Distributed Trust Empowerment for Secure Offline Communications

Endre Medhus Mæland, Sigmund Bernhard Berbom, Besmir Tola, and Yuming Jiang

Department of Information Security and Communication Technology
NTNU, Norwegian University of Science and Technology, Trondheim, Norway
Email:{endrebm, sigmunbb}@stud.ntnu.no | {besmir.tola, yuming.jiang}@ntnu.no

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 an extensive experimental study aiming at optimizing system parameters and investigating the performance of the application is conducted. The results from these measurements indicate that the solution performs well on small to medium-scale networks.

Keywords: Decentralized Authentication · Distributed Ledger Technology · Device-to-Device Communication · Peer-to-Peer Network · mTLS · Mobile Social Network.

1 Introduction

The use of certificates and private/public key pairs is a common way to provide authentication over the Internet [6]. Certificates enable end users to be authenticated from a centralized Certification Authority (CA), a trusted third party, in order to retrieve service [10]. 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 [1]. 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 [19]. However, most public DLTs have resource-consuming security

mechanisms tailored for financial transactions. Transactions 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 [12]. In article [5], 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 and the embedded consensus mechanisms can be unsuitable for resource-scarce mobile devices when exchanging authentication material.

The mutual authentication in mutual Transport Layer Security (mTLS) allows all parties to be authenticated, and the protocol also provides data encryption [11]. Furthermore, using symmetric encryption combined with mTLS can improve connection establishment times compared to only using mTLS. This is the case of a previous work addressing authentication in offline networks [15]. The work proposes a system where users receive authentication material when an Internet connection is available. 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. The study [17] improves this solution by using peer signed certificates to allow offline registration. The solution establishes a trust chain where CA-signed users can vouch for others, i.e., peer authenticated users. However, it is difficult to draw a clear line to what point in the chain a user is no longer trusted since only CA-signed users can vouch for others while peer authenticated users cannot. 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.

The remainder of the paper is structured as follows: Section 2 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 3. Successively, the validation of the security adopted in the architecture and the analysis of the experimental results are presented in Section 4. Finally, Section 5 concludes the paper.

2 Proposed solution

This section illustrates how the proposed solution enables authenticated and secure device-to-device communication in offline scenarios.

2.1 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 multicast

[1] `https://github.com/sberbom/masterProject`

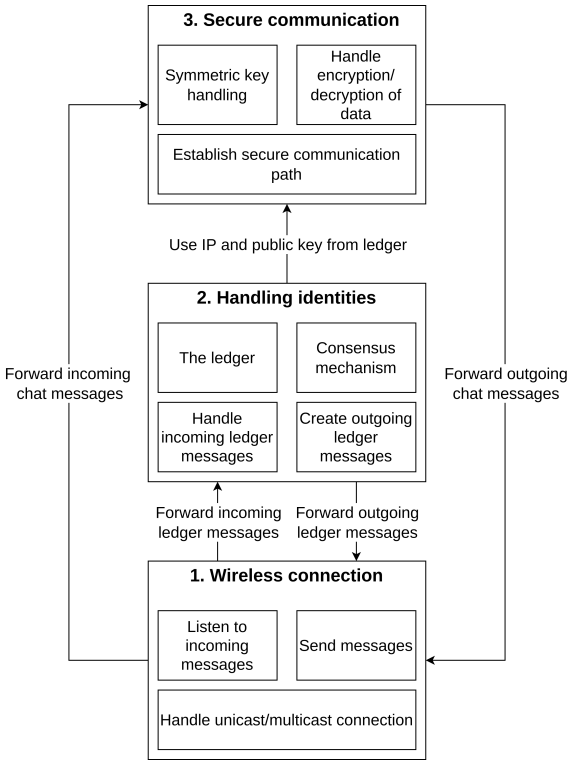


Fig. 1: High-level architecture of the solution.

and unicast transmissions utilized for ledger management and the actual service, respectively. This layer can also be built on other technologies but we chose Wi-Fi given the wide adoption of Wi-Fi in mobile devices. In addition, other Wi-Fi operating modes such as Wi-Fi Direct [16] or Wi-Fi Aware [17] can also be used but our choice of using infrastructure mode is driven by the higher coverage and speed provided by the use of a router access point. Moreover, using extended service set configuration of multiple access points acting as one single network allows for even larger coverage. Note that the router acts as a mere base station that performs forwarding and routing between mobile devices and does not provide access to the internet, de-facto enabling offline communication among peers.

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 wireless 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 [9] to be used the next time they communicate. From the next time these users connect, they will establish a TCP connection secured with Advanced Encryption Standard (AES) in Galois/Counter Mode (GCM) [18]. In addition, DH- and symmetric ratchets are used to improve the security of the symmetric keys [8] while reducing the time to establish the connection.

2.2 Distributed Authentication

This subsection describes how the private ledger is used to achieve distributed authentication. The ledger is an agreement among the users within the network on which users are part of the network and how to authenticate them. This achieved by distributing ledger blocks, i.e., entries, and maintaining a fully synchronized ledger containing network users authentication material.

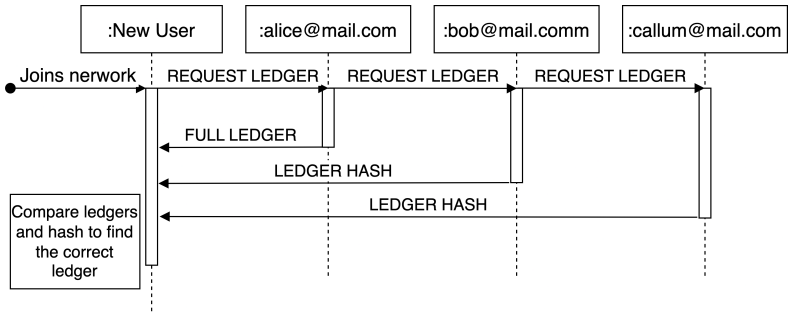


Fig. 2: Multicast message exchange when a new user joins the network.

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 [3] and an IP address. The certificate contains information about the username and public key only known to the user, and 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, which must be verified before receiving the CA-signed certificate. The self-signed certificate holds the same information but the binding between the user identity and its public key is not verified by a certification authority. Identifying the signature origin would simply require a comparison of the certificate issuer and the relative subject. In case they coincide, the certificate is self-issued.

Joining the ledger: Figure 2 illustrates the process of a new user joining an already established network. The new device 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. This request triggers a voting process where all responses are considered as votes. 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 the consensus mechanism, refer to the following section.

After receiving the ledger responses and achieving consensus on the valid ledger, the user store it locally and then creates 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.

In case the user does not receive any responses within α seconds, the user assumes there are no other users in the network. The user will then proceed to create its own LE, which at this point will make up the ledger. If the user has not received the full ledger, it will randomly pick one of the users who sent the hash of the 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 values of the α and β parameters are further discussed in Section 4.1.

All messages regarding the ledger are sent using UDP multicast. The choice of using UDP is primarly due to the support of multicast transmissions, as opposed to TCP. This is because the synchronization of the ledger is more efficient through one-to-many transmissions. In addition, UDP comes with lower overhead and various measures on the application layer have been added to alleviate the lack of reliable transfer of messages, refer to Section 3.4. Moreover, the ledger messages are signed with Secure Hash Algorithm-256 (SHA256) Elliptic Curve Digital Signature Algorithm (ECDSA) using the sender's private key and taking the message payload and the unique nonce as input.

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. The voting is triggered by the ledger request message sent by the new user. All users send a full ledger or a hash of their ledger and these messages are

interpreted as votes for the validation of the correct ledger by the new user. The full ledger is hashed before all hashes are compared. The process of learning the ledger ensures that the ledger accepted by the new user is the one held by the majority of the users. 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 [2] whose consequences would be a Denial of Service (DoS). To mitigate this, votes from users with CA-signed certificates are given priority in the consensus mechanism. 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. 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.

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 the following, and ensures no LEs are lost.

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.

Mitigating Ledger attacks: There are several security mechanisms implemented in the proposed solution to mitigate various types of security attacks. In the following we summarize the most important and present the mechanisms to ensure protection.

1) Vote Replay attack: if a valid vote in one voting process could also be valid in another voting, a malicious actor could exploit this to execute 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.

2) Ledger Rollback attack: an extension to the previous 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 occurrence, LEs are not removed from the ledger even when they are not a part of the accepted ledger.

3) DoS attack through request flooding: 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

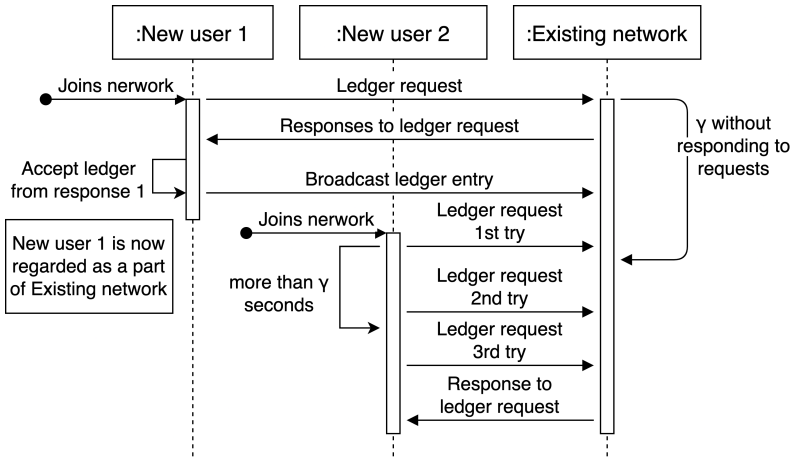


Fig. 3: If one ledger request falls within the time window where a request is dropped, another will fall outside it.

received within γ seconds after responding to a request for the ledger. The choice of the parameter γ is discussed in Section 4.1. 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 3.

4) Sybil attack: as previously mentioned, a Sybil attack entails an attacker simulating multiple users that send false information with the objective of steering a decision process such as the consensus voting. To protect from this form of attack, the consensus mechanism prioritizes votes from CA-signed users for the majority decision of the ledger validity. Obtaining a CA-signed certificate is the default behaviour when a user signs up for the service for the first time while having access to the internet. Only in the edge case of a user or a group of users being first time user(s) while offline they will use self-signed certificates but this remote possibility will be overrun with more users joining the network.

5) Spoofing attack: in this kind of attack, a malicious user tries to impersonate a legitimate user in order to make use of his privileges in the voting process. In order to succeed, it requires the knowledge of the users's private key hence, an attacker cannot spoof another user's identity.

6) Message forging attack: a ledger message forging attack would try to deceive the recipient(s) of a ledger message as if it had been sent by the original sender. In our setup all ledger messages are signed with sender's private key and hashed with SHA256 which ensures both origin authentication and message integrity.

2.3 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. With the introduction of a 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 4. 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. On the second time of interaction, 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.

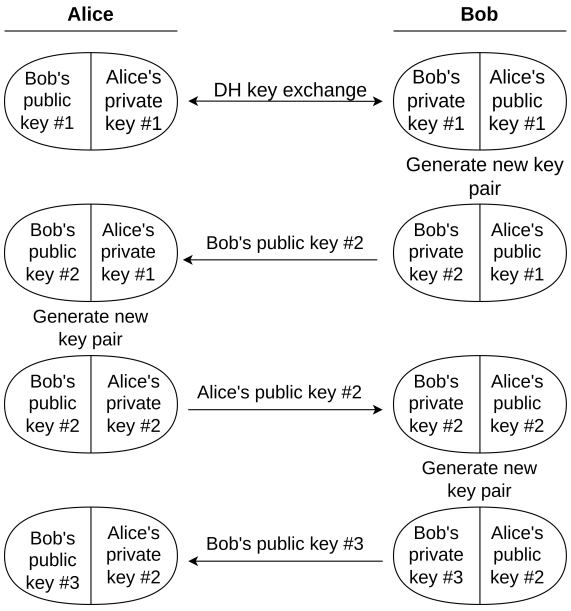


Fig. 4: DH-ratchet

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 [8].

3 System Design

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: i) a multicast server, ii) a multicast client, iii) a unicast server, iv) a unicast client, v) the ledger, and vi) the voting handler.

3.1 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 5 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.

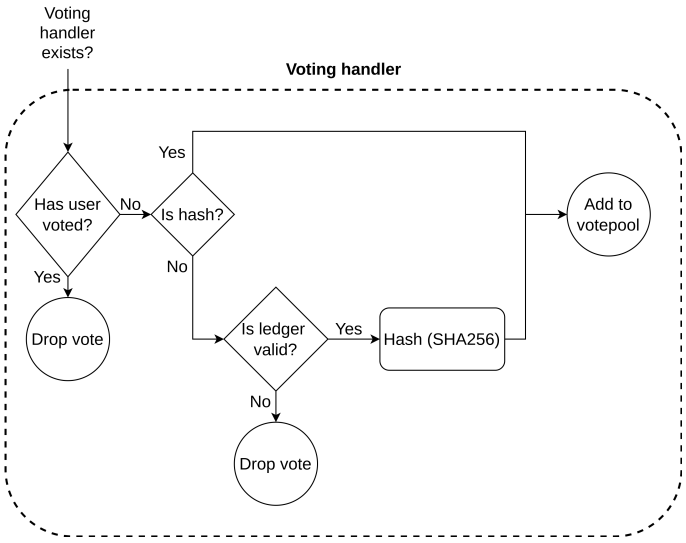


Fig. 5: Voting handler flowchart.

3.2 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.

3.3 Activities

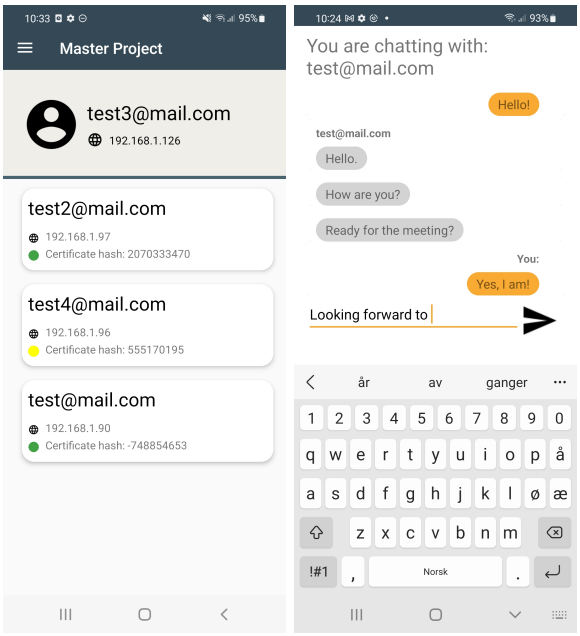
Android divides applications into different interfaces called activities. The proof-of-concept application consists of two primary activities: the main and the chat activity.

Figure 6(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 6(b) starts. In this activity, the users can read and send messages.

3.4 Ledger Design Parameters

The system must achieve a consensus on the ledger content to enable authentication of users. The ledger distribution can be challenged by the use of unreliable access and transport layers, i.e., wireless channel and UDP. The following parameters are used to tune mechanisms on the application layer that have been added to alleviate the eventual loss of ledger distribution messages: 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 4.



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

Fig. 6: Android Application Activities.

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

ii) *Fragment size*: When a sufficiently large ledger is sent, the message is divided into several fragments. This is motivated by various studies reporting that the packet loss rate is generally an increasing function of packet size for contention-based multiple access wireless channels, e.g. 802.11 [7] and 802.15.4 [4]. In the proof-of-concept application, one fragment holds n LEs. Every fragment must be received for the ledger to be counted as a valid vote.

iii) *Time between transmissions*: The time between transmissions is defined as the time from sending the last fragment of the ledger in one message 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 transmissions will increase the time it takes to sign up, potentially affecting the user experience.

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

4 Results

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

4.1 Optimizing Parameters

Specific tests have been conducted for optimizing the parameters identified in Section 3. 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. They 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 other parameters fixed. The fixed parameters' values are

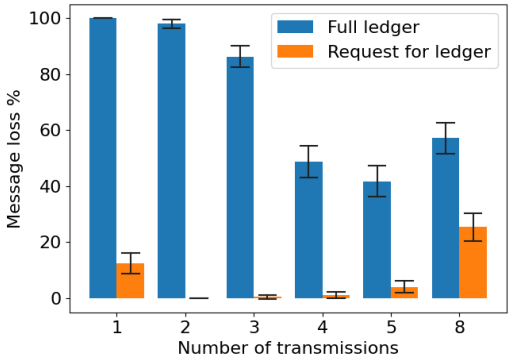


Fig. 7: Number of transmissions.

chosen to be near-optimal, based on a smaller sample of the performed tests. Unless otherwise specified, the results show the average of 300 runs and the related standard deviation.

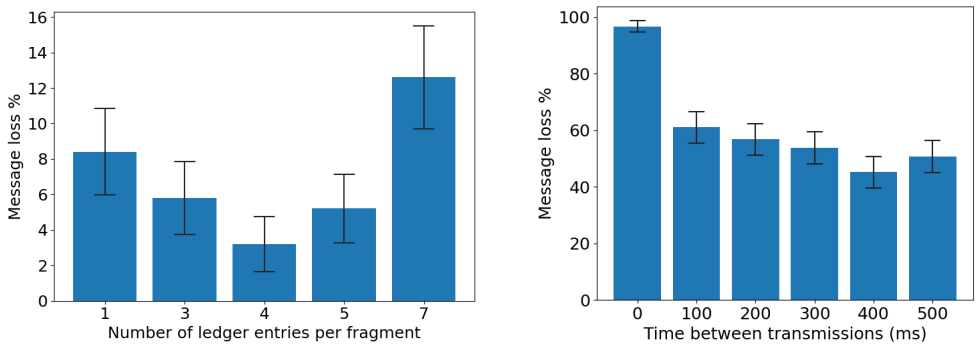
The test environment of the following tests is constructed to have a large message loss to increase the statistical significance. As the number of ledger entries in the pre-programmed ledger negatively affects the message loss, the number of LEs is chosen so that the message loss will be near 50% for 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 4.2.

Number of Transmissions: Figure 7 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, i.e., simulating 100 users in the network, 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 important, even though the message loss for larger ledgers is also of significance. Choosing 4 transmissions achieves a low message loss of the small messages while achieving an acceptable message loss of the larger messages, i.e., full ledgers.

Time Between Transmissions: Figure 8(b) 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.

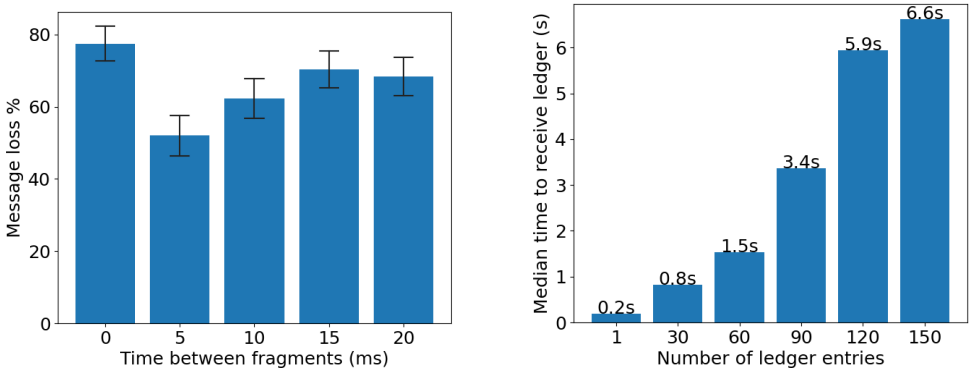
The number of transmissions has to be considered when analyzing the test results. While 400 ms is the optimal time between transmissions for four transmissions, according to these results, that might not be the case for another number of transmissions. With four transmissions, 400 ms between each transmission 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.

Ledger Fragment Size: Figure 8(a) shows the test results for message loss for different fragment sizes. On the one hand, a smaller number of ledger entries per fragment requires a higher number of fragments to be successfully received for a vote to be considered valid. Therefore, smaller fragment sizes exhibits higher message loss. On the other hand, a higher number of entries per



(a) Fragment size in terms of number of LEs. (b) Time between transmissions.

Fig. 8: Impact on the message loss as a result of varying time between transmissions and fragment size.



(a) Time between fragments. (b) Median time to receive full ledgers of length n.

Fig. 9: 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.

fragment increases the actual fragment size which is reflected in an increase of the message loss, as also reported on related literature investigating the impact that packet size has on packet loss rate [4, 7]. The observed results indicate that the best trade-off between too many fragments per ledger message and fewer but much larger fragments can be achieved for a fragment size of 4 LEs.

Time Between Fragments: Figure 9(a) shows how the message loss is affected by the time between fragments. As shown, message loss decreases from 0ms to 5ms between 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.

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 9(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.

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 4.1 each packet is sent four times. The time between each transmission is chosen to be 400ms, as also observed in Section 4.1. 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.

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.

Each request is transmitted four times with 400ms between each transmission, as discussed in the analysis regarding the *number of transmissions* and *the time between transmissions*. 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.

4.2 Performance

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

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.

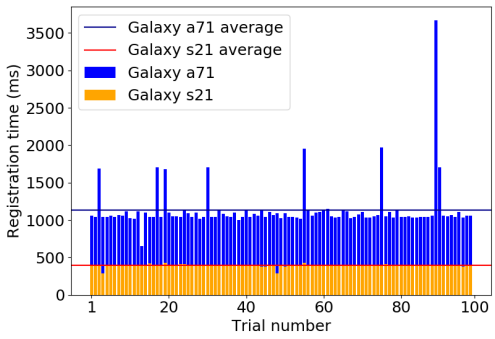
The test starts when the user has typed in the username and presses the sign-up button and finishes when the LE has been generated. For the following tests we have used two different Samsung devices, a Galaxy s21 and a Galaxy a71, and Figure 10(a) shows the results from 100 experimental runs.

On average, the Samsung Galaxy s21 used 393 ms for sign-up, while the Samsung Galaxy a71 used 1136 ms. The worst test runs resulted in test times of over 2 seconds, which is a noticeable amount of time, yet their occurrence was limited to just a few experimental runs.

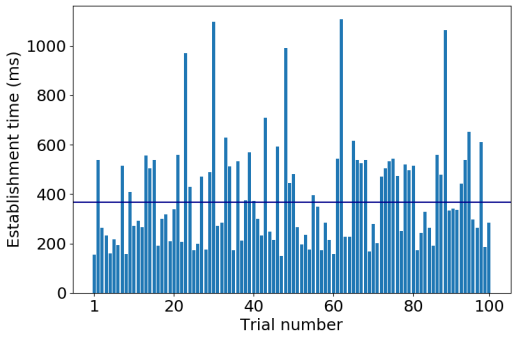
The difference in results between the phones is partly due to the difference in clock speed between the phones [13] [14]. The volatility in test times is related to the variability in the time it takes to ping the Google open DNS (8.8.8.8) to check the Internet availability.

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 finishes when the connection is established. Figure 10(b) shows the connection establishment times for mTLS, and Figure 10(c) shows the connection establishment times for TCP.

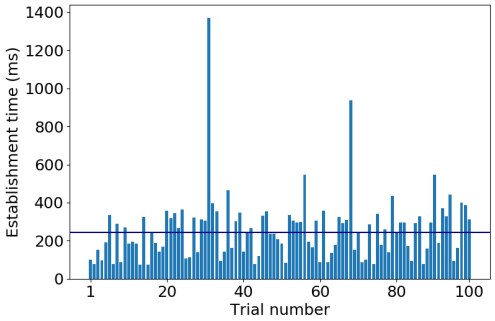
The test results show that, on average, it takes 368 ms to establish a connection using mTLS and 243 ms using TCP. Even though establishing the pure TCP connection takes 34% less time than mTLS, the difference can be negligible.



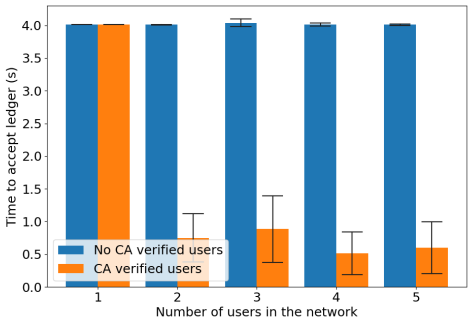
(a) Sign up time.



(b) mTLS establishment times.



(c) TCP establishment times.



(d) Time to accept a ledger vs. #users.

Fig. 10: Time required to sign up and establish mTLS and TCP connections.

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 2.2.

Figure 10(d) shows how the time to accept the ledger is affected by the number of users in the network. Tests have been conducted for a network with CA-verified users and in a network without any CA-signed users. The decreased time for more than one CA verified user shows how meeting the first condition of acceptance, refer to Section 2.2, not only ensures a higher level of trust but also increases performance as a result of avoiding timer overheads. In addition, the probability that the n'th user joining the network accepts a ledger containing the all the LEs of the already present users has been found to be 99% for up to 5 users, thus enabling the last user to establish communication with any of the other members within half a second.

Multicast Packet Loss: A test has been conducted to measure the ledger message loss under the optimal consensus parameter values identified in Section 4.1. The test measures how many full ledger messages are lost during a voting. The tests consists of one user sending a variable-length ledger to another user only once and recording if the message is lost. For each length of the ledger, the test is repeated 300 times and average values with related standard deviation are reported.

Figure 11 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 additional times when the message loss is high. In particular, we observe that for up to 30 ledger entries, i.e., users in the network, the packet loss is limited to 1.7% which means that the systems enables

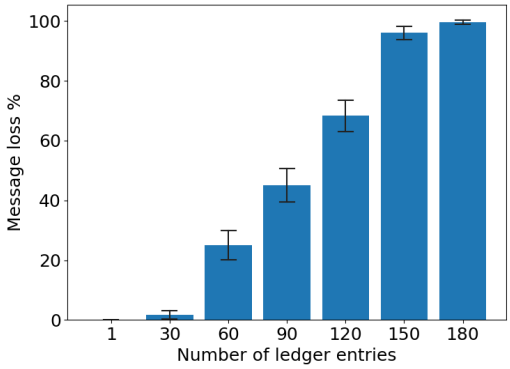


Fig. 11: Effect of ledger size on the message loss.

a fairly robust ledger distribution for 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.

4.3 Security

Packets have been captured with and analyzed with Wireshark [20] 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.

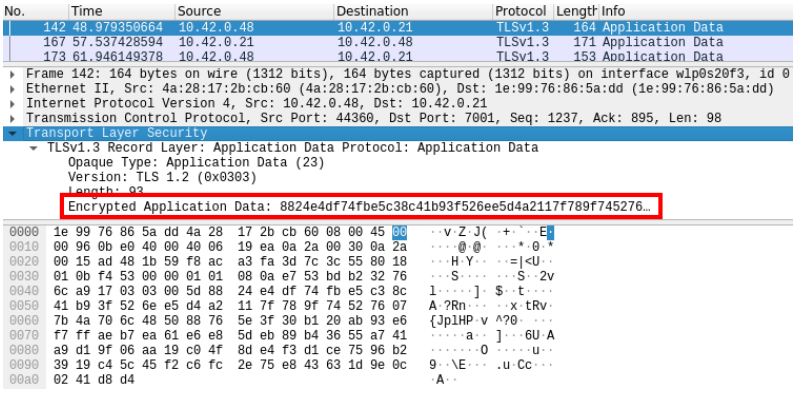


Fig. 12: Wireshark capture of TLS traffic.

Message security: Figure 12 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. Note that in Wireshark the TLSv1.3 gets the default value of 0x0303 - "TLS 1.2" because the version number field is not in use [11]. The protocol column shows the correct protocol version, which is TLSv1.3.

Figure 13 shows a Wireshark capture of a data packet the second time two users communicate. As expected by design, refer to Section 2.3, 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 13. The encrypted payload while using TCP shows that the application works as expected.

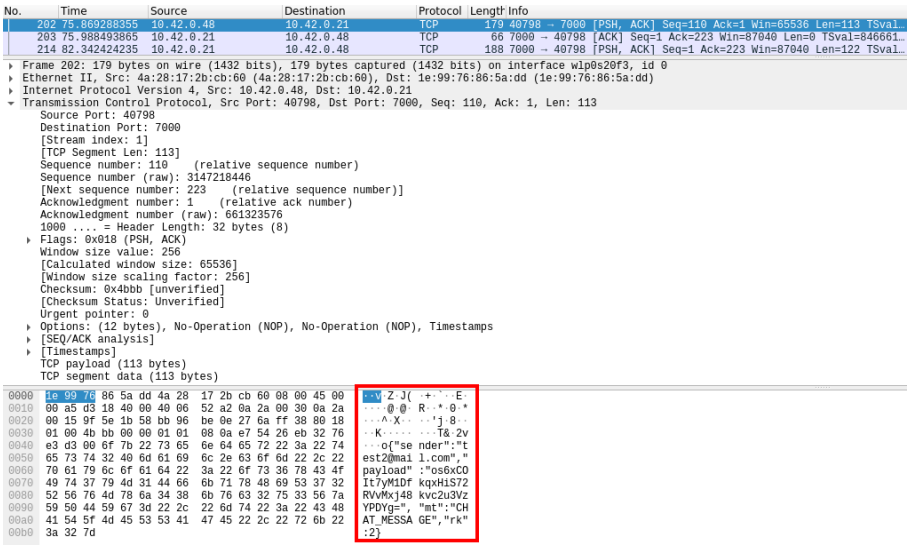


Fig. 13: Wireshark capture of TCP traffic.

The multicast messages sent from the application are sent through one-to-many communications using UDP. Figure 14 shows a multicast message broadcasting a LE. The message data is not encrypted and can be read by anyone listening to the multicast group since we consider this a public communication within the network but the messages preserve integrity and enable origin authentication through secure hashing and the signing using the sender's private key, as shown in the figure.

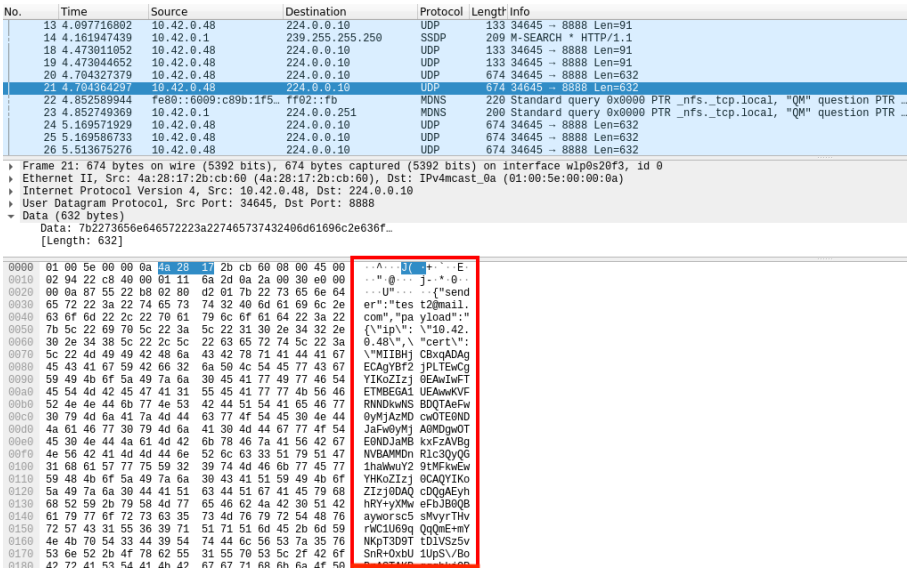


Fig. 14: Wireshark capture of UDP traffic.

5 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 offline communication. The solution is divided into three independent layers. The first layer presents a solution

for setting up offline communication between network peers by 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 and a 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. 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)
2. Aggarwal, S., Kumar, N.: Chapter twenty - attacks on blockchain. In: Aggarwal, S., Kumar, N., Raj, P. (eds.) The Blockchain Technology for Secure and Smart Applications across Industry Verticals, Advances in Computers, vol. 121, pp. 399–410. Elsevier (2021)
3. Cooper, D., Santesson, S., Farrel, S., Boeyen, S., Housley, R., Polk, T.: Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. RFC 5280, IETF (5 2008)
4. Fu, S., Zhang, Y., Ceriotti, M., Jiang, Y., Packeiser, M., Marrón, P.J.: Modeling packet loss rate of IEEE 802.15.4 links in diverse environmental conditions. In: 2018 IEEE Wireless Communications and Networking Conference, WCNC 2018, Barcelona, Spain, April 15-18, 2018. pp. 1–6. IEEE (2018)
5. Hammi, M.T., Hammi, B., Bellot, P., Serhrouchni, A.: Bubbles of trust: A decentralized blockchain-based authentication system for IoT. Computers & Security **78**, 126–142 (2018)
6. (IETF), I.E.T.F.: The transport layer security (TLS) protocol version 1.3 (2018), `https://datatracker.ietf.org/doc/html/rfc8446#appendix-C.2`
7. Korhonen, J., Wang, Y.: Effect of packet size on loss rate and delay in wireless links. In: IEEE Wireless Communications and Networking Conference, 2005. vol. 3, pp. 1608–1613 Vol. 3 (2005). https://doi.org/10.1109/WCNC.2005.1424754
8. Marlinspike, M., Perrin, T.: The double ratchet algorithm. Tech. rep., Open Whisper System (Nov 2016)
9. Maurer, U., Wolf, S.: The diffie–hellman protocol. In: Designs, Codes and Cryptography 19. p. 147–171 (2000). https://doi.org/10.1023/A:1008302122286
10. Oracle: Certificate-based authentication (2010), `https://docs.oracle.com/cd/E19575-01/820-2765/6nebir7eb/index.html`
11. Rescorla, E.: The transport layer security (TLS) protocol version 1.3. RFC 8446, IETF (8 2018), `https://datatracker.ietf.org/doc/html/rfc8446`
12. Saleh, F.: Blockchain without Waste: Proof-of-Stake. The Review of Financial Studies **34**(3), 1156–1190 (07 2020). https://doi.org/10.1093/rfs/hhaa075, `https://doi.org/10.1093/rfs/hhaa075`
13. SAMSUNG ELECTRONICS CO., L.: Specifications galaxy s21 fe — s21 — s21+ 5g (2022), `https://www.samsung.com/no/smartphones/galaxy-s21-5g/specs/`
14. SAMSUNG ELECTRONICS CO., L.: Specifications galaxy s21 fe — s21 — s21+ 5g (2022), `https://www.samsung.com/no/business/smartphones/galaxy-a/galaxy-a71-a715-sm-a715fzkunee/`
15. Sigholt, Ø., Tola, B., Jiang, Y.: Keeping connected when the mobile social network goes offline. In: 2019 International conference on wireless and mobile computing, networking and communications (WiMob). pp. 59–64. IEEE (2019)
16. Sigholt, O.L., Tola, B., Jiang, Y.: Keeping Connected When the Mobile Social Network Goes Offline. Master's thesis, Norwegian University of Science and Technology (2019)
17. Skaug, K.L., Smebye, E.B., Tola, B., Jiang, Y.: Keeping connected in internet-isolated locations. In: 2022 Seventh International Conference On Mobile And Secure Services (MobiSecServ). pp. 1–7. IEEE (2022)
18. Stallings, W.: Cryptography and Network Security: Principles and Practice, Global Edition. Pearson Education Limited (2016)
19. Sunyaev, A.: Distributed Ledger Technology, pp. 265–299. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-34957-8˙9, `https://doi.org/10.1007/978-3-030-34957-8_9`
20. Wireshark: Wireshark (2022), `https://www.wireshark.org/`