REDPANDA: A SECURE PEER-TO-PEER INSTANT MESSAGING SYSTEM

ROBIN FLOHR, MALTE FLOHR, AND STEFAN BERENS

Abstract. In the aftermath of the Snowden revelations, the issue of secure electronic communication is attracting new interest. In this paper we propose a protocol for secure instant messaging without central authority. All messages will be secured via end-to-end encryption and relayed via peer-to-peer network. The proposed instant messaging system is primarily designed to be used on mobile devices.

1. Introduction

Edward Snowden raised public awareness for the issues of mass surveillance and information privacy by releasing classified documents in 2013. The revelations, both the initial and the later ones, sparked new (public) interest in secure electronic communication. In this context new instant messaging systems emerged and gathered media attention (cf. Threema [6], Bleep [2] or Telegram [8]). However, all of these systems have one characteristic in common: A single point of failure. In all cases the server or the source code is (in parts) controlled by one party - the operators or developers. Now, a single point of failure always leads to potential security issues. The solution proposed in this paper is to remove the central structure inherent in current systems and replace it with a peer-to-peer network with an open protocol and an open source implementation. In order to guarantee security, the system relies on end-to-end encryption as a default.

The remainder of the paper is organized as follows. Section 2 provides an overview with respect to related systems. Section 3 presents the assumptions of the proposed system. Section 4 describes the protocol and its implementation. Section 5 concludes.

2. Related systems

2.1. **WhatsApp Messenger.** Currently, WhatsApp Messenger is one of the most popular instant messaging application in the world, cf. [10]. In its core the system is based on the open standard of the Extensible Messaging and Presence Protocol (XMPP). However, WhatsApp in itself is not an open system, i.e. it is not intended to be used with different servers or clients. In at least one case concerning this issue, namely the case of Disa [4], WhatsApp in fact filed a cease and desist order [3].

In addition, the general security issues mentioned in the introduction also exist, i.e. servers all controlled by the operators and source code not completely available

Robin Flohr is a PhD-student in mathematics at Karlsruhe Institute of Technology (Germany).
Malte Flohr is a MSc-student in mathematics at TU Dortmund University (Germany).
Stefan Berens is a PhD-student in economics at Bielefeld University (Germany).
Our Bitmessage address is BM-2cTFkMRbvmMMgA4z5qYSpXtEr3EGuWFxDi.

to the public. Thus, the system of WhatsApp suffers from the general problem of centralized systems, the problem of a single point of failure.

Another major issue is the lack of independently verifiable security, especially with respect to the end-to-end encryption. It is not possible to verify the privacy of the encryption keys and therefore exclude the possibility of third-party access.

An additional criticism is the possibility to track the online status of every user provided knowledge of their mobile phone number, c.f. [13] and [11].

2.2. **Bitmessage.** 'A Peer-to-Peer Message Authentication and Delivery System', cf. [12], which is similar to the current e-mail system in its application. Thus, the Bitmessage system is not an instant messaging system by design. It differs in terms of features as well as challenges. However, it nevertheless deserves to be mentioned here because of its similar goal(s).

The major difference between Bitmessage (or e-mail) and any instant message system is the delay between sending and receiving a message. In addition to that, Bitmessage (or - again - e-mail) requires no (initial) trust between any participants of a conversation. In this context, the required trust refers to the required trust to protect against spam. In the system of Bitmessage, minimal trust is replaced by minimal proof-of-work. In order to fulfill its purpose, the minimal proof-of-work for each message unfortunately exceeds the threshold for what is acceptable for an instant messaging system on a mobile device - at least when considering the requirement of minimal delay.

Thus, the (current) design of the Bitmessage system is suitable for a system similar to e-mail. However, it is not suitable for an instant messaging system that aims to include mobile devices. Conversely, in our proposed system a minimal level of trust is required, which allows us to avoid concepts like proof-of-work and therefore enables the use of mobile devices.

2.3. **Other systems.** The majority of issues raised with respect to WhatsApp can be applied to other systems as well. For example systems like BitTorrent Bleep [2] fulfill only part of the requirements established earlier. It is a system that features end-to-end encryption, but both servers and source code are controlled by the operators/developers. A similar critique can also be applied to other systems like Threema [6] or Telegram [8]. An exception to this is XMPP itself, which actually allows decentralized end-to-end communication (up to a certain degree). However, even though it is achievable, it is not central to the design philosophy of the system. As a result, end-to-end encryption requires additional effort to use it instead of being the default and remains a niche feature.

Furthermore, the issue of a single point of failure is also relevant on a meta level. In order to decrease the vulnerability of society with respect to mass surveillance the existence of multiple (independent) systems for secure electronic communication is ultimately necessary. Thus, irrespective of the systems in place, each new system (potentially) increases information privacy.

3. Assumptions

Any (non-trivial) conversation in day-to-day life usually takes place between people who already know each other to a certain degree. Consequently, it is safe to assume that there also exists a certain degree of trust between those people. In our protocol we make a specific assumption based on this general assumption.

First of all, if Alice wants to send a message to Bob, then Bob has to trust Alice to have no interest in causing extraordinary bandwidth or computational effort. Second, if Alice and Charlie are sending messages to Bob, then Bob has to trust Alice and Charlie to not fake their identities.

An additional assumption is that the distance between any two nodes in the network is bounded from above by six. In other words, going from one person to another person takes at most five intermediate steps (or hops). The concept, now called the 'six degrees of separation', was first introduced in 1929 by the author *Frigyes Karinthy* in [7]. In 2011 the *Facebook Data Science* team reported in [1] that the average distance between all their active users was 4.74 and therefore the average number of hops 3.74. Even though these numbers are average numbers and not maximal numbers, we still believe this (and other research in this area) to be enough justification for our assumption.

4. The protocol and its implementation

In the following we provide an overview of the protocol and its implementation (in its current form). Note, that the majority of this section is still work in progress. It should be understand as a rough sketch of our ideas. The affected parts are marked gray.

4.1. **Messages.**

In general, messages in the redPanda protocol are exchanged via conversations - which can include any (evolving) number of participants. The protocol differentiates between two different types of access to these conversations, namely read access and write access. This is provided by the use of signing algorithms and symmetric encryption. The signing algorithm uses a key-pair containing a public and a private key (control of write access), whereas the symmetric encryption uses one private key (control of read access). The check of write access is enforced by every receiving node. A usual conversation, also called channel, is realized by the use of both private keys - resulting in read and write access - of all participants. Another variant is the conversation with asymmetric access where there are two groups of people, namely one group with full access and the other with only read access. The participants with read access can, in this case, be sure that only a full access participant was able to write a message. It is also possible to form a channel with another asymmetric access, i.e. full access and only write access without read access, e.g. a conversation to share debugging information to the developers.

4.2. **Transfering Messages.**

The transfer of a message between two nodes is implemented as synchronization. Each node can mark specific channels for receiving header information. These header information will be send to all nodes to which one is connected to and to those that are willing to receive messages of a channel. The download of a message starts only from one node and usually from the one who send the header first. To provide offline messages, if one node is not connected, then there will be a resync when the connection is reestablished. This resync will only work for a limited time (7 days), after this time the hole synchronization information will be deleted. A history can be generated and shared for a conversation with the so called blocks. Note, that these blocks do not have anything in common with the blocks from bitcoin.

In the early stage of this project (and in order to simplify things) the private keys of read access and write access coincide. Therefore, in the current implementation every participant in a conversation automatically has both read and write access. The current announcement channel is an exception and an example for a conversation with asymmetric access, i.e. write access for the developers and read access for every user. In this specific case however, the private key for read access is actually hard-coded into every node.

4.3. **Forming a chat group.** The usual way to create a new chat group (conversation) with people is to create a new private key and exchange this key via QR-code to your chat partners. Another way to exchange the private key is to use Diffie Hellman: Alice sends Bob an invitation message including the public key via SMS (or alike) and Bobs client will generate an answer - again including Bobs public key - to be send again via SMS. Those messages will be generated by the client and ensure that the key exchange is nearly as secure as using QR-codes. This security works up to the assumption that the SMS can be read by other but not be modified or intercept.

One way to introduce convenience is to add friends by mobile phone number (or any other identification like e-mail addresses, etc.), if desired. To improve anonymity, phone numbers are hashed before they are used to find other users. The benefit is that if a malevolent person has not the phone number of a hash, he can not reveal the actual phone number. The private key exchange for the new conversation will be done with the trust to your conversations which coincide with the person you want to create a new conversation with. In detail, lets assume there are three such paths between Alice and Bob who want to create a new conversation by phone numbers. Every path will be used for a Diffie Hellman key exchange by sending three different public keys of Alice and Bob over these paths. Now the concatenation of the three secure-transferred shared secrets are SHA-256 hashed and result to the private secret of the new conversation. Thus, if there is at least one trusted path, the combination complexity for a brute-force attack is still 256-bits (the same for a full private key). Hence, you have to only trust one path to your new chat partner. The trust from a central authority, which confirms the identity of a phone number, is shifted to your friends.

In the current implementation you can only use QR-codes or send the private keys as a string like

```
pr7768EZp1Y8PwQhozCfacTNjL4iRxgJqcm8NDVuyvhfYJDwkzJFTtCZ6
```

but this way is very insecure but will be replaced by the key exchange described above. The first two characters are a prefix for decoding the type of the channel. The next 32 bytes decoded in Base58 are the private key and the last bytes are a checksum to prevent copy or decode errors. You may also have a look in the future section for creating chat groups via mobile phone number.

4.4. **Routing of Messages.** We use our restrictions/assumptions to arrange the peer-to-peer network by our chat partners. Since light clients, e.g. mobile phones, should focus on using a minimum of internet bandwidth and computational effort, our plan is to transfer the main routing part, if possible, of the messages to normal and super nodes on the network.

4.4.1. *Transfer routing to non-light nodes.* Lets assume that Charlie is linked to a super node (S2), i.e. he is a operator of that node, and that Bob has a conversation-chain to Charlie. In addition, Alice is connected via a conversation-chain to a further super node (S1). Our goal is now to route messages between Alice and Bob: Since Charlie is linked to S2, thus has the minimum hop of 1, he will broadcast S2 to Bob (via hops of conversations). If Bob knows multiple servers, he will choose 3-5 of the best using measurements like load, registered users on the server, etc., and will register to them (using the conversation-chain to the server via Charlie), to ensure that the super nodes will do their best to collect and send messages for his conversations. The server will prioritize registrations using the hop count and probably will de-/unregister users which have a higher hop number than a registered user (this information will be send to the node once he will connect again/is connected). If the registration process was successful for Alice to S1 and Bob to S2, they will establish a direct connection to the super nodes and we have now transferred the complex routing parts from the light clients to the super nodes.

These concepts are currently not realized in our implementation but we are working on it.

4.4.2. *Routing between non-light nodes.* Since we optimized the node communication in our implementation - using polling instead of forking mechanisms - for many connections, we expect that a single node/server will be able to serve about 500-1000+ users at once using decent hardware. A hop count of infinity will be allowed and will allow users to register to super nodes when they are new to the network and have no extended friends linked to a super node. There are many ideas for the routing of the servers and this will be continuously improved. We first propose an easy-to-implement routing algorithm, which will use the Kademlia distributed hash table (DHT) to find IPs of super nodes serving messages for a given channel. Easy said, the DHT is a decentralized key-value database which in our chase will store and link channels (their public key) to IP addresses for the super nodes. For every channel, which was registered by a user to a super node, the super node puts his IP address into the DHT database and thus will be found by others, who are interested in these channels. Since all messages will be downloaded and relaid by super nodes, its sufficient to connect to 3-5 nodes for each channel (already connected nodes are preferred). Since friends are more likely to connect/register to the same super node, the number of connections of each node should be not to big. The key in the database will be secured by hashing algorithms, in addition there are two possibilities to secure the DHT value: Since every super node has a direct connection to a node who knows usually the private keys of a channel, the super node can encrypt and sign messages published into the DHT database with the help of the connected user. Signing messages in this context will secure from publishing IPs from malevolent nodes, which will not send us any messages, and thus will break down the communication of the channel, if there are much more malevolent nodes the real ones. The benefit of encryption here will lead to less deanonymization of the IPs of the super nodes, and perhaps of the user it self. The transfer of messages and their synchronization needs to be adjusted for the super node to super node synchronization and will include a full synchronization to unknown nodes. Every node in the network will have an identity, which will be shared between differend devices of a user. This identity will be userd for registration processes and the authoriziation to a server by hop count. The main parts of the mentioned decisions/registrations will be done by the client and not by the user. The user will maybe able to adjust their prioritys which the client will take in count.

The current implementation is using no explicit routing at all but we will begin the implementation in the near future, when there is a need for routing algorithms.

4.5. **Spam.** First of all, network spam can only occur when someone generates a lot of messages for a given channel where the malevolent person knows the private signing key of the channel. Because (almost) every node will check the signature of a message which will guarantee that the sender of the message has write access for the given channel. Only after this verification the node will introduce this message to other connected nodes. Using our assumptions, this leads to the ability for every node to block a malevolent channel but punishing only bad people. Beside this, every user can track the chain of trust to a malevolent channel and remove/block a person/channel in this chain of trust (especially remove the friend/channel who gave trust to this malevolent person).

4.6. **Anonymity.** Our first goal is security and not anonymity but we try to add as much anonymity as possible until this will inhibit our main goals . The main problems for anonymity arises in the structure of the network. For example, it may be easy to be tracked by your chat partners. To get in details, others can collect information at which times you are sending/receiving messages and may guess your IP address. For these cases we will introduce so called dust messages. To enhance your anonymity your client will send random messages to your chat partners and also will exchange dust channels. A person who is tracking you may see messages to persons who you maybe don't even know. Hence, this person can not be 100 percent sure that you are chatting with a specific person. In addition, in the future we probably want to introduce (onion, garlic) routing algorithms to establish connections to a clients super nodes by routing the traffic using other (random) super nodes. In this case the tracking is nearly impossible.

4.7. **Multiple Devices per User.** The protocol will be designed to handle more than one client for each user. This gives the ability to continue a chat from your mobile phone to your tablet or even your computer. The nessesary informations will be shared between a users devices by a so called master channel. New/delete operations of conversations/user information will be shared and decision-making-processes will be handled by devices with a high battery charge or even by non-light client, if available.

4.8. **Encryption algorithms.** In this section we list the encryption algorithms used for different features. We use common abbreviations:

Feature	Algorithm
Conversation (key exchange)	ECDH
Conversation	AES-256-CBC
Node key exchange	ECDH
Node communication	AES-CTR
Backup encryption	AES-256-CBC

The choice for AES was made due to hardware support of some devices.

5. Conclusion

The peer-to-peer instant messaging system presented in this paper features decentralized communication with end-to-end encryption. It is a protocol designed to be used both on desktop as well as mobile devices. Furthermore, it aims for a reasonable balance between anonymity/security and usability.

The redPanda project is free and open source software (FOSS). It can therefore be independently verified to be a secure way of communication, which reduces the risk of backdoors or similar anti-features. Furthermore, the current implementation only serves as a first example (or reference). Everyone is invited to write their own (independent) implementation of the protocol.

References

[1] Lars Backstrom et al. *Four Degress of Separation*. 2011.

[2] *Bittorrent Bleep*. 2014. URL: `http://labs.bittorrent.com/bleep/`.

[3] *Disa Messenger - cease and desist order*. 2013. URL: `https://plus.google.com/101758828974303616894/posts/Wtyy5sGvyaQ`.

[4] *Disa Messenger*. 2014. URL: `http://disa.im`.

[5] Threema GmbH. *Threema Cryptography Whitepaper*. 2014. URL: `https://threema.ch/press-files/cryptography_whitepaper.pdf`.

[6] Threema GmbH. *Threema*. 2012. URL: `https://threema.ch`.

[7] Frigyes Karinthy. *Chains (Láncszemek)*. 1929.

[8] Telegram Messenger LLP. *Telegram Messenger*. 2013. URL: `https://telegram.org`.

[9] Petar Maymounkov and David Mazières. *Kademlia: A Peer-to-peer Information System Based on the XOR Metric*. 2002. URL: `https://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia-lncs.pdf`.

[10] Parmy Olson. *WhatsApp Hits 600 Million Active Users, Founder Says*. 2014. URL: `http://www.forbes.com/sites/parmyolson/2014/08/25/whatsapp-hits-600-million-active-users-founder-says/`.

[11] *Online Status Monitor*. 2014. URL: `https://www.onlinestatusmonitor.com/`.

[12] Jonathan Warren. *Bitmessage: A Peer-to-Peer Message Authentication and Delivery System*. 2012. URL: `https://bitmessage.org/bitmessage.pdf`.

[13] Heinz Heise publishing house. *c't - magazine for computer technology*. 2015. URL: `http://shop.heise.de/katalog/ct-2-2015`.