Distributed Trust Empowerment for Secure Offline Communications

Endre Medhus Mæland, Sigmund Bernhard Berbom, Besmir Tola, Yuming Jiang
Department of Information Security and Communication Technology
NTNU, Norwegian University of Science and Technology

***Abstract*— Most of today's digital communications over the Internet rely on central entities, such as certificate authority servers, to provide secure and authenticated communication. In situations when the Internet is unavailable due to lack of reception in remote areas, natural disasters destroying network infrastructure, or congestion due to large amounts of traffic, these central entities may not be available. This causes secure communication, even among users in the vicinity of each other, to become a challenge. This paper contributes with a solution that enables peers within the vicinity to communicate securely without a connection to the Internet backbone. The solution operates on the Wi-Fi infrastructure mode and exploits a private distributed ledger to ensure a trusted authorization among users without a third party. Moreover, the solution enables users to set up secure communication channels using mutual authentication for exchanging data securely. Finally, the solution is validated through a proof of concept application and extensive experiments aiming at optimizing system parameters and investigating the performance of the application are carried out. The results from these measurements indicate that the solution performs well on small to medium-scale networks.**

***Index Terms*—Decentralized Authentication, Distributed Ledger Technology, Device-to-Device Communication, Peer-to-Peer Network, mTLS, Mobile Social Network.**

I. INTRODUCTION

The use of certificates and private/public key pairs is a common way to provide authentication over the Internet [1]. Certificates enable end users to be authenticated from a centralized Certification Authority (CA), a trusted third party, in order to retrieve service [2]. However, the process of authentication is challenged in situations where there is a lack of Internet access or backbone connectivity. In case natural disasters, power outages, or human-caused accidents impact the Internet infrastructure, end users will not be able to retrieve authentication and consequently use a secure service, or establish secure communication.

Although the Internet may be unavailable due to disaster impacts, mobile-equipped end users can still establish network connectivity within the range of their mobile radios. Almost all modern mobile devices are equipped with Wi-Fi radios and this technology can be exploited for re-establishing connectivity in a Peer-to-peer (P2P) fashion for offline communication. However, the lack of a trusted third party will still prevent the users from establishing and providing secure and authenticated communication. Wi-Fi can provide authentication through WPA-Enterprise [3]. However, WPA-Enterprise requires an authentication server, such as a RADIUS server, for establishing user identities but if the server is not available due to Internet connectivity issues, Wi-Fi cannot provide authentication. Furthermore, in a scenario without access to the Internet, the users in the vicinity will not be able to establish a secure communication channel due to the lack of a central unit orchestrating the communication. As a result, additional security mechanisms need to be built on upper layers to enable data confidentiality, integrity and authenticity.

A distributed authentication mechanism can solve the issue of a CA not being available on the network by allowing benign nodes to agree on an immutable record of authentication material, despite the existence of malicious nodes. To achieve such an agreement, Distributed Ledger Technologies (DLTs) can be utilized [4]. However, most public DLTs have resource-consuming security mechanisms tailored for financial transactions. Transaction are typically secured by consensus mechanisms such as Proof-of-Work (POW) and Proof-of-Stake (POS), requiring large amount of resources or financial transactions [5]. Therefore, these consensus mechanisms are unsuitable for resource-scarce mobile devices when exchanging authentication material. The mutual authentication in mutual Transport Layer Security (mTLS) [6] allows all parties to be authenticated, and the protocol also provides data encryption. Furthermore, using symmetric encryption combined with mTLS can improve connection establishment times compared to only using mTLS.

A previous work addressing authentication in offline networks proposes a system where users receive authentication material when an Internet connection is available [7]. If the user loses the Internet connection, the authentication material could still be used. However, this solution does not support offline registrations, hence preventing new users to be authenticated after the loss of Internet connection. [8] improves this solution by using peer signed certificates to allow offline registration. The peer signed certificates establish a trust chain where users vouch for each other. In this solution, it is difficult to draw a clear line to what point in the chain a user is no longer trusted. In [9], the authors propose a blockchain-based solution for decentralized authentication of IoT devices. The proposal is based on the public blockchain Ethereum and envisions the creation of specific trusted zones, called Bubbles, where IoT devices within the zone establish a level of trust. The trust is confined within each zone and inter-zone authentication and trust is left for future work. However, as also identified by the authors, the solution presents several open issues related to the use of a public DLT. The solution is associated with economic cost, is unsuited for real-time applications, and requires an initialization phase with a node assuming the role of a certification authority. Such solution is infeasible for use cases with high user mobility. Henceforth, a problem yet to be solved is *how to efficiently authenticate users in a mobile offline environment where users are able to register offline, establish a secure communication path,* and have a clear separation between trusted and untrusted users.

This paper aims to design, implement, and validate a solution for application layer security in offline networks. The proposed solution exploits Wi-Fi Infrastructure mode as the technology for offline communication, mTLS in combination with a symmetric key solution to provide mutual authentication and encrypted data exchange among peers, and implements a private distributed ledger and a consensus mechanism to agree on users' authentication material. A proof-of-concept instant messaging application has been developed to validate the solution and the implementation source code is publicly available[1]. An extensive experimental campaign on real equipment has been performed for optimizing the system parameters and analyzing its performance and security features.

The remainder of the paper is structured as follows: Section II illustrates the proposed system architecture for enabling secure and trustworthy communication over Wi-Fi Infrastructure mode. The implementation on a real testbed of smart devices running the Android OS is presented in Section III. Successively, the validation of the security adopted in the architecture and the analysis of the experimental results are presented in Section IV. Finally, Section V concludes the paper.

II. PROPOSED SOLUTION

This section illustrates how the proposed solution enables authenticated and secure communication.

A. Overall Architecture

Figure 1 illustrates the high-level architecture of the proposed solution.

Going bottom-up, the first layer is the wireless connection layer. This layer exploits Wi-Fi in infrastructure mode and is responsible for handling wireless connections, including both unicast and multicast transmissions utilized for ledger management and the actual service. This layer can also be built on other technologies but we chose Wi-Fi given the wide adoption of Wi-Fi radios in mobile devices.

The second layer is responsible for handling identities and enabling authentication. This layer consists of the ledger and the consensus mechanism. The ledger contains authentication material for all users in the network. By having all users agree on the ledger's content, the responsibility of authenticating users is moved from a single entity to the network as a whole. This layer is further discussed in the following sections.

The third layer is responsible for enabling secure communication. The first time two users connect, an mTLS connection will be established with the authentication material found in the ledger. During this connection, the users will negotiate a symmetric key using the Diffie-Hellman key exchange [10] to be used the next time they communicate. From the next time these users connect, they will establish a TCP connection secured with AES in Galois/Counter Mode (GCM) [11]. Using symmetric keys reduces the time to establish the connection. To improve the security of the symmetric keys, DH- and symmetric ratchets are used [12].

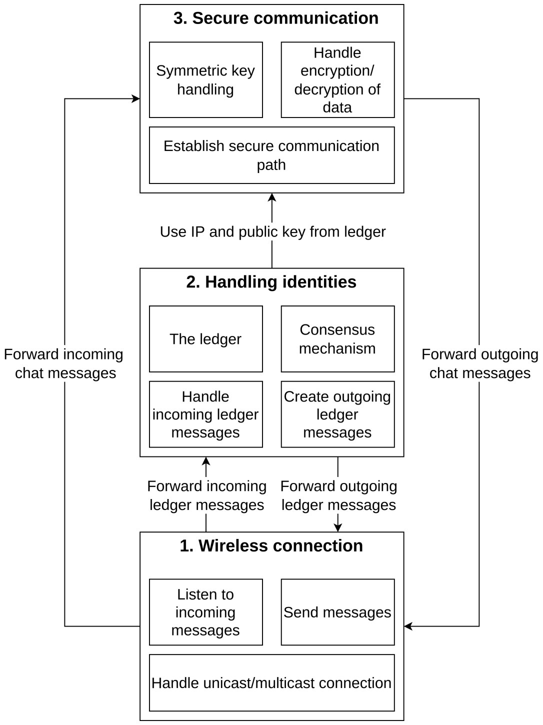

3. Secure communication
Symmetric key handling
Handle encryption/ decryption of data
Establish secure communication path
Use IP and public key from ledger
2. Handling identities
The ledger
Consensus mechanism
Handle incoming ledger messages
Create outgoing ledger messages
Forward incoming chat messages
Forward outgoing chat messages
Forward incoming ledger messages
Forward outgoing ledger messages
1. Wireless connection
Listen to incoming messages
Send messages
Handle unicast/multicast connection

Fig. 1: High-level architecture of the solution.

B. Distributed Authentication

This subsection describes how a ledger is used to achieve distributed authentication.

1) Ledger Entry: The ledger consists of Ledger Entries (LEs), where one LE represents one user in the network. When new users want to join a network, they have to create a valid LE and distribute it. Each LE contains an X.509 certificate [13] and an IP address. The certificate can be signed by a CA or by the users themselves. To obtain a CA-signed certificate, the user has to sign up to the system while the application is online using a password and an email address as the username. The email address must be verified before receiving the CA-signed certificate. Whether the certificate is CA- or self-signed plays an important role in the consensus mechanism. This is further described in Section II-B3.

2) Joining the ledger: When a new device connects to a network, it first joins a predefined multicast group to which all users attached to the network are listening. Secondly, it broadcasts a request for the ledger to all the users in this group. If the user does not receive any responses within α seconds, the user assumes there are no other users in the network. The

---

[1] Will be disclosed at a later time.

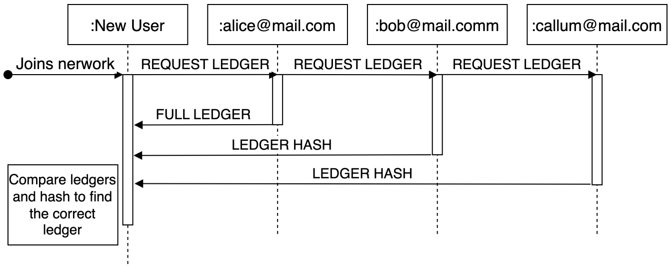


Fig. 2: Multicast message exchange when a new user joins the network.

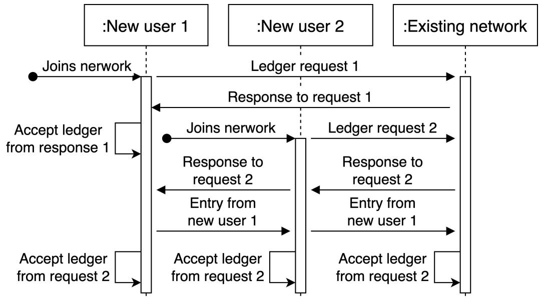


Fig. 3: LE of new user 1 is not included in the ledger from request 2, and is therefore lost after new user 1 has registered.

user will then proceed to create its own LE, which at this point will make up the ledger. The value of α is further discussed in Section IV-A6.

Figure 2 illustrates the process of a new user joining an already established network. The last CA-signed user to join the ledger will respond by sending a full ledger, while the others will send the hash of their ledger. If there are no CA-signed users in the network, the last user to join the network will send the full ledger. The new user will use these responses to select the correct ledger according to the criteria described in Section II-B3.

All messages regarding the ledger are sent using UDP multicast. If the user has not received the full ledger that won the voting, the user will randomly pick one of the users who sent the hash of the winning ledger and request the full ledger. If the user does not receive the full ledger within β seconds, this process is repeated until the full ledger is received. The value of β is further discussed in Section IV-A5.

After receiving the ledger, the user can create and broadcast its own LE. The username used by the new user cannot already exist in the ledger unless the new user can obtain a CA-signed certificate with that username. If so, the LE with the CA-signed certificate will replace the one with the self-signed certificate, and the old user will no longer exist in the ledger.

3) Consensus in the ledger: When a new user joins the network, a voting is initiated to agree upon and distribute the ledger to the new user. As shown in Figure 2, all users send a full ledger or a hash of their ledger. These messages are interpreted as votes for the validation of the correct ledger by the new user. The messages have to be signed, and their certificate must be included to ensure that each user can vote only once. A malicious user could generate many fake users with self-signed certificates, corrupt the ledger with fake entries, and drive the consensus. This would enable a Sybil attack [14] whose consequences would be a Denial of Service (DoS). To mitigate this, votes from users with CA-signed certificates are given priority. This is because CA-signed certificates contain usernames, i.e., email addresses, that have to be validated online thus making the process of impersonation much harder. As a result, the following restrictive criteria, with a decreasing priority, must be met before a user accepts a ledger:

1) If at least two CA-certified users distribute the same ledger or corresponding hash, and they make up more than 50% of the CA-certified users in the ledger, that ledger will be accepted, given the full ledger has been received.
2) If β seconds have passed from the time the ledger was requested and at least one CA-certified user has responded with the ledger, either full or hashed, the ledger with most votes from CA-certified users is accepted.
3) If β seconds have passed from the ledger was requested and no CA-certified users have responded with the ledger, the ledger with most votes will be accepted.

β has to be set so that users can expect to have received all the votes within that time. This value is discussed in Section IV-A. Note that the above criteria are not exclusive of each other. Priority is given to CA-signed users, however, self-signed users are considered in some scenarios to ensure the service is available even though no CA-signed users are present. This may pose a risk in scenarios without CA-signed users, but the risk will decrease as the number of user increases.

4) Synchronizing the ledger: When a new user joins the network, all users can listen to the following voting process because all messages are broadcast. They can therefore see which ledger is correct by comparing responses, in the same way as the user joining. Hence, users with an incorrect ledger can update their ledger. If the accepted ledger has LEs that does not exist in the user's ledger, they are added. If a user's ledger holds any LEs that do not exist in the accepted ledger, those LEs are not removed. This mechanism combats a possible ledger rollback attack, further mentioned in Section II-B5, and ensures no LEs are lost. Figure 3 shows how a LE could potentially have been lost if two users joined at the same time.

An LE conflict means that two LEs with different certificates have the same username. If there are conflicting LEs between the accepted ledger and the ledger held by a user, the users will update their ledger as long as the conflicting LE they hold does not have a CA-signed certificate.

5) Mitigating attacks: There are several security mechanisms added to the proposed solution to mitigate various types of security attacks.

If a valid vote in one voting process could also be valid in another voting, a malicious actor could exploit this to execute

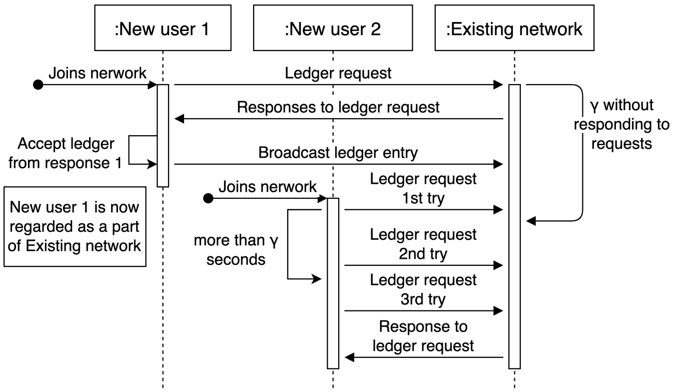


Fig. 4: If one ledger request falls within the time window where a request is dropped, another will fall outside it.

a replay attack. To mitigate this, every voting has a unique nonce where all messages related to this vote have to contain this nonce. Because the messages are signed, and an attacker cannot forge a signature, there is no way for them to obtain a valid response signed by another user, and the attack is prevented from occurring.

An extension to the above attack is a ledger rollback where an attacker stores both the request for the ledger and its responses. As a result, the ledger could be reset to a previous state by replaying these messages later, potentially removing users from the ledger. To avoid this from happening, LEs are not removed from the ledger even when they are not a part of the accepted ledger.

Every request for the ledger broadcast in the network triggers a response from the other users. All the users in the network then handle these responses. Therefore, flooding the network with requests will cause an increase in computational load on the devices, potentially leading to a DoS attack. To combat this attack, a user will drop requests received within γ seconds after responding to a request for the ledger. The choice of the parameter γ is discussed in Section IV-A7. Multicast packets are transmitted multiple times to avoid packet loss. As long as γ is less than the time between the first and last request transmission, dropping these packets will not lead to requests not getting a response, as also shown in Figure 4.

C. Secure communication path

The first time two users connect, they establish an mTLS connection using the authentication material found in the ledger. During this connection, the peers negotiate a symmetric key to be used for the next connection. The second time two peers communicate, they use a pure TCP connection with AES for encryption. Every time they communicate, the peers negotiate a key they will use the next time they communicate.

Diffie-Hellman (DH) key exchange is used to negotiate the symmetric keys. By introducing DH-ratchet, this process gets faster as two messages are needed the first time two users communicate, while only one message is required from there on. The process is illustrated in Figure 5. The first DH key exchange between two users initiates a DH-ratchet where one user's private key and the other user's public key are used to calculate a shared DH secret. From the second time the two users interact, only one user updates their key pair to generate the new symmetric key. Therefore, only one key is sent, and only one message is required to update the symmetric key. The users renegotiate the symmetric key every time they set up a new connection.

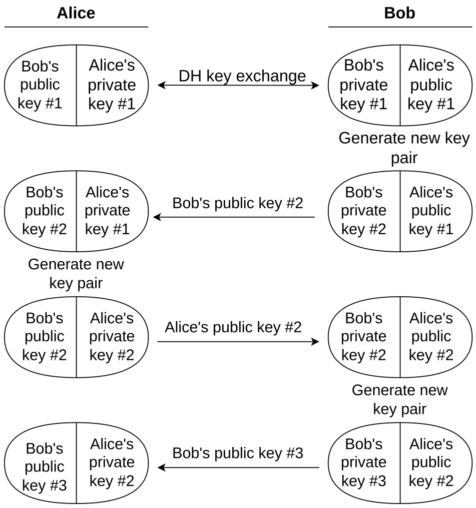


Fig. 5: DH-ratchet

If an attacker can break one of the symmetric keys, they will be able to read all messages within that conversation. By introducing double ratchets, this problem is reduced to a backward secrecy problem. Double ratchets use a key derivation function (KDF) on every key after use to ensure a key is only used once. Hence, it is not possible to find an old key given a new one, but it is possible to find a new key given an old one. Using double ratchets and DH-ratchets provides forward secrecy within a conversation and forward- and backward secrecy between conversations [12].

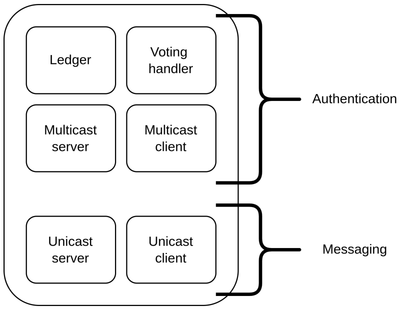


Fig. 6: Application overview.

III. PROOF-OF-CONCEPT IMPLEMENTATION

In order to validate the proposed solution, the system has been implemented in a proof-of-concept Instant Messaging application. The application consists of the six components, shown in Figure 6.

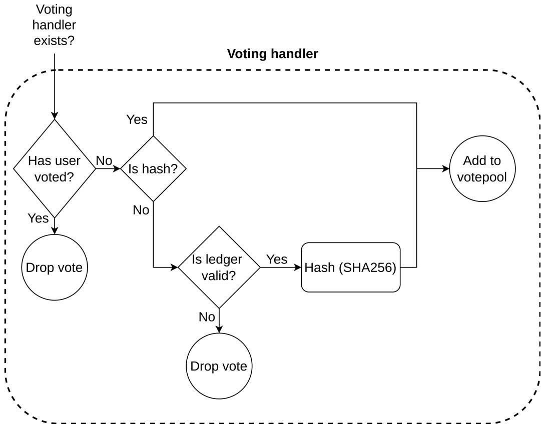


Fig. 7: Voting handler flowchart.

A. Authentication components

The multicast client and server are responsible for sending messages related to the ledger. The multicast server is implemented as an Android Service while the client is a Kotlin class. Both are initiated when the application starts.

The voting handler is responsible for handling the process related to achieving consensus on the ledger. For every new voting, the application initiates a new voting handler. The voting handler ensures the votes are in the correct format, counts votes, selects the correct ledger, and is responsible for updating the ledger after a finished voting.

When a new vote is received by the multicast server, it checks if there exists a voting handler for that nonce, and if so, forwards it to the correct handler. Figure 7 shows how a new vote is processed by the voting handler. The process ensures the vote is related to an existing voting, that a user does not vote multiple times, and ensures the vote is in the correct format.

The ledger module contains the ledger itself and methods for updating it and creating new LEs. The application updates the information in the ledger after every voting.

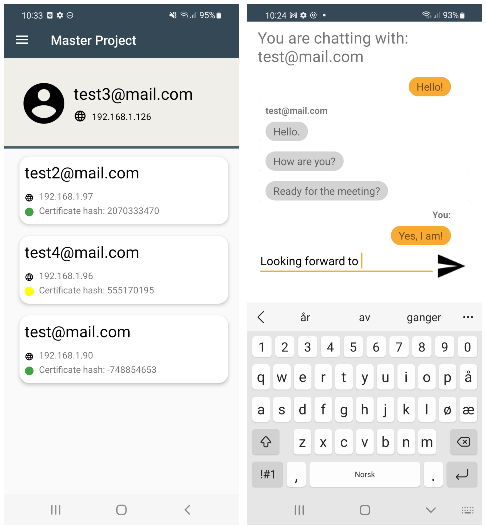
B. Messaging components

When the application starts, it initiates a unicast server. The unicast server listens for incoming requests to set up unicast communication. The server opens a new port for every new connection request, allowing the application to support receiving messages from multiple devices simultaneously. Depending on the available authentication material, an mTLS over TCP or pure TCP connection is established.

When the user starts a chat, the application initiates a unicast client. The unicast client sends a connection request to its peer's unicast server and establishes a connection. After a unicast connection is established, both server and client send and receive messages.

C. Activities

Android divides applications into different interfaces called activities. The proof-of-concept application consists of two primary activities: the main and the chat activity.



test3@mail.com
192.168.1.126

test2@mail.com
192.168.1.97
Certificate hash: 2070333470

test4@mail.com
192.168.1.96
Certificate hash: 555170195

test@mail.com
192.168.1.90
Certificate hash: -748854653

You are chatting with:
test@mail.com

Hello!

test@mail.com
Hello.
How are you?
Ready for the meeting?

You:
Yes, I am!

Looking forward to

(a) Main Activity. (b) Chat Activity.

Fig. 8: Android Application Activities.

Figure 8(a) shows the main activity interface. The interface displays the content of the ledger as a list of users. Each list entry displays the username, IP address, and a colored dot indicating if the user's certificate is self-signed (yellow) or CA-signed (green). The application will initiate a chat with a peer when the user taps a list entry. The application does not limit the users' ability to connect to other users based on their certificate type. The users themselves have to decide whether or not to trust a user with a self-signed certificate.

When a connection between two peers has been initiated, the chat activity shown in Figure 8(b) starts. In this activity, the users can read and send messages.

D. Ledger design parameters

The system must achieve a consensus on the ledger content to enable authentication. The ledger distribution can be challenged by the use of unreliable access and transport layers, i.e., wireless channel and UDP. Henceforth, the following parameters are used to tune mechanisms on the application layer that have been added to address the eventual packet loss: i) *number of transmissions*; ii) *time between transmissions*; iii) *fragment size*; and iv) *time between fragments*. A description of each of the parameters is provided below, while tests used for optimizing the value of each parameter can be found in Section IV.

i) *Number of transmissions*: All messages used for ledger management are sent multiple times to reduce packet loss through redundancy.

ii) *Time between transmissions*: The time between transmissions is defined as the time from sending the last fragment of the ledger in one transmission until sending the first fragment of the ledger in the subsequent transmission. Increasing this parameter may reduce the probability that the same factor, e.g., propagation conditions, will affect multiple transmissions. However, too much time between packets will increase the time it takes to sign up, potentially affecting the user experience.

iii) *Fragment size*: When a sufficiently large ledger is sent, the message is divided into several fragments. In the proof-of-concept application, one fragment holds n LEs. Every fragment must be received for the ledger to be counted as a vote. When the size of each fragment is reduced, the packet loss is also reduced as found in [15]. However, with smaller fragments the number of fragments that has to be successfully received increases.

iv) *Time between fragments* When a ledger is fragmented, the time between fragments may affect the packet loss similarly to the time between transmissions.

IV. RESULTS

This section describes the test conducted to evaluate the ledger parameter values and assess the security and performance of the proof of concept application.

A. *Optimizing parameters*

Specific tests have been conducted for optimizing the parameters identified in Section III. In the tests, one user sends a pre-programmed ledger to another user, and the messages sent and received are recorded. The tests only regard message loss, so there is no need for multiple peers to vote for a correct ledger. Each ledger is seen as one message, although it is transmitted in fragments to reduce message loss, with each fragment containing 1 LE. Therefore, a ledger is considered lost if one or more fragments are not received. That is because the entire ledger will have to be received in order for it to be considered a vote. The tests have been conducted using two Samsung Galaxy S21 5G phones and a Netgear Nighthawk M2 wireless router. The optimal value of each parameter has been found by varying its value, while keeping the parameters fixed. The fixed parameters' values are chosen to be near-optimal, based on a smaller sample of the performed tests.

The test environment of the following tests is constructed to have a large message loss to increase the statistical significance. As the number of ledger entries in the pre-programmed ledger negatively affects the message loss, the number of LEs is chosen so that the message loss will be near 50% for what is believed to be the optimal value for the tested parameter. Such message loss will increase the statistical significance of the results compared to a very high or very low message loss. The test environment is designed solely to investigate the near-optimal values for the parameters tested and does not represent how the system will perform outside of the test environment. The latter is investigated in more detail in Section IV-B.

1) Number of transmissions: Figure 9 shows how the message loss is affected by the number of transmissions of the same packet. The requests for ledgers messages are sent in one fragment while the full ledger contains 100 LEs, and with a fragmentation size of 1, it is delivered in 100 fragments. As expected, the message loss decreases as the number of transmissions increases for the full ledgers. However, the message loss for small packets, here represented by request messages for the ledger, increases with a higher number of transmissions. Most of the messages sent in the system, including hashes of ledgers, consist of one packet, so the message loss of small messages is very important, even though the message loss for larger ledgers is also of significance. Choosing 4 transmissions achieves a low message loss of the important small messages while achieving an acceptable message loss of the larger messages, i.e., full ledgers.

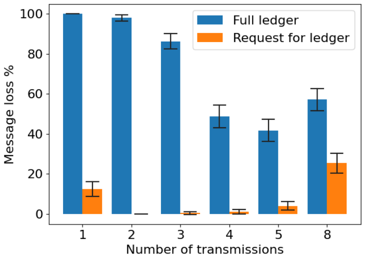
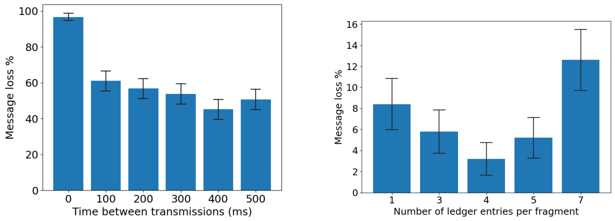


Fig. 9: Number of transmissions.



(a) Time between transmissions.

(b) Fragment size.

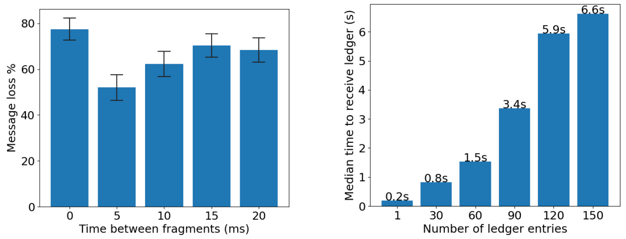
Fig. 10: Impact on the message loss as a result of varying time between transmissions and fragment size.

2) Time between transmissions: Figure 10(a) shows how the message loss is affected by the time between transmissions. Increasing the time between transmissions will decrease the message loss until 400 ms. That is because what causes packets to be lost in one transmission might be gone before the next one. From 400 ms and upwards, the message loss increases with more time between transmissions. This indicates that the advantage of stretching out the transmissions in time is reduced for values larger than 400 ms.

The number of transmissions has to be considered when analyzing the test results. While 400 ms is the optimal time between transmissions for four transmissions, according to these results, that might not be the case for another number of transmissions. With four transmissions, 400 ms between each transmission will stretch the transmission out to 1200 ms in total. The total stretch in time will be lower for fewer transmissions, and the optimal value for the time between transmissions might be higher.

3) Ledger fragment size: Figure 10(b) shows the test results for message loss for different fragment sizes. The figure shows that the message loss is lowest for a fragment size of 4 LEs.

4) Time between fragments: Figure 11(a) shows how the message loss is affected by the time between fragments. As shown, message loss decreases from 0ms to 5ms between



(a) Time between fragments.

(b) Median time to receive full ledgers of length n.

Fig. 11: Impact on the message loss as a result of varying time between fragments and the number of ledger entries' effect on time to receive ledger.

fragments. However, for values higher than 5ms, the message loss increases with the time between fragments. Based on these results, 5ms has been chosen as the optimal value.

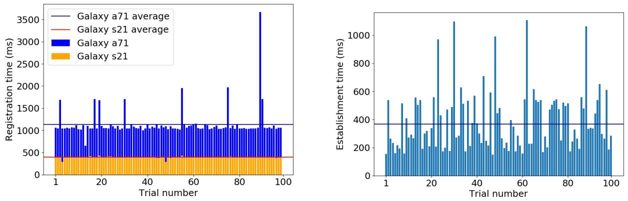
5) Accept ledger timer: The time, β, a user waits before accepting a ledger according to consensus criteria 2 and 3 should be chosen to ensures that most voting messages are received before selecting a ledger. Figure 11(b) shows the median time it takes to receive a full ledger of length n, which is the largest voting message for ledgers of size. n. This value is used to decide β. In reality, the users would receive several hashes in addition to the full ledger. Because all the messages are handled in the same thread on the device, many hashes would increase the time it takes to handle the responses, and should be considered when choosing the value of the timer.

If the value of β is set too low, a ledger might be accepted while a vote is still missing. As all legitimate users will most likely vote for the same ledger, one vote missing will not affect the system for large ledgers. If the value is too large, the user will have to wait longer before they can join the network.

By setting the value of β to 4000ms, full ledgers as large as 90 LEs are most likely to be received and the time is not expected to reduce the user experience.

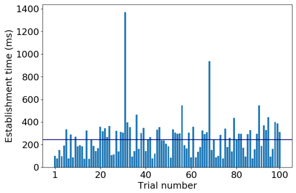
6) Alone in network timer: The time, α, a new user waits before concluding they are alone in the network should be chosen to give other users time to respond to the new user. As described in Section IV-A1 each packet is sent four times. The time between each transmission is chosen to be 400ms, as also observed in Section IV-A2. Therefore, the total time it takes from the first to the last transmission is 1200ms. That is the case for both the ledger request and its responses. If all but the last transmission is lost, it will take 2400ms plus the transmission time from the user sends the request until it receives a response. Hence, the user should wait 2500ms from the request is sent until the user concludes that they are alone in the network.

7) Idle time after request: To avoid a DoS attack by request flooding, a user waits γ seconds after responding to a request before the user will respond to new requests. The value for the idle time should be chosen to avoid all transmissions of a request from a legitimate user falling into this window.



(a) Sign up time.

(b) mTLS establishment times.



(c) TCP establishment times.

Fig. 12: Time required to sign up and establish mTLS and TCP connections.

Each request is transmitted four times with 400ms between each transmission, as discussed in Sections IV-A1 and IV-A2. The total time from the first to the last transmission is 900ms. At least two transmissions should always fall outside this window to increase the likelihood of receiving legitimate requests. The idle time must be less than 600ms to ensure that. Therefore the idle time, γ, is chosen to be 590ms.

B. Performance

This section presents the results from testing the applications' performance.

1) Sign up time: To use the application, the user signs up by creating an LE. For users without an existing certificate, this includes checking that the username is available, checking if the device has an Internet connection, and generating keys and certificates.

The test starts when the user has typed in the username and presses the sign-up button and finishes when the LE has been generated. Figure 12(a) shows the results from 100 tests using two different Samsung devices, Galaxy s21 and Galaxy a71.

On average, the Samsung Galaxy s21 used 393ms for sign-up, while the Samsung Galaxy a71 used 1136ms. The worst test runs resulted in test times of over 2s, which is a noticeable amount of time.

The difference in results between the phones is partly due to the difference in clock speed between the phones [16] [17]. The volatility in test times is related to the variability in the time it takes to ping the Google open DNS to check the Internet availability.

2) Connection establishment time: The proof of concept application uses a combination of asymmetric and symmetric encryption to achieve a lower connection establishment time. Two tests have been conducted to find the time it takes to establish a connection, one for each type of connection. The tests start when the user presses the LE of its peer and finish

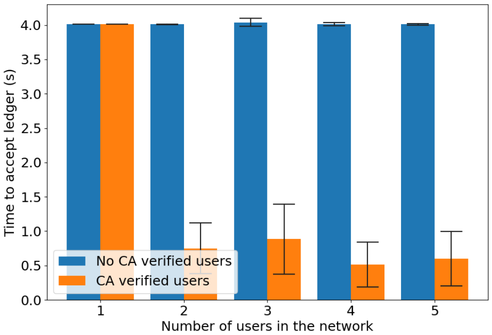


Fig. 13: The time it takes to accept a ledger is affected by the number of peers in the network.

when the connection is established. Figure 12(b) shows the connection establishment times for mTLS, and Figure 12(c) shows the connection establishment times for TCP.

The test results show that, on average, it takes 368ms to establish a connection using mTLS and 243ms using TCP. Even though establishing the pure TCP connection takes 34% less time than mTLS, the difference can be negligible.

3) Ledger acceptance: Before a user can sign up for the application, the user has to get the correct ledger or conclude that the user is alone in the network. Therefore, the time for accepting the ledger is an important performance metric. Tests have been conducted to measure this.

The test starts when the user opens the application and finishes when the application has accepted a ledger. The time it takes to accept a ledger depends on the number of users with CA-signed certificates, as described in the ledger acceptance criteria introduced in Section II-B3.

Figure 13 shows how the time to accept the ledger is affected by the number of users in the. Tests have been conducted for a network with CA verified users and in a network without any. The decreased time for more than one CA verified user shows how the first acceptance increases performance.

The probability that the n'th user joining the network accepts a ledger containing the LEs of all the users that have previously joined has been found to be 99% for up to 5 user.

4) Multicast packet loss: A test has been conducted to measure message loss. The test measures how many full ledger messages are lost during a voting. The test is conducted by making one user send a variable-length ledger to another user and recording if the message is lost. The tests have been conducted with the optimal variables found in Section IV-A.

Figure 14 shows that the size of the ledger affects the message loss negatively. Hence, the ledger size affects the performance, as the user is more likely to have to request the full ledger more times when the message loss is high. In particular, we observe that up to 30 ledger entries, i.e., users in the same network, the packet loss is limited to 1.7% which means that the systems enables a fairly robust ledger distribution for up to 30 users, i.e., small to medium-scale networks. Note that although the packet loss increases significantly as the number of users increase, this is not an upper bound on the network scalability as it simply indicates that there is a higher probability that they would be required to forward an additional full ledger request for reaching the consensus.

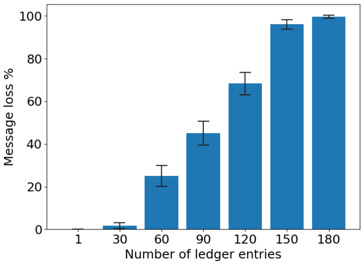
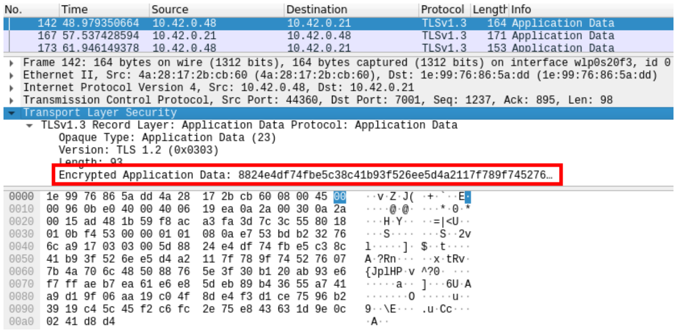


Fig. 14: Effect of ledger size on the message loss.



```
No.   Time          Source       Destination  Protocol Length Info
  142 48.979350664  10.42.0.48   10.42.0.21   TLSv1.3    164 Application Data
  167 57.537428594  10.42.0.21   10.42.0.48   TLSv1.3    171 Application Data
  173 61.946149378  10.42.0.48   10.42.0.21   TLSv1.3    153 Application Data
▸ Frame 142: 164 bytes on wire (1312 bits), 164 bytes captured (1312 bits) on interface wlp0s20f3, id 0
▸ Ethernet II, Src: 4a:28:17:2b:cb:60 (4a:28:17:2b:cb:60), Dst: 1e:99:76:86:5a:dd (1e:99:76:86:5a:dd)
▸ Internet Protocol Version 4, Src: 10.42.0.48, Dst: 10.42.0.21
▸ Transmission Control Protocol, Src Port: 44360, Dst Port: 7001, Seq: 1237, Ack: 895, Len: 98
▾ Transport Layer Security
  ▾ TLSv1.3 Record Layer: Application Data Protocol: Application Data
      Opaque Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 93
      Encrypted Application Data: 8824e4df74fbe5c38c41b93f526ee5d4a2117f789f745276…
0000  1e 99 76 86 5a dd 4a 28  17 2b cb 60 08 00 45 00   ··v·Z·J( ·+·`··E·
0010  00 96 0b e0 40 00 40 06  19 ea 0a 2a 00 30 0a 2a   ····@·@· ···*·0·*
0020  00 15 ad 48 1b 59 f8 ac  a3 fa 3d 7c 3c 55 80 18   ···H·Y·· ··=|<U··
0030  01 0b f4 53 00 00 01 01  08 0a e7 53 bd b2 32 76   ···S···· ···S··2v
0040  6c a9 17 03 03 00 5d 88  24 e4 df 74 fb e5 c3 8c   l·····]· $··t····
0050  41 b9 3f 52 6e e5 d4 a2  11 7f 78 9f 74 52 76 07   A·?Rn··· ··x·tRv·
0060  7b 4a 70 6c 48 50 88 76  5e 3f 30 b1 20 ab 93 e6   {JplHP·v ^?0· ···
0070  f7 ff ae b7 ea 61 e6 e8  5d eb 89 b4 36 55 a7 41   ·····a·· ]···6U·A
0080  a9 d1 9f 06 aa 19 c0 4f  8d e4 f3 d1 ce 75 96 b2   ·······O ·····u··
0090  39 19 c4 5c 45 f2 c6 fc  2e 75 e8 43 63 1d 9e 0c   9··\E··· .u·Cc···
00a0  02 41 d8 d4                                        ·A··
```

Fig. 15: Wireshark capture of TLS traffic.

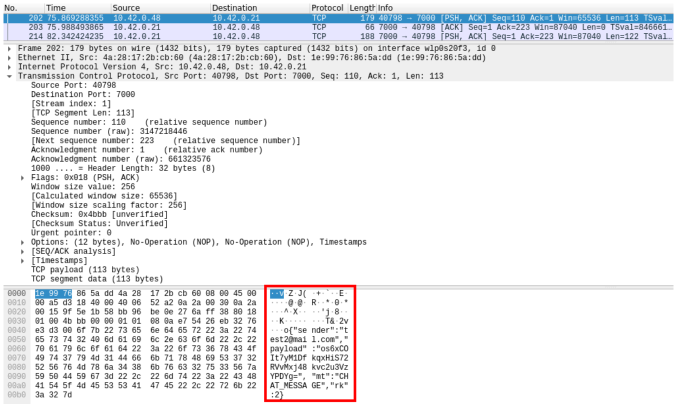
C. Security

Packets have been captured with and analyzed with Wireshark [18] to verify that the instant messaging traffic is encrypted. A computer was used as an AP to capture packets, and all packets were routed through it and saved. This section presents the security-related findings.

1) Message security: Figure 15 shows a Wireshark capture of a data packet the first time two users communicate. As expected, the packet payload is encrypted using TLS, and the plaintext can not be read without knowledge of the encryption key. In TLSv1.3 the version number gets the default value of 0x0303 - "TLS 1.2", as the version number field is not in use [6]. Hence, Wireshark shows TLS 1.2 in the version field even when TLSv1.3 is used. The protocol column shows the correct protocol, which is TLSv1.3.

Figure 16 shows a Wireshark capture of a data packet the second time two users communicate. The protocol column shows that TCP is used as the transport layer protocol. The payload is encrypted with AES on the application layer, and the ciphertext is shown in the red box in Figure 16. The encrypted payload while using TCP shows that the application works as expected.

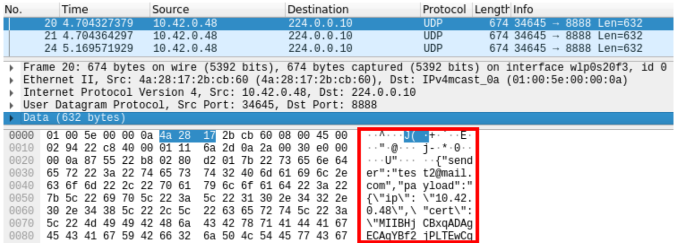
The multicast messages sent from the application are sent in cleartext using UDP. Figure 17 shows a multicast message



```
No.   Time          Source       Destination   Protocol Length Info
  202 75.869288355  10.42.0.48   10.42.0.21    TCP       179 40798 → 7000 [PSH, ACK] Seq=110 Ack=1 Win=65536 Len=113 TSval…
  203 75.988493865  10.42.0.21   10.42.0.48    TCP        66 7000 → 40798 [ACK] Seq=1 Ack=223 Win=87040 Len=0 TSval=846661…
  214 82.342424235  10.42.0.21   10.42.0.48    TCP       188 7000 → 40798 [PSH, ACK] Seq=1 Ack=223 Win=87040 Len=122 TSval…
▸ Frame 202: 179 bytes on wire (1432 bits), 179 bytes captured (1432 bits) on interface wlp0s20f3, id 0
▸ Ethernet II, Src: 4a:28:17:2b:cb:60 (4a:28:17:2b:cb:60), Dst: 1e:99:76:86:5a:dd (1e:99:76:86:5a:dd)
▸ Internet Protocol Version 4, Src: 10.42.0.48, Dst: 10.42.0.21
▾ Transmission Control Protocol, Src Port: 40798, Dst Port: 7000, Seq: 110, Ack: 1, Len: 113
    Source Port: 40798
    Destination Port: 7000
    [Stream index: 1]
    [TCP Segment Len: 113]
    Sequence number: 110    (relative sequence number)
    Sequence number (raw): 3147218446
    [Next sequence number: 223    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    Acknowledgment number (raw): 661323576
    1000 .... = Header Length: 32 bytes (8)
  ▸ Flags: 0x018 (PSH, ACK)
    Window size value: 256
    [Calculated window size: 65536]
    [Window size scaling factor: 256]
    Checksum: 0x4bbb [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▸ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▸ [SEQ/ACK analysis]
  ▸ [Timestamps]
    TCP payload (113 bytes)
    TCP segment data (113 bytes)
0000  1e 99 76 86 5a dd 4a 28  17 2b cb 60 08 00 45 00   ··v·Z·J( ·+·`··E·
0010  00 a5 d3 18 40 00 40 06  52 a2 0a 2a 00 30 0a 2a   ····@·@· R··*·0·*
0020  00 15 9f 5e 1b 58 bb 96  be 0e 27 6a ff 38 80 18   ···^·X·· ··'j·8··
0030  01 00 4b bb 00 00 01 01  08 0a e7 54 26 eb 32 76   ··K····· ···T&·2v
0040  e3 d3 00 6f 7b 22 73 65  6e 64 65 72 22 3a 22 74   ···o{"se nder":"t
0050  65 73 74 32 40 6d 61 69  6c 2e 63 6f 6d 22 2c 22   est2@mai l.com","
0060  70 61 79 6c 6f 61 64 22  3a 22 6f 73 36 78 43 4f   payload" :"os6xCO
0070  49 74 37 79 4d 31 44 66  6b 71 78 48 69 53 37 32   It7yM1Df kqxHiS72
0080  52 56 76 4d 78 6a 34 38  6b 76 63 32 75 33 56 7a   RVvMxj48 kvc2u3Vz
0090  59 50 44 59 67 3d 22 2c  22 6d 74 22 3a 22 43 48   YPDYg=", "mt":"CH
00a0  41 54 5f 4d 45 53 53 41  47 45 22 2c 22 72 6b 22   AT_MESSA GE","rk"
00b0  3a 32 7d                                           :2}
```

Fig. 16: Wireshark capture of TCP traffic.

broadcasting a LE. The message data is not encrypted and can be read by anyone listening to the multicast group. The message has been signed with the sender's private key.



```
No.   Time         Source       Destination   Protocol Length Info
   20 4.704327379  10.42.0.48   224.0.0.10    UDP       674 34645 → 8888 Len=632
   21 4.704364297  10.42.0.48   224.0.0.10    UDP       674 34645 → 8888 Len=632
   24 5.169571929  10.42.0.48   224.0.0.10    UDP       674 34645 → 8888 Len=632
▸ Frame 20: 674 bytes on wire (5392 bits), 674 bytes captured (5392 bits) on interface wlp0s20f3, id 0
▸ Ethernet II, Src: 4a:28:17:2b:cb:60 (4a:28:17:2b:cb:60), Dst: IPv4mcast_0a (01:00:5e:00:00:0a)
▸ Internet Protocol Version 4, Src: 10.42.0.48, Dst: 224.0.0.10
▸ User Datagram Protocol, Src Port: 34645, Dst Port: 8888
▸ Data (632 bytes)
0000  01 00 5e 00 00 0a 4a 28  17 2b cb 60 08 00 45 00   ··^···J( ·+·`··E·
0010  02 94 22 c8 40 00 01 11  6a 2d 0a 2a 00 30 e0 00   ··"·@··· j-·*·0··
0020  00 0a 87 55 22 b8 02 80  d2 01 7b 22 73 65 6e 64   ···U"··· ··{"send
0030  65 72 22 3a 22 74 65 73  74 32 40 6d 61 69 6c 2e   er":"tes t2@mail.
0040  63 6f 6d 22 2c 22 70 61  79 6c 6f 61 64 22 3a 22   com","pa yload":"
0050  7b 5c 22 69 70 5c 22 3a  5c 22 31 30 2e 34 32 2e   {\"ip\": \"10.42.
0060  30 2e 34 38 5c 22 2c 5c  22 63 65 72 74 5c 22 3a   0.48\",\ "cert\":
0070  5c 22 4d 49 49 42 48 6a  43 42 78 71 41 44 41 67   \"MIIBHj CBxqADAg
0080  45 43 41 67 59 42 66 32  6a 50 4c 54 45 77 43 67   ECAgYBf2 jPLTEwCg
```

Fig. 17: Wireshark capture of UDP traffic.

V. CONCLUSION

This paper presents a solution for authentication without a central trusted unit using a distributed authentication scheme and public and private cryptographic keys to provide secure communication. The solution is divided into three independent layers. The first layer presents a solution for setting up communication between the peers relying on Wi-Fi infrastructure. The second layer provides authentication by using a distributed ledger. All users receive the authentication material needed to authenticate their peers. The consensus mechanism ensures that the network as a whole agrees upon a user's identity and thus the user's authentication material. This solution relocates the responsibility of authentication from one single trusted unit to the whole network through cooperation between all network participants. Finally, the third layer provides secure communication relying on mTLS and symmetric encryption. The application's performance has been tested, and its encryption confirmed. In addition, challenges regarding ledger distribution have been identified and addressed through various parameterized mechanisms.

REFERENCES

[1] I. E. T. F. (IETF). (2018) The transport layer security (TLS) protocol version 1.3. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8446#appendix-C.2

[2] Oracle. (2010) Certificate-based authentication. [Online]. Available: https://docs.oracle.com/cd/E19575-01/820-2765/6nebir7eb/index.html

[3] "IEEE standard for information technology–telecommunications and information exchange between systems - local and metropolitan area networks–specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications," *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)*, pp. 1–4379, 2021.

[4] A. Sunyaev, *Distributed Ledger Technology*. Cham: Springer International Publishing, 2020, pp. 265–299. [Online]. Available: https://doi.org/10.1007/978-3-030-34957-8_9

[5] F. Saleh, "Blockchain without Waste: Proof-of-Stake," *The Review of Financial Studies*, vol. 34, no. 3, pp. 1156–1190, 07 2020. [Online]. Available: https://doi.org/10.1093/rfs/hhaa075

[6] E. Rescorla, "The transport layer security (TLS) protocol version 1.3," Internet Requests for Comments, IETF, RFC 8446, 8 2018. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8446

[7] Ø. Sigholt, B. Tola, and Y. Jiang, "Keeping connected when the mobile social network goes offline," in *2019 International conference on wireless and mobile computing, networking and communications (WiMob)*. IEEE, 2019, pp. 59–64.

[8] K. L. Skaug, E. B. Smebye, B. Tola, and Y. Jiang, "Keeping connected in internet-isolated locations," in *2022 Seventh International Conference On Mobile And Secure Services (MobiSecServ)*. IEEE, 2022, pp. 1–7.

[9] M. T. Hammi, B. Hammi, P. Bellot, and A. Serhrouchni, "Bubbles of trust: A decentralized blockchain-based authentication system for iot," *Computers & Security*, vol. 78, pp. 126–142, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404818300890

[10] U. Maurer and S. Wolf, "The diffie–hellman protocol," in *Designs, Codes and Cryptography 19*, 2000, p. 147–171.

[11] W. Stallings, *Cryptography and Network Security: Principles and Practice, Global Edition*. Pearson Education Limited, 2016.

[12] M. Marlinspike and T. Perrin, "The double ratchet algorithm," Open Whisper System, Tech. Rep., Nov. 2016.

[13] D. Cooper, S. Santesson, S. Farrel, S. Boeyen, R. Housley, and T. Polk, "Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile," Internet Requests for Comments, IETF, RFC 5280, 5 2008. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5280

[14] S. Aggarwal and N. Kumar, "Chapter twenty - attacks on blockchain," in *The Blockchain Technology for Secure and Smart Applications across Industry Verticals*, ser. Advances in Computers, S. Aggarwal, N. Kumar, and P. Raj, Eds. Elsevier, 2021, vol. 121, pp. 399–410. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0065245820300759

[15] J. Korhonen and Y. Wang, "Effect of packet size on loss rate and delay in wireless links," in *IEEE Wireless Communications and Networking Conference, 2005*, vol. 3, 2005, pp. 1608–1613 Vol. 3.

[16] L. SAMSUNG ELECTRONICS CO. (2022) Specifications galaxy s21 fe — s21 — s21+ 5g. [Online]. Available: https://www.samsung.com/no/smartphones/galaxy-s21-5g/specs/

[17] ——. (2022) Specifications galaxy s21 fe — s21 — s21+ 5g. [Online]. Available: https://www.samsung.com/no/business/smartphones/galaxy-a/galaxy-a71-a715-sm-a715fzkunee/

[18] Wireshark. (2022) Wireshark. [Online]. Available: https://www.wireshark.org/