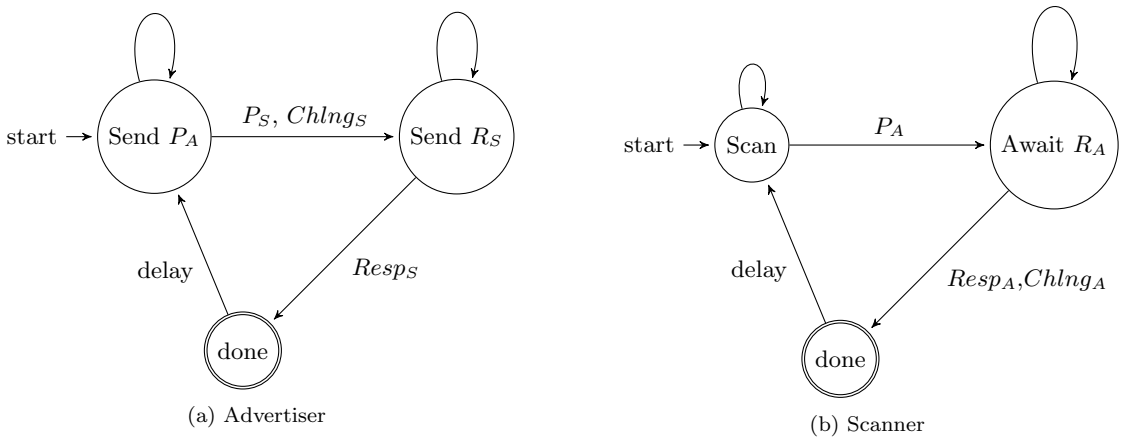


Figure 6: Secret handshake on top of Advertiser-Scanner interaction. The diagram illustrates simultaneous advertisement and scanning by the same party at certain stages, and switching between advertising and scanning to communicate.



(a) Advertiser

(b) Scanner

Figure 7: State machines for Advertiser and Scanner.

putes a shared key according to one of Eq. 2 or 4, depending on whether we use linkable or unlinkable handshakes. It then starts advertising a response packet containing its own pseudonym P_S, and a challenge $Challenge_S$. The challenge is AES–encrypted using the computed symmetric key. Shortly after, it reactivates scanning in order to receive a response from the advertiser. There is a small delay between starting to advertise and reactivating the scanning, in order to avoid receiving an advertisement with a pseudonym, before the Advertiser had a change to receive the Scanner's pseudonym and switch to a new state. Once the Advertiser receives an advertisement with the scanner's pseudonym, it pauses scanning and computes the shared key using one of Eq. 1 or 3. It then decrypts the challenge, using the computed symmetric key and calculates a response $Resp_A$. It also generates its own challenge $Challenge_A$ and encrypts it using the same key. It resumes advertising to send the response to the Scanner. The Scanner decrypts the advertisement packet using its key, and if it verifies correctly, assumes that the handshake succeeded. It computes and encrypts a response $Resp_S$ to $Challenge_A$ and starts advertising to send it to the Advertiser. Once the Advertiser receives the response $Resp_S$, it verifies it, and if it passes, assumes that the handshake succeeded.

The source code of our implementation is available on https://github.com/ymcrcat/MASHaBLE.

4.3 Evaluation

For evaluating our implementation we created a simple app that can broadcast, scan, and perform a secret handshake with another phone running the same app.

4.3.1 *Functionality testing*

Our devices perform simultaneously Bluetooth advertising and scanning. It is important to verify experimentally that this unusual practice results in successful execution of the handshake protocol with high probability. We ran our application on two Windows Phone devices, repeatedly performing handshakes between the two, for 8296 seconds ($\sim$ 2 hours 18 seconds). We performed 1 handshake every 8 seconds, resulting in a total number of 1068 handshake attempts. 1025 attempts resulted in a successful shared key establishment, and only 43 handshake attempts have failed. The failures were due to occasional lack of synchronization between the states of the two devices. Overall, in 96% of the cases the handshake procedure succeeded. This result suggests that our protocol is fairly resistant to synchronization failures.

4.3.2 *Energy overhead*

We measured the energy overhead of our Windows Phone implementation on a Nokia Lumia 920 phone. In each of the following scenarios, we started with the phone with its battery 100% charged, WiFi and location services off, and screen dimmed to minimum brightness level:

1. Baseline: the phone is not running the app.
2. Advertising: the app is continuously publishing BLE advertisements of its pseudonym (no response is sent meaning no actual handshake process is initiated).
3. Scanning: the app is continuously scanning for advertisements (no advertisement is received meaning no actual handshake process is initiated).
4. Handshake Advertiser: the app, acting as an advertiser, is constantly performing handshakes with another phone acting as a scanner.
5. Handshake Scanner: the app, acting as a scanner, is constantly performing handshakes with another phone acting as an advertiser.

We ran the experiment for 3 hours for each one of the modes, and took note of the battery drain percentage. The results of the experiment are presented in Fig. 8. It is interesting to notice that scanning actually consumes more than advertising (despite the fact that advertising involves transmission). This result was previously confirmed in other experiments such as [32]. The reason is the design rational behind BLE modes of operation. Since it is likely that a beacon device has to be in advertising mode for a long period of time, it is designed for maximal energy saving. It transmits a short packet each time and returns to sleep. Scanning is intended to be initiated less often. While scanning, the device constantly searches for beacons, performing intense processing.

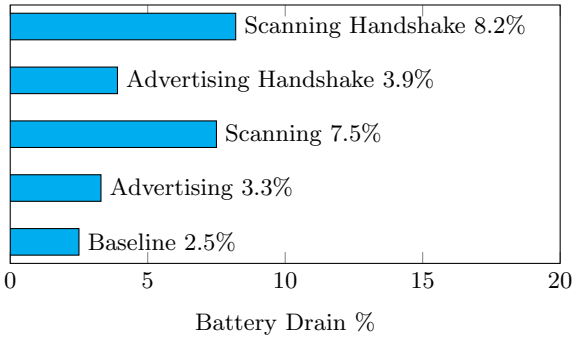


Figure 8: Battery drain in different modes after 3 hours. We can see that cryptographic operations add little overhead on top of BLE energy consumption.

The baseline we compare to is battery drain without any activity, which was measured to be 2.5% per hour. Advertising resulted in additional 0.8% per hour. Scanning consumes 5% per hour on top of the baseline, which is non-negligible, but nevertheless still enables almost 13 hours of operation. Advertising followed by handshake operations resulted in a battery drain of 3.9% per hour, which enables approximately 26 hours of operation. Finally, scanning followed by handshakes resulted in a drain of 8.2% per hour, which enables 12 hours of operation. Overall, these results are encouraging in the sense that they enable operation times that are on the order of what is required to be able to normally use a smartphone before recharging.

By subtracting the results when handshakes are not performed from the results when they are, we get the percentage that the handshake logic adds on top of the power consumption due to BLE scanning or advertising. In the case of advertising, handshakes added 0.6% per hour on top of advertising without proceeding further with the protocol. In the case of scanning, proceeding with a handshake added 0.7% on top of scanning only.

4.3.3 *Communication overhead*

Each advertisement packet consists of 47 bytes[10]. The handshake initiator (Alice) sends 1 packets to scanner (Bob) containing a pseudonym. Bob sends back 1 packet with his pseudonym and a challenge. Alice replies with 1 packet containing her response to Bob's challenge and a challenge of her own, and finally, Bob responds to Alice's challenge with 1 packet. Alice sends $47 \times 2 = 94$ bytes total, and Bob sends $47 \times 2 = 94$ bytes. Overall we have 4 communication rounds with 4 packets, which results in 188 bytes.

Depending on Alice and Bob's decision whether to use the established encrypted channel for further communication, e.g. exchanging messages, more data can be transmitted. However, for all further communication, the overhead will stem solely from the Bluetooth packet structure, since they use the computed symmetric key for encryption, which results in the same length as the plaintext.

5. FURTHER IMPROVEMENTS

We propose possible enhancements to the current implementation. Some of them are immediately applicable, and some are subject to further research.

[10]In our case we use the maximum available size for advertisement packets.

5.1 Speeding up computation

Chapter 6.11 of Lynn's thesis [37] covers several possible opportunities for precomputation that can speed up subsequent computation of pairings. Those opportunities arise from the fact that some of the computation steps depend entirely on only one of the two inputs to the pairing. For each handshake initiated using the same credentials, the pairing $\mathbb{G}_1$ input is the same. Therefore, Lynn's proposals can be applied. Moreover, preprocessing is supported by the PBC library and is part of its API. It would be beneficial to extend PbcProxy to support preprocessed pairings.

5.2 Using BLE identifiers as pseudonyms

As already mentioned, BLE supports using a randomized device address instead of the permanent public address. If we could set the source device address we would be able to use it as a pseudonym, instead of setting it in the advertisement data. We saw that having 128-bit security for the unlinkable scheme requires broadcasting two subsequent advertisement packets in order to transmit the pseudonym. Having additional space in the form of the Bluetooth device address field enables to transmit 32-byte long credentials in a single packet.

5.3 Organizational role authentication

The secret handshake scheme described in 2.2 supports, with a slight modification, establishing the corresponding roles of the parties within the organization, in addition to proving affiliation. We do not provide the details here, but it is described in [13]. We note, however, that the method in [13] assumes using pseudonyms which are public strings, to which the other party can attach an additional string indicating the role. That doesn't settle with the unlinkable handshake scheme in 2.2.3, proposed as a countermeasure against tracking, because it involves transmitting encodings of elliptic curve elements that mask the pseudonym.

5.4 Additional applications of PbcProxy and ported PBC library

PbcProxy, the glue between the .NET framework and a C implementation of pairings compiled for the ARM processor, is an opening for many other Windows Phone applications that rely on pairing-based cryptography. One prominent application is Identity-based Encryption [17, 15, 16]. It enables encrypting sending encrypted messages to another party solely based on its publicly known identifier such as an email or a phone number. For instance, a company or organization's employees have each other's contacts. Using identity-based encryption they can securely communicate with each other without the need to exchange public keys directly. While the concept itself is not new, and even commercialized by Voltage, the software libraries we provide enable easy development of this application for the Windows Phone platform.

5.5 Usability

While we discuss the technical aspects of secret handshakes between mobile devices, usability issues were left outside the scope of this work. For successful adoption of our, or any other library and protocol, there has to be a good understanding of how vendors and users would use this ca-

pability in practice. We suggested several use cases in the introduction. It would be beneficial to validate them by surveying vendors and users, as well as to find additional use-case scenarios.

6. RELATED WORK

We survey related work in the relevant domains, focusing on theoretic work applicable to our setting, and on systems projects with similar goals.

Hiding the Rumor Source by Fanti et al. [22] provides message source obfuscation by using *adaptive diffusion* spreading. They operate under a model that assumes an adversary that has access to metadata, and and also to some corrupted nodes. Their method can be complementary to ours. If our application uses the established shared key to diffuse messages among society members, we can apply their scheme to dictate the message diffusion policies, so that an adversary that colludes with corrupted members of the secret group, would not be able to identify the message source.

RevCast by Schulman et al. [42] is related to our proposal in the sense that it disseminates messages over a radio channel[11]. Thus the message transfer is independent of a central server. It also aims to facilitate clients' privacy by obviating requests to a server, an act that discloses a client's interest in particular data.

Since the scheme we have chosen [13] was proposed, there have been more works on the subject of secret handshakes. Some of them provide alternatives to pairing-based crypto [19, 45], while others address unlinkability and more flexible policies [12, 29, 34]. We discuss some of them below.

In [19] the authors propose a scheme for secret handshakes based on CA-Oblivious encryption. CA-Oblivious encryption relies on providing a certificate to the user from which one cannot learn about the signing authority. Its security proof relies on the Computational Diffie-Hellman assumption (CDH). This scheme avoids using pairing-based cryptography and can be built based on more standard PKI. This can be useful on platforms on which it is currently difficult to run one of the existing libraries for pairing-based cryptography, such as PBC or JPBC. While this scheme doesn't require pairings, more data has to be sent by the handshake initiating party than in the scheme based on pairings that only requires sending a short pseudonym. Considering the very constrained amount of data we are allowed to send in an advertisement packet, the pairing-based scheme seems more suitable for our task. Additionally, their scheme doesn't provide unlinkability that is not based on issuing multiple credentials, while the scheme of Balfanz et al. [13] is conveniently transformed into an unlinkable SH scheme with reusable credentials.

Secret handshakes with dynamic and fuzzy matching, by Ateniese et al. [12], adds the ability to specify a set of attributes, required from the other party, in order for the handshake to succeed. It is essentially an attribute-based secret handshake scheme which also provides unlinkability and supports roles. The authors integrated the scheme into the IPSec protocol, by extending the Internet Key Exchange protocol [26]. While the scheme we chose is not the state of the art, as of today, it is simpler to understand, and requires sending less data upon handshake initiation. The applications we proposed so far can do without the additional features provided by the dynamic or fuzzy handshake schemes. In addition, our protocol requires performing a single bilinear pairing computation, while that scheme requires performing three pairings. However, providing additional functionality by using more novel schemes would be a desirable follow-up work, as well as identifying how mobile applications can benefit from fuzzy attribute-based secret handshakes.

Hidden Credentials by Holt et al. [27] proposes a method for Bob to send a message to Alice depending only on Alice's credentials, without having any credentials of his own. It is based on Boneh-Franklin IBE [17] which is in turn based on the Weil pairing. The notion of hidden credentials could be useful in case there is a messenger, who is an outsider to the organization, but wants to send an encrypted message to any member that has a certain role or clearance in the organization.

[33] is an example of incorporating custom identifiers in the BLE MAC address, as we suggested in 5.2. While that work also pertains to protecting user's privacy, the guarantees provided by their Incognito system are different and are useful under different assumptions.

Prior work also investigated techniques for preserving privacy of end-users by changes of IP and MAC addresses [40, 38], identifier-free link layer protocol [14, 24] and short-lived credentials or pseudonyms [30]. However, all these techniques aim to ensure unlinkability only.

[32] is a useful reference for understanding the intricacies of BLE scanning and advertisement. The power consumption model and design guidelines in that work can be useful for optimizing future versions of our protocol.

7. CONCLUSION

We propose a mobile application that enables members of a secret community to discover other affiliates that are in proximity to their mobile device. It enables the creation of an authenticated and encrypted communication channel over which the two members can communicate. We provide the background needed to understand our scheme and a novel analysis of the technicalities related to implementing it on top of a Bluetooth LE stack on a mobile device. Current state of BLE support poses many challenges which we examine. We discuss the design considerations and alternatives and describe our prototype implementation of the secret handshake scheme. The experimental evaluation of our implementation suggests that cryptographic secret handshakes for mobile devices are highly practical. In particular, we show that our protocol plays well with Bluetooth LE, and its energy overhead is acceptable for mobile devices. Finally, we propose several ideas for further enhancement and research.

Acknowledgments

We would like to thank Dan Boneh from Stanford University for advice on pairing-based cryptography, identity-based encryption and related works. We would like to thank members of the Sensing and Energy Research Group at Microsoft Research: Bodhi Priyantha, for advising on Bluetooth LE and Di Wang for helping with the mobile setup used in our experiments. Finally, we thank the anonymous reviewers and the MobiCom 2016 TPC for their valuable comments.

[11] They use FM radio broadcasts to spread certificate revocation updates.

8. REFERENCES

[1] Afterschool, http://afterschoolapp.com/app.

[2] Complete keyless, www.completekeyless.com.

[3] Connect2car, www.connect2car.com.

[4] Dubbledutch.me, http://doubledutch.me/attendance-tracking.html.

[5] Legatalk, http://legatalk.com.

[6] Multiple precision integers and rationals library, mpir.org.

[7] Open whisper systems, https://whispersystems.org.

[8] Sharpen - Automated Java to C# coversion.

[9] Telegram messenger, https://telegram.org.

[10] Yik yak, https://www.yikyak.com/home.

[11] "iBeacon" technology that will make possible Internet of Things. pages 159–165. Institution of Engineering and Technology, 2014.

[12] Giuseppe Ateniese, Marina Blanton, and Jonathan Kirsch. Secret handshakes with dynamic and fuzzy matching. In *Network and Distributed System Security Symposium*, pages 159–177, 2007.

[13] D. Balfanz, G. Durfee, N. Shankar, D. Smetters, J. Staddon, and Hao-Chi Wong. Secret handshakes from pairing-based key agreements. *2003 Symposium on Security and Privacy, 2003.*, 2003.

[14] Kevin Bauer, Damon McCoy, Ben Greenstein, Dirk Grunwald, and Douglas Sicker. Physical layer attacks on unlinkability in wireless lans. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 108–127. Springer, 2009.

[15] Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In *Advances in Cryptology–Crypto 2004*, pages 443–459. Springer, 2004.

[16] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *Advances in Cryptology–EUROCRYPT 2005*, pages 440–456. Springer, 2005.

[17] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. Advances in Cryptology - CRYPTO 2001, 2001.

[18] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. Advances in Cryptology - ASIACRYPT 2001, 2001.

[19] Claude Castelluccia, Stanislaw Jarecki, and Gene Tsudik. Secret Handshakes from CA-Oblivious Encryption. In *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*, volume 3329, pages 293–307, 2004.

[20] Angelo De Caro and Vincenzo Iovino. jpbc: Java pairing based cryptography. In *Proceedings of the 16th IEEE Symposium on Computers and Communications, ISCC 2011*, pages 850–855. IEEE, 2011.

[21] Andreas Enge. Bilinear pairings on elliptic curves, 2013.

[22] Giulia Fanti, Peter Kairouz, Sewoong Oh, Kannan Ramchandran, and Pramod Viswanath. Hiding the rumor source. *arXiv preprint arXiv:1509.02849*, 2015.

[23] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers, 2008.

[24] Ben Greenstein, Damon McCoy, Jeffrey Pang, Tadayoshi Kohno, Srinivasan Seshan, and David Wetherall. Improving wireless privacy with an identifier-free link layer protocol. In *Proceedings of the 6th international conference on Mobile systems, applications, and services*, pages 40–53. ACM, 2008.

[25] Jie Gu and Zhi Xue. An Improved Efficient Secret Handshakes Scheme with Unlinkability. *IEEE Communications Letters*, 15(2):259–261, feb 2011.

[26] Dan Harkins and Dave Carrel. Rfc 2409: The internet key exchange (ike), november 1998. *Status: Proposed Standard*.

[27] Jason E. Holt, Robert W. Bradshaw, Kent E. Seamons, and Hilarie Orman. Hidden Credentials. *Proceeding of the ACM workshop on Privacy in the electronic society - WPES '03*, page 1, 2003.

[28] Hai Huang and Zhenfu Cao. A Novel and Efficient Unlinkable Secret Handshakes Scheme. *Ieee Communications Letters*, 13(5):363–365 ST – A Novel and Efficient Unlinkable Sec, 2009.

[29] Stanislaw Jarecki and Xiaomin Liu. Unlinkable Secret Handshakes and Key-Private Group Key Management Schemes. In *Applied Cryptography and Network Security*, volume 4521, pages 270–287. 2007.

[30] Qi Jiang, Jianfeng Ma, Guangsong Li, and Li Yang. An efficient ticket based authentication protocol with unlinkability for wireless access networks. *Wireless personal communications*, 77(2):1489–1506, 2014.

[31] Antoine Joux. A one round protocol for tripartite diffie–hellman. In *Algorithmic number theory*, pages 385–393. Springer, 2000.

[32] Philipp Kindt, Daniel Yunge, Robert Diemer, and Samarjit Chakraborty. Precise Energy Modeling for the Bluetooth Low Energy Protocol. mar 2014.

[33] Robin Kravets, Güliz Seray Tuncay, and Hari Sundaram. For your eyes only. In *Proceedings of the 6th International Workshop on Mobile Cloud Computing and Services*, pages 28–35. ACM, 2015.

[34] Preeti Kulshrestha and Arun Kumar. A New Unlinkable Secret Handshakes Scheme based on ZSS.

[35] Ben Lynn. Pbc benchmarks, https://crypto.stanford.edu/pbc/times.html.

[36] Ben Lynn. Pbc library, https://crypto.stanford.edu/pbc.

[37] Ben Lynn. *On the Implementation of Pairing-Based Cryptosystems*. PhD thesis, 2007.

[38] Shrirang Mare, Jacob Sorber, Minho Shin, Cory Cornelius, and David Kotz. Hide-n-sense: preserving privacy efficiently in wireless mhealth. *Mobile Networks and Applications*, 19(3):331–344, 2014.

[39] Victor S Miller. The weil pairing, and its efficient calculation. *Journal of Cryptology*, 17(4):235–261, 2004.

[40] Barath Raghavan, Tadayoshi Kohno, Alex C Snoeren, and David Wetherall. Enlisting isps to improve online privacy: Ip address mixing by default. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 143–163. Springer, 2009.

[41] Mike Ryan. Bluetooth: With Low Energy Comes Low Security. In *Proceedings of the 7th USENIX Conference on Offensive Technologies*, page 4, 2013.

[42] Aaron Schulman, Dave Levin, and Neil Spring. Revcast: Fast, private certificate revocation over fm radio. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 799–810. ACM, 2014.

[43] Renwang Su. On the security of a novel and efficient unlinkable secret handshakes scheme. *IEEE Communications Letters*, 13(9):712–713, sep 2009.

[44] Kevin Townsend, Carles Cufi, and Robert Davison. Getting Started with Bluetooth Low Energy, 2014.

[45] Damien Vergnaud. RSA-Based Secret Handshakes. pages 252–274. 2006.

[46] Eun-Jun Yoon. Cryptanalysis of an Efficient Secret Handshakes Scheme with Unlinkability. *Procedia Engineering*, 24:128–132, 2011.

[47] Taek-Young Youn and Young-Ho Park. Security analysis of an unlinkable secret handshakes scheme. *IEEE Communications Letters*, 14(1):4–5, jan 2010.