2023 IEEE 10th International Conference on Cyber Security and Cloud Computing (CSCloud)/2023 IEEE 9th International Conference on Edge Computing and Scalable Cloud (EdgeCom)

2023 IEEE 10th International Conference on Cyber Security and Cloud Computing (CSCloud)/2023 IEEE 9th International Conference on Edge Computing and Scalable Cloud (EdgeCom) | 979-8-3503-1246-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/CSCloud-EdgeCom58631.2023.00053

Quarks: A Secure and Decentralized Blockchain-Based Messaging Network

Mirza K. B. Shuhan[1], Tariqul Islam[2], Enam A. Shuvo[3], Faisal H. Bappy[4], Kamrul Hasan[5], and Carlos Caicedo [6]
[1] Software Research & Engineering, bKash Limited, Dhaka, Bangladesh
[2,4,6] School of Information Studies (iSchool), Syracuse University, Syracuse, NY, USA
[3] School of Computing, Asia Pacific University of Technology and Innovation, Kuala Lumpur, Malaysia
[5] Tennessee State University, Nashville, TN, USA
Email: *[shuhan.mirza@gmail.com, mtislam@syr.edu, enam.ahmed.shuvo@gmail.com, fbappy@syr.edu, mhasan1@tnstate.edu, ccaicedo@syr.edu]*

***Abstract*—Over the past two decades, the popularity of messaging systems has increased both in enterprise and consumer level. Many of these systems used secure protocols like end-to-end encryption to ensure strong security features such as "future secrecy" for one-to-one communication. However, the majority of them rely on centralized servers owned by big IT companies, which allows them to use their users' personal data. Also it allows the government to track and regulate their citizens' activities, which poses significant threats to "digital freedom". Also, these systems have failed to achieve security attributes like confidentiality, integrity, privacy, and future secrecy for group communications. In this paper, we present a novel blockchain-based secure messaging system named `Quarks` that overcomes the security pitfalls of the existing systems and eliminates the centralized control. We have analyzed our design of the system with security models and definitions from existing literature to demonstrate the system's reliability and usability. We have developed a Proof of Concept (PoC) of the `Quarks` system leveraging Distributed Ledger Technology (DLT), and conducted load testing on that. We noticed that our PoC system achieves all the desired attributes that are prevalent in a traditional centralized messaging scheme despite the limited capacity of the development and testing environment. Therefore, this assures us the applicability of such systems in near future if scaled up properly.**

***Index Terms*—instant messaging, blockchain, group communication, system security.**

I. Introduction

The popularity of Messaging Systems has been increasing for its prompt response time, convenient user experience, and ease of multi-tasking in both informal and formal communication and collaboration. Despite it's uprising popularity, there is also some concerns about the resiliency of these services and users' control over data. Because all popular messaging systems leverage central servers to route text messages [1]. This gives more control to big tech companies and government to provision the user activities.

The Internet's provision of digital freedom is not a new thing. It started with the use of metadata by big tech companies for profit-making opportunities. Centralized systems provide more control over user personal data, which can be sold to third parties or used for targeted marketing. Moreover, centralized systems are frequently monitored by governments in order to keep track of their citizens' activities, and some countries impose regulatory laws that potentially jeopardize user privacy. Telegram, for example, was banned in Russia in April 2018 because it refused to provide the Russian Federal Security Service (FSB) with access to encrypted messages, as required by anti-terrorism laws [2]. Centralized architectures are more likely to pose the biggest threats to privacy and freedom of speech, including single point of failure, data leakage, and control over private conversations [3]. Therefore, the current Web 2 foundation cannot guarantee privacy or freedom of expression.

To tackle mass surveillance of conversations by government agencies and large corporations, all major messaging applications are integrating end-to-end encryption to their protocols, such as Signal [4], WhatsApp [5], Threema [6], Google Allo [7], and Facebook Messenger [8]. End-to-end encryption has been proven [9] to be an outstanding way to implement secure messaging protocol that ensures additional security properties like *future secrecy* [10]. However, we have seen these messaging systems are vulnerable to malicious attacks due to the improper implementations and imbalance between "usability first" or "privacy first". These attacks include server-based attacks, such as vulnerability of Apple's iMessage [11] and Signal Protocol [12]. Moreover, researchers have been able to decipher the message database at end-users' devices of major messaging apps like WhatsApp, WeChat and Viber [13], [14], [15], [16].

Conventional models of data security rely on creating harder and harder walls– adding multiple factors of authentication to ensure access control and emphasizing stronger encryption. With Blockchain, there exists the potential to scatter the stack, rendering the cost of any one breach or combination of breaches much lower. Combined with strong encryption methods and zero knowledge proofs, Blockchain-based messaging systems can be a much more secure method of storing, accessing, and transmitting data; enhancing the ability of data managers to protect critical information.

Research Questions. The following research questions (RQs) intrigued us to investigate existing messaging systems and conduct this research. RQ_1: Can blockchain-based messaging systems be utilized to ensure digital freedom and eliminate control over user data? RQ_2: Is it possible to build a blockchain-based messaging system that includes all the features of a traditional messaging system? and RQ_3: Can blockchain-based

979-8-3503-1246-1/23/$31.00 ©2023 IEEE
DOI 10.1109/CSCloud-EdgeCom58631.2023.00053

(decentralized) messaging systems overcome the security flaws of existing centralized messaging models?

In this paper, we try to address the above-mentioned research questions by proposing Quarks, a novel application of Blockchain in decentralized messaging system which has not been explored before. Quarks eliminates central control over data and ensures true decentralization, security, privacy, and trust. The following are the major contributions of the paper.

- We developed a PoC of Quarks system that ensures "Message Integrity" and "Trust in Federation" using Distributed Ledger Technology (DLT).
- Quarks ensures digital freedom, future secrecy, and guarantees no single point of failure.
- We conducted performance and security analysis, which demonstrate that our PoC meets all intended features of a truly decentralized messaging system.

The rest of the paper is organized as follows. In Section II, we discuss some related work on the existing messaging systems. The architecture of the Quarks is presented in Section III. In Section IV, we describe the protocol flow of our Quarks system. We present implementation, system performance, and informal security analysis of Quarks in Section V. Finally, we conclude the paper in Section VI.

II. Related Works

Sarıtekin et al. [17] and Mirzaei et al. [18] propose communication applications named 'CrypTouch' and 'Simorgh' respectively, where both the schemes used IPFS (InterPlanetary File System) to store the messages off-chain. However, details about their network structure and protocol flow was not presented in the paper. Also, IPFS is suitable only for storing large files. In case of messaging systems, storing the messages in IPFS will add extra overhead as most of the messages are smaller in size. And it will require extra resources for each nodes to store and maintain the whole messaging system.

A chat application using Ethereum's Whisper protocol is proposed by Abdulaziz et al. [19]. This protocol is designed to "communicate darkness" (i.e., the content of the message is inaccessible to those who intercepts the messages, and that communicating nodes cannot be easily identified) at high cost. It is a off-chain protocol in which every message is sent to every node, and every node tries to decrypt the message. Consequently, the protocol has high traffic, high processor and memory usages. Hence, this is not an efficient protocol for chatting applications. Menegay et al. [20] attempt to implement a communication application using 'Steem', which is commonly referred as the "social blockchain", designed to power blockchain-based blogging and social media platforms. However, "Steem" is a public content platform which is not suitable for implementing communication application. A blockchain-based secure communication framework for community interaction is proposed by Sharma et al. [21]. Their scheme manages identity of network's user through third party centralized service (Google) and all communication data are kept in centralized database.

Currently, in most of the popular messaging systems, text messages are routed through central servers [1]. It makes the system prone to single-point of failure, although it provides the service-provider with fine-grained control over the system. Many companies, namely Signal [4], WhatsApp [5], Threema [6], Google Allo [7], and Facebook Messenger [8], have servers that are maintained by the service providers and they have full access to the sensitive conversation data which raises privacy concerns. Furthermore, researchers have been able to intercept sensitive information from WhatsApp [22], [23], [24], which has over two billion users worldwide. Rösler et al. [25] have analyzed the group communication of Whatsapp, Threema, and Signal. Their analysis disclosed that the *communication integrity* and the *groups' closeness* are not end-to-end protected. In addition, they proved that Signal protocol can not maintain strong security properties like *future secrecy* in group communication.

III. Background and System Architecture

In this section, we first review some preliminaries and then present our system architecture.

A. Background

Blockchain. Blockchain is a smartly engineered distributed system featuring an immutable ledger of transactions shared and validated by a number of distributed Peer-to-Peer (P2P) nodes [26]. The ledger is an ordered data structure consisting of many blocks chained together by cryptographic mechanisms.

Smart Contract (SC). Smart Contracts are computer programs deployed on top of the respective blockchain [27]. Being part of the ledger makes SC and their executions immutable and irreversible, a sought-after property having a wide range of applications in different domains.

Future Secrecy. Future secrecy is a prime feature of key agreement protocols in messaging systems which prevents an adversary (i.e., who compromises the message keys of a target user) from decrypting any future messages in the conversation to some extent. This is achieved by a unique technique called "ratcheting" in which session keys are updated with every message sent [28].

B. Quarks Components

The following four are the key components and participants of our proposed Quarks System. **i. User:** Users are the actors who use the system to communicate with their acquaintances; **ii. Quarks Channel:** A channel is a message thread between two or more users; **iii. Quarks Node:** A Quarks node independently hosts multiples users and their messages and **iv. Quarks Network:** A Quarks Network consists of multiple Quarks nodes where users from different nodes interact with each other.

C. High Level View of Quarks System

The high level semantics of the proposed system is illustrated in Fig. 1, where we can see that, our decentralized network can consist of multiple nodes that are hosted independently. A person can register as a user of a node. A user can: (a) create a channel, (b) invite other users to a channel, (c) join a channel upon invitation, (d) send messages to a channel, and (e) read

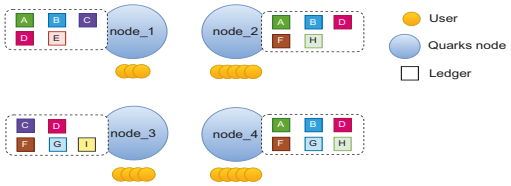


Fig. 1: High level view of Quarks System

TABLE I: Notations for Protocol Flow

Notations	Description
Q_i	A Quarks node in the network
CN_H	Channel Name of Channel H
SK_H	Symmetric Secret Key of Channel H
J_{Q_i}	A user registered in node Q_i
UN_J	Username of J_{Q_i}
$Adrs_{Q_i}$	Domain address of Q_i
$Adrs^J_{Q_i}$	Domain address of Q_i where J_{Q_i} has registered
W_J	Wallet of J_{Q_i}
$C_U^{Q_i}$	Certificate of J_{Q_i} issued by Q_i
K_J	Public key of J_{Q_i}
K_J^{-1}	Private key of J_{Q_i}
N_i	A fresh nonce
$\{\}_K$	Encryption operation using a public key K
$\{\}_{K^{-1}}$	Signature using a private key K^{-1}
$H(.)$	A hash function
$[]_{https}$	Communication over an HTTPS channel
msg	A text message
msg_i	ith msg in list of all textual messages
$MsgL$	List of text messages
$MsgL^K$	Every element is encrypted using K in this list
ts	A timestamp in nanoseconds
QB	Quarks Blockchain Network
SC_H	Smart Contract in Channel H
$Ldgr_H$	Ledger of Channel H

messages from a channel. Each node creates a blockchain ledger for every channel and hosts the messages of that channel in that ledger. A node hosts only the ledgers of which it's users are part of.

IV. Quarks Protocol Flow

In this section, we present the protocol flow between different components of our proposed system. The mathematical notations and symbols used to describe this protocol flow are listed in Table I. The protocol has seven phases: i) user registration; ii) channel creation; iii) node addition; iv) channel member addition; v) channel secret key retrieval; vi) message sending and vii) receiving.

1. User Registration Phase. [Table II]

M_1. Firstly, the user (A_{Q_1}) generates a pair of public and private keys (K_A, K_A^{-1}). Next, the user sends it's username, public key, and digital signature to the Quarks node (Q_1) through a https request.

M_2. The node (Q_1) validates the signature, generates a digital certificate ($C_A^{Q_1}$) for the user, and makes an entry in the off-chain database. Finally, the node returns the certificate and the nonce (N_1) to the user. A fresh nonce is used in every transmission to combat replay attacks.

2. Channel Creation Phase. [Table III]

M_3. The user (A_{Q_1}) requests the node (Q_1) to create a new channel through a https request. The request includes the name of the new channel (CN_H), certificate ($C_A^{Q_1}$), signature, and the channel secret key encrypted with the user's public key ($\{SK_H\}_{K_A}$). The secret key will be generated by the user creating the channel.

M_4. The node (Q_1) validates the user's certificate ($C_A^{Q_1}$), digital signature, and then creates a new ledger for this channel. Furthermore, the node sets up the smart contract for this channel and initiates the ledger for messaging. At initiation, the smart contract stores the encrypted secret key (SK_H) for this channel.

TABLE II: User Registration Protocol

M_1	$A_{Q_1} \rightarrow Q_1:$	$[N_1, UN_A, K_A, \{N_1, UN_A\}_{K_A^{-1}}]_{https}$
M_2	$Q_1 \rightarrow A_{Q_1}:$	$[N_1, C_A^{Q_1}]_{https}$

TABLE III: Channel Creation Protocol

M_3	$A_{Q_1} \rightarrow Q_1:$	$[N_1, C_A^{Q_1}, \{SK_H\}_{K_A}, CN_H, \{N_1, CN_H\}_{K_A^{-1}}]_{https}$
M_4	$Q_1 \rightarrow A_{Q_1}:$	$[N_1]_{https}$

3. Node Addition Phase. [Table IV and Algorithm-1 (lines 8-14)]

M_5. A member (A_{Q_1}) of the channel sends a request to the node (Q_1) for federating with a new node (Q_2) for the channel. The request includes member's certificate, signature, name of the channel, and the domain address of the new node.

M_6. The node (Q_1) validates the request, fetches the certificate, and invokes smart contract function for adding the node (Q_2) in the channel. The node (Q_1) then sends it's own certificate ($C_A^{Q_1}$) and the new node's certificate ($C_A^{Q_2}$).

M_7. The smart contract function validates the request and add the new node's certificate (Algorithm 1, lines 9-10) in the authorized list of nodes and sends back a response (N_2) to the node (Q_1).

M_8. After receiving the success response from the smart contract function, the node asks the new node to join the channel, synchronizes the ledger ($Ldgr_H$), and the smart contract (SC_H). The message contains the encrypted ledger and the smart contract.

M_9. The new node (Q_2) validates the request, creates a replica of the ledger, and sets up the smart contract in the ledger. Finally, the new node starts synchronizing the ledger and the smart contract with all the participating nodes of the channel and sends back a nonce (N_3) to the node(Q_1).

M_{10}. Upon successful addition of a new node (Q_2) in the channel, the node (Q_1) sends back a "success message" to the the requesting member (A_{Q_1}) of the channel.

4. Channel Member Addition Phase. [Table V and Algorithm-1 (lines 15-19)]

M_{11}. At first, a member (A_{Q_1}) of the channel sends a request to the node (Q_1) to add a user (UN_B) to the channel. The request contains certificate of the member ($C_A^{Q_1}$), the signature of the member, new username (UN_B), and the node's address ($Adrs^B_{Q_1}$) of the user.

TABLE IV: Node Addition Protocol

M_5	$A_{Q_1} \rightarrow Q_1$:	$[N_1, C_A^{Q_1}, Adrs_{Q_2}, CN_H, \{N_1, CN_H\}_{K_A^{-1}}]_{https}$
M_6	$Q_1 \rightarrow SC_H$:	$[N_2, C_{Q_1}, C_{Q_2}]_{https}$
M_7	$SC_H \rightarrow Q_1$:	$[N_2]_{https}$
M_8	$Q_1 \rightarrow Q_2$:	$[N_3, C_{Q_1}, Ldgr_H, SC_H]_{https}$
M_9	$Q_2 \rightarrow Q_1$:	$[N_3]_{https}$
M_{10}	$Q_1 \rightarrow A_{Q_1}$:	$[N_1, \text{success msg}]_{https}$

Algorithm 1: Smart Contract

1 **Input:** req ▷ function name and parameters
2 **Output:** $resp$ ▷ output of function
3 **Start**
4 $ChNodes \leftarrow$ `GetChNodeCerts()`
5 ▷ get certificate list of the nodes of the channel
6 $ChUsers \leftarrow$ `GetChUsersCerts()`
7 ▷ get certificate list of the users of the channel
8 ***fn*** **addNode**(N_2, C_{Q_1}, C_{Q_2})
9 **if** $C_{Q_1} \in ChNodes$ **then**
10 $ChNodes \leftarrow ChNodes \cup C_{Q2}$
11 ▷ add the new certificate to the set
12 `PutChNodeCerts(`$ChNodes$`)`
13 ▷ put the updated list in the ledger
14 **return** N_2
15 ***fn*** **addMember**$(N_3, C_{Q_1}, C_A^{Q_1}, C_B^{Q_1}, \{SK_H\}_{K_B})$
16 **if** $C_{Q_1} \in ChNodes \wedge C_A^{Q_1} \in ChUsers$ **then**
17 `PutSK(`$C_B^{Q_1}, \{SK_H\}_{K_B}$`)`
18 ▷ put encrypted secret key in the ledger
19 **return** N_3
20 ***fn*** **getChannelSK**$(N_2, C_{Q_1}, C_A^{Q_1})$
21 **if** $C_{Q_1} \in ChNodes \wedge C_A^{Q_1} \in ChUsers$ **then**
22 $\{SK_H\}_{K_A} \leftarrow$ `GetSK(`$C_A^{Q_1}$`)`
23 ▷ get encrypted secret key from ledger
24 **return** $N_2, \{SK_H\}_{K_A}$
25 ***fn*** **sendMsg**$(N_2, C_{Q_1}, C_A^{Q_1}, \{msg\}_{SK_H})$
26 **if** $C_{Q_1} \in ChNodes \wedge C_A^{Q_1} \in ChUsers$ **then**
27 $ts \leftarrow$ `GetTs()`
28 ▷ get current timestamp in nanoseconds
29 `PutState(`$ts, \{msg\}_{SK_H}$`)`
30 ▷ store encrypted message in the ledger
31 **return** N_2
32 ***fn*** **readMsg**$(N_2, C_{Q_1}, C_A^{Q_1}, ts)$
33 **if** $C_{Q_1} \in ChNodes \wedge C_A^{Q_1} \in ChUsers$ **then**
34 $ts_f \leftarrow ts$
35 $ts_t \leftarrow$ `GetTs()`
36 ▷ get current timestamp in nanoseconds
37 $MsgL^{SK_H} \leftarrow$ `GetStateByRange(`ts_f, ts_t`)`
38 ▷ get messages sent between ts_f and ts_t
39 **return** $N_2, MsgL^{SK_H}$

M_{12}. After request validation, the node fetches the user's certificate using user's node address and username. If the new member was registered on the same node, the node will be able to fetch it from it's local database. Otherwise, the node has to request the user's node to share the certificate of the user. Next, the node sends the requesting member (A_{Q_1}) the public key (K_B) of the user.

M_{13}. The member (A_{Q_1}) now encrypts the channel's secret key (SK_H) using the received public key (K_B) and send it to the node (Q_1).

TABLE V: Add Member Protocol

M_{11}	$A_{Q_1} \rightarrow Q_1$:	$[N_1, C_A^{Q_1}, UN_B, Adrs_{Q_1}^B, CN_H, \{N_1, CN_H\}_{K_A^{-1}}]_{https}$
M_{12}	$Q_1 \rightarrow A_{Q_1}$:	$[N_1, K_B]_{https}$
M_{13}	$A_{Q_1} \rightarrow Q_1$:	$[N_2, \{SK_H\}_{K_B}]_{https}$
M_{14}	$Q_1 \rightarrow SC_H$:	$[N_3, C_{Q_1}, C_A^{Q_1}, C_B^{Q_1}, \{SK_H\}_{K_B}]_{https}$
M_{15}	$SC_H \rightarrow Q_1$:	$[N_3]_{https}$
M_{16}	$Q_1 \rightarrow A_{Q_1}$:	$[N_2, \text{success msg}]_{https}$

M_{14}. The node invokes the smart contract of the channel to store the encrypted channel secret key. The node then sends the certificate of itself, the requesting member's certificate, the new user's certificate, and the encrypted channel secret key.

M_{15}. The smart contract function (Algorithm 1, line 17) stores the encrypted secret key in the ledger. As the key was encrypted using user's public key, only the entity having the private key will be able to decrypt this secret key.

M_{16}. Finally, after successfully invoking add member function of the smart contract, the node sends the requesting member a success message.

5. Channel Secret Key Retrieval Phase. [Table VI and Algorithm-1 (lines 20-24)]

M_{17}. The user requests the node to get the encrypted channel secret key. The request includes name of the channel, user's certificate, and the digital signature.

M_{18}. The node validates the user's request and checks if the channel exists. If the channel is found, the node invokes smart contract function (Algorithm 1, lines 21-22) for retrieving the encrypted secret key (SK_H) of the channel.

M_{19}. Upon validating the parameters, smart contract retrieves the encrypted secret key from the ledger and sends back the encrypted key to the node.

M_{20}. The node (Q_1) responds to user's request by sending back the encrypted key. The user (A_{Q_1}) will be able to decipher the key (SK_H) using it's private key (K^{-1}).

TABLE VI: Get Channel Secret Key Protocol

M_{17}	$A_{Q_1} \rightarrow Q_1$:	$[N_1, C_A^{Q_1}, CN_H, \{N_1, CN_H\}_{K_A^{-1}}]_{https}$
M_{18}	$Q_1 \rightarrow SC_H$:	$[N_2, C_{Q_1}, C_A^{Q_1}]_{https}$
M_{19}	$SC_H \rightarrow Q_1$:	$[N_2, \{SK_H\}_{K_A}]_{https}$
M_{20}	$Q_1 \rightarrow A_{Q_1}$:	$[N_1, \{SK_H\}_{K_A}]_{https}$

6. Message Sending Phase. [Table VII and Algorithm-1 (lines 25-31)]

M_{21}. The user (A_{Q_1}) sends, the message encrypted with the channel secret key, user's certificate, name of the channel, and the signature to the node (Q_1).

M_{22}. The node (Q_1) validates the request, invokes the smart contract function (Algorithm 1, line 25) for sending messages in the channel.

M_{23}. The smart contract (SC_H) function validates the user's membership in the channel (Algorithm 1, line 26) and if succeeds, puts the encrypted message in the ledger.

M_{24}. Finally, the node (Q_1) sends back a success message to the user (A_{Q_1}) after a successful smart contract invocation.

TABLE VII: Send message protocol

M_{21}	$A_{Q_1} \rightarrow Q_1$:	$[N_1, C_A^{Q_1}, \{msg\}_{SK_H}$ $CN_H, \{N_1, CN_H\}_{K_A^{-1}}]_{https}$
M_{21}	$Q_1 \rightarrow SC_H$:	$[N_2, C_{Q_1}, C_A^{Q_1},$ $\{msg\}_{SK_H}]_{https}$
M_{23}	$SC_H \rightarrow Q_1$:	$[N_2]_{https}$
M_{24}	$Q_1 \rightarrow A_{Q_1}$:	$[N_1, \text{success msg}]_{https}$

7. Message Reading Phase. [Table VIII and Algorithm-1 (lines 32-39)]

M_{25}. The user (A_{Q_1}) asks the node (Q_1) to fetch all messages in a period of time. The request includes the channel's name, certificate, signature, and a timestamp (in nanoseconds).

M_{26}. After validating the request, the node invokes the smart contract function (Algorithm 1, lines 32) to read the contents of the channel. The node passes the timestamp along with the certificates of the node and the user as function parameters.

M_{27}. Once the validation of parameters is done, the smart contract queries the encrypted messages from the ledger and sends back the message list ($MsgL^{SK_H}$) to the node.

M_{28}. The node (Q_1) sends back the message list to the user (A_{Q_1}). Finally, the user will be able to read the message contents by decrypting the messages using channel secret key.

TABLE VIII: Read message protocol

M_{25}	$A_{Q_1} \rightarrow Q_1$:	$[N_1, C_A^{Q_1}, ts$ $CN_H, \{N_1, CN_H\}_{K_A^{-1}}]_{https}$
M_{26}	$Q_1 \rightarrow SC_H$:	$[N_2, C_{Q_1}, C_A^{Q_1},$ $ts]_{https}$
M_{27}	$SC_H \rightarrow Q_1$:	$[N_2, MsgL^{SK_H}]_{https}$
M_{28}	$Q_1 \rightarrow A_{Q_1}$:	$[N_1, MsgL^{SK_H}]_{https}$

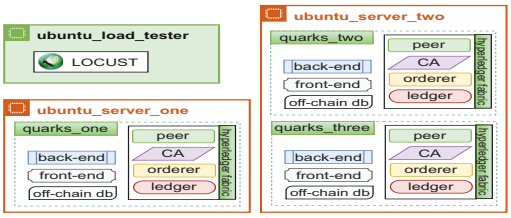


Fig. 2: Proof-of-Concept of Quarks System

V. Implementation

We have implemented a proof-of-concept and analyzed it's performance. We have recorded the throughput and latency of (a) Send Message and (b) Read Message; simulating with increasing loads using an open source load testing tool named Locust [29].

A. Proof of Concept (PoC)

We leveraged *Hyperledger Fabric* [30] to implement the blockchain network. Here, every Quarks node have their own Certificate Authority (CA), an Orderer, and multiple peers. We developed fabric chaincode using golang [31] for implementing *Smart Contract*. For *Ledger*, a fabric-channel [32] is opened for every quarks channel. We built a backend service using NodeJs [33] so that our clients can interact with the blockchain. We used ReactJS [34] for our *Frontend* and MongoDB [35] for the off-chain database.

B. Performance Analysis

For testing the performance of our PoC, we have set up a controlled simulation environment targeting a maximum of 100 users. Figure 2 shows the structure of our environment. It contains a Locust and 2 Ubuntu servers containing 3 nodes of the Quarks system. The simulation is carried out in a series of cycles that vary depending on the number of users. First, we start with 20 users and add 20 more users in each cycle till 100 users. Each cycle contains the following three steps: i) each simulated user sends and reads messages from the Quarks using a REST API; ii) for each API request, the response time is measured (in ms); and iii) on the Locust end, the throughput is measured by calculating the number of requests served per second.

After simulating for 100 users, we also performed a stress test on our environment. We added 50 additional users in 5 more cycles to check if the system can handle excessive loads. Figure 3(a) and 3(b) shows the median response time for all our test cycles. For usual test cycles (20-100 users), the response time is decent considering the decentralized architecture. It increases linearly as the number of users increases. And for stress testing cycles, the response time is respectively higher. However, our system can handle all requests of up to 150 users with its limited capacity. In figure 3(c), the system's throughput increases with the number of users in usual test cycles. As the number of users grows, so does the number of requests per second (throughput). However, in stress test cycles figure 3(d), the throughput reaches a saturated level. This indicates that our PoC has reached its maximum capacity.

Though we tested our PoC in a small simulation environment, it showed 100% availability and decent performance in sending and reading messages. We believe that with proper resources, Quarks could be scaled to use as a decentralized messaging system. Also, in the upcoming era of 5G and the increasing popularity of DApps (Decentralized Applications), Quarks will be more feasible to use and can be integrated with other decentralized systems for secured message sharing.

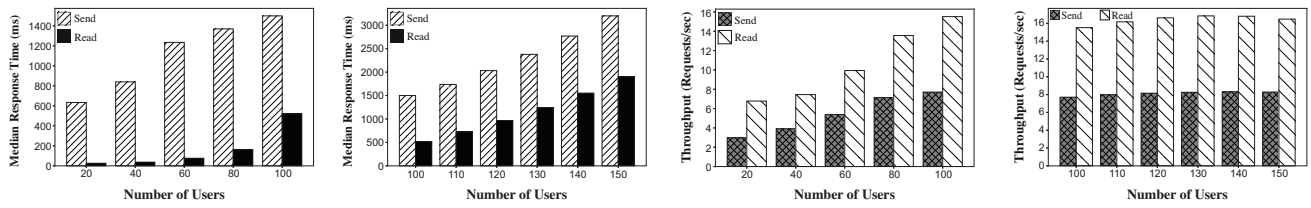
C. Informal Security analysis of Quarks

Confidentiality. Every message in the ledger in Quarks is encrypted with a secret key unique to a channel. This channel's secret key is only known to the members of the channel. This guarantees protection against potential data breaches.

Integrity. In Quarks all the messages are stored on-chain. That means it is impossible to delete or tamper the previous messages.

Non-Repudiation. As every message-write is digitally signed by the members and signatures are being verified by smart contract, Quarks diminishes repudiation attempts.

Authentication and Authorization. Users interact with the Quarks by authenticating with their private keys which enables



(a) Number of users VS Median response time in usual test cycles

(b) Number of users VS Median response time in stress test cycles

(c) Number of users VS Throughput in usual test cycles

(d) Number of users VS Throughput in stress test cycles

Fig. 3: Performance evaluation of Quarks (Median Response Time and Throughput)

the `Quarks` nodes to prevent unauthorized access to the users' data.

Resilient to DDoS Attacks. DDoS attackers may successfully make a single node unavailable for some time; however, the decentralized nature of the `Quarks` network will keep the service intact.

VI. Conclusion

To ensure private and secure communication over a decentralized network, we have designed and developed a messaging system leveraging distributed ledger technology. Going forward, we believe that our design will empower individuals to set-up nodes and communicate with their peer nodes securely. Our implemented PoC assured us of the feasibility of such systems. We envision that, our novel approach to solve the current security issues of the existing messaging systems will open up new school of thoughts and pioneer the next generation messaging services that we would be using for secure communication and collaboration.

References

[1] Z. Xiao, L. Guo, and J. Tracey, "Understanding instant messaging traffic characteristics," in *27th International Conference on Distributed Computing Systems (ICDCS'07)*. IEEE, 2007, pp. 51–51.

[2] M. Wijermars, "Selling internet control: the framing of the russian ban of messaging app telegram," *Information, Communication & Society*, pp. 1–17, 2021.

[3] K. E. F. Musiani *et al.*, "Concealing for freedom: The making of encryption, secure messaging and digital liberties," 2022.

[4] "Signal," https://signal.org/, accessed: 2022-02-02.

[5] "Whatsapp security," https://www.whatsapp.com/security/, accessed: 2022-02-02.

[6] "Threema security," https://threema.ch/en/security, accessed: 2022-02-02.

[7] "Open whisper systems partners with google on end-to-end encryption for allo," https://whispersystems.org/blog/allo/, accessed: 2022-02-02.

[8] "Facebook messenger deploys signal protocol for end to end encryption," https://whispersystems.org/blog/facebook-messenger/, accessed: 2022-02-02.

[9] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila, "A formal security analysis of the signal messaging protocol," *Journal of Cryptology*, vol. 33, no. 4, pp. 1914–1983, 2020.

[10] "Advanced cryptographic ratcheting," https://signal.org/blog/advanced-ratcheting/, accessed: 2022-02-02.

[11] C. Garman, M. Green, G. Kaptchuk, I. Miers, and M. Rushanan, "Dancing on the lip of the volcano: Chosen ciphertext attacks on apple {iMessage}," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 655–672.

[12] N. Kobeissi, K. Bhargavan, and B. Blanchet, "Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach," in *2017 IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 2017, pp. 435–450.

[13] K. Rathi, U. Karabiyik, T. Aderibigbe, and H. Chi, "Forensic analysis of encrypted instant messaging applications on android," in *2018 6th international symposium on digital forensic and security (ISDFS)*. IEEE, 2018, pp. 1–6.

[14] L. P. Gudipaty and K. Y. Jhala, "Whatsapp forensics: decryption of encrypted whatsapp databases on non rooted android devices," *Journal Information Technology & Software Engineering*, vol. 5, p. 2, 2015.

[15] J. Grover, "Android forensics: Automated data collection and reporting from a mobile device," *Digital Investigation*, vol. 10, pp. S12–S20, 2013.

[16] S. Wu, Y. Zhang, X. Wang, X. Xiong, and L. Du, "Forensic analysis of wechat on android smartphones," *Digital investigation*, vol. 21, pp. 3–10, 2017.

[17] R. A. Sarıtekin, E. Karabacak, Z. Durgay, and E. Karaarslan, "Blockchain based secure communication application proposal: Cryptouch," in *2018 6th International Symposium on Digital Forensic and Security (ISDFS)*. Ieee, 2018, pp. 1–4.

[18] E. Mirzaei and M. Hadian Dehkordi, "Simorgh, a fully decentralized blockchain-based secure communication system," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–19, 2022.

[19] M. Abdulaziz, D. Çulha, and A. Yazici, "A decentralized application for secure messaging in a trustless environment," in *2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT)*. IEEE, 2018, pp. 1–5.

[20] P. Menegay, J. Salyers, and G. College, "Secure communications using blockchain technology," in *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*. IEEE, 2018, pp. 599–604.

[21] R. Sharma, M. Wazid, and P. Gope, "A blockchain based secure communication framework for community interaction," *Journal of Information Security and Applications*, vol. 58, p. 102790, 2021.

[22] F. Karpisek, I. Baggili, and F. Breitinger, "Whatsapp network forensics: Decrypting and understanding the whatsapp call signaling messages," *Digital Investigation*, vol. 15, pp. 110–118, 2015.

[23] D. Wijnberg and N. Le-Khac, "Identifying interception possibilities for whatsapp communication," *Forensic Science International: Digital Investigation*, vol. 38, p. 301132, 2021.

[24] R. Cents and N. Le-Khac, "Towards a new approach to identify whatsapp messages," in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, 2020, pp. 1895–1902.

[25] P. Rösler, C. Mainka, and J. Schwenk, "More is less: On the end-to-end security of group chats in signal, whatsapp, and threema," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2018, pp. 415–429.

[26] M. J. M. Chowdhury, M. S. Ferdous, K. Biswas, N. Chowdhury, A. Kayes, M. Alazab, and P. Watters, "A comparative analysis of distributed ledger technology platforms," *IEEE Access*, vol. 7, pp. 167 930–167 943, 2019.

[27] M. S. Ferdous, F. Chowdhury, and M. Alassafi, "In search of self-sovereign identity leveraging blockchain technology," *IEEE Access*, vol. 7, pp. 103 059–103 079, 2019.

[28] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila, "A formal security analysis of the signal messaging protocol," *J. Cryptol.*, vol. 33, no. 4, pp. 1914–1983, 2020.

[29] "Locust," https://locust.io/, accessed: 2022-02-06.

[30] "Hyperledger Fabric," https://www.hyperledger.org/use/fabric.

[31] Google, "Go: Build fast, reliable, and efficient software at scale," https://go.dev/, accessed: 2022-02-06.

[32] "Channels," https://hyperledger-fabric.readthedocs.io/, accessed: 2022-02-06.

[33] O. Foundation, "Nodejs," https://nodejs.org/en/, accessed: 2022-02-06.

[34] F. O. Source, “Reactjs,” https://reactjs.org/, accessed: 2022-02-06.
[35] M. Inc., “Mongodb,” https://www.mongodb.com/, accessed: 2022-02-06.