Moritz Würth Oliver Weinmeier Renato Farruggio

Project: 03-doubleRatchet

Abstract: We extended BACnet with a forward- and backward-secure chat, by implementing the Extended Triple Diffie-Hellman key exchange protocol X3DH and a double ratchet algorithm for generating subsequent message keys. Our implementation runs on Linux and MacOS. In a first step we implemented X3DH and double ratchet and testet its functionality over TCP. In a second step, we replaced TCP in the transport layer with cbor files so that it is BACnet conform. The X3DH algorithm is designed to have a server to support the key exchange. Since BACnet is decentralized, we had to slightly adapt our protocol so that it works for a serverless setup, such as BACnet. The double ratchet protocol is designed for secure asynchronous messaging, and we were able to implement it as defined on the Signal website [1]. Finally we wrote an implementation that works over rsync.

Github: https://github.com/cn-uofbasel/BACnet/tree/master/redez-sem-hs20/groups/03-doubleRatchet

Project Idea

Secure Chat is an application intended to extend BACnet. It allows two parties to have Signal-like secure chats that follow the the double ratchet algorithm, thus guaranteeing forward-secure and backward-secure chats. This also means that all messages are being saved encrypted and are being decrypted only once.

Components

All helper functions are located in src/helper_functions and are not important for the user.

The class *TCP_chat.py* is being used for secure chat over TCP. This is our prototype for testing doubleratchet.
The class *BACnet_local_chat.py* is being used for chatting over a log file.
The class *chat_over_rsync.py* allows users to chat over a log file, using rsync. This log file can be synchronized via dropbox for example.

All 3 implementations offer secure chatting.

Deployment / Integration

For the exact instructions on how to run the code, please have a look at our README.md on the git repository.

[1]https://signal.org/docs/specifications/doubleratchet/

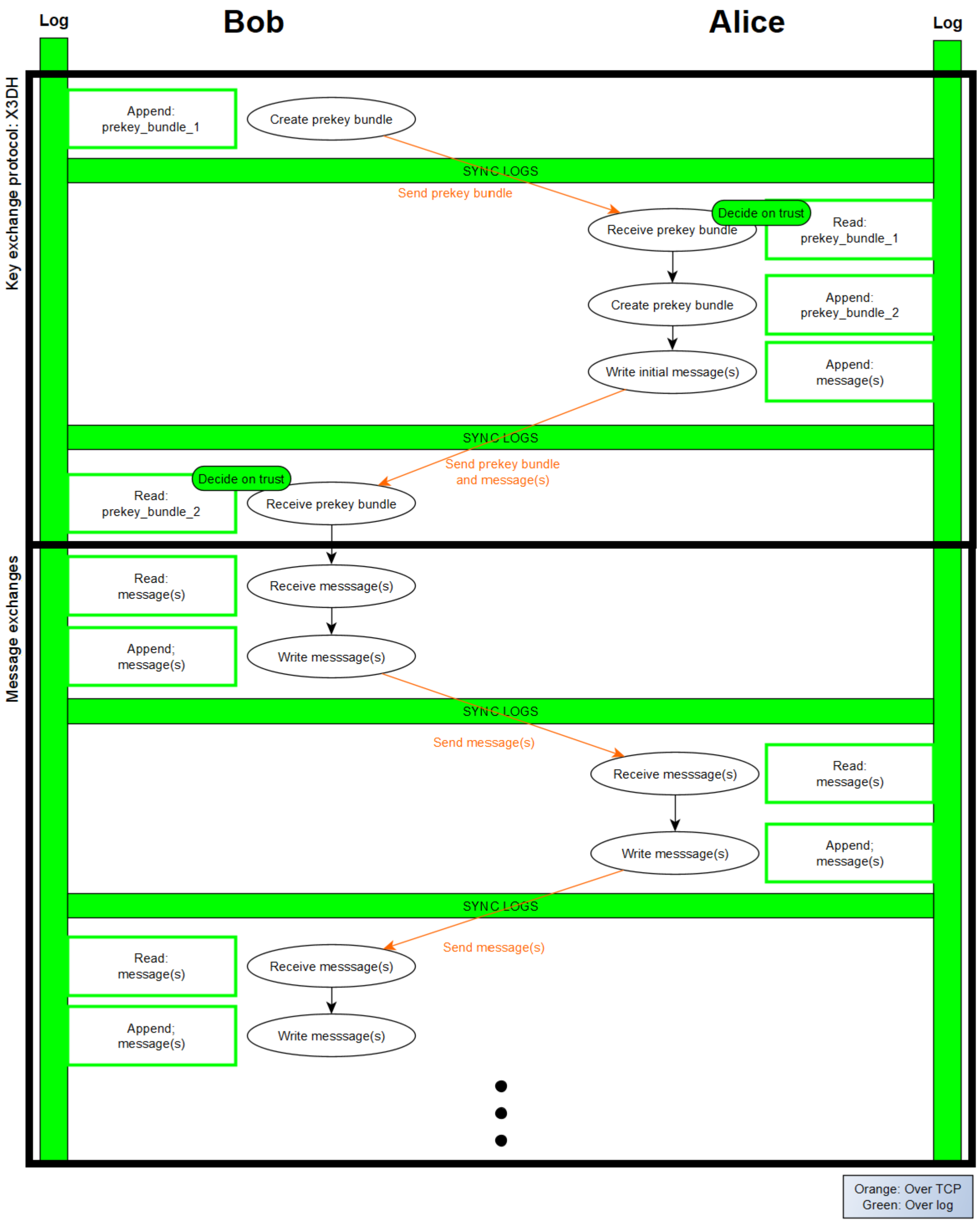
Conclusion

We managed to implement a chat that is encrypted and Perfect Forward Secrecy and Backward Security by design. The BACnet structure partly comes in handy, since we don't have to worry about out-of-order messages because the correct order of messages is guaranteed in a log.

Solvable but unsolved problems:
We currently save our private key locally, encrypted by a password. The current password is hardcoded as "pw". To make it secure, even when the data on the phone is compromised, the password should be entered by the user, everytime he runs the program.
The key exchange protocol is prone to errors. When Bob writes a message before he received the first message from Alice, all messages will be shown as empty, and there is no way of recovering the conversation, other than resetting both sides.
Trying to retrieve a message when there is no new message will break the algorithm, and then both sides have to reset the conversation.

Inherent problems:
BACnet has no server. The key exchange protocol X3DH requires Bob to publish his information (prekey_bundle) on a server, so that Alice can ask the server for Bob's information. Normally, Bob would publish several signed onetime prekeys, and the server would make sure that keys are being used only once. In a log, as BACnet, information is not being deleted, and keys could potentially be used by several people, which would result in a problem (the key exchange would not work for them). Therefore, Bob only publishes one onetime prekey when he wants to establish contact. Alice then has to know that Bob wants to establish contact, so that she can complete the key exchange.

Protocol


Log
Bob
Alice
Log
Key exchange protocol: X3DH
Append: prekey_bundle_1
Create prekey bundle
SYNC LOGS
Send prekey bundle
Decide on trust
Receive prekey bundle
Read: prekey_bundle_1
Create prekey bundle
Append: prekey_bundle_2
Write initial message(s)
Append: message(s)
SYNC LOGS
Send prekey bundle and message(s)
Decide on trust
Read: prekey_bundle_2
Receive prekey bundle
Message exchanges
Read: message(s)
Receive messsage(s)
Append; message(s)
Write messsage(s)
SYNC LOGS
Send message(s)
Receive messsage(s)
Read: message(s)
Write messsage(s)
Append; message(s)
SYNC LOGS
Send message(s)
Read: message(s)
Receive messsage(s)
Append; message(s)
Write messsage(s)
Orange: Over TCP
Green: Over log