BlockChat - Blockchain Based Messaging

CPEN442 Project Final Report

Clarence Adrian, Cody Li, Jacky Le, Jayant Bhardwaj

Abstract

Many popular messaging applications require a centralized server to deliver messages between users. We propose our application, BlockChat, which uses blockchain to deliver encrypted messages as an alternative to existing messaging applications. Our approach should give users confidence that messages can only be read by the sender and receiver, and the properties inherent to blockchain should provide message integrity and authentication. However, decentralization and the costs associated with sending messages makes BlockChat a hard choice to consider as an individual's main secure messaging application.

1 Introduction

1.1 Problem and Significance

Over the past decade, the exponential increasing reliance on the internet for communication has brought large concern toward internet censorship and privacy. Countries such as China, Saudi Arabia, and Turkey are a few cases where the government has implemented unethical censorship policies in order to curb public dissent among its citizens [1].

However, censorship of social media is only one part of the problem. In another example, Iran blocked an open-source end-to-end encrypted messaging app, Signal, to prevent citizens from communicating [2] and uniting in their protests. Without access to instant messaging or communication with other protesters or organizers, fighting and participating in protests can become incredibly isolating and difficult.

1.2 System Design Summary

Our solution to the problem is BlockChat, a blockchain based messaging application that enables communication between any two participants on the blockchain. By using blockchain technology to help send and receive messages, our solution provides a decentralized method to online communication that supports message confidentiality and integrity via encryption. Messages are secured using the Signal Protocol, the same protocol other messaging applications such as Signal use.

1.3 Related Work

There has been much discourse regarding the advantages and disadvantages of a blockchain-based messaging system, however there have been no large-scale implementations of such a system to date. There exist popular messaging applications that provide end-to-end encryption, but part of the safety of these applications depend on the integrity of an unknown third party to deliver all messages. Decentralized messaging apps offer more security and privacy, but are less accessible and subject to network issues.

1.4 Evaluation Summary

Our evaluation methodology consisted of usability and security evaluations. For usability, we conducted multiple trials of essential tasks to gauge learnability, speed, efficiency and user preference. The results were largely positive thanks to our UI design being similar to existing messaging apps. For security, we found that due to how much power a government controlled ISP has, while we are resilient from simpler attacks such as 51% and port blocking, we are not able to fully deal with DPI (Deep Packet Inspection) attacks.

1.5 Contribution to Society and Cybersecurity Community

This application demonstrates how blockchain technology can be used as part of our daily lives. It focuses on providing message integrity and confidentiality without a need for a central authority. Popular pre-existing applications have had to face requests from governments demanding essentially backdoor access to end-to-end encrypted messaging applications [3]. We hope our work on this application focuses people's attention on how important online encrypted messaging is and how dependence on a single application can be detrimental.

2 Related Work

There has been much discourse regarding the advantages and disadvantages of a blockchain-based messaging system, but still there have been no large-scale implementations of such a system to date. Popular messaging apps like Signal provide end-to-end encryption on all messages in such a way that even Signal can read anyone's messages [4]. Other similar messaging applications exist like Whatsapp or Viber. However, these messaging platforms are often owned and operated by large companies, making them susceptible to government surveillance and targeting by malicious actors.

Decentralized messaging apps, on the other hand, do not rely on a central server and messages are delivered by following a peer-to-peer (P2P) model. However, these applications are not as widely used as end-to-end encrypted messaging platforms, and their complexity and technical requirements make them less accessible to the average user. Furthermore, decentralized P2P messaging apps are subject to network issues due to their decentralized nature. In a P2P network, each user's device acts as both a client and a server, allowing messages to be transmitted directly between devices without the need for a central server. This decentralized architecture makes the network more resilient and resistant to censorship, but it also means that the reliability of the messaging service is dependent on the availability and performance of individual devices on the network.

3 Adversary Model

We may assume that governments, nation-state actors, and wealthy competitors all have access to experienced personnel and have the ability to provide these personnel with as many resources as needed to attack a decentralized messaging platform. These entities may view the platform as a potential threat and attempt to attack it to gather intelligence, disrupt activities, or protect their interests. Blockchain-based chat applications are generally considered to be secure, but they can still be vulnerable to attacks. Some potential attacks by these entities include: denial of service (DoS), man-in-the-middle (MitM), Sybil, 51% attacks, and blockchain bloat. Government or nation-state actors may utilize more advanced capabilities, such as advanced persistent threat (APT) attacks, quantum computing attacks, blockchain fork attacks, or regulation/legislation.

Hackers and cybercriminals may also attempt to attack a blockchain-based chat application for financial gain or other malicious purposes. These individuals can range from highly skilled attackers to low skill phishers. Since they cannot directly alter the blockchain network, they are limited to attacks targeting individual users such as ransomware, phishing, malware, and wallet stealing.

4 System Design

Our project can be split into four major components: the implementation of the blockchain network, smart contracts, message encryption, and local storage. The blockchain network acts as a distributed data structure that stores messages in encrypted form. Using blockchain technology helps ensure message integrity and authenticity. Smart contracts act as the delivery mechanism on the blockchain that allows users to send and fetch messages from the blockchain. To ensure confidentiality, the messages are encrypted using the Signal protocol, which utilizes the Double Ratchet algorithm to generate new keys for each message while also allowing messages to be received out of order. Finally, the application will securely store the sent and received messages in an encrypted local database. Storing the messages on the user's system allows for messages to be reviewed offline, and encrypting the messages ensures that malicious actors will have a hard time reading the messages even if they have access to the user's system.

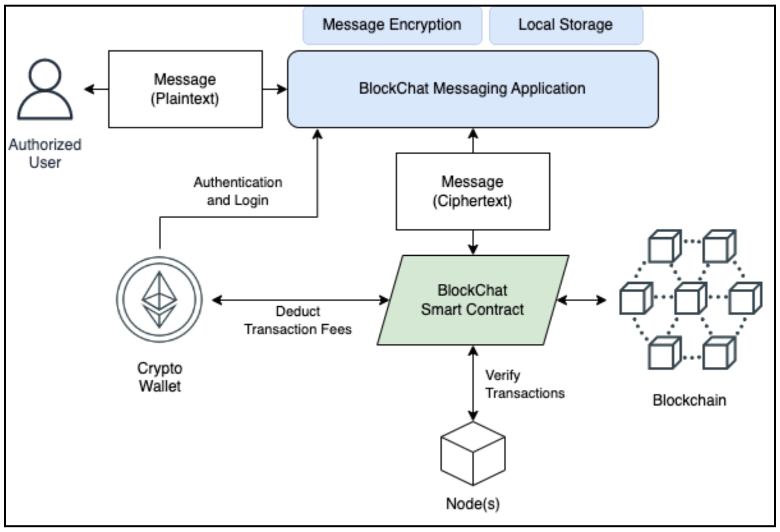


Figure 1: Diagram of Overall System Design

4.1 Blockchain Network

The blockchain is an implementation of a decentralized, digital ledger of transactions. Anything that can be recorded digitally can be stored on the blockchain and anyone with an adequate machine can access and contribute to it. This way, the use of a blockchain offers both transparency and accessibility of its data.

The immutability of data is another key feature offered by blockchains. Data in the blockchain is stored in blocks, where each block contains a list of validated transactions and other essential metadata. Additionally, each block must have a way to link to its previous block, such that multiple blocks will form a linked "chain". By taking advantage of a number of cryptographic principles, participants cannot change/tamper with data blocks that have already been recorded on the blockchain because they will "break" the links between blocks. A more detailed explanation of this functionality is visualized in Appendix A.

Decentralization is realized by relying on a network of nodes to each store a copy of the blockchain. Instead of depending on a central authority to manage a single source of truth, the blockchain relies on massive data replication to maintain data quality. Additionally, specialized nodes called validators are typically held responsible for participating in the validation of new data blocks. In this process, nodes need to agree on a set of rules, called a consensus mechanism. Nodes will use the consensus mechanism to verify new transactions and decide on the current state of the blockchain in the event of a conflict.

Beside the benefits of decentralization and immutability, a blockchain can be used to verify the transfer of digital assets (transactions) without the use of middlemen. However, because any mutative interaction with the blockchain requires some amount of work from both nodes and miners, some amount of payment is required in exchange for publishing new data (blocks/transactions) to the blockchain.

Our usage of the blockchain was decided largely because of the properties it offers in terms of immutability and decentralization. We recognized that these were imperative to have in a modern messaging app. By relying on a decentralized ledger, a central authority should have a hard time censoring communication between participants or preventing them altogether.

4.1.1 Ethereum

Currently, there are a number of blockchain networks that have been widely adopted like Ethereum and Bitcoin. Our choice to rely on the Ethereum network is grounded on its popularity as a time-tested tool and its ability to offer smart contract functionality.

4.1.2 Cryptocurrency Wallets

In order to conduct transactions on the blockchain, participants require some type of "identifier" that can be used to identify them as a sender/receiver in a transaction. The blockchain relies on public key cryptography, where participants are associated with a public key and a private key. The private key is used to sign transactions as the authorized user, while the public key is used to identify the participant. A "wallet" can be used to securely store the private key, while the public key can be included in transaction blocks or shared across the network without concern. Thus, the security of a person's message and identity will also depend on the security of the user's crypto wallet.

In theory, a wallet's public address cannot be traced back to the true identity of its user(s) - an exception to this is when the original user purposely or carelessly associates their wallet's public key to their own identity. An example of this is by obtaining a wallet from an identifiable cryptocurrency exchange [5].

4.2 Smart Contracts

Our system's messaging logic is governed largely by smart contracts that are written on top of the Ethereum blockchain. The smart contracts are written in Solidity, a high-level programming language specifically created for the development of smart contracts on a number of blockchain networks. Conveniently, Solidity features functionality that supports modular design and data type validation.

In our system, the smart contract automatically validates and executes the logic behind sending and receiving messages. Because interactions with the blockchain require some form of payment to compensate for the computational power consumed, the smart contract is also responsible for managing the fees for each transaction. Figure 2 shows a general breakdown of the functions of the smart contract and how it interacts with the blockchain, the sender/recipient, and other participating services in our system.

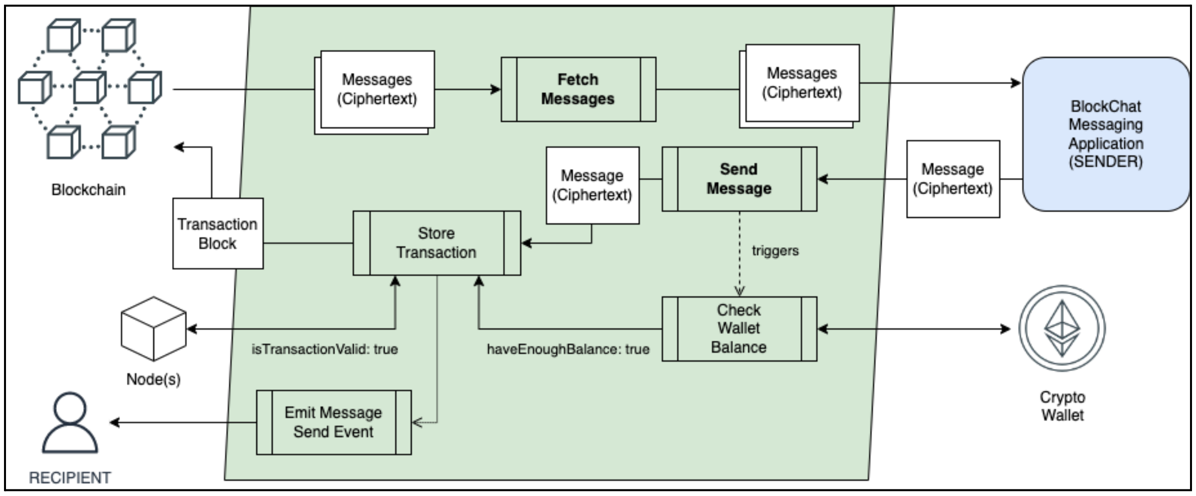


Figure 2: Diagram of Smart Contract Functionality and Design

During every interaction with the smart contract, it will typically check for the identity of the user by accepting the public key of a cryptocurrency wallet. At the beginning, a newly-opened application will typically want to get all the existing messages related to a certain other participant (the recipient). By sending the public address of the desired recipient, the smart contract will simply fetch all the existing messages from the blockchain and return it to the client (the sender).

In the case that a client wants to start sending messages, the smart contract will first check that the wallet associated with the sender has sufficient funds to send the transaction. Following this, it will wait for the transaction to be added to a block and validated. Once a message has been pushed onto the blockchain, it will emit an event that the recipient can listen for. As illustrated in figure 2, the smart contract does not handle message encryption/decryption and, since the blockchain will only store messages in encrypted form, will only ever manage data that have already been encrypted.

4.3 Signal Protocol

Our system provides secure messaging by using the Signal Protocol - more specifically, our system uses the Double Ratchet algorithm to provide message confidentiality and integrity [6]. The Double Ratchet protocol makes use of key derivation functions which take keys and an application specific input to produce different keys. The main goal of this algorithm is to be able to use unique message keys for each message and to allow parties to communicate without having to exchange the needed decryption keys beforehand. To begin, both communicating parties will have a shared key called the Root Key. This Root Key (labeled "RK" in Figure 3), will be used to generate sending (encryption) and receiving (decryption) keys using a key derivation function. The output of the key derivation function will be two keys - one key will be used for sending/receiving and another will be used as input to generate the next message key if the user wants to send the message. This process is illustrated in the "Root", "Sending", and "Receiving" columns in Figure 3, where the gray boxes represent the use of a key derivation function and the lines display show how the output of previous key derivation functions will be used as input for another key derivation function. The "RK" blocks in Figure 3 represent the Root Key, "CK" blocks represents Chain Keys that are used to generate sending/receiving keys, and "A#" and "B#" blocks represent message keys. The two communicating parties should have reversed chains, as in the sending chain of one party will be the receiving chain of the other party. This whole process gets reset every full message exchange where Diffie-Hellman (DH) public key values are exchanged to generate new keys. This is demonstrated in the "Ratchet" and "Root" Figure 3 columns where the public Diffie-Hellman key values are in cyan and the private values are in yellow. A more detailed explanation can be found in Appendix B.

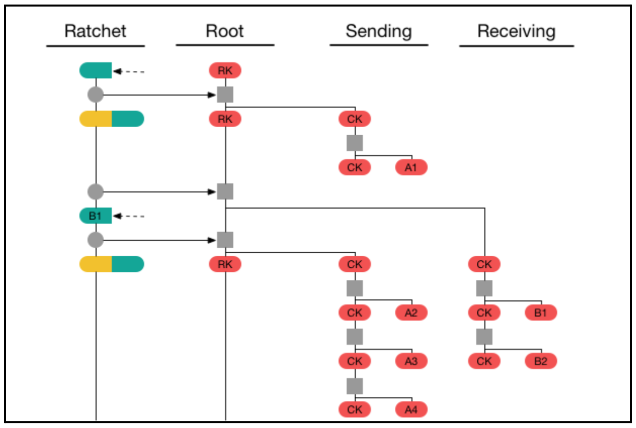


Figure 3: Diagram of Double Ratchet Algorithm from Signal [6]

Messages are encrypted using AES-256 in CBC cipher mode and an HMAC will be used to provide integrity. Using this algorithm also allows for messages to be received out of order since the algorithm will store previous message keys, and the only public information about the keys will be the public Diffie-Hellman key values used to help generate the Root Key.

Following this algorithm for message encryption and decryption applies the principle of open design. We are using a public algorithm that has existed for years and is widely used by other messaging applications such as Signal and Whatsapp who tout end-to-end encryption and security. Additionally, the algorithm applies the principle of time tested tools, as it utilizes well-tested and well-known cryptographic primitives for hashing, encryption, and key derivation. The algorithm also provides perfect forward secrecy as each message is encrypted with a separate encryption key. In the event that an attacker manages to learn one of the Root Keys, or one of the sending/receiving key keychains, the two communicating parties will continuously 'reset' their ratchets using the DH key exchange. Finally, this design applies the principle of sufficient work factor, as an attacker would need to compute a unique key and IV for every message sent that was encrypted using AES-256. The effort needed to decrypt an entire conversation between two parties by brute force is currently impossible relative to the length of a human lifetime.

4.4 Local Storage

The messages a user sends and receives will be stored locally in an encrypted database. By storing the messages locally, it gives users the ability to read messages offline, and the application can show past messages to the user more efficiently. Similarly, Signal stores messages locally on the user's device [7]. Since the application will manage the stored database, it reduces the need for other security measures like a password that would create more work for the user. The application will need the user's cryptocurrency wallet to begin with, and the user's system will already be locked by their system password or biometrics like Windows Hello.

Since we emulate Signal's methodology for message storage, we can consider our solution on par with other security focused applications. This design follows the security principle of open design as we rely on the same public methodology other applications apply. Additionally, it applies the principle of time tested tools as we can use an open source encrypted database such as SQLCipher instead of creating our own encrypted database to implement this functionality.

5 System Prototype

We evaluated our system design by developing a prototype web application using React and NodeJS. We also utilized Metamask and Ganache to test the functionality of the smart contract on a local blockchain. Metamask is a popular browser addon that allows users to create a cryptocurrency hot wallet. Ganache is a downloadable application that can create a local Ethereum-based blockchain. Our system used React to create the front-end of the application and NodeJS to write the message encryption protocols.

5.1 Web Application

In our system, we wanted to utilize available open-source frameworks such as the *web3* Javascript library which had been built to assist the development of applications that needed to interact with the Ethereum blockchain [8]. React was the library chosen for creating the user interface because it offers a lightweight and efficient solution that can take advantage of the existing *web3* library.

The first time a user opens the application, the user will have to create a password that will be used to authenticate the user for future uses and used to encrypt the plaintext messages the user sends and receives that will be stored locally. The password is hashed with PBKDF2 using SHA-256 and 310,000 iterations following the recommendations of the Open Web Application Security Project (OWASP) [9] and uses a randomly generated salt. To generate the encryption and authentication key, it uses PBKDF2 using SHA-512, 120,000 iterations, and a random salt as recommended by OWASP [9] to generate two 32 byte keys. The messages are encrypted using AES-256 in CBC ciphermode with a randomly generated IV, and the integrity of the messages is assured by using a HMAC that takes the ciphertext, IV, the random salt used to generate the keys concatenated together as input and hashed with the aforementioned generated authentication key from the password.

We used a password instead of an encrypted database since it was easier to prototype, but adding the password does make the application harder to use. The password puts additional work onto the user, and the application already has two barriers in place that block access to the application. One barrier is the password or biometric implementation required to login to the system and the other is the user's authentication for accessing Metamask, their digital wallet.

Next, the user will be able to select the address of the person they want to talk to on the sidebar. All the public addresses of nodes on the blockchain are visible in this sidebar. On an actual blockchain network with potentially thousands or millions of users, this method of selecting public addresses would be inefficient and user-unfriendly. However, using the local blockchain allows us to specify a small number of auto-generated public addresses which eases the process of testing the application.

Once the user has selected a recipient's public address, they are prompted by an interface similar to most other messaging applications. The interface will display messages previously exchanged between both parties as well as provide an input box for the user to type and send new messages. We designed this part of the application so it would closely mirror other messaging applications so users would have an easier time learning how to use the application.

Once the user closes the web application and tries to open the application again, the user will be prompted for their password.

5.2 Crypto API

The encryption algorithms were implemented using the Crypto library available from NodeJS. We predominantly used the Subtle module in Crypto, which provides an implementation of the Web Crypto API as recommended by the W3C [10]. Alongside the client application, we had to create a backend server that would port out of localhost to handle all the necessary cryptographic computations. We could not get a client side cryptography library working, so this was the alternative we used instead. This should still be safe as the API calls should only be exposed if someone has access to the user's system since the server ports out of localhost not a public IP address.

5.3 Local Blockchain

We used an open-source application, Ganache, to create a local blockchain for testing. Additionally, we used Metamask, a browser add-on which gives users access to a cryptocurrency hot wallet that is used for authentication and storing wallet data. In order to test both of them in unison, we needed to initialize test wallets on the local Ganache blockchain and imported them into Metamask for further use.

6 Evaluation

6.1 Evaluation Methodology

Evaluation of our system was focused on two areas: usability and security. We describe the methodologies below.

6.1.1 Useability Evaluation Methodology

We used four metrics to measure usability of our design: speed, efficiency, learnability and user preference.

For our first task, we measured how long it would take to send a message from first launching the web application. We calculated efficiency by counting the number of accidental clicks the user performed when attempting to complete the task. We measured learnability by having the user repeat the task over three trials, making sure to have an adequate amount of time between each trial. For user preference, we collected qualitative feedback by noting down users' opinions after performing the task as well as quantitative feedback (rating from 1-10) assessing the difficulty of the task.

For our second task, we measured how long it would take to view a message from the moment it is sent from a different user. We measured efficiency, learnability and user preference the same way as in the previous task.

6.1.2 Security Evaluation Methodology

We did not consider replay attacks or any kind of mutation attack of either message content, message sender or message receiver as the properties of blockchain is known to prevent them.

We focused on attacks that could be conducted by the adversaries we identified and considered attacks that can take down Ethereum nodes or compromise the consensus algorithm. We specifically looked at DPI (Deep Packet Inspection) and port blocking attacks possible by government controlled ISPs as well as 51% attacks commonly attempted on proof of stake cryptocurrencies.

6.2 Evaluation Results

6.2.1 Useability Evaluation Results

For task one, the learnability of sending messages is shown below:

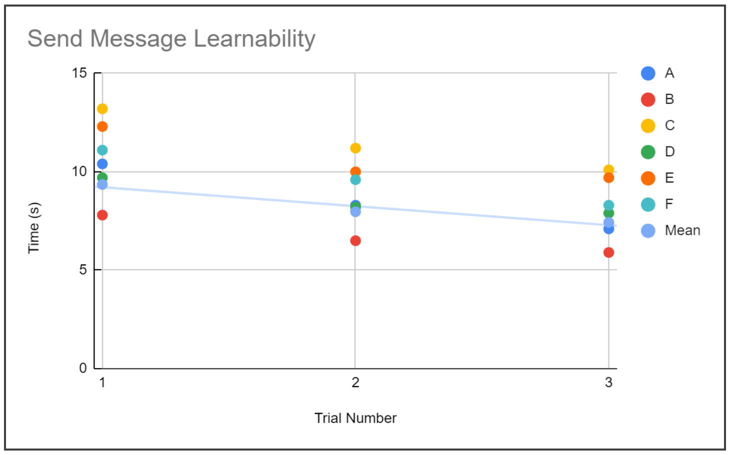


Figure 4: Learnability of Sending Message Task

Users took an overall average of 8.25s to complete the task of sending a message, with averages of 9.35s, 7.97s and 7.43s on each of the three trials respectively. The learnability was 29% from the first to third trial which indicated that the learnability is high. Users also made no mistakes and scored the task as 2.3 out of 10 in difficulty.

For task two, the learnability of viewing messages is shown below:

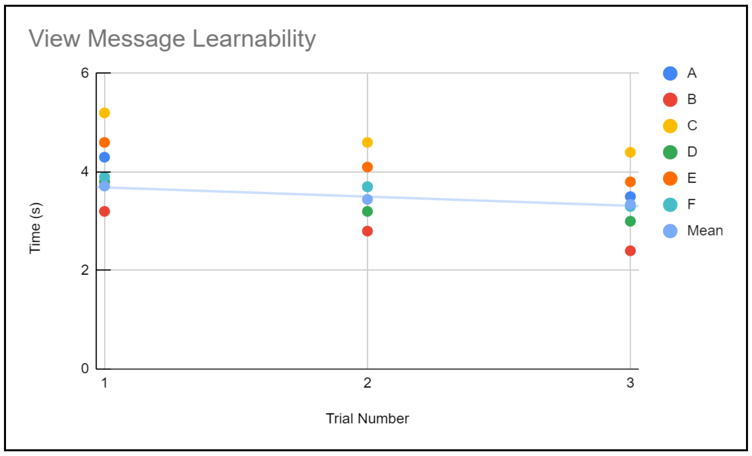


Figure 5: Learnability of Viewing Message Task

Users took an overall average of 3.5s to complete the task of viewing a message, with averages of 3.72s, 3.44s and 3.34s on each of the three trials respectively. The learnability was 11% from the first to third trial which is lower than ideal but it is likely due to the task already being fairly fast. Users also made no mistakes and scored the task as 2 out of 10 in difficulty.

Overall, the results were positive and users were able to comfortably complete the tasks. User surveys suggested that users were indifferent to using this user interface as opposed to other common messaging platforms such as Whatsapp or Messenger.

6.2.2 Security Evaluation Results

A port blocking attack is where an ISP blocks all traffic through certain ports. Ethereum nodes commonly use port 30303 by default and can change their port number, making sure to update their advertised port number. Nodes can try to jump around ports, potentially also communicating through third party channels to universally use a different port number. As a last resort, nodes can also use port 80, which is commonly used for HTTP. An internet service provider's (ISP) decision to block port 80 would effectively mean cutting off all general internet access. In that case, hypothetically no messaging apps would be able to function normally beside those that utilize short-range wireless technology such as bluetooth. Overall, our system is fairly resilient to this kind of attack.

Deep Packet Inspection (DPI) is a method of examining data traveling via networks. Using DPI, a malicious actor can look for common fields between ethereum packets and then block the IP addresses associated with the identified nodes. Since there are currently more than 400,000 staked validator nodes [11], assuming some are still discoverable through third party channels, it would be extremely difficult to block every single node. This is one advantage of using a large established cryptocurrency blockchain as opposed to creating one specifically for this application. Additionally, a VPN can be used to encrypt packets so that they cannot be examined by an ISP [12]. The use of a VPN can be effective in preventing DPI if the IP addresses of the VPNs are not blocked separately. Because DPI is a very resource-intensive attack method, only the most authoritarian governments would ever use this and the method is only effective as long as the ISP has the resources.

The last attack we are considering is a 51% attack where an attacker obtains control of a majority of the validator nodes and can therefore control a consensus. Ethereum requires a two third majority to reach a consensus. Attackers are much more likely to be incentivised by the monetary value of publishing an incorrect transaction than with changing an encrypted message. This attack can be very expensive because each validator node needs to stake 32 ETH (56000 CAD) which is burned when they go against the majority. Each transaction is also validated by a random pool of 128 validators. It would be very unlikely to obtain a majority without a large number of nodes. As previously stated with 400,000 staked validator nodes you would need more than 48 billion CAD to stake a majority, making this attack heavily reliant on economic power. Even if an attacker does have enough capital to fund this attack, the Ethereum community can mount a counter-attack by either creating a fork of the main blockchain or forcibly removing the attacker [13].

6.3 Discussion of Evaluation Results

Overall, our usability testing results demonstrate the user friendliness of our UI design. Tasks were completed quickly and efficiently with no mistakes as users took on average 8.25s and 3.5s for task 1 and 2 respectively. Additionally, tasks demonstrated high learnability as users improved 29% and 11% for task 1 and 2 respectively between trials. Lastly, users did not have any issues with task difficulty.

For the security evaluation, government controlled ISP's are a very strong adversary with likely vast amounts of resources and personnel available for attacks. This meant we would likely not be able to protect against all types of attacks from them, but we do have strategies of mitigation. In summary, our system is resistant to port blocking attacks unless port 80 is blocked and the internet is shut down. DPI attacks can be mitigated through the use of VPNs, unless the ISP chooses to block all VPN connections as well. The DPI attack is also only effective as long as the ISP has the resources to perform it. 51% attacks are mitigated largely due to the Etherum community and how costly the buy-in is. Overall, our approach still demonstrates good security against attacks by hacktivists or cybercriminals, as we are protected from any form of mutation or replay attack due to the properties of the blockchain.

7 Discussion

Our application utilizes the decentralized and distributed nature of blockchain technology to offer a high level of data integrity and authenticity. This allows for secure and transparent record-keeping of messages, making it difficult for anyone to tamper with or alter them once they have been sent. The decentralized nature of the technology also ensures that the application is not reliant on a single company or point of failure, making it more resilient and less vulnerable to downtime or disruption. The ability to create unique digital identities on the blockchain network allows for increased privacy and security, as it can potentially make it difficult for others to trace the origin of a message.

However, there are also disadvantages to using a blockchain-based chat app. For instance, users may need to pay gas or other cryptocurrency fees in order to send messages on the network, which could limit accessibility for some users. Additionally, messages on a blockchain-based chat application may take longer to be delivered than on traditional chat apps due to the need for verification and record-keeping on the blockchain network. While the application may be more difficult for a government to shut down from our security evaluation, it is still possible for it to be shut down through government legislation if it is deemed illegal or poses a threat to national security.

In comparison to traditional messaging apps, which are often owned and operated by large companies and therefore susceptible to government surveillance and data collection, decentralized messaging apps offer enhanced security and privacy. However, they are not as widely used due to their technical complexity and potential network issues arising from their decentralized nature. Our application addresses these limitations by leveraging the decentralized and encrypted data storage capabilities of blockchain technology, but this inclusion of blockchain comes with its own limitations. Requiring to pay cryptocurrency fees, and slow sending time make it hard to justify usage of our messaging app.

8 Conclusion

By using a blockchain to deliver messages between users, we provide a peer-to-peer based messaging application that does not depend on a central authority to securely store and deliver messages. The blockchain helps ensure message integrity and authenticity as the blockchain provides immutability of data. Messages can also be easily verified to ensure we users are communicating with the correct public address. However, the application suffers from the same drawbacks as cryptocurrencies when it comes to privacy. Our solution provides only pseudonymity not true anonymity, and the application requires ownership of some cryptocurrency to send messages. The message fee may deter users from wanting to use the application, and even if someone was willing to exchange fiat currency with the required cryptocurrency, that transaction between crypto wallets will be public and can be traced. Ultimately, this design makes it possible for encrypted messages to be delivered without a need for a central authority but this application does suffer from a lack of true anonymity that may make it hard for users to trust.

References

[1] Nairobi Newsroom. 2021. Uganda orders all social media to be blocked - letter. (January 2021). Retrieved October 10, 2022 from https://www.reuters.com/business/media-telecom/uganda-orders-all-social-media-be-blocked-letter-2021-01-12/

[2] Meredith Whittaker. 2022. Help people in Iran reconnect to Signal – a request to our community. (September 2022). Retrieved October 10, 2022 from https://www.signal.org/blog/run-a-proxy/

[3] Sebastian Klovig Skelton. 2022. Online Safety Bill returns to Parliament. (December 2022). Retrieved December 13, 2022 from https://www.computerweekly.com/news/252528153/Online-Safety-Bill-returns-to-Parliament

[4] Signal. Signal >> Home. Retrieved December 13, 2022 from https://signal.org/en/

[5] Peili Li, Haixia Xu, and Tianjun Ma. 2021. An efficient identity tracing scheme for blockchain-based systems. Information Sciences 561 (February 2021), 130–140. DOI:http://dx.doi.org/10.1016/j.ins.2021.01.081

[6] Moxie Marlinspike and Trevor Perrin. 2016. The Double Ratchet Algorithm. (November 2016). Retrieved December 5, 2022 from https://signal.org/docs/specifications/doubleratchet/

[7] Signal. Signal and the General Data Protection Regulation (GDPR). Retrieved December 6, 2022 from https://support.signal.org/hc/en-us/articles/360007059412-Signal-and-the-General-Data-Protection-Regulation-GDPR-

[8] Colin Adams. 2022. Web3.js repository migration announcement. (September 2022). Retrieved December 6, 2022 from https://blog.chainsafe.io/web3-js-repository-migration-announcement-96cbb34e0c7e

[9] Open Web Application Security Project. 2022. Password Storage Cheat Sheet. (October 2022). Retrieved December 6, 2022 from https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

[10] W3C. 2017. Web Cryptography API. (January 2017). Retrieved December 6, 2022 from https://www.w3.org/TR/WebCryptoAPI/

[11] Anon. Ethereum Validator Count Surge to over 435,000 after the Merge. Retrieved December 12, 2022 from https://www.binance.com/en/news/top/7214655

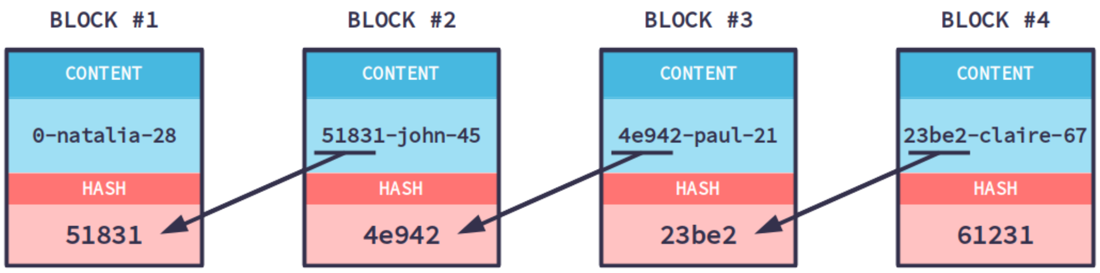
[12] Martynas Klimas. 2022. How do isps block sites & how to access them anyway. (September 2022). Retrieved December 12, 2022 from https://surfshark.com/blog/how-do-isps-block-sites.

[13] Anon. Proof-of-stake (POS). Retrieved December 12, 2022 from https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/

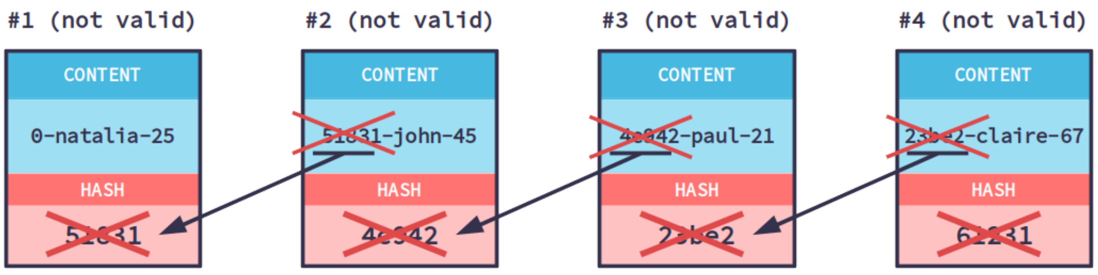
9 Appendices

Appendix A

The figure below illustrates a simplified view of a few linked blocks in a blockchain. In order for a block to be valid, a cryptographic problem needs to be solved by calculating an appropriate cryptographic hash based on the content of each block. This resulting hash is included in the content of the succeeding block. Shown below, the cryptographic hash of block 1 (*51831*) is included in the content of block 2, typically indicated as the "previous hash".



In the case that the content of block 1 is changed, or tampered with, the cryptographic problem needs to be solved again and its cryptographic hash needs to be re-calculated. This means that there is an inconsistency with the content of the next block, the link is "broken", and therefore the blockchain is invalid. In order to re-validate the blockchain, the cryptographic hashes of all the preceding blocks need to be re-calculated and re–validated, which is a time and resource-consuming process. The figure below illustrates this phenomenon.



Since most blockchains rely on a "longest valid chain" consensus mechanism, nodes with the tampered blockchain will need to compute the correct cryptographic hashes faster than the rest of the network in order to have a chance at competing to be the longest chain. Otherwise, non-malicious nodes will simply ignore this shorter chain.

Appendix B

The protocol for message encryption is as follows:

1. A root key needs to be established first between the two communicating parties. This can be established using Diffie-Hellman key exchange. For all the Diffie-Hellman key exchanges, we use Curve25519 (X25519) which is an algorithm that generates Diffie-Hellman key exchange values based on elliptic curves that Signal recommends [6]. The exchange is sent by sending a message request and a confirmation to that message request with both containing the involved parties' public key value of the Diffie-Hellman exchange. The message confirmation will also contain

another Diffie-Hellman public key that will be used for the next step. We will call the party that sent the message request, Alice, and the party that sends the message request confirmation, Bob.

2. Alice will generate another Diffie-Hellman pair using X25119 then generate two keys using a function we will call HK_RK. This function will take in two inputs, the previously generated root key and the public Diffie-Hellman key exchange value given in the message request confirmation. HK_RK generates two new keys by using a hash-based key derivation function (HKDF). The HKDF uses SHA-256 as the hash, the given root key as the salt, the computed output Diffie-Hellman key exchange output generated using the given public key as the input key material, and a constant byte sequence that will only be used in this function as the info value. One of the key outputs generated by HK_RK will replace the current root key and the other will be used as input for what will be called the Sending Chain key. The Diffie-Hellman values first generated in the first part of this step and the root key derived from the HK_RK function will be called the Diffie-Hellman Ratchet.
3. When Alice wants to send a message, Alice will call a function, KDF_CK, to generate two keys. The function KDF_CK takes one input, the Sending Chain key generated from the previous step, and it generates keys by using a hash-based message authentication code (HMAC) with SHA-256. The function generates two keys by using the same key on different inputs. Signal recommends using separate constants as inputs to the HMAC, one for each key [6]. Finally, one of the two keys generated by KDF_CK will be the new Chain Sending key for future use and the other key will be used for message encryption.
4. Alice will then create a message header containing the public value of the Diffie-Hellman key pair generated in Step 2, the number of messages sent using the previous Sending Chain by Bob (initialized to 0 for Alice's first message), and the number of sent messages using the most recent public value from Bob's Diffie-Hellman key pair.
5. To encrypt the message, Signal recommends using a form of Authenticated encryption with Associated Data (AEAD). More specifically, Signal recommends AES-256 in CBC mode with an HMAC for integrity [6]. Initially, it uses the message encryption key generated in Step 3 as input to a HKDF using SHA-256 to generate 80 bytes of output. The HKDF will take a sequence of 32 bytes all set to zero as the salt, the input will be the message encryption key mentioned previously, and the info will be a bye sequence used only in this function. The 80 bytes will be split into three parts, 32 bytes will be used as an encryption key for AES-256, another 32 bytes will be used as an authentication key, and the last 16 bytes will be used as the IV for AES-256. The application then takes the plaintext and encrypts it using AES-256 in CBC mode with the aforementioned key and IV. The HMAC will then be generated by concatenating the header and cipher text together and using the authentication key determined earlier from the 80 bytes. However, before it calculates the HMAC, it concatenates the header with a fixed constant set to 0. This constant is called the Associated Data. By including the extra constant into the HMAC, it helps provide extra integrity for the message as the associated data should be unique to this context, and it can be fixed to a constant since every message should be encrypted with a unique key.
6. On the receiving end, once Bob gets the message from Alice, Bob needs to set up their keys to decrypt the message. Bob will use the public key Diffie-Hellman value from the message header and their initial root key established between both Alice and Bob to generate Bob's Chain Sending and Chain Receiving keys. The Chain Receiving key will later be used as part of the process of decrypting Alice's message. The KDF_CK function will be called using the Chain Receiving key to generate the next receiving key and the message key required to decrypt Alice's message. It decrypts the message by breaking the message keys into different parts to decrypt the AES-256 encryption and verify the HMAC.
7. If a message gets received out of order, the application will save the message keys indexed by the public key Diffie-Hellman and the message position in the key chain if possible. It checks if any past key used the same public key value in the message header and if the message count in the

header matches any previously stored then it uses that message key for verification and decryption.