Mesh Messaging in Large-scale Protests: Breaking Bridgefy

Martin R. Albrecht, Jorge Blasco, Rikke Bjerg Jensen, and Lenka Mareková

Royal Holloway, University of London
{martin.albrecht,jorge.blascoalis,rikke.jensen,lenka.marekova}@rhul.ac.uk

Abstract. Mesh messaging applications allow users in relative proximity to communicate without the Internet. The most viable offering in this space, Bridgefy, has recently seen increased uptake in areas experiencing large-scale protests (Hong Kong, India, Iran, US, Zimbabwe, Belarus), suggesting its use in these protests. It is also being promoted as a communication tool for use in such situations by its developers and others. In this work, we report on a security analysis of Bridgefy. Our results show that Bridgefy, as analysed, permitted its users to be tracked, offered no authenticity, no effective confidentiality protections and lacked resilience against adversarially crafted messages. We verified these vulnerabilities by demonstrating a series of practical attacks on Bridgefy. Thus, if protesters relied on Bridgefy, an adversary could produce social graphs about them, read their messages, impersonate anyone to anyone and shut down the entire network with a single maliciously crafted message.

Keywords: Mesh messaging · Bridgefy · Security analysis.

1 Introduction

Mesh messaging applications rely on wireless technologies such as Bluetooth Low Energy (BLE) to create communication networks that do not require Internet connectivity. These can be useful in scenarios where the cellular network may simply be overloaded, e.g. during mass gatherings, or when governments impose restrictions on Internet usage, up to a full blackout, to suppress civil unrest. While the functionality requirements of such networks may be the same in both of these scenarios – delivering messages from A to B – the security requirements for their users change dramatically.

In September 2019, Forbes reported "Hong Kong Protestors Using Mesh Messaging App China Can't Block: Usage Up 3685%" [46] in reference to an increase in downloads of a mesh messaging application, Bridgefy [1], in Hong Kong. Bridgefy is both an application and a platform for developers to create their own mesh network applications.[1] It uses BLE or Bluetooth Classic and is designed for use cases such as "music festivals, sports stadiums, rural communities, natural disasters, traveling abroad", as given by its Google Play store

[1] As we discuss in Section 2.4, alternatives to Bridgefy are scarce, making it the predominant example of such an application/framework.

description [20]. Other use cases mentioned on its webpage are ad distribution (including "before/during/after natural disasters" to "capitalize on those markets before anybody else" [1]) and turn-based games. The Bridgefy application has crossed 1.7 million downloads as of August 2020 [58].

Though it was advertised as "safe" [20] and "private" [18] and its creators claimed it was secured by end-to-end encryption [46,52,68], none of the aforementioned use cases can be considered as taking place in adversarial environments, such as situations of civil unrest where attempts to subvert the application's security are not merely possible, but to be expected, and where such attacks can have harsh consequences for its users. Despite this, the Bridgefy developers advertised the application for such scenarios [46,72,74,75] and media reports suggest the application is indeed relied upon.

Hong Kong International news reports of Bridgefy being used in anti-extradition law protests in Hong Kong began around September 2019 [60,82,46,17], reporting a spike in downloads that was attributed to slow mobile Internet speeds caused by mass gatherings of protesters [22]. Around the same time, Bridgefy's CEO reported more than 60,000 installations of the application in a period of seven days, mostly from Hong Kong [60]. However, a Hong Kong based report available in English [15] gave a mixed evaluation of these claims: in the midst of a demonstration, not many protesters appeared to be using Bridgefy. The same report also attributes the spike in Bridgefy downloads to a DDoS attack against other popular communication means used in these protests: Telegram and the Reddit-like forum LIHKG.

India The next reports to appear centred on the Citizenship Amendment Act protests in India [10] that occurred in December 2019. Here the rise in downloads was attributed to an Internet shutdown occurring during the same period [48,64]. It appears that the media narrative about Bridgefy's use in Hong Kong might have had an effect: "So, Mascarenhas and 15 organisers of the street protest decided to take a leaf out of the Hong Kong protesters' book and downloaded the Bridgefy app" [55]. The Bridgefy developers reported continued adoption in summer 2020 [76].

Iran While press reports from Iran remain scarce, there is evidence to suggest that some people are trying to use Bridgefy during Internet shutdowns and restrictions: the rise of customer support queries coming from Iran and a claim by the Bridgefy CEO that it is being distributed via USB devices [49].

Lebanon Bridgefy now appears among recommended applications to use during an Internet shutdown, e.g. in the list compiled by a Lebanese NGO during the October 2019 Lebanon protests [63]. A media report suggests adoption [68].

US The Bridgefy developers reported uptake of Bridgefy during the Black Lives Matter protests across the US [77,75]. It is promoted for use in these protests by the developers and others on social media [69,75,70].

Zimbabwe Media and social media reports advertised Bridgefy as a tool to counter a government-mandated Internet shutdown [50,47] in summer 2020. The Bridgefy developers reported an uptick in adoption [78].

Belarus Social media posts and the Bridgefy developers suggest adoption in light of a government-mandated Internet shutdown [79].

Thailand Social media posts encouraged student protesters to install the Bridgefy application during August 2020 [71].

1.1 Contributions

We reverse engineered Bridgefy's messaging platform, giving an overview in Section 3, and in Section 4 report several vulnerabilities voiding both the security claims made by the Bridgefy developers and the security goals arising from its use in large-scale protests. In particular, we describe various avenues for tracking users of the Bridgefy application and for building social graphs of their interactions both in real time and after the fact. We then use the fact that Bridgefy implemented no effective authentication mechanism between users (nor a state machine) to impersonate arbitrary users. This attack is easily extended to an attacker-in-the-middle (MITM) attack for subverting public-key encryption. We also present variants of Bleichenbacher's attack [12] which break confidentiality using $\approx 2^{17}$ chosen ciphertexts. Our variants exploit the composition of PKCS#1 v1.5 encryption and Gzip compression in Bridgefy. Moreover, we utilise compression to undermine the advertised resilience of Bridgefy: using a single message "zip bomb" we could completely disable the mesh network, since clients would forward any payload before parsing it which then caused them to hang until a reinstallation of the application.

Overall, we conclude that using Bridgefy, as available prior to our work, represented a significant risk to participants of protests. In October 2020 and in response to this work, the Bridgefy developers published a revision of their framework and application adopting the Signal protocol. We discuss our findings and report on the disclosure process in Section 5.

2 Preliminaries

We denote concatenation of strings or bytes by $||$. Strings of byte values are written in hexadecimal and prefixed with `0x`, in big-endian order.

We analysed the Bridgefy apk version 2.1.28 dated January 2020 and available in the Google Play store. It includes the Bridgefy SDK version 1.0.6. In what follows, when we write "Bridgefy" we mean this apk and SDK versions, unless explicitly stated otherwise. As stated above, the Bridgefy developers released an update of both their apk and their SDK in response to a preliminary version of this work and our analysis does not apply as is to these updated versions (cf. Section 5).

2.1 Reverse engineering

Since the Bridgefy source code was not available, we decompiled the apk to (obfuscated) Java classes using Jadx [62]. The initial deobfuscation was done automatically by Jadx, with the remaining classes and methods being done by hand using artefacts left in the code and by inspecting the application's execution.

This inspection was performed using Frida, a dynamic instrumentation toolkit [28], which allows for scripts to be injected into running processes, essentially treating them as black boxes but enabling a variety of operations on them. In the context of Android applications written in Java, these include tracing class instances and hooking specific functions to monitor their inputs/outputs or to modify their behaviour during runtime.

2.2 Primitives used

Message encoding To encapsulate Bluetooth messages and their metadata, Bridgefy uses MessagePack [29], a binary serialisation format that is more compact than and advertised as an alternative to JSON.

It is then compressed using Gzip [40], which utilises the widely-used DEFLATE compressed data format [39]. The standard implementation found in the `java.util.zip` library is used in the application. A Gzip file begins with a 10-byte header, which consists of a fixed prefix `0x1f8b08` followed by a flags byte and six additional bytes which are usually set to 0. Depending on which flags are set, optional fields such as a comment field are placed between the header and the actual DEFLATE payload. A trailer at the end of the Gzip file consists of two 4-byte fields: a CRC32 and the length, both over the uncompressed data.

RSA PKCS#1 v1.5 Bridgefy uses the (now deprecated) PKCS#1 v1.5 [41] standard. This standard defines a method of using RSA encryption, in particular specifying how the plaintext should be padded before being encrypted. The format of the padded data that will be encrypted is `0x0002` || $<$*random non-zero bytes*$>$ || `0x00` || $<$*message*$>$. If the size of the RSA modulus and hence the size of the encryption block is k bytes, then the maximum length of the message is $k - 11$ bytes to allow for at least 8 bytes of padding.

This padding format enables a well-known attack by Bleichenbacher [12] (for variants/improvements of Bleichenbacher's attack see e.g. [45,7,6,14]). The attack requires a padding oracle, i.e. the ability to obtain an answer to whether a given ciphertext decrypts to something that conforms to the padding format. Sending some number of ciphertexts, each chosen based on previous oracle responses, leads to full plaintext recovery. For RSA with $k = 128$, the number of chosen ciphertexts required has been shown to be between 2^{12} and 2^{16} [16].

In more detail, let c be the target ciphertext, n the public modulus, and (e, d) the public and private exponents, respectively. We have $\mathrm{pad}(m) = c^d \mod n$ for the target message m. The chosen ciphertexts will be of the form $c^* = s^e \cdot c \mod n$ for some s. If c^* has correct padding, we know the first two bytes of $s \cdot \mathrm{pad}(m)$, and hence a range for its possible values. The attack thus first finds small values of s which result in a positive answer from the oracle and for each of them computes a set of ranges for the value of $\mathrm{pad}(m)$. Once there is only one possible range, larger values of s are tried in order to narrow this range down to only one value, which is the original message.

2.3 Related work

Secure messaging Message layer security has received renewed attention in the last decade, with an effort to standardise a protocol – simply dubbed Messaging Layer Security (MLS) – now underway by the IETF [67], with several academic works proposing solutions or analysing security [23,5,4]. The use of secure messaging by "high-risk users" is considered in [26,34]. In particular, those works analyse interviews with human rights activists and secure messaging application developers to establish common and diverging concerns.

Compression in security The first compression side channels in the context of encryption were described by [44], based on the observation that the compression rate can reveal information about the plaintext. Since then, there have been practical attacks exploiting the compression rate "leakage" in TLS and HTTP, dubbed CRIME [25] and BREACH [32], which enabled the recovery of HTTP cookies and contents, respectively. Similarly, [31] uses Gzip as a format oracle in a CCA attack. Beyond cryptography, compression has also been utilised for denial of service attacks in the form of so-called "zip bombs" [27], where an attacker prepares a compressed payload that decompresses to a massive message.

Mesh networking security Wireless mesh networks have a long history, but until recently they have been developed mainly in the context of improving or expanding Wi-Fi connectivity via various ad hoc routing protocols, where the mesh usually does not include client devices. Flood-based networks using Bluetooth started gaining traction with the introduction of BLE, which optimises for low power and low cost, and which has been part of the core specification [13] since Bluetooth 4.0. BLE hardware is integrated in all current major smartphone brands, and the specification has native support in all common operating systems.

Previous work on the security analysis of Bluetooth focused on finding vulnerabilities in the pairing process or showing the inadequacy of its security modes, some of which have been fixed in later versions of the specification (see [24,35] for surveys of attacks focusing on the classic version of Bluetooth). As a more recent addition, BLE has not received as much comprehensive analysis, but general as well as IoT-focused attacks exist [57,42,81,61]. Research on BLE-based tracking has looked into the usage of unique identifiers by applications and IoT devices [83,9]. The literature on security in the context of BLE-based mesh networks is scarce, though the Bluetooth Mesh Profile [59] developed by the Bluetooth SIG is now beginning to be studied [2,3].

2.4 Alternative mesh applications

We list various alternative chat applications that target scenarios where Internet connectivity is lacking, in particular paying attention to their potential use in a protest setting.

FireChat FireChat [53] was a mobile application for secure wireless mesh networking meant for communication without an Internet connection. Though it was not built for protests, it became the tool of choice in various demonstrations since 2014, e.g. in Iraq, Hong Kong and Taiwan [8,11,43], and since then was also promoted as such by the creators of the application. However, it had not received any updates in 2019 and as of April 2020, it is no longer available on the Google Play store and its webpage has been removed, so it appears that its development has been discontinued.

BLE Mesh Networking Bluetooth itself provides a specification for building mesh networks based on Bluetooth Low Energy that is referred to as the Bluetooth Mesh Profile [59]. While it defines a robust model for implementing a flood-based network for up to 32,000 participating nodes, its focus is not on messaging but rather connectivity of low-power IoT devices within smart homes or smart cities. As a result, it is more suitable for networks that are managed centrally and whose topology is stable over time, which is the opposite of the unpredictable and always-changing flow of a crowd during a mass protest. Further, it makes heavy use of the advertising bearer (a feature not widely available in smartphones), which imposes constraints on the bandwidth of the network – messages can have a maximum size of 384 bytes, and nodes are advised to not transmit more than 100 messages in any 10 second window. The profile makes use of cryptography for securing the network from outside observers as well as from outside interference, but it does expect participating nodes to be benign, which cannot be assumed in the messaging setting. From within the network, a malicious node can not only observe but also impersonate other nodes and deny them service.

HypeLabs The Hype SDK offered by HypeLabs [38] sets out a similar goal as Bridgefy, which is to offer secure mesh networks for a variety of purposes when there is no Internet connection. Besides Bluetooth, it also utilises Wi-Fi, and supports a variety of platforms. Among its use cases, the Hype SDK whitepaper [37] lists connectivity between IoT devices, social networking and messaging, distributed storage as well as connectivity during catastrophes and emergency broadcasting. While an example chat application is available on Google Play (with only 100+ downloads), HypeLabs does not offer the end-user solutions for those use cases themselves, merely offering the SDK as a paid product for developers. There is no information available on what applications are using the SDK, if any.

Briar Briar [56] describes itself as "secure messaging, anywhere" [56] and is referenced in online discussions on the use of mesh networking applications in protests [51]. However, Briar does not realise a mesh network. Instead it opens point-to-point sockets over a Bluetooth Classic (as opposed to Low Energy) channel to nearby nodes. Its reach is thus limited to one hop unless users manually forward messages.

Serval Serval Mesh [30] is an Android application implementing a mesh network using Wi-Fi that sets its goal as enabling communication in places which lack infrastructure. Originally developed for natural disasters, the project includes special hardware Mesh Extenders that are supposed to enhance coverage. While the application is available for download, it cannot be accessed from Google Play because it targets an old version of Android to allow it to run on older devices such as the ones primarily used in rural communities. Work on the project is still ongoing, as it is not ready for deployment at scale. Hence its utility in large-scale protests where access to technology itself is not a barrier is currently limited.

Subnodes The use of additional hardware devices enables a different approach to maintaining connectivity, which is taken by the open source project Subnodes [66]. It allows local area wireless networks to be set up on a Raspberry Pi, which then acts as a web server that can provide e.g. a chat room. Multiple devices can be connected in a mesh using the BATMAN routing protocol [54], which is meant for dynamic and unreliable networks. However, setting up and operating such a network requires technical knowledge. In the setting of a protest, even carrying the hardware device for one of the network's access points could put the operator at risk.

3 Bridgefy architecture

In this section, we give an overview of the Bridgefy messaging architecture. The key feature of Bridgefy is that it exchanges data using Bluetooth when an Internet connection is not available. The application can send the following kinds of messages:

- one-to-one messages between two parties
  - sent over the Internet if both parties are online,
  - sent directly via Bluetooth if the parties are in physical range, or
  - sent over the Bluetooth mesh network, and
- Bluetooth broadcast messages that anyone can read in a special "Broadcast mode" room.

Note that the Bluetooth messages are handled separately from the ones exchanged over the Internet using the Bridgefy server, i.e. there is no support for communication between one user who is on the Internet and one who is on the mesh network.

3.1 Bluetooth messages

Bridgefy supports connections over both BLE and Bluetooth Classic, but the latter is a legacy option for devices without BLE support, so we focus on BLE. How the Generic Attribute Profile (GATT) protocol is configured is not relevant for our analysis, so we only consider message processing starting from and up to

characteristic read and write requests. BLE packet data is received as an array of bytes, which is parsed according to the MessagePack format and processed further based on message type. At the topmost level, all messages are represented as a BleEntity which has a given entity type et. Table 1 matches the entity type to the type of its content ct and the class that implements it. Details of all classes representing messages can be found in Figures 6, 8, 7 and 9 in Appendix A.

Table 1: Entity types.

BleEntity types

et	content	class for ct
0	handshake	BleHandshake
1	direct message	BleEntityContent
3	mesh message	ForwardTransaction

AppEntity types

et	content	extending class
0	encrypted handshake	AppEntityHandShake
1	any message	AppEntityMessage
4	receipt	AppEntitySignal

Encryption scheme One-to-one Bluetooth (mesh and direct) messages in Bridgefy, represented as MessagePacks, are first compressed using Gzip and then encrypted using RSA with PKCS#1 v1.5 padding. The key size is 2048 bits and the input is split into blocks of size up to 245 bytes and encrypted one-by-one in an ECB-like fashion using Java SE's "RSA/ECB/PKCS1Padding", producing output blocks of size 256 bytes. Decryption errors do not produce a user or network visible direct error message.

Direct messages Messages sent to a user who is in direct Bluetooth range have et = 1 and hence the content ct is of type BleEntityContent. Upon reception, its payload is decrypted, decompressed and then used to construct the content of a Message object. Note that the receiver does not parse the sender ID from the message itself. Instead, it sets the sender to be the user ID which corresponds to the device from which it received the message. This link between user IDs and Bluetooth devices is determined during the initial handshake that we describe in Section 3.2.

The content of the Message object is parsed into an AppEntity, which also contains an entity type et that determines the final class of the message. A direct message has et = 1 here as well, so it is parsed as an AppEntityMessage. Afterwards, a delivery receipt for the message that was received is sent and the message is displayed to the user. Receipts take the format of AppEntitySignal: one is sent when a message is delivered as described above, and another one when the user views the chat containing the message.

Mesh messages Bridgefy implements a managed flood-based mesh network, preventing infinite loops using a time-to-live counter that is decremented whenever a packet is forwarded; when it reaches zero the packet is discarded. Messages that are transmitted using the mesh network, whether it is one-to-one messages encrypted to a user that is not in direct range, or unencrypted broadcast messages that anyone can read, have $\mathsf{et} = 3$. Such a BleEntity may hold multiple mesh messages of either kind. We note that these contain the sender and the receiver of one-to-one messages in plaintext.

The received one-to-one mesh messages are processed depending on the receiver ID – if it matches the client's user ID, they will try to decrypt the message, triggering the same processing as in the case of direct messages, and also send a special "mesh reach" message that signals that the encrypted message has found its recipient over the mesh. If the receiver ID does not match, the packet is added to the set of packets that will be forwarded to the mesh.

The received broadcast messages are first sent to the mesh. Then the client constructs AppEntityMessages and processes them the same as one-to-one messages before displaying them.

3.2 Handshake protocol

Clients establish a session by running a handshake protocol, whose messages follow the BleEntity form with $\mathsf{et} = 0$. The content of the entity is parsed as a BleHandshake which contains a request type rq and response rp. The handshake protocol is best understood as an exchange of requests and responses such that each message consists of a response to the previous request bundled with the next request. There are three types of requests:

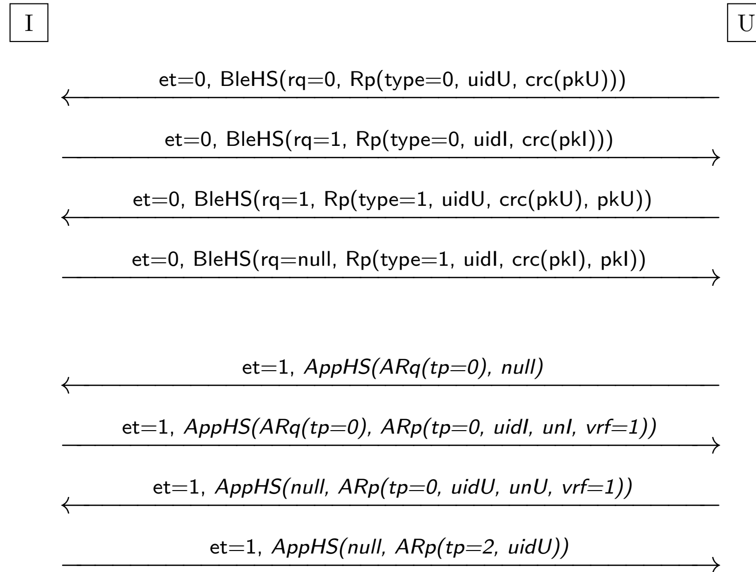
- $\mathsf{rq} = \mathsf{null}$: no request,
- $\mathsf{rq} = 0$: general request for user's information,
- $\mathsf{rq} = 1$: request for user's public key.

The first handshake message that is sent when a new BLE device is detected, regardless of whether they have communicated before, has $\mathsf{rq} = 0$ and also contains the user's ID, supported versions of the SDK and the CRC32 of the user's public key. The processing of received handshake messages depends on whether the two users know each other's public keys (either because they have connected before, or because they are contacts and the server supplied the keys when they were connected to the Internet).

Key exchange In the case when the parties do not have each other's public keys, this exchange is illustrated in Figure 1: Ivan is already online and scanning for other users when Ursula comes into range and initiates the handshake. The protocol can be understood to consist of two main parts, first the key exchange that occurs in plaintext, and second an encrypted "application handshake" which exchanges information such as usernames and phone numbers. Before the second

part begins, the devices may also exchange recent mesh messages that the device that was offline may have missed.[2]

Fig. 1: Handshake protocol including key exchange between Ivan and Ursula.



We abbreviate HandShake with HS. Here uidI, uidU are user IDs, pkI, pkU are the public keys and unI, unU are usernames of Ivan and Ursula, and crc(pkU) $>$ crc(pkI). Messages in *italics* are encrypted.

In Figure 1 some fields of the objects are omitted for clarity. Rp represents a response object (ResponseJson) while ARq and ARp are application requests and responses (AppRequestJson and AppResponseJson). The AppHandShake(rq, rp) object is wrapped in an AppEntityHandShake which forms the content of the Message that is actually compressed and encrypted. Note that the order of who initialises the BleHandshake depends on which user came online later, while the first AppHandShake is sent by the party whose CRC32 of their public key has a larger value. We are also only displaying the case when a user has not verified their phone number (which is the default behaviour in the application), i.e. vrf=1. If they have, AppHandShake additionally includes a request and a response for the phone number.

[2] This is facilitated by the dump flag in ForwardTransaction, but we omit this exchange in the figure as it is not relevant to the actual handshake protocol.

Known keys In the case when both parties already know each other's public keys, there are only two BleHandshake messages exchanged, and both follow the format of the first message shown in Figure 1, where $\mathsf{rq} = 0$. The exchange of encrypted AppHandShake messages then continues unchanged.

Conditions When two devices come into range, the handshake protocol is executed automatically and direct messages can only be sent after the handshake is complete. Only clients in physical range can execute the BleHandShake part of the protocol. Devices that are communicating via the mesh network do not perform the handshake at all, so they can only exchange messages if they already know each other's keys from the Bridgefy server or because they have been in range once before.

3.3 Routing via the Bridgefy server

An Internet connection is required when a user first installs the application, which registers them with the Bridgefy server. All requests are done via HTTPS, the APIs for which are in the package me.bridgefy.backend.v3.

The BgfyUser class that models the information that is sent to the server during registration contains the user's chosen name, user ID, the list of users blocked by this user, and if they are "verified" then also their phone number. Afterwards, a contacts request is done every time an Internet connection is available (regardless of whether a user is verified or not) and the user refreshes the application. The phone numbers of the user's contacts are uploaded to the server to obtain a list of contacts that are also Bridgefy users. BgfyKeyApi then provides methods to store and retrieve the users' public keys from the server.

Messages sent between online users are of a simpler form than the Bluetooth messages: an instance of BgfyMessage contains the sender and receiver IDs, the RSA encryption of the text of the message and some metadata, such as a timestamp and the delivered/read status of the message in plaintext. The server will queue messages sent to users who are not currently online until they connect to the Internet again.

4 Attacks

In this section, we show that Bridgefy does not provide confidentiality of messages and also that it does not satisfy the additional security needs arising in a protest setting: privacy, authenticity and reliability in adversarial settings.

4.1 Privacy

Here, we discuss vulnerabilities in Bridgefy pertaining to user privacy in contrast to confidentiality of messages. We note that Bridgefy initially made no claim about anonymity in its marketing but disabled mandating phone number verification to address anonymity needs in 2019 [73].

Local user tracking To prevent tracking, Bluetooth-enabled devices may use "random" addresses which change over time (for details on the addressing scheme see [13, Section 10.8]). However, when a Bridgefy client sends BLE ADV_IND packets (something that is done continuously while the application is running), it transmits an identifier in the service data that is the CRC32 value of its user ID, encoded in 10 bytes as decimal digits. The user ID does not change unless the user reinstalls the application, so passive observation of the network is enough to enable tracking all users.

In addition, the automatic handshake protocol composed with public-key caching provides a mechanism to perform historical contact tracing. If the devices of two users have been in range before, they will not request each other's public keys, but they will do so automatically if that has not been the case.

Participant discovery Until December 2019 [73], Bridgefy required users to register with a phone number. Users still have the option to do so, but it is no longer the default. If the user gives the permission to the application to access the contacts stored on their phone, the application will check which of those contacts are already Bridgefy users based on phone numbers and display those contacts to the user. When Bridgefy is predominantly installed on phones of protesters, this allows the identification of participants by running contact discovery against all local phone numbers. While an adversary with significant control over the network, such as a state actor, might have alternative means to collect such information, this approach is also available to e.g. employers or activists supporting the other side.

Social graph All one-to-one messages sent over the mesh network contain the sender and receiver IDs in plaintext, so a passive adversary with physical presence can build a social graph of what IDs are communicating with whom. The adversary can further use the server's API to learn the usernames corresponding to those IDs (via the getUserById request in BgfyUserApi). In addition, since three receipts are sent when a message is received – "mesh reach" in clear, encrypted "delivery" receipt, encrypted "viewed" receipt – a passive attacker can also build an approximate, dynamic topology of the network, since users that are further away from each other will have a larger delay between a message and its receipts.

4.2 Authenticity

Bridgefy does not utilise cryptographic authentication mechanisms. As a result, an adversary can impersonate any user.

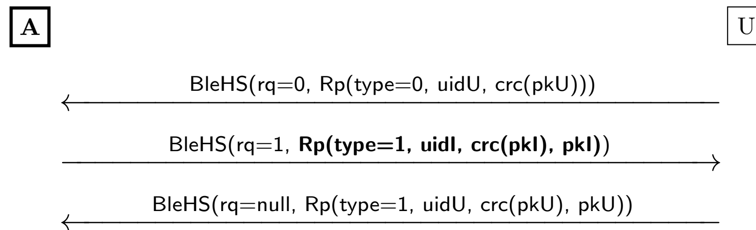
The initial handshake through which parties exchange their public keys or identify each other after coming in range relies on two pieces of information to establish the identities: a user ID and the lower-level Bluetooth device address. Neither of these is an actual authentication mechanism: the user ID is public information which can be learned from observing the network, while [57] shows that it is possible to broadcast with any BLE device address.

However, an attacker does not need to go to such lengths. Spoofing can be done by sending a handshake message which triggers the other side to overwrite the information it currently has associated with a given user. Suppose there are two users who have communicated with each other before, Ursula and Ivan, and the attacker wishes to impersonate Ivan to Ursula. When the attacker comes into range of Ursula, she will initiate the handshake. The attacker will send a response of type 1, simply replacing its own user ID, public key and the CRC of its public key with Ivan's, and also copies Ivan's username, as shown in Figure 2.

This works because the processing of handshakes in Bridgefy is not stateful and parts of the handshake such as the request value rq and type of rp act as control messages. This handshake is enough for Ursula's application to merge the attacker and Ivan into one user, and therefore show messages from the attacker as if they came from Ivan. If the real Ivan comes in range at the time the attacker is connected to Ursula, he will be able to communicate with her and receive responses from her that the attacker will not be able to decrypt. However, he will not be able to see the attacker's presence. We implemented this attack and verified that it works, see Section 4.3.

The messages exchanged over the mesh network (when users are not in direct range) merely contain the user ID of the sender, so they can be spoofed with ease. We also note that although the handshake protocol is meant for parties in range, the second part of the handshake (i.e. AppHandshake) can also be sent over the mesh network. This means that users can be convinced to change the usernames and phone numbers associated with their Bridgefy contacts via the mesh network.

Fig. 2: Impersonation attack, with attacker modifications in **bold**.



4.3 Confidentiality

Confidentiality of message contents is both a security goal mentioned in Bridgefy's marketing material and relied upon by participants in protests. In this section, we show that the implemented protections are not sufficient to satisfy this goal.

IND-CPA Bridgefy's encryption scheme only offers a security level of 2^{64} in a standard IND-CPA security game, i.e. a passive adversary can decide whether a message m_0 or m_1 of its choosing was encrypted as c. The adversary picks messages of length 245 bytes and tries all 255^8 possible values for PKCS#1 v1.5 padding until it finds a match for the challenge ciphertext c.

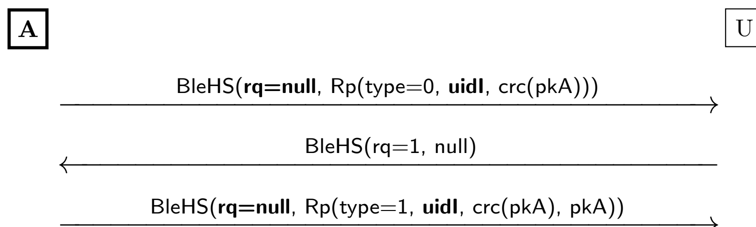
Plaintext file sharing Bridgefy allows its users to send direct messages composed of either just text or containing a location they want to share. The latter is processed as a special text message containing coordinates, and so these two types are encrypted, but the same is not true for any additional data such as image files. Only the payload of the BleEntityContent is encrypted, which does not include the byte array BleEntity.data that is used to transmit files. While the application itself does not currently offer the functionality to share images or other files, it is part of the SDK and receiving media files does work in the application. The fact that files are transmitted in plaintext is not stated in the documentation, so for developers using the SDK it would be easy to assume that files are shared privately when using this functionality.

MITM This attack is an extension of the impersonation attack described in Section 4.2 where we convince the client to change the public key for any user ID it has already communicated with. Suppose that Ivan is out of range, and the attacker initiates a handshake with Ursula where rq = null, rp is of type 0 and contains the CRC of the attacker's key as well as Ivan's user ID as the sender ID (the user ID being replaced in all following handshake messages as well). The logic of the handshake processing in Ursula's client dictates that since the CRC does not match the CRC of Ivan's key that it has saved, it has to make a request of type 1, i.e. a request for an updated public key. Then the attacker only needs to supply its own key, which will get associated with Ivan's user ID, as shown in Figure 3. Afterwards, whenever Ursula sends a Bluetooth message to Ivan, it will be encrypted under the attacker's key. Further, Ursula's client will display messages from the attacker as if they came from Ivan, so this attack also provides impersonation. If at this stage Ivan comes back in range, he will not be able to connect to Ursula. The attack is not persistent, though – if the attacker goes out of range, Ivan (when in range) can run a legitimate handshake and restore communication.

We verified this and the previous impersonation attack in a setup with four Android devices, where the attacker had two devices running Frida scripts that modified the relevant handshake messages. Two attacker devices were used to instantiate a full attacker in the middle attack, which is an artefact of us hotpatching the Bridgefy application using Frida scripts: one device to communicate with Ursula on behalf of Ivan and another with Ivan on behalf of Ursula.

We also note that since the Bridgefy server serves as a trusted database of users' public keys, if compromised, it would be trivial to mount an attacker in the middle attack on the communication of any two users. This would also impact

Fig. 3: One side of the MITM attack, with attacker modifications in **bold**.



users who are planning to only use the application offline since the server would only need to supply them the wrong keys during registration.

Padding oracle attack The following chosen ciphertext attack is enabled by the fact that all one-to-one messages use public-key encryption but no authentication, so we can construct valid ciphertexts as if coming from any sender. We can also track BLE packets and replay them at will, reordering or substituting ciphertext blocks.

We instantiate a variant of Bleichenbacher's attack [12] on RSA with PKCS#1 v1.5 padding using Bridgefy's delivery receipts. This attack relies on distinguishing whether a ciphertext was processed successfully or not. The receiver of a message sends a message status update when a message has been received and processed, i.e. decrypted, decompressed and parsed. If there was an error on the receiver's side, no message is sent. No other indication of successful delivery or (type of) error is sent. Since the sender of a Bridgefy message cannot distinguish between decryption errors or decompression errors merely from the information it gets from the receiver, we construct a padding oracle that circumvents this issue.

Suppose that Ivan sends a ciphertext c encrypting the message m to Ursula that we intercept. In the classical Bleichenbacher's attack, we would form a new ciphertext $c^* = s^e \cdot c \mod n$ for some s where n is the modulus and e is the exponent of Ursula's public key. Now suppose that c^* has a correct padding. Since messages are processed in blocks, we can prepend and append valid ciphertexts. These are guaranteed to pass the padding checks as they are honestly generated ciphertexts (we recall that there is no authentication). We will construct these blocks in such a way that decompression of the joint contents will succeed with non-negligible probability, and therefore enable us to get a delivery receipt which will instantiate our padding oracle.

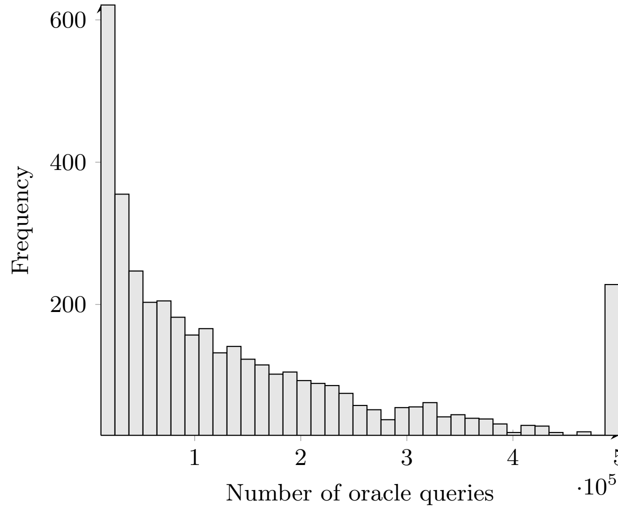
The Gzip file format [40] specifies a number of optional flags. If the flag FLG.FCOMMENT is set, the header is followed by a number of "comment" bytes, terminated with a zero byte, that are essentially ignored. In particular, these bytes are not covered by the CRC32 checksum contained in the Gzip trailer. Thus, we let c_0 be the encryption of a 10-byte Gzip header with this flag set followed by up to 245 non-zero bytes, and let c_1 be the encryption of a zero byte

followed by a valid compressed MessagePack payload (i.e. of a message from the attacker to Ursula) and Gzip trailer.

When put together, $c_0||c^*||c_1$ encrypts a correctly compressed message as long as $\text{unpad}(s \cdot \text{pad}(m))$ (which is part of the comment field) does not contain a zero byte, and therefore Ursula will send a delivery receipt for the attacker's message. The probability that the comment does not contain a zero byte for random s is $\geq (1 - \frac{1}{256})^{245} \approx 0.383$.

To study the number of adaptively chosen ciphertexts required, we adapted the simulation code from [16] for the Bleichenbacher-style oracle encountered in this attack: a payload will pass the test if it has valid padding for messages of any valid length ("FFT" in [7] parlance) and if it does not contain a zero byte in the "message" part after splitting off the padding. We then ran a Bleichenbacher-style attack $4,096$ times (on 80 cores, taking about 12h in total) and recorded how often the oracle was called in each attack. We give a histogram of our data in Figure 4. The median is $2^{16.75}$, the mean $2^{17.36}$. Our SageMath [65] script, based on Python code in [16], and the raw data for Figure 4 are attached to the electronic version of this document.

Fig. 4: Density distribution for number of ciphertexts required to mount a padding-oracle attack via Gzip comments.



We have verified the applicability of this attack in Bridgefy using ciphertexts c^* constructed to be PKCS#1 v1.5-conforming (i.e. where we set $s = \text{pad}(m)^{-1} \cdot \text{pad}(r) \mod n$ where r is 245 random bytes). We used Frida to run a script on the attacker's device that would send $c_0||c^*||c_1$ to the target Bridgefy user via Bluetooth, and record whether it gets a delivery receipt for the message contained in c_1 or not. The observed frequency of the receipts matched the probability

given earlier. This oracle suffices to instantiate Bleichenbacher's original attack. In our preliminary experiments we were able to send a ciphertext every 450ms, suggesting 50% of attacks complete in less than 14 hours. We note, however, that our timings are based on us hotpatching Bridgefy to send the messages and that a higher throughput might be achievable.

Padding oracles from a timing side-channel Our decompression oracle depends on the Bridgefy SDK processing three blocks as a joint ciphertext. While we verified that this behaviour is also exhibited by the Bridgefy application, the application itself never sends ciphertexts that span more than two blocks as it imposes a limit of 256 bytes on the size of the text of each message. Thus, a stopgap mitigation of our previous attack could be to disable the processing of more than two blocks of ciphertext jointly together.

We sketch an alternative attack that only requires two blocks of ciphertext per message. It is enabled by the fact that when a receiver processes an incorrect message, there is a difference in the time it takes to process it depending on what kind of error was encountered. This difference is clearly observable for ciphertexts that consist of at least two blocks, where the error occurs in the first block. We note that padding errors occurring in the second block can be observed by swapping the blocks, as they are decrypted individually.

Figure 5 (raw data is attached to the electronic version of this document) shows the differences for experiments run on the target device, measured using Frida. A script was injected into the Bridgefy application that would call the method responsible for extracting a message from a received BLE packet (including decryption and decompression) on given valid or invalid data. The execution time of this method was measured directly on the device using Java.

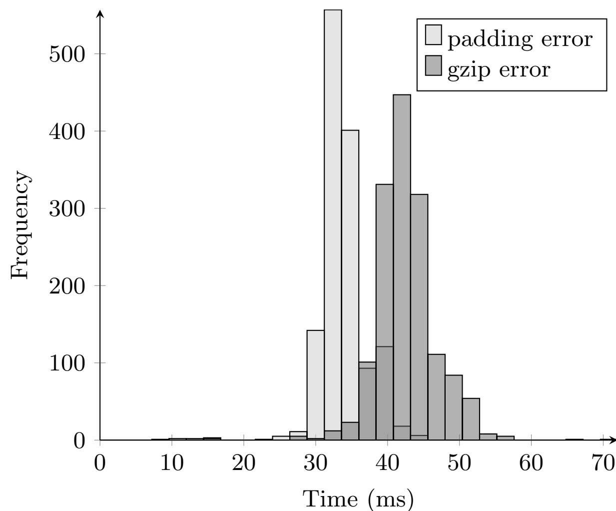
If multiple messages are received, they are processed sequentially, which enables the propagation of these timing differences to the network level. That is, the attacker sends two messages, one consisting of $c^*||c'$ where $c^* = s^e \cdot c \bmod n$ is the modified target ciphertext and c' is an arbitrary ciphertext block, and one consisting of some unrelated message, either as direct messages one after another or a mesh transaction containing both messages among its packets. The side-channel being considered is then simply the time it takes to receive the delivery receipt on the second valid message.

We leave exploring whether this could be instantiated in practice to future work, since our previous attacks do not require this timing channel. We note, though, that an adversary would likely need more precise control over the timing of when packets are released than that offered by stock Android devices in order to capture the correct difference in a BLE environment.

4.4 Denial of service

Bridgefy's appeal to protesters to enable messaging in light of an Internet shutdown makes resilience to denial of service attacks a key concern. While a flood-based network can be resilient as a consequence of its simplicity, some particularities of the Bridgefy setup make it vulnerable.

Fig. 5: Execution time of ChunkUtils.stitchChunksToEntity for 2 ciphertext blocks in milliseconds. In the table, N is the number of samples in each experiment.



error type	N	μ	σ	$\sigma/\sqrt{N}$
bad padding	1360	33.882956	3.137260	0.085071
gzip error	1508	42.557275	4.273194	0.110040

Broad DoS Due to the use of compression, Bridgefy is vulnerable to "zip bomb" attacks. In particular, compressing a message of size 10MB containing a repeated single character results in a payload of size 10KB, which can be easily transmitted over the BLE mesh network. Then, when the client attempts to display this message, the application becomes unresponsive to the point of requiring reinstallation to make it usable again. Sending such a message to the broadcast chat provides a trivial way of disabling many clients at the same time, since clients will first forward the message further and only then start the processing to display it which causes them to hang. As a consequence, a single adversarially generated message can take down the entire network. We implemented this attack and tested it in practice on a number of Android devices.

Targeted DoS A consequence of the MITM attack from Section 4.3 is that it provides a way to prevent given two users from connecting, even if they are in Bluetooth range, since the attacker's key becomes attached to one of the user ids.

5 Discussion

While our attacks reveal severe deficiencies in the security of both the Bridgefy application (v2.1.28) and the SDK (v1.0.6), it is natural to ask whether they are valid and what lessons can be drawn from them for cryptographic research.

Given that most of our attacks are variants of attacks known in the literature, it is worth asking why Bridgefy did not mitigate against them. A simple answer to this question might be that the application was not designed for adversarial settings and that therefore our attacks are out of scope, externally imposing security goals. However, such an account would fail to note that Bridgefy's security falls short also in settings where attacks are not expected to be the norm, i.e. Bridgefy does not satisfy standard privacy guarantees expected of any modern messaging application. In particular, prior to our work, Bridgefy developers advertised the app/SDK as "private" and as featuring end-to-end encryption; our attacks thus broke Bridgefy's own security claims.

More importantly, however, Bridgefy *is* used in highly adversarial settings where its security ought to stand up to powerful nation-state adversaries and the Bridgefy developers advertise their application for these contexts [46,72,74,75]. Technologies need to be evaluated under the conditions they are used in. Here, our attacks highlight the value of secure by design approaches to development. While designers might envision certain use cases, users, in the absence of alternatives, may reach for whatever solution is available.

Our work thus draws attention to this problem space. While it is difficult to assess the actual reliance of protesters on mesh communication, the *idea* of resilient communication in the face of a government-mandated Internet shutdown is present throughout protests across the globe [8,11,43,74,77,46,10,74,75,47,79,71]. Yet, these users are not well served by the existing solutions they rely on. Thus, it is a pressing topic for future work to design communication protocols and tools that cater to these needs. We note, though, that this requires understanding "these needs" to avoid a disconnect between what designers design for and what users in these settings require [26,34].

5.1 Responsible disclosure

We disclosed the vulnerabilities described in this work to the Bridgefy developers on 27 April 2020 and they acknowledged receipt on the same day. We agreed on a public disclosure date of 24 August 2020. Starting from 1 June 2020, the Bridgefy team began informing their users that they should not expect confidentiality guarantees from the current version of the application [80]. On 8 July 2020, the developers informed us that they were implementing a switch to the Signal protocol to provide cryptographic assurances in their SDK. On 24 August 2020, we published an abridged[3] version of this paper in conjunction with a media article [33]. The Bridgefy team published a statement on the same day [19]. On 30 October 2020, an update finalising the switch to Signal was released [21]. If implemented correctly, it would rule out many of the attacks described in this work. Note, however, that we have not reviewed these changes and we recommend an independent security audit to verify they have been implemented correctly.

[3] We had omitted details of the Bridgefy architecture, as the attacks had not been mitigated at that point in time.

Acknowledgements

Part of this work was done while Albrecht was visiting the Simons Institute for the Theory of Computing. The research of Mareková was supported by the EPSRC and the UK Government as part of the Centre for Doctoral Training in Cyber Security at Royal Holloway, University of London (EP/P009301/1). We thank Kenny Paterson and Eamonn Postlethwaite for comments on an earlier version of this paper.

References

1. Bridgefy. https://web.archive.org/web/20200411143157/https://www.bridgefy.me/ (Apr 2020)
2. Adomnicai, A., Fournier, J.J.A., Masson, L.: Hardware security threats against Bluetooth Mesh networks. In: 2018 IEEE Conference on Communications and Network Security, CNS 2018, Beijing, China, May 30 - June 1, 2018. pp. 1–9. IEEE (2018), https://doi.org/10.1109/CNS.2018.8433184
3. Álvarez, F., Almon, L., Hahn, A., Hollick, M.: Toxic friends in your network: Breaking the Bluetooth Mesh friendship concept. In: Mehrnezhad, M., van der Merwe, T., Hao, F. (eds.) Proceedings of the 5th ACM Workshop on Security Standardisation Research Workshop, London, UK, November 11, 2019. pp. 1–12. ACM (2019), https://doi.org/10.1145/3338500.3360334
4. Alwen, J., Capretto, M., Cueto, M., Kamath, C., Klein, K., Pascual-Perez, G., Pietrzak, K., Walter, M.: Keep the dirt: Tainted treekem, an efficient and provably secure continuous group key agreement protocol. Cryptology ePrint Archive, Report 2019/1489 (2019), https://eprint.iacr.org/2019/1489
5. Alwen, J., Coretti, S., Dodis, Y., Tselekounis, Y.: Security analysis and improvements for the IETF MLS standard for group messaging. Cryptology ePrint Archive, Report 2019/1189 (2019), https://eprint.iacr.org/2019/1189
6. Aviram, N., Schinzel, S., Somorovsky, J., Heninger, N., Dankel, M., Steube, J., Valenta, L., Adrian, D., Halderman, J.A., Dukhovni, V., Käsper, E., Cohney, S., Engels, S., Paar, C., Shavitt, Y.: DROWN: Breaking TLS using SSLv2. In: Holz and Savage [36], pp. 689–706
7. Bardou, R., Focardi, R., Kawamoto, Y., Simionato, L., Steel, G., Tsay, J.K.: Efficient padding oracle attacks on cryptographic hardware. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 608–625. Springer, Heidelberg (Aug 2012). https://doi.org/10.1007/978-3-642-32009-5_36
8. BBC News: Iraqis use Firechat messaging app to overcome net block. http://web.archive.org/web/20190325080943/https://www.bbc.com/news/technology-27994309k (Jun 2014)
9. Becker, J.K., Li, D., Starobinski, D.: Tracking anonymized Bluetooth devices. Proceedings on Privacy Enhancing Technologies **2019**(3), 50–65 (2019)
10. Bhavani, D.K.: Internet shutdown? why Bridgefy app that enables offline messaging is trending in India. http://web.archive.org/web/20200105053448/https://www.thehindu.com/sci-tech/technology/internet-shutdown-why-bridgefy-app-that-enables-offline-messaging-is-trending-in-india/article30336067.ece (Dec 2019)
11. Bland, A.: FireChat – the messaging app that's powering the Hong Kong protests. http://web.archive.org/web/20200328142327/https://www.theguardian.com/world/2014/sep/29/firechat-messaging-app-powering-hong-kong-protests (Sep 2014)

12. Bleichenbacher, D.: Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In: Krawczyk, H. (ed.) CRYPTO'98. LNCS, vol. 1462, pp. 1–12. Springer, Heidelberg (Aug 1998). https://doi.org/10.1007/BFb0055716
13. Bluetooth SIG: Core specification 5.1. https://www.bluetooth.com/specifications/bluetooth-core-specification/ (Jan 2019)
14. Böck, H., Somorovsky, J., Young, C.: Return of bleichenbacher's oracle threat (ROBOT). In: Enck, W., Felt, A.P. (eds.) USENIX Security 2018. pp. 817–849. USENIX Association (Aug 2018)
15. Borak, M.: We tested a messaging app used by Hong Kong protesters that works without an internet connection. http://web.archive.org/web/20191206182048/https://www.abacusnews.com/digital-life/we-tested-messaging-app-used-hong-kong-protesters-works-without-internet-connection/article/3025661 (Sep 2019)
16. Boyle, G.: 20 years of Bleichenbacher attacks. Tech. Rep. RHUL–ISG–2019–1, Information Security Group, Royal Holloway University of London (2019)
17. Brewster, T.: Hong Kong protesters are using this 'mesh' messaging app—but should they trust it? http://web.archive.org/web/20191219071731/https://www.forbes.com/sites/thomasbrewster/2019/09/04/hong-kong-protesters-are-using-this-mesh-messaging-app--but-should-they-trust-it/ (Sep 2019)
18. Bridgefy: Developers. https://blog.bridgefy.me/developers.html, https://archive.vn/yjg9f
19. Bridgefy: Bridgefy's commitment to privacy and security. http://web.archive.org/web/20200826183604/https://bridgefy.me/bridgefys-commitment-to-privacy-and-security/ (Aug 2020)
20. Bridgefy: Offline messaging. https://web.archive.org/20200411143133/https://play.google.com/store/apps/details?id=me.bridgefy.main (Apr 2020)
21. Bridgefy: Technical article on our security updates. http://web.archive.org/web/20201102093540/https://bridgefy.me/technical-article-on-our-security-updates/ (Nov 2020)
22. Cortés, V.: Bridgefy sees massive spike in downloads during Hong Kong protests. http://web.archive.org/web/20191013072633/http://www.contxto.com/en/mexico/mexican-bridgefy-sees-massive-spike-in-downloads-during-hong-kong-protests/ (Aug 2019)
23. Cremers, C., Hale, B., Kohbrok, K.: Efficient post-compromise security beyond one group. Cryptology ePrint Archive, Report 2019/477 (2019), https://eprint.iacr.org/2019/477
24. Dunning, J.P.: Taming the blue beast: A survey of bluetooth based threats. IEEE Secur. Priv. **8**(2), 20–27 (2010), https://doi.org/10.1109/MSP.2010.3
25. Duong, T., Rizzo, J.: The CRIME attack. Presentation at ekoparty Security Conference (2012)
26. Ermoshina, K., Halpin, H., Musiani, F.: Can johnny build a protocol? co-ordinating developer and user intentions for privacy-enhanced secure messaging protocols. In: European Workshop on Usable Security (2017)
27. Fifield, D.: A better zip bomb. In: 13th USENIX Workshop on Offensive Technologies (WOOT 19). USENIX Association, Santa Clara, CA (Aug 2019)
28. Frida: A dynamic instrumentation framework, v12.8.9. https://frida.re/ (Feb 2020)
29. Furuhashi, S.: MessagePack. https://msgpack.org/ (2008)
30. Gardner-Stephen, P.: The Serval project. http://www.servalproject.org/ (2017)
31. Garman, C., Green, M., Kaptchuk, G., Miers, I., Rushanan, M.: Dancing on the lip of the volcano: Chosen ciphertext attacks on apple iMessage. In: Holz and Savage [36], pp. 655–672

32. Gluck, Y., Harris, N., Prado, A.: BREACH: reviving the CRIME attack. Black Hat USA (2013)
33. Goodin, D.: Bridgefy, the messenger promoted for mass protests, is a privacy disaster. https://arstechnica.com/features/2020/08/bridgefy-the-app-promoted-for-mass-protests-is-a-privacy-disaster/ (Aug 2020)
34. Halpin, H., Ermoshina, K., Musiani, F.: Co-ordinating developers and high-risk users of privacy-enhanced secure messaging protocols. In: Cremers, C., Lehmann, A. (eds.) Security Standardisation Research - 4th International Conference, SSR 2018. Lecture Notes in Computer Science, vol. 11322, pp. 56–75. Springer (2018), https://doi.org/10.1007/978-3-030-04762-7_4
35. Hassan, S.S., Bibon, S.D., Hossain, M.S., Atiquzzaman, M.: Security threats in Bluetooth technology. Comput. Secur. **74**, 308–322 (2018), https://doi.org/10.1016/j.cose.2017.03.008
36. Holz, T., Savage, S. (eds.): USENIX Security 2016. USENIX Association (Aug 2016)
37. HypeLabs: The Hype SDK: A technical overview. https://hypelabs.io/documents/Hype-SDK.pdf (2019)
38. HypeLabs: https://hypelabs.io (2020)
39. IETF: DEFLATE compressed data format specification version 1.3. https://tools.ietf.org/html/rfc1951 (May 1996)
40. IETF: GZIP file format specification version 4.3. https://tools.ietf.org/html/rfc1952 (May 1996)
41. IETF: PKCS #1: RSA encryption version 1.5. https://tools.ietf.org/html/rfc2313 (March 1998)
42. Jasek, S.: GATTacking Bluetooth smart devices. https://github.com/securing/docs/raw/master/whitepaper.pdf (2016)
43. Josh Horwitz, T.i.A.: Unblockable? unstoppable? FireChat messaging app unites China and Taiwan in free speech… and it's not pretty. http://web.archive.org/web/20141027180653/https://www.techinasia.com/unblockable-unstoppable-firechat-messaging-app-unites-china-and-taiwan-in-free-speech-and-its-not-pretty/ (Mar 2014)
44. Kelsey, J.: Compression and information leakage of plaintext. In: Daemen, J., Rijmen, V. (eds.) FSE 2002. LNCS, vol. 2365, pp. 263–276. Springer, Heidelberg (Feb 2002). https://doi.org/10.1007/3-540-45661-9_21
45. Klíma, V., Pokorný, O., Rosa, T.: Attacking RSA-based sessions in SSL/TLS. In: Walter, C.D., Koç, Çetin Kaya., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 426–440. Springer, Heidelberg (Sep 2003). https://doi.org/10.1007/978-3-540-45238-6_33
46. Koetsier, J.: Hong Kong protestors using mesh messaging app china can't block: Usage up 3685%. https://web.archive.org/web/20200411154603/https://www.forbes.com/sites/johnkoetsier/2019/09/02/hong-kong-protestors-using-mesh-messaging-app-china-cant-block-usage-up-3685/ (Sep 2019)
47. Magaisa, A.T.: https://twitter.com/wamagaisa/status/1288817111796797440 (Jul 2020), http://archive.today/DVRZf
48. Mihindukulasuriya, R.: Firechat, Bridgefy see massive rise in downloads amid internet shutdowns during caa protests. http://web.archive.org/web/20200109212954/https://theprint.in/india/firechat-bridgefy-see-massive-rise-in-downloads-amid-internet-shutdowns-during-caa-protests/340058/ (Dec 2019)
49. Mohan, P.: How the internet shutdown in Kashmir is splintering India's democracy. http://web.archive.org/web/20200408111230/https://www.fastcompany.com/

90470779/how-the-internet-shutdown-in-kashmir-is-splintering-indias-democracy (Mar 2020)
50. Mudzingwa, F.: This offline messenger that might keep you connected if the govt decides to shut down the internet. https://web.archive.org/web/20200816101930/https://www.techzim.co.zw/2020/07/bridgefy-is-an-offline-messenger-that-might-keep-you-connected-if-the-govt-decides-to-shut-down-the-internet/ (Aug 2020)
51. News, H.: Hong Kong protestors using Bridgefy's Bluetooth-based mesh network messaging app. https://web.archive.org/web/20191016114954/https://news.ycombinator.com/item?id=20861948 (Aug 2019)
52. Ng, B.: Bridgefy: A startup that enables messaging without internet. http://archive.today/2020.06.07-120425/https://www.ejinsight.com/eji/article/id/2230121/20190826-bridgefy-a-startup-that-enables-messaging-without-internet (Aug 2019)
53. Open Garden: FireChat. http://web.archive.org/web/20200111174316/https://www.opengarden.com/firechat/ (Oct 2019)
54. Open Mesh: B.A.T.M.A.N. Advanced. https://www.open-mesh.org/projects/batman-adv/wiki (2020)
55. Purohit, K.: Whatsapp to Bridgefy, what Hong Kong taught India's leaderless protesters. http://web.archive.org/web/20200406103939/https://www.scmp.com/week-asia/politics/article/3042633/whatsapp-bridgefy-what-hong-kong-taught-indias-leaderless (Dec 2019)
56. Rogers, M., Saitta, E., Grote, T., Dehm, J., Wieder, B.: Briar. https://web.archive.org/web/20191016114519/https://briarproject.org/ (Mar 2018)
57. Ryan, M.: Bluetooth: With low energy comes low security. In: Proceedings of the 7th USENIX Conference on Offensive Technologies. p. 4. WOOT'13, USENIX Association, USA (2013)
58. Schwartz, L.: The world's protest app of choice. https://restofworld.org/2020/the-worlds-protest-app-of-choice/ (Aug 2020), http://archive.today/5kOhr
59. SIG, B.: Mesh profile specification 1.0.1. https://www.bluetooth.com/specifications/mesh-specifications/ (Jan 2019)
60. Silva, M.D.: Hong Kong protestors are once again using mesh networks to preempt an internet shutdown. http://archive.today/2019.09.20-220517/https://qz.com/1701045/hong-kong-protestors-use-bridgefy-to-preempt-internet-shutdown/ (Sep 2019)
61. Sivakumaran, P., Blasco, J.: A study of the feasibility of co-located app attacks against BLE and a large-scale analysis of the current application-layer security landscape. In: Heninger, N., Traynor, P. (eds.) USENIX Security 2019. pp. 1–18. USENIX Association (Aug 2019)
62. Skylot: Jadx - Dex to Java decompiler, v1.1.0. https://github.com/skylot/jadx (Dec 2019)
63. SMEX: Lebanon protests: How to communicate securely in case of a network disruption. https://smex.org/lebanon-protests-how-to-communicate-securely-in-case-of-a-network-disruption-2/ (Oct 2019), http://archive.today/hx1lp
64. Software Freedom Law Centre, India: Internet shutdown tracker. https://internetshutdowns.in/ (2020)
65. Stein, W., et al.: Sage Mathematics Software Version 9.0. The Sage Development Team (2019), http://www.sagemath.org
66. Subnodes: Subnodes. http://subnodes.org/ (2018)
67. Sullivan, N., Turner, S., Kaduk, B., Cohn-Gordon, K., et al.: Messaging Layer Security (MLS). https://datatracker.ietf.org/wg/mls/about/ (Nov 2018)

68. Teknologiia Lebanon: Lebanese Protesters are using this 'Bridgefy' Messaging App — What is it? https://medium.com/@teknologiialb/lebanese-protesters-are-using-this-bridgefy-messaging-app-what-is-it-74614e169197 (Jan 2020), https://archive.vn/udqly
69. The Stranger: How to message people at protests even without internet access. https://www.thestranger.com/slog/2020/06/03/43829749/how-to-message-people-at-protests-even-without-internet-access (Jun 2020), http://archive.is/8UrWQ
70. Twitter: Bridgefy search. https://twitter.com/search?q=bridgefy (Jun 2020), http://archive.today/hwklY
71. Twitter – B1O15J: https://twitter.com/B1O15J/status/1294603355277336576 (Aug 2020), https://archive.vn/dkPqD
72. Twitter – Bridgefy: https://twitter.com/bridgefy/status/1197191632665415686 (Nov 2019), http://archive.today/aNKQy
73. Twitter – Bridgefy: https://twitter.com/bridgefy/status/1209924773486170113 (Dec 2019), http://archive.today/aQZDL
74. Twitter – Bridgefy: https://twitter.com/bridgefy/status/1216473058753597453 (Jan 2020), http://archive.today/x1gG4
75. Twitter – Bridgefy: https://twitter.com/bridgefy/status/1268905414248153089 (Jun 2020), http://archive.today/odSbW
76. Twitter – Bridgefy: https://twitter.com/bridgefy/status/1287768436244983808 (Jul 2020), https://archive.vn/WQfZm
77. Twitter – Bridgefy: https://twitter.com/bridgefy/status/1268015807252004864 (Jun 2020), http://archive.today/uKNRm
78. Twitter – Bridgefy: https://twitter.com/bridgefy/status/1289576487004168197 (Aug 2020), https://archive.vn/zbxgR
79. Twitter – Bridgefy: https://twitter.com/bridgefy/status/1292880821725036545 (Aug 2020), https://archive.vn/tKr0t
80. Twitter – Bridgefy: https://twitter.com/bridgefy/status/1267469099266965506 (Jun 2020), http://archive.today/40pzC
81. Uher, J., Mennecke, R.G., Farroha, B.S.: Denial of sleep attacks in Bluetooth Low Energy wireless sensor networks. In: Brand, J., Valenti, M.C., Akinpelu, A., Doshi, B.T., Gorsic, B.L. (eds.) 2016 IEEE Military Communications Conference, MILCOM 2016, Baltimore, MD, USA, November 1-3, 2016. pp. 1231–1236. IEEE (2016), https://doi.org/10.1109/MILCOM.2016.7795499
82. Wakefield, J.: Hong Kong protesters using Bluetooth Bridgefy app. http://web.archive.org/web/20200305062625/https://www.bbc.co.uk/news/technology-49565587 (Sep 2019)
83. Zuo, C., Wen, H., Lin, Z., Zhang, Y.: Automatic fingerprinting of vulnerable BLE IoT devices with static UUIDs from mobile apps. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 1469–1483. ACM (2019)

A Bridgefy message classes

These classes can be found in three packages:

- me.bridgefy.entities,
- com.bridgefy.sdk.framework.entities and

– com.bridgefy.sdk.client (the class `Message`).

In the figures, data types are Java SE data types. In particular an `int` is a 32-bit integer, a `long` is a 64-bit integer and `Integer` and `Long` are their object forms.

Fig. 6: Classes of me.bridgefy.entities.

```
Message:
  String conversation, messageId, offlineId
  String receiver, sender, otherUsername
  String dateSent, serverDate
  int messageType
  String text  // text of the actual message
  String fileName
  byte[] fileContent
  int status  // sent/delivered/read

AppHandshake:
  AppRequestJson rq
  AppResponseJson rp

AppRequestJson:
  int tp  // type
  String dt  // device type (Android)

AppResponseJson:
  int tp  // type: general, phone or finished
  String uid  // user ID
  String ph  // phone number
  String un  // username
  boolean dn  // no phone
  int vrf  // verified user
```

Fig. 7: Class com.bridgefy.sdk.client.Message.

```
Message:
  HashMap content  // payload
  String receiverId, senderId, uuid
  long dateSent
  byte[] data  // media file content
  boolean isMesh
  int hop, hops  // time to live counter
```

Fig. 8: Classes of com.bridgefy.sdk.framework.entities.

```
BleEntity:
  String id
  int et  // entity type
  T ct  // content
  byte[] binaryPart  // encrypted mesh messages
  byte[] data  // media file content

BleHandshake:
  Integer rq  // request type
  ResponseJson rp  // response

ResponseJson:
  int type  // general or key
  String uuid  // user ID
  String v, lcv  // SDK version
  long crcKey  // CRC of public key
  int dt
  String key  // public key

BleEntityContent:
  String id
  HashMap<String, Object> pld  // payload

ForwardTransaction:
  String sender
  String mesh_reach  // mesh delivery receipt
  boolean dump
  List<ForwardPacket> mesh

ForwardPacket:
  String id, sender, receiver
  int receiver_type  // user or broadcast
  int enc_payload  // index into binaryPart
  HashMap<String, Object> payload
  long creation, expiration
  int hops, profile, propagation
  ArrayList<Long> track
  byte[] forwardedPayload
```

Fig. 9: Classes of me.bridgefy.entities.transport.

```
AppEntity:
  String ct, mi  // content, message id
  long ds  // date sent
  int et  // entity type

AppEntityMessage extends AppEntity:
  int mt  // message type
  int ku  // unused
  String nm  // username of the sender

AppEntitySignal extends AppEntity:
  int ms  // signal type

AppEntityHandShake extends AppEntity:
  AppHandShake hi
```