Onion Chain Messenger - Anonymous messenger on blockchain using onion routing

Bipin Dhoddamane Ravi
Department of Computer Science
University of California, Riverside
SID: 862315841
bdhod001@ucr.edu

Kaushik Sai Kadali
Department of Computer Science
University of California, Riverside
SID: 862315842
kkada001@ucr.edu

Siddhant Purohit
Department of Computer Science
University of California, Riverside
SID: 862254690
spuro001@ucr.edu

ABSTRACT

Anonymity and privacy have always been a concern in traditional messaging applications as they can reveal sensitive information about the sender and receiver, making it vulnerable to surveillance and breaches.In this paper, we propose a secure messaging application that utilizes the strengths of both onion routing and blockchain technology. The application utilizes onion routing to provide multiple layers of encryption for the communication and protect the identity of the users. Blockchain technology is used to ensure the integrity and confidentiality of the messages by providing a tamper-proof and decentralized ledger for the messages. Our implementation uses a custom blockchain based on the Ethereum protocol, which allows for the creation of smart contracts to manage the access control of the messages. We evaluate the performance and security of the proposed system and show that it provides a high level of security and privacy for the users while also being efficient and easy to use.

KEYWORDS

Onion Routing, Blockchain, Web3, Ethereum, Smart Contracts, Cryptography

TECHNICAL STACK

Solidity, Truffle, Ganache, React.js, Python

1 INTRODUCTION

Traditional messaging applications have shown vulnerability in protecting the anonymity and privacy of their users. These applications often collect and store user data, which can be accessed by third parties through surveillance or data breaches. This data can include sensitive information such as location, message content, and contact lists. Additionally, traditional messaging applications often rely on centralized servers, which can be targeted by hackers and government agencies. Furthermore, in recent years we have seen an exponential rise in the number of these breaches. This has increased the need for alternative solutions that can provide stronger anonymity and privacy protection for users. Techniques such as end-to-end encryption and onion routing have been proposed as potential solutions to these issues. These techniques aim to protect the identity and communication of users by encrypting the data and routing it through multiple layers, making it more difficult to intercept and decode.

Although encrypted, storing messages in a centralized server is vulnerable as it can be a single point of failure, making it easy for hackers or government agencies to access and intercept the messages, which can compromise the anonymity and privacy of the users. In 2008, the bitcoin white paper [2], proposed a robust, decentralized, peer-to-peer electronic cash system using proof-of-work [7] to record a public history of transactions on a network called blockchain. Blockchain technology can be used to increase anonymity in a chat messenger by providing a decentralized and tamper-proof ledger for storing and transmitting messages. Instead of relying on centralized servers, a blockchain-based chat messenger would use a distributed network of nodes to store and transmit messages. This would make it more difficult for third parties to intercept and access the messages, as they would have to compromise multiple nodes in the network. Additionally, blockchain technology can be used to provide end-to-end encryption for the messages, ensuring that only the intended recipients can read the messages. Smart contracts [5] can also be used to manage the access control of the messages, allowing for greater control over who can view and interact with the messages. Another important aspect of blockchain technology is **pseudonymity**, which allows users to communicate without revealing their true identity, this way the anonymity of the users is protected. All these features of blockchain technology can be integrated to create a chat messenger that is more secure and private than traditional messaging applications.

Our goal with this project is to merge and integrate the prior mentioned solutions like onion routing [9] and blockchain based approaches in order to design and develop a complete and robust application which allows users to communicate and exchange messages and tokens anonymously and securely. The proposed solution can be broken down into two aspects which lead to its success:

1. Research and Literature Survey
2. Development and Implementation

In section 2 of the paper, we will go into detail about individual elements of the project like blockchain and onion routing, about their prior implementations, encryption strategies used, advantages and disadvantages, along with in-depth analysis for cost, vulnerability and failures in existing applications. Using these extensive surveys helped us build and implement a simple, easy to use, and robust messaging application on Ethereum blockchain.

Another challenging aspect of the project was design, development, and implementation of the application which demonstrated the confidentiality and anonymity of onion routing coupled with the security and integrity of using blockchain networks. From the design aspect, we have used tools like Ganache, which enables you to set a personal Ethereum blockchain on the local network for testing and development along with Truffle, a framework used for development of smart contracts [5]. Smart contracts are self-executing contracts with the terms of the agreement written directly into lines of code. Smart contracts are used to communicate with the blockchain and manage access control of messages, ensuring only authorized parties can view and interact with the message. They are used to implement various routing protocols in order to determine how the blockchain will transfer the message.

In Onion Routing, the data is encrypted multiple times, with each layer of encryption being peeled off by a "relay" or "node" on the network. These relays are selected randomly and the data is sent through them in a random order, making it difficult for third parties to trace the origin or destination of the data. Each relay only knows the relay it received the data from and the relay it is sending the data to, but not the ultimate source or destination of the data. This provides a high level of anonymity for the users as the data is encrypted and sent through multiple layers, making it difficult for third parties to intercept and access the data. Additionally, onion routing can also be used to protect against network-level adversaries and to resist traffic analysis.

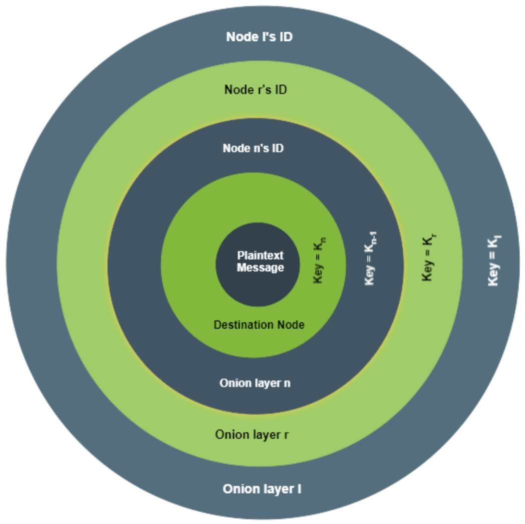


Figure 1: Onion Routing Model: Model displays the onion which is sent by the source node to node L, which removes a layer of encryption to learn only where to send it next and where it came from (though it does not know if the sender is the origin or just another node). Node L sends it to Node R, which decrypts another layer to learn its next destination. Node R sends it to Node N, which removes the final layer of encryption and transmits the original message to its destination node.

Since all the transactions that occur on the network are recorded and stored in a global ledger on blockchain, it becomes challenging to share and exchange keys required to decrypt the onion at each stage. In order to overcome this, we have implemented an asymmetric key encryption method, where in each layer of the onion we append the next node it is required to send the message before encrypting using RSA public-private key pairs of varying sizes. The advantage of using asymmetric key cryptography is that messages can be encrypted using a node's public key which is known by all other nodes, and then when the message is received by the associated node it can use its own private key to decrypt the message.

In order to help us visualize the routing and the network we have also created a front-end web application using React.js which allows us to send and receive messages and ethereum tokens on blockchain anonymously.

2 Literature Survey

Designing and implementing a robust end-to-end messaging application requires a lot of research and development. In order to achieve our goal in developing a solution which maintains privacy, confidentiality and integrity of users data we plan on implementing a custom onion routing protocol on a decentralized blockchain application. Hence, to streamline the research effort we can break down our literature survey into two parts:

1. **Survey on blockchain:** Blockchain is considered to be one of the relatively newly introduced concepts in 2008. Hence, it is necessary to study in detail not only the initial introduction but also recent developments along with its analysis, advantages and disadvantages.
    a. Understanding blockchain and initial implementation
    b. Recent developments and trends in blockchain
2. **Survey on onion routing:** Although first introduced in 1996, by U.S Naval Research Laboratory we take a deeper look into individual aspects like encryption strategies, attacks and vulnerabilities and countermeasures and aim to effectively compare the use over
    a. Onion routing over traditional networks (Tcp/Ip,Tor or traditional communication networks).
    b. Onion routing over blockchain (Existing Implementation)

To successfully cover each of the above mentioned sections, we have studied and analyzed the publications mentioned in table 1. This not only helps us in analyzing traditional systems like onion routing but gives us a better perspective to benefits and challenges faced when integrating it with a comparatively newer technology like blockchain and decentralized networks.

2.1 Survey on blockchain

A. Understanding blockchain and initial implementation

We try to understand the basic structure and the layers in the node. We then observe the initial implementations like bitcoin and ethereum, discuss the pros and cons of blockchain systems, and discuss smart contracts with its applications. As defined by F. Casino et al. [1], a blockchain is a "distributed append-only timestamp data structure", where non-trusting members form a peer-to-peer network without any trusted authority and interact with each other. Signed transactions authorize blocks of physical or digital assets as an agreement between these participants at the lowest level. This ensures the trust and avoids corruption and divergences in the network, when actions are performed sans any validating authority. Typically, an entity that connects individually to the chain is called a node. It basically consists of a block and a signature key. The nodes that satisfy all the blockchain rules are called full nodes. Most blockchain nodes today are divided into four layers, which can be defined as -

1. Transaction layer
2. Consensus layer
3. Computing interface
4. Governance layer

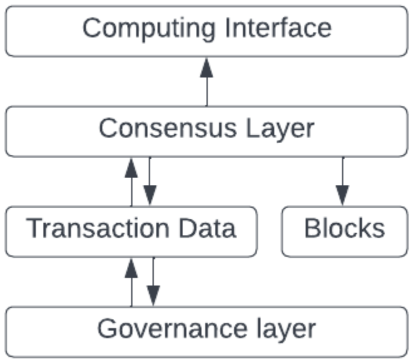


Figure 2: Layers in the Blockchain Node[1]: This diagram shows the various layers present typically in a blockchain node, namely the transaction, consensus, governance and the computing interface layers.

The transaction layer authorizes the transaction and contains the data blocks, the consensus layer maintains the agreement between different nodes, the computing interface layer acts as the interface to access additional computing capabilities, and the governance layer governs the human interactions with the network.

Bitcoin:
Blockchain as a technology started out as a peer-to-peer cash exchange system controlled by a digital signature ledger, as defined by the Bitcoin paper by Satoshi Nakamoto [2]. It was designed on the idea that an electronic coin consisting of a chain of signatures is transferred between the owners, and every new buyer has to digitally sign the hash of the previous transaction by identifying its public key. A timestamp server would hash a specific block of items in this transaction, and widely publish this hash. Proof-of-work concept was introduced, where the allotment of this timestamp on the blockchain would be determined by the work performed and the representation of the payer in the blockchain. The incentive for the seller for having this coin would be the transaction fees they would demand from the buyer, which is the expense of the mining. The transaction ledger is maintained by hashing them in a Merkle Tree data structure, which helps the blockchain to discard the previous transactions that are spent, without affecting the integrity of the blockchain. The root is only saved for the next hash, while the remaining branches are stubbed off to retain the space.

Privacy is maintained in every transaction as the public key is kept anonymous, and only shared between the buyer and the seller. A new pair key is generated, which shall delink the chain from a common owner, making it as decentralized as possible. The probability of an attacker catching up to the system's chain is calculated to be less as the number of the members of a chain increases. Chains as such, form decentralized networks, which make them harder to crack for the hackers.

The functioning of this whole system was analyzed by Juan et. al [3] in their research, "The Bitcoin Backbone Protocol" where they term the core operation in the Bitcoin network as the backbone of the Bitcoin. This backbone protocol has been used by various others who build a blockchain, with functions that essentially determine the structure, content and the context of the chain's application. They formalize a few fundamental properties it possesses, namely the common prefix, chain quality and the chain growth. They provide applications for this system, which include Byzantine Agreement protocol which can be built underneath a robust public transaction ledger, with a bounded-delay model being explored over this system. Byzantine agreement, or commonly called as the "consensus", is implemented with the proof-of-work mechanism to solve the Byzantine fault in the decentralized functioning of this system. They suggest certain directions of further development in the applications, like secure multiparty computation, while noting a newer direction of implementing this consensus in the paper by Kiayias et al. [3] called "proof-of-stake".

Ethereum:
Ethereum was conceived by Butarin et al. [4] as an upgraded version of the cryptocurrency, as it provides withdrawal limits, financial contracts and markets with a highly-generalized programming language. The usage of ethereum protocol can cover beyond currency, and include decentralized file systems, decentralized computation, decentralized prediction markets, and many more. Ethereum solves the problem of scalability, standardization, with additional attributes like feature-completeness, ease of development and interoperability, allowing anyone to write smart contracts and decentralized applications. Each user is given an account, and it contains the transaction details, or "message". Major utilities of ethereum are stated, like creation of sub currencies, hedging contracts, creating wallets, wills and employment contracts, online voting and decentralized currency systems. We wish to utilize this technology in our work, with a decentralized system to focus on end-to-end encryption of data.

Pros and Cons:
As with every technology, blockchain is bound to have its own set of pros and cons that determine its use cases and development. Niranjanamurthy et al. [6] weigh them in detail in their paper as they analyze blockchain and its usage and characteristics. We list out a few of them in the table 2 below.

Pros	Cons
Decentralized systems work independently without any central storage and administrator application.	Computation heavy, does three operations of signature verification, consensus management and repeated metadata for every transaction.
Data is consistent, complete, accurate, public and consistent	The technology is nascent, challenges, optimizations are still to be addressed
Durable, Reliable and has	Unregulated by

high longevity.	governments, so widespread adoption by public will take time
Processes are maintained by the protocol commands, ensuring integrity	Heavy on resource usage, both energy and computing power
All transactions are transparent and immutable throughout the blockchain	Security and Privacy concerns still exist with no centralized control system
Faster and economical for transactions as the overhead costs and use of intermediaries are bypassed	Integration and migration of business operations can be complex to strategize for organizations

Table 2: Blockchain utilization analysis and differences.

Smart Contracts:

A smart contract can be defined as a common agreement between two or more parties, where the contract terms are automated to control and perform transactions accordingly. It contains states and functions. States are the variables that contain the user data regarding the transaction status. Two kinds of functions exist; read-only functions return the value in the node, while write functions utilize the gas fee to initiate the transaction into a new node.

Ethereum was the first blockchain platform that developed smart contracts on the blockchain system. The language used to implement the smart contracts is Solidity, which is then compiled to the bytecode of the ethereum runtime, called Ethereum Virtual Machine (EVM).

Smart contracts would be a very important feature of the future of the internet, dubbed as Web3. Web3 technology, which involves decentralized user activity alongside monetizing their content. As blockchains are currently used to implement these functionalities, smart contracts will be critical in validating each transaction on Web3.

There are not many applications which would replicate today's web functionalities in Web3, like mail, chat messenger, etc. There has been limited research regarding a chat application on Web3, and one such paper is titled "Secure Communications Using Blockchain Technology" by Peter et al., where they discuss blockchain based applications for mail and chat. They use the Steem blockchain to implement this system, alongside the SteemJS used to handle the chat messages. The application was implemented using the React JS library for the UX, while using sockets and IO.js library to handle the events within the application. They encrypt the messages to further enhance the security. Another chat application was also developed, but this time using the BitShares blockchain along with IRC protocol, enabling any IRC client to chat.

B. Recent Developments and trends in blockchain technology

Proof-of-Work Vs Proof-of-Stake: A Comparative Analysis and an Approach to Blockchain Consensus Mechanism

A recent development in blockchain technology is the switch from Proof of Work to Proof of Stake. The paper [7] published in IJRASET in 2018 by Husneara Sheikh, Rahima Meer Azmathullah, presents a comparative study of the two most commonly implemented consensus algorithms for blockchain: the Proof of Work (PoW) algorithm and the Proof of Stake (PoS) algorithm.

The need for consensus algorithms and the need to validate the transaction is to prevent double spending. Double spending is a problem where a user can spend the coins more than once, creating false transactions. This jeopardizes the blockchains integrity.

Our implementation of Onion routing on blockchain creates a lot of intermediate transactions which might prove to be expensive if running on the Proof of Work algorithm. Hence as future work we depend on the implementation of Proof of Stake algorithms being implemented in the ethereum blockchain to reduce the transaction cost.

PoW: Proof-of-Work

In the process of mining, all miners or validators participate in the process by carefully validating and confirming the network transactions in order to get some of the cryptocurrency as a reward. The distributed ledger gathers and arranges into blocks all of the network's validated transactions. Miners validate by solving complex mathematical problems and it is called the proof of work.

Mining uses inverse hashing to discover numbers such that the block information's hash method will be less than the specified threshold. This essentially becomes a race to the first one to come up with the unique hash number. A new block is created every 10 minutes during this update, which happens about every 14 days. To increase the security of the node and blockchain in the network, miners put in all the labor and are paid.

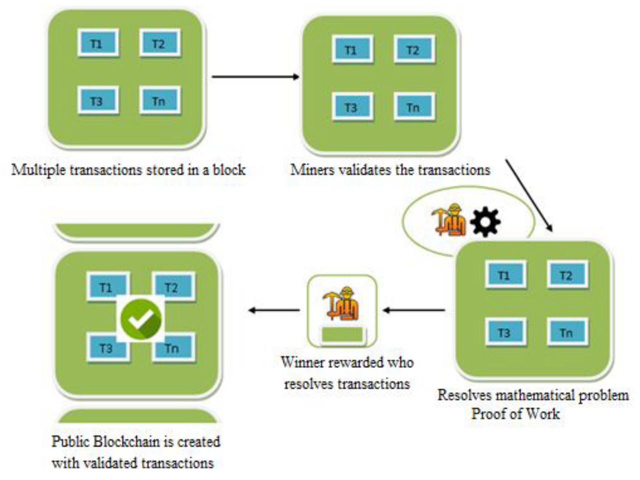

T1
T2
T3
Tn
Multiple transactions stored in a block
Miners validates the transactions
Resolves mathematical problem Proof of Work
Winner rewarded who resolves transactions
Public Blockchain is created with validated transactions

Figure 3: Proof of Work[7]: The diagram shows the working of a Proof of Work validation system. A miner who has a block that holds validated transactions tries to solve the computation to be rewarded by joining this block in the blockchain

PoS: Proof-of-stake

It is another consensus algorithm that is similar to Proof of work, all the steps except the validation of work done by miners is the same. Proof of work depends on "stake" , that is the amount of cryptocurrency kept in the chain for a fixed amount of time by the miners. Then miners are now called "forgers".

The size of the network and the total number of users using Proof-of-Stake are indirectly correlated. There will be less prizes when a lot of people stake the coin. The network should share the process in order to prevent one individual from controlling the currency. If users hold onto more cryptocurrency for a longer length of time, they will get greater transaction fees as compensation.

In order to add a new block and validate the transactions, the forger stakes their currency. If the validator determines that the transaction was fraudulent, they might potentially lose their transaction fees, their stake and authorization for future procedures.

Unlike Proof of work, there is no race to validate the transaction. The forger assigned to validate a transaction is done by a process of random selection. The forger responsible is randomly selected from the network of forgers. The random selection also takes into account the amount of coins they have staked and the age of the stake, called "Coin Age-Based Selection". Since the forgers don't have any complex mathematical problem to solve, they save on electricity, and hence are rewarded a smaller amount of cryptocurrency.

Proof of work	Proof of Stake
Validation and computational work is done by miner	Validation is done by a forger who has a stake.
First one to finish the computation is given the reward	A randomly selected user selected for the transaction is given the reward
Electricity and computation cost is high and requires expensive hardware	Low computation and less electricity is used resulting is a smaller transaction fee
The best immutable consensus approach is this one. Each block takes 10 minutes to mine.	Nothing immutable. By avoiding the need to choose a new block signer, Proof-of-Stake instantaneously establishes a block.
Cryptocurrencies: BTC, ETH, other major coins	BlackCoin, Peercoin, Nxt Coin

Table 3: Comparison of the characteristics of PoW and PoS

Disadvantages and vulnerabilities of Proof of Stake

1. **"Nothing at stake" problem**

Forking is to create a copy of the blockchain with common ancestry. Miners must choose whether to adhere to the original blockchain or make adjustments to the split blockchain during forking. Miners split their processing power between the original and branched block chains in order to sustain both the blockchains. After signing out of each split blockchain, the validator obtains a duplicate copy of his stake on the forked blockchain and is entitled to the double transaction fee as compensation. This agreement called for enforcing a deposit that was locked for a specific amount of time to fix the issue.

2. **Bribe Attack**

After the transaction, the attacker surreptitiously creates an alternate chain and gains confirmations. A fresh blockchain is taken into account once the transaction is reversed.

3. **51% Attack**

The attackers control the majority of the coins in stake, that is they have 51% of coins in circulation to control the network and forcefully accept fraudulent transactions. High cost and need to achieve 50%+ computation power in the network makes it impractical.

Combining GHOST and Casper

With the shift from PoW to PoS, different cypyto-currencies have come up with their own protocols for the implementation of PoS, trying to make it more secure and viable.

The paper[8] presents "Gasper" written by founder of Ethereum blockchain Vitalik Buterin, presents a proof of stake based consensus protocol, which is in WIP development to be implemented on the Ethereum 2.0 blockchain. The protocol combines Casper the Friendly Finality Gadget with LMD GHOST, a fork-choice rule to form their protocol "GASPER".

Casper

Casper is a gadget written in a smart contract to enforce protocols. This tool describes the terms "justification" and "finalization," which were taken from the literature on Practical Byzantine Fault Tolerance (PBFT) and is similar to "prepare" and "commit". For each transaction there are a set of chosen validators who have a stake.

In a given view *G*, there is a set of checkpoint blocks (blocks to be validated and added to the blockchain) that need to be "justified" *J(G)* and a subset of those blocks that need to be "finalized" *F(G)*. If *A* is an existing justified block then block *B* needs to reference block *A*, a majority of 2/3 of the validators need to "attest" vote that $A \rightarrow B$ for the block *B* to be marked as "justified" that is $A \in F(G)$. The 2/3 majority voting is called a supermajority link $A \rightarrow B$ with $h(B) = h(A) + 1$.

Slashing Conditions

Slashing is the protocol to penalize validators who validate fraudulent transactions. A separate validator W can slash V (destroy V's stake and perhaps receive some form of "slashing incentive") by providing evidence V breached the requirements when assumptions about honest validators are violated by validator V.

- With checkpoint edges s1 → t1 and s2 → t2, respectively, no validator produces two unique attestations, a1 and a2, with h(t1) = h(t2)
- No validator produces two separate attestations (S1) and (S2), which correspond to the checkpoint edges s1 and t1 and s2 and t2, respectively, so that h(s1) < h(s2) < h(t2) < h(t1)

The main theorems of casper are

- Two checkpoints on distinct branches cannot be finalized simultaneously unless a group of validators with stakes greater than a certain amount are shown to have broken the protocol.
- It is always feasible for new checkpoints to be finalized as long as the underlying blockchain can produce new blocks.

LMD GHOST Fork-Choice rule

Sompolinsky and Zohar developed the Greediest Heaviest Observed SubTree rule (GHOST), which governs fork selection.

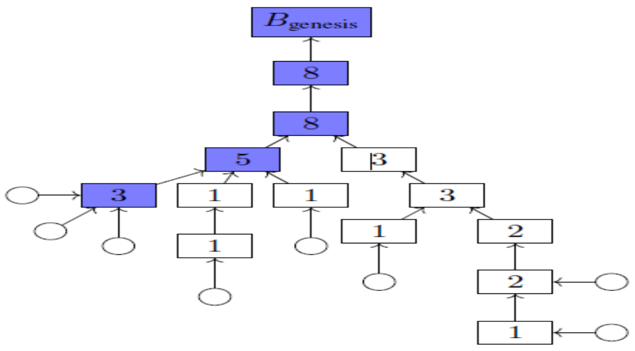


Figure 4: Presents[8] an example where the chain marked in Blue is the correct one, the circles are the validators and the aggregated weight of validators is inside the block.

The principle behind LMD GHOST is that, as implied by the algorithm's name, we utilize the weights of the forked subtrees as a heuristic and presume the subtree with the highest weight is the "correct" one.

Main Protocol of GASPER

1. Epoch boundary

In the chain supplied, some blocks are chosen to act as Casper's checkpoints, ideally one every epoch and each block is unique to an epoch. An epoch is 32 time slots of 6 minutes and 24 seconds.

2. Committees

Validators are grouped into committees. One validator is picked to propose a new block on which the other validators will vote on what they think is the right chain using the Hybrid LMD GHOST rule.

3. Justification and Finalization

Same as implemented in CASPER, blocks are justified and finalized to be added to the chain.

2.2 Survey on onion routing

A. Onion routing over traditional networks

The first onion routing research paper is "Anonymous Connection and Onion Routing" by Paul Syverson, Michael G. Reed and David Goldschlag [9], which was presented at the 1997 IEEE Symposium on Security and Privacy. This paper introduced the concept of onion routing and described the design and implementation of the first onion routing network, called The Onion Routing (TOR) project. The paper presented the fundamental design principles of onion routing, such as using multiple layers of encryption, randomly selected relays, and the use of public-key cryptography to establish secure connections between nodes. This research laid the foundation for the development of the TOR network, which is now one of the most widely used anonymity networks in the world.

The authors implemented a network of relays, also known as nodes, that are used to forward traffic in a randomized and layered fashion. When a user wants to send a message, they encrypt it multiple times, creating multiple layers of encryption, similar to the layers of an onion. The user then sends the message to an entry relay, which is the first relay in the network. The entry relay peels off the first layer of encryption and forwards the message to the next relay, which peels off the next layer of encryption and so on. The message is then sent through multiple relays until it reaches the exit relay, which sends the message to its final destination.

Encryption strategies use a combination of symmetric-key and public-key cryptography to secure the communication. The encryption of each layer is done using symmetric-key encryption, which is fast and efficient, while the establishment of secure connections between the nodes is done using public-key cryptography, which is more secure and resistant to eavesdropping. The use of multiple layers of encryption and randomized relay selection makes it difficult for third parties to trace the origin or destination of the message.

One of the main advantages of paper [9] is that it provides a high level of anonymity and privacy for the users. The use of multiple layers of encryption and randomized relay selection makes it difficult for third parties to trace the origin or destination of the message. It also protects against network-level adversaries and resists traffic analysis. Additionally, it is also resistant to certain types of censorship, as it allows users to access blocked content by routing their traffic through nodes in other countries.

However, authors also mention some disadvantages. One of the main disadvantages is that it can be relatively slow, as the message has to be sent through multiple relays and the encryption and decryption process can add latency. Additionally, the anonymity provided by onion routing is not perfect, and it can be possible for an attacker with enough resources and knowledge to de-anonymize users. Furthermore, as the network relies on volunteer-run relays, the anonymity provided is only as good as the number and diversity of the relays.

Another analysis that authors fail to do is the scalability of the network, as the number of users and relays increases, the network can become congested, and the performance may decrease. It's also important to consider that Onion routing is not a magic bullet to solve all privacy concerns, it has its own limitations, and it should be used in conjunction with other privacy-enhancing technologies.

"Probabilistic Analysis of Onion Routing in a Black-box model" [10] by Paul Syverson, Aaron Johnson, Joan Feigenbaum. This paper was published in the Proceedings of the 14th USENIX Security Symposium in 2005.This paper is considered as an important research in the field of

Onion routing and it is widely cited in the literature. It provides a theoretical foundation for the security of onion routing and has been used as a reference for many other studies in the field.

In this paper, the authors present a probabilistic analysis of onion routing in a black-box model, which allows them to evaluate the security of onion routing without making any assumptions about the behavior of the individual routers. They show that onion routing can provide strong protection against traffic analysis attacks, even in the presence of a large number of compromised relays. They also demonstrate that the security of onion routing is not significantly affected by a small number of compromised relays, and that it is possible to achieve a high degree of anonymity with a relatively small number of relays.

The authors first show that onion routing can provide strong protection against traffic analysis attacks. They do this by analyzing the probability that an attacker can link a message to a specific user, given the number of relays and the number of compromised relays. They demonstrate that the probability of a successful attack decreases exponentially with the number of relays, even in the presence of a large number of compromised relays. They also show that the security of onion routing is not significantly affected by a small number of compromised relays, and that it is possible to achieve a high degree of anonymity with a relatively small number of relays. They demonstrate that the probability of a successful traffic analysis attack decreases exponentially with the number of relays in the network. Specifically, they show that when the number of relays in the network is increased from 5 to 25, the probability of a successful attack drops from approximately 0.4 to less than 0.0001.

The authors then present a new attack, called the "end-to-end correlation attack". In this attack, the attacker correlates the timing of messages at the entry and exit relays, and then uses this information to deanonymize a user. The authors show that this attack can be used to deanonymize a significant number of users in a realistic setting. However, they also provide a number of countermeasures to mitigate this attack, such as adding random delays to the messages, and suggest that onion routing can provide a high degree of anonymity and security in practice. The authors also present statistics on the effectiveness of the "end-to-end correlation attack" that they propose. They show that in a network with 100 relays, where 10% of the relays are compromised, this attack can be used to deanonymize approximately 12% of the users in the network. However, they also show that by adding a random delay of just 10 seconds to each message, the effectiveness of this attack can be significantly reduced, to less than 1%.

Finally, the authors also provide a number of other analyses, including the analysis of the number of users that can be de-anonymized in a single attack, the trade-off between anonymity and communication overhead, and the impact of different network topologies on the security of onion routing. They show that the overhead of onion routing is relatively low, with a communication overhead of less than 1% for a network with 25 relays. And finally, the authors also provide statistics on the impact of different network topologies on the security of onion routing, and demonstrate that a random network topology is more resistant to attacks than a regular or a hierarchical topology.

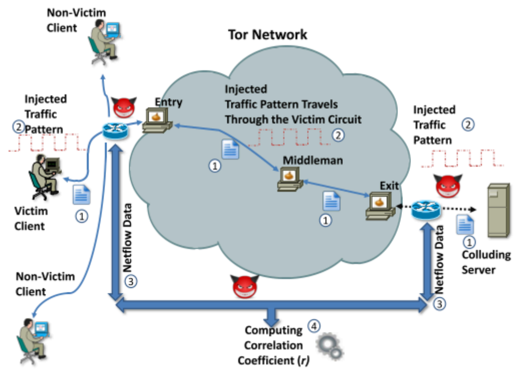

Non-Victim Client
Tor Network
Injected Traffic Pattern ②
Entry
Injected Traffic Pattern Travels Through the Victim Circuit ②
Injected Traffic Pattern ②
①
Middleman
Exit
Victim Client
①
①
Netflow Data ③
Colluding Server ①
Netflow Data ③
Non-Victim Client
Computing Correlation Coefficient (r) ④

Figure 5: Overall process for Netflow based Traffic Analysis against Tor [15]. Client downloads a file from server 1, while the server injects a traffic pattern into TCP connections it sees arising from exit node 2. When connection is terminated and the adversary obtains flow of data corresponding to server and exit node to client traffic 3, and computes correlation coefficient between server and exit traffic and entry to client.

"LiLAC: Lightweight Low-latency Anonymous Chat" [11] which is a research paper that discusses a proposed chat application that combines the features of onion routing and group messaging to provide a secure and anonymous communication system. The paper describes how the proposed system, called LiLAC, uses onion routing to conceal the identity of the users and the group messaging protocol to enable a scalable and efficient communication. The authors of the paper performed several experiments to evaluate the performance of the proposed system and they reported that LiLAC can achieve a low latency and high throughput while maintaining a high level of anonymity and security. The authors also discuss the related work in this area and the advantages and limitations of the proposed system.

The authors of the paper propose a chat application that uses onion routing and group messaging to provide secure and anonymous communication. The approach they use for encryption is based on the use of onion routing, which is a technique that uses a series of nodes (or relays) to conceal the identity of the users and the location of the communication. The users first establish a set of connections to the onion routing network and then use these connections to send and receive messages. Each message is encrypted and wrapped in a series of layers, called onion layers, that conceal the identity of the sender and the recipient. The message is then sent through the onion routing network, where it is decrypted and unwrapped at each node until it reaches its final destination.

In terms of implementation, the authors propose a group messaging protocol that is built on top of onion routing. The protocol uses a set of group keys to encrypt and decrypt the messages and a set of group identities to identify the group members. The group keys are generated by the group owner and are shared among the group members. The group identities are used to authenticate the group members and to prevent unauthorized access to the group.

The authors also mention some advantages of their proposed system, such as low-latency, high-throughput, and strong anonymity. They also mention that the LiLAC system is lightweight and efficient, which makes it suitable for mobile devices and resource-constrained environments.

In terms of vulnerabilities, the authors mention that the LiLAC system is vulnerable to traffic analysis, which is a technique that can be used to infer the identity of the users based on the patterns of their communication. They also mention that the system is vulnerable to denial of service (DoS) attacks, which can disrupt the communication and prevent the users from sending and receiving messages. The authors also mention that the system is vulnerable to group key compromise, which can occur if an attacker can obtain the group key and use it to decrypt and read the messages.

In terms of cost, the authors mention that the LiLAC system is relatively low-cost, since it does not require any special hardware or software. They also mention that the system is relatively easy to deploy and maintain, since it does not require any complex configuration or management.

B. Onion routing over blockchain

Since blockchain is a relatively new technology, it goes through continuous research and development. In recent years onion routing on blockchain networks has been attempted in a few different ways which differ in the encryption and propagation strategies, scalability, vulnerability and overall effectiveness. In this section, we will visit and understand the existing implementations of onion routing protocols on blockchain.

The paper “Onionchain: Towards Balancing Privacy and Traceability of Blockchain-Based Applications” [12] by Yue Zhang, Jian Weng, Jiasi Weng, Ming Li, and Weiqi Luo was published to IEEE in 2020. This paper mainly focuses on exploring onion routing on blockchains particular for vehicular communication as well as systematic security analysis and extensive experiments to validate security and cost-effectiveness of the system.

In this paper, the authors present their approach of implementing onion routing protocol using blockchain. The author's implementation goes in detail about each design and development attribute. They propose a end to end protocol implementation which can be broken down into:-

I. **Registration Protocol:** Each node on registering to the blockchain creates a symmetric keys which will be used to encrypt and decrypt the layers of onion message.
II. **Message Transmission:** The sender chooses intermediate nodes, negotiates keys with other nodes, encrypts and propagates the onion. Each connection is recorded as a transaction on blockchain.
III. **Validation protocol:** The authors propose an identity disclosure protocol which can be used to verify and validate each transaction and whether the message has been tampered with along the transmission.

The authors have demonstrated a case study of implementing the above protocol on vehicular communication networks and analyzed it against known threats and vulnerabilities. Authors demonstrated the strength of protocol by successfully hindering 5 main types of attacks:

- **Malicious Transmitter Attack:** is when a malicious node tries to create a false message intentionally, not to propagate but to generate evidence. This is done to evade responsibility when false message is detected but fails since each transaction is immutable and signed using private key on the blockchain

- **Malicious Messenger Attack:** is when one of the relay nodes is compromised and replaces the sender's message with a false message. Such attacks fail due to use of identity disclosure protocol
- **Replay Attack:** occurs when compromised node resend previous message, in order to use same evidence but fails due to use of timestamp with message
- **Calumniating Attack:** is when a malicious receiver pretends to receive a false message from sender, but fails since this message will not be signed by previous relay nodes.
- **Collusion Attack:** When multiple nodes are compromised and try to create false messages. Although authors mention that increasing the number of relay nodes decreases these attacks but fail to go into detail regarding the number of nodes which can be compromised over the entire network.

For the purpose of demonstration the authors use AES-128 bit symmetric key encryption and ECDSA for signature generation and validation using JAVA 1.8. The authors also exclude the propagation delay between sending messages as it is negligible compared to time/cost of encryption. The Evaluation aspects of the protocol include studying effects of the number of relay nodes. The authors use various box plots and line chart visualizations to analyze these effects. We can conclude that adding a number of relay nodes will linearly increase the time/cost it takes to propagate the message from sender to destination node. It is also worth noting that Identity disclosure requires lesser time compared to transmitting messages. As mentioned in the paper, the average time cost of message transmitting with 3 hops is around 18 ms, increasing to 28 ms with 5 hops and 57 ms with 8 hops. Identity disclosure for the same were around 11 ms for 3 hops, 18 ms for 5 hops, 34 ms for 8 hops.

The time cost also varies on the size of the message which is transmitted and stored on the blockchain since each transaction has a gas fee. The time to propagate messages ranging from sizes 1 MB to 10 MB all take approximately the same amount of time (~ 1 ms difference). A message size of 1 MB requires 18 ms time and 10 MB requires around 18.98 ms. The gas fee charged by the blockchain per transaction is dependent on various factors like size of message as well as other factors decided by the blockchain. The authors also fail to analyze the cost of transmitting messages in terms of the amount of money it will require to transfer messages in terms of tokens.

Blockchain and Onion-Routing-Based Secure Message Exchange System for Edge-Enabled IIoT

This paper is authored by Rajesh Gupta , Nilesh Kumar Jadav, Harsh Mankodiya, Mohammad Dahman Alshehri , Sudeep Tanwar and will be published in IEEE in February 2023. The paper discusses M2M(machine to machine) communication using an LSTM(long short-term memory) AI model to filter malicious messages before passing it into the OR network.

A secure message exchange between the machines *Mi* and *Mj* is made possible by conventional security algorithms like AES, RSA, etc. However, the aforementioned algorithms are extremely vulnerable to attacks from cutting-edge computer power. Because it uses many levels of encryption, the OR (O) protocol can be used in place of standard routing for security-related reasons.

Onion Routing model

In the OR model the message in encrypted as

S_Message = Enc(Enc ... Enc((Message)))

If any device If one machine *Mi* wants to send another machine *Mj* a message request, *Mi* must first create a shared secret *Sl* for the appropriate onion layer *{O1,...Ol,...OL}* ∈ *O*. Using a Diffie- Hellman method, the quantity of keys *{S1,...Sl,... SL}* ∈ *S* created must be exchanged with corresponding *Ol*. *Mi* and *Ol* safely communicate with each other their public keys using SSH tunnels to conduct the key exchange.

Attackers are unable to use the initial message request even if they discover the public keys. This is because it will take some time for the attackers to examine and uncover the private keys after they discover the public key.

Using shared secret keys *S1, Sl, and SL*, the message is encrypted and sent to the first onion node *O1*. With *S1, O1* decrypts the top encryption layer and sends information to *O2*, the following onion node. Following that, *O2* uses *S2* to decode the message, which is then sent to *O3* via *O2*. This procedure keeps on until the last *OL* decrypts the final encrypted layer and sends the necessary message to the receiving device *Mj*.

Each machine *Mi* creates *VF* tokens *(VF1, VF2…., VFm)* using a pseudorandom number generator and keeps them in a private Blockchain.

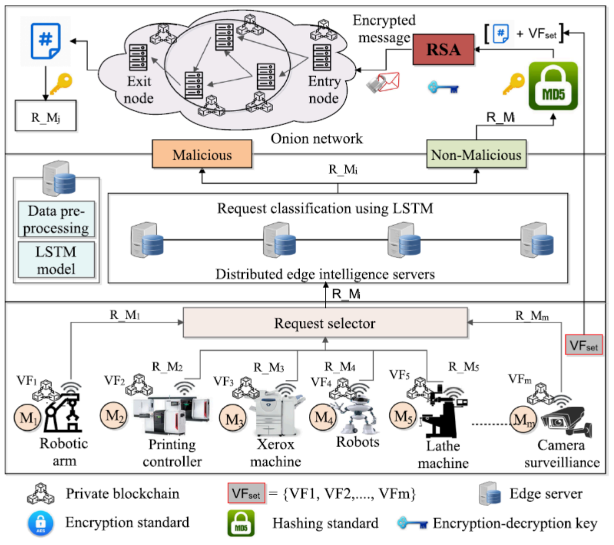


Figure 6: Shows the architecture of the proceed system [13]

After classifying the message as malicious or non malicious using the LSTM model, it is sent to the OR layer. The *RMi* is first hashed using the MD5 technique, which converts each character into binary form. The hashed message *Hi* is sent to the OR network. However, if any *Ol* from O is compromised for any Mi, an attacker can use this to follow the journey of all the *Ol* in the OR circuit. The OR network can be broken with contemporary computers, despite the fact that it may appear like a difficult undertaking. The *VF* token and TTL on the private blockchain network are therefore included as two extra fields to the hashed *Hi* in the conventional OR. TTL field in *Hi* determines whether or not the message received at *Mj* is attacked. It is dependent on how long it takes the *RMi* to travel between *Ol* and *Ol+1*. If the TTL is greater than this difference then there is a chance that the message has been tampered. The VF token is transformed into an integer number and prepended with "*Hi*" to provide a message input *M* for the OR network. Before *M* is sent to the OR network, it is triple encrypted for each layer with RSA, however how they are able to achieve multiple encryption is left out, as it isn't possible to encrypt an already encrypted message using the same value of bits in RSA.

Menc = E3(E2(E1(M)))

Menc is only able to be decrypted when it has successfully completed the two-step authentication procedure, which involves comparing the VF token attached to *Menc* with the VF token stored inside the private blockchain.

However the paper does not mention how they are able to achieve multiple encryption while creating the onion. The challenge being encrypting an already encrypted message is not possible when using a constant number of bits for RSA.

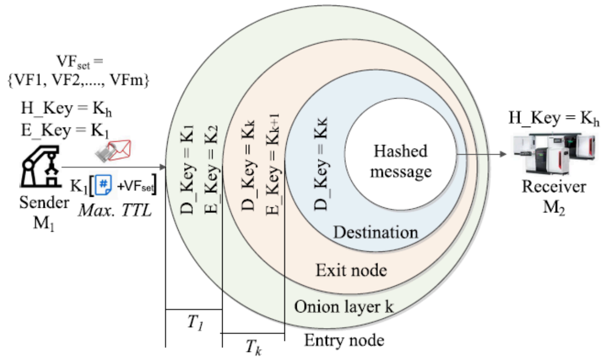


Figure 7: Represents [13] created onion cipher text for propagation.

Blockchain's role

By safely keeping the message request inside the distributed, immutable ledger, the blockchain protects against data integrity assaults. Each onion router Ol of the OR network is coupled with the blockchain node in the suggested design. However the message data is not stored on the blockchain, as is transmitted through traditional OR method.

C. Key Takeaways

In section 2, parts A and B help by providing a comprehensive overview of the current state of the field and the research that has already been conducted. This helped us to understand the existing solutions and techniques that have been proposed and the limitations and challenges they face. It also provides insights into the future direction of the field and helps us to identify potential areas for further research. Additionally, it helped us to evaluate the strengths and weaknesses of different approaches, and to determine which methods are the most effective and efficient for our specific needs. The key takeaways on benefits and drawbacks of existing solutions and implementations are

- **Benefits of existing solutions:**
  - Does not store message data on the blockchain and only uses blockchain to leverage immutable verification data
  - Is fast as it uses SSH tunnels to transfer the data using traditional OR system therefore is faster
- **Drawbacks of existing solutions:**
  - Dependent of systems being able to SSH to other systems to transfer data
  - The message itself is not written on the blockchain and is mutable
  - The last message layer is not encrypted but only hashed with MD5 and is easier to crack
  - Anonymity of the sender for the receiver is non existent
  - Encryption keys are computed on demand for each node/layer

3 Proposed Solution and Implementation

3.1 Overview

With this project , we are introducing a new and secure chat messenger application that implements the onion routing protocol on a blockchain-based platform. The application will offer its users the ability to communicate and share information privately and securely, without the worry of their data being intercepted or monitored. The onion routing protocol will add multiple layers of encryption to the user's data, making it nearly impossible for anyone to intercept and read their messages.

The application will be built on blockchain technology, which will ensure the security and privacy of the user's data. The use of blockchain technology will also allow the application to be decentralized, giving users control over their data and preventing a single entity from having access to it. Additionally, the use of smart contracts will enable users to conduct secure and transparent transactions within the chat application. The combination of onion routing and blockchain technology will provide users with the ultimate privacy and security, making it the perfect chat messenger for those who value their privacy.

In order to demonstrate the effectiveness and capabilities of the proposed system, we implement our application on a locally deployed Etherum blockchain. The primary reasons behind building our application on Ethereum network are available development and documentation resources as well as ethereum's recent switch to proof- of- stake consensus. The latter will make our proposed system get cheaper in terms of gas fee and cost. Ethereum is also one of the most popular networks and also widely used making it one of the obvious choices for a scalable messaging application.

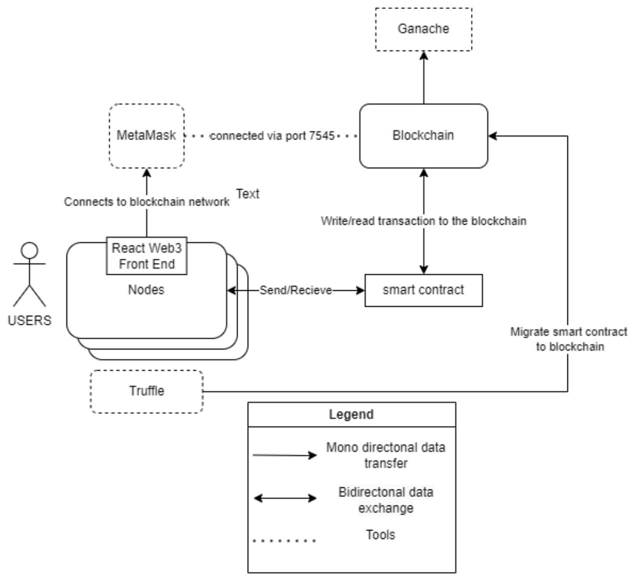

Ganache
MetaMask
connected via port 7545
Blockchain
Connects to blockchain network
Text
Write/read transaction to the blockchain
React Web3 Front End
Nodes
USERS
Send/Recieve
smart contract
Migrate smart contract to blockchain
Truffle
Legend
Mono directonal data transfer
Bidirectonal data exchange
Tools

Figure 8: shows us the architectural diagram and individual components of the proposed system.

The proposed system locally creates an ethereum blockchain of 'N' nodes using Ganache. Smart contracts are used to communicate between different nodes on the block chain. Smart contracts allow us to tell the nodes what to do when it receives a message on the blockchain. We use a tool called Truffle which allows us to migrate these smart contracts written in solidity to the blockchain. In order to visualize the creation and propagation of onion messages, we have built a React.js based front end web3 application which gives users an User Interface (UI) to send and receive the messages from other nodes. Metamask , an online software wallet, is used to connect our front-end web3 application to our local backend blockchain server via port 7545. The above enables us to send and receive text as well as ethereum tokens anonymously and securely.

Before we look at the design criteria in detail, for the purpose of demonstrating locally we initialize all the nodes with a fixed starting balance and also the required public and private RSA keys are generated for each node. Next we will take a look at important notations and symbols in Table 4 which we will be using in order to explain working of individual components and functions.

Notation	Description
$\mathbf{N_1 ..... N_n}$	Nodes in blockchain
$\mathbf{N_i \rightarrow N_j}$	Message sent from Node 'i' to Node 'j'
$\mathbf{Priv_i^b}$	Private key for node 'i' of size 'b' bits
$\mathbf{Pub_i^b}$	Pubic key for node 'i' of size 'b' bits
$ENC(M , Ni, b)$	RSA encryption function which encrypts message using $\mathbf{Pub_i^b}$
$DEC(M , Ni, b)$	RSA encryption function which encrypts message using $\mathbf{Priv_i^b}$
$VERIFY(TR , TS , \boldsymbol{\tau}))$	Verify if the message is fresh and not stale
CipherText	Encrypted text string

Table 4: Notation Table describing key notations used in implementation of Onionchain Messenger.

3.2 Design Criteria

In this section, we will focus on elaborating the design criteria for the implementation of our application. We will look in detail about individual modules used for Onion creation, Message transmission & decryption and Verification protocols established to better understand our implementation.

- **RSA & Onion Encryption**

As previously mentioned, we will be initializing an Etherum blockchain locally with a fixed amount of ethereum balance along with the required RSA public - private key pairs which will be useful for encrypting and decrypting the message. This is done only to achieve the local implementation. In a real world application use of our project, whenever a new user/ node joins the app, we will create the public private key pairs for each node, broadcast the public keys to the public ledger of the blockchain where other nodes can find it and the private key will be given to the user to be securely stored.

RSA Algorithm:

Let's take a detailed look at the RSA public key cryptography algorithm. As we know, we generate public and private keys of fixed size bits **'b'**. We will now study the meaning of this size and how these can be used to encrypt and decrypt a plaintext message **'M'**. The algorithm has 3 steps to it.

A. **Key generation:** In order to generate public and private keys of size b bits we
   a. Choose 2 large prime numbers, p and q to compute n = p*q such that 'n' is at least a 'b' bit long number.
   b. Calculate (p-1)(q-1) & choose a public key exponent **'e'** such that they are both coprime numbers.
   c. Compute private key $\mathbf{Priv_i^b}$ for node 'i' using e^{-1}(mod (p-1)(q-1)).
   d. Publish the public key $\mathbf{Pub_i^b}$ for i'th node and keep the $\mathbf{Priv_i^b}$ a secret.

B. **Encryption:** To encrypt a plaintext message **'M'** , the sender needs to
   a. Convert the plaintext message into a number **'m'** , such that 0 < m < n.
   b. Encrypted ciphertext represented by **'c'** is obtained by $\mathbf{m^e (mod\ n)}$.
   c. Send the ciphertext 'c' to the recipient.

C. **Decryption:** To decrypt a ciphertext 'c' back to plaintext message 'm' and private key exponent 'd' using $\mathbf{c^d (mod\ n)}$.

Let us further look at how the above algorithm helps us encrypt and decrypt using an example of RSA 200 for encrypting a message. RSA 200 means the size (b) of our public and private keys is 200 bits.

Key Generation:

a. Choose two large prime numbers, p and q, such that their product n = pq is at least 200 bits long. Let's say p = 257 and q = 263.

b. Calculate the totient of n, φ(n) = (p-1)(q-1). In this case, φ(n) = 65856.

c. Choose a public key exponent e, such that 1 < e < φ(n) and e is coprime to φ(n). A commonly used value for e is 65537.

d. Calculate the private key $\mathbf{Priv_i^{200}}$ exponent 'd' such that $d \equiv e^{-1} \pmod{\varphi(n)}$. In this case, d = $Priv_i^{200}$ = 41871.

e. The public key is (e, n) = (65537, 170521) and the private key is (d, n) = (41871, 170521).

Encryption:

a. The message is converted into a number, let's say m = 123.

b. The ciphertext is calculated as c ≡ m^e (mod n). In this case, c = 123^65537 (mod 170521) = 122613.

Decryption:

a. The original message is calculated as m ≡ c^d (mod n). In this case, m = 122613^41871 (mod 170521) = 123.

Note: This is a very simple example to illustrate the mathematical concepts behind the RSA algorithm. In real-world implementations, much larger prime numbers and more sophisticated methods are used to ensure the security of the encryption.

RSA keys are of fixed size and can only encrypt messages which are within its space constraints. For messages which are larger can be broken and individually encrypted and decrypted. A message encrypted by RSA 200 will be of size 200 bits after encryption no matter the size of the message. For the purpose of implementation of onion routing we will be performing multiple encryption operations on the messages. In order to avoid the hassle of breaking and re-encrypting for each layer with the same size keys, we choose to change the size of encryption at each layer. Say that we encrypt the innermost layer of onion with RSA 500 the next with RSA 1000 the next with RSA 2000 and the outermost with RSA 4000 encryption making it exponentially harder to break at each layer.

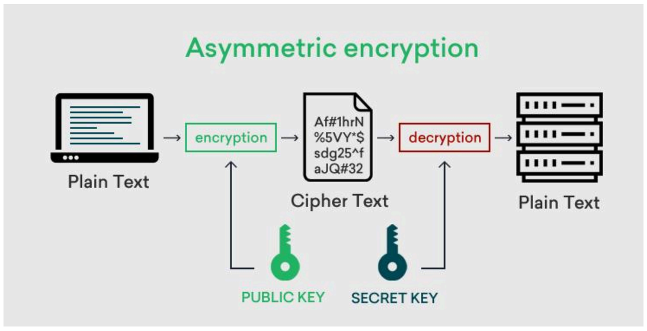


Figure 9: Explains Asymmetric encryption which uses a public key for encryption and private (secret) key for decryption. [16]

Onion Encryption

Onion routing protocol aims at enhancing the users anonymity by providing the interlinked nodes very little information about the message route. At any stage in the process, when a node receives a message it can only decrypt a part of a ciphered onion message which will give information about the next node in the chain. The intermediate node which receives the onion cipher has no idea about whether the previous node was a sender or intermediate node itself. It also cannot tell whether the next node it forwards the message to is the final destination node or not. This makes our messages highly secure since at each stage, no information is compromised. To achieve this, we need to implement the above discussed RSA Asymmetric encryption algorithm multiple times in a nested manner on top of one another.

$$\textit{Size of Message}(m) < \textit{Size of Encryption Key}(b) \qquad ....(i)$$

Equation (i), shows one key constraint of using RSA encryption is that the size of the message is always less than size of RSA key(b). Also, once encrypted using an RSA key of size **'b'** the generated ciphertext is also of size **b**. For example, a message which needs to be encrypted using RSA 200 should be less than 200 bits long. Also, when encrypted the generated ciphertext will also be 200 bits long. Therefore, it makes it impossible to re-encrypt an already encrypted ciphertext using the same key length. Hence, to overcome this problem we will incrementally use larger encryption keys at each layer in the onion. This will allow us to maintain the integrity of the message without splitting it into parts to re-encrypt. This will also increase the strength of the encryption at each layer, adding a stronger encryption on top.

For the case of demonstrating our principal we will select the number of intermediate relay nodes as 3. This is the minimum number of nodes which are required for effectively using onion routing protocol. The number of intermediate relay nodes can be increased for higher anonymity and privacy by the user. Once we get the user-determined number of relay nodes, using the following steps we implement the creation of onion cipher text:

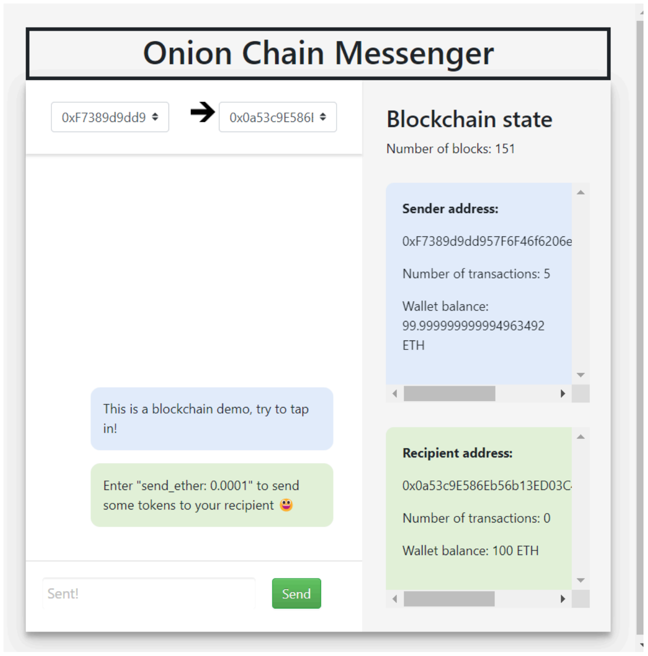
1) Initially, the sender who wants to send the message randomly chooses three relay nodes in the blockchain network except the sender and receiver nodes. The three nodes are denoted as $\mathbf{N_A}$, $\mathbf{N_B}$ and $\mathbf{N_c}$ respectively. Sender also needs the public keys associated with these nodes to encrypt the message. We denote the encryption keys as $\mathbf{K^b}_{\mathbf{A\text{-}B}}$, where the superscript represents the size of the encryption key and the first letter in the subscript is the sender, while the second letter is the receiver. $\mathbf{K^{200}}_{\mathbf{A\text{-}B}}$ represents an encryption key of size 200 bits used to encrypt the message sent from A to B. Modular encrypt function used to encrypt the message(M) with b bits for node 'i' can be expressed as

$$ENC(M\,,\; Ni,\; b) \rightarrow Pub_i^{\,b} \;\Rightarrow\; CipherText \quad \ldots \text{(ii)}$$

2) In order to achieve multiple encryption, we have implemented the *makeOnion(nodes)* function Using this the sender 'S' encrypts the message (m) with three encryption keys recursively. The function takes a list of intermediate nodes and starts encrypting in a reverse order i.e encrypting the inner layers along with adding the next-hop information at each layer. Eg $N_A \rightarrow N_B$ is appended to the message received by node A indicating it to forward the remaining message to node B. The cipherText (C) is the final encrypted onion message generated by our function can be represented as equation (iii):

$$\text{ENC}(\mathbf{K^{4000}}_{\mathbf{S\text{-}A}}, (\text{N}_\mathbf{A}\rightarrow\text{N}_\mathbf{B} \,||\, \text{ENC}(\mathbf{K^{2000}}_{\mathbf{A\text{-}B}}, (\text{N}_\mathbf{B}\rightarrow\text{N}_\mathbf{C} \,||\, \text{ENC}(\mathbf{K^{1000}}_{\mathbf{B\text{-}C}}, (\text{N}_\mathbf{C}\rightarrow\text{N}_\mathbf{R} \,||\, \text{ENC}(\mathbf{K^{500}}_{\mathbf{R}}, \text{m})))))) \Rightarrow \text{C}$$

3) The generated cipherText 'C' will be a result of final layer encryption of size 4000 bits generated due to encrypting the next-hop info for receiving node A telling it to forward the remaining onion to node B. Figure 10 shows the front-end web application console where we log the entire process of creating the ciphertext, as well as backend ganache transaction logged by the blockchain when sender initially creates the onion cipher and sends it to the first relay node.


Onion Chain Messenger

0xF7389d9dd9 → 0x0a53c9E586l

This is a blockchain demo, try to tap in!

Enter "send_ether: 0.0001" to send some tokens to your recipient 😜

Sent!

Blockchain state

Number of blocks: 151

Sender address:

0xF7389d9dd957F6F46f6206e

Number of transactions: 5

Wallet balance: 99.999999999994963492 ETH

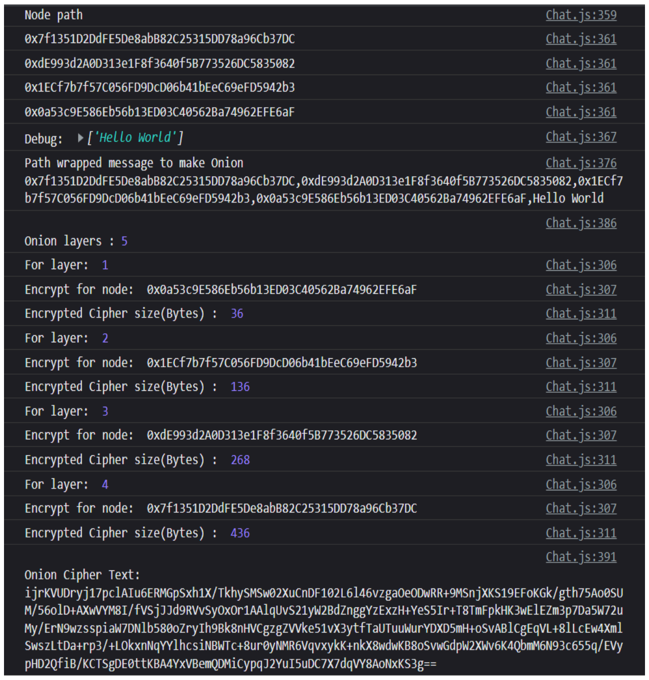
Recipient address:

0x0a53c9E586Eb56b13ED03C

Number of transactions: 0

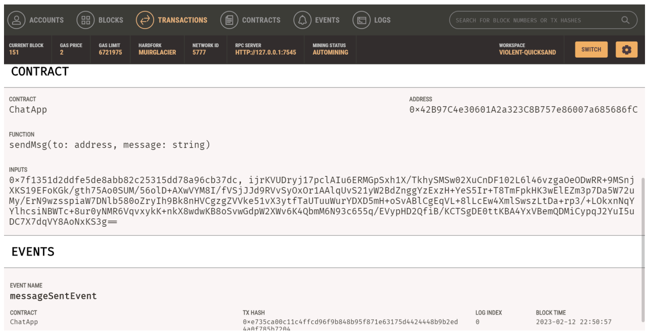
Wallet balance: 100 ETH

(10.1) Front-end Web application interface to send and receive messages between nodes



```
Node path                                                          Chat.js:359
0x7f1351D2DdFE5De8abB82C25315DD78a96Cb37DC                         Chat.js:361
0xdE993d2A0D313e1F8f3640f5B773526DC5835082                         Chat.js:361
0x1ECf7b7f57C056FD9DcD06b41bEeC69eFD5942b3                         Chat.js:361
0x0a53c9E586Eb56b13ED03C40562Ba74962EFE6aF                         Chat.js:361
Debug:  ▸['Hello World']                                           Chat.js:367
Path wrapped message to make Onion                                 Chat.js:376
0x7f1351D2DdFE5De8abB82C25315DD78a96Cb37DC,0xdE993d2A0D313e1F8f3640f5B773526DC5835082,0x1ECf7
b7f57C056FD9DcD06b41bEeC69eFD5942b3,0x0a53c9E586Eb56b13ED03C40562Ba74962EFE6aF,Hello World
                                                                   Chat.js:386
Onion layers : 5
For layer:  1                                                      Chat.js:306
Encrypt for node:  0x0a53c9E586Eb56b13ED03C40562Ba74962EFE6aF      Chat.js:307
Encrypted Cipher size(Bytes) :  36                                 Chat.js:311
For layer:  2                                                      Chat.js:306
Encrypt for node:  0x1ECf7b7f57C056FD9DcD06b41bEeC69eFD5942b3      Chat.js:307
Encrypted Cipher size(Bytes) :  136                                Chat.js:311
For layer:  3                                                      Chat.js:306
Encrypt for node:  0xdE993d2A0D313e1F8f3640f5B773526DC5835082      Chat.js:307
Encrypted Cipher size(Bytes) :  268                                Chat.js:311
For layer:  4                                                      Chat.js:306
Encrypt for node:  0x7f1351D2DdFE5De8abB82C25315DD78a96Cb37DC      Chat.js:307
Encrypted Cipher size(Bytes) :  436                                Chat.js:311
                                                                   Chat.js:391
Onion Cipher Text:
ijrKVUDryj17pclAIu6ERMGpSxh1X/TkhySMSw02XuCnDF102L6l46vzgaOeODwRR+9MSnjXKS19EFoKGk/gth75Ao0SU
M/56olD+AXwVYM8I/fVSjJJd9RVvSyOxOr1AAlqUvS21yW2BdZnggYzExzH+YeS5Ir+T8TmFpkHK3wElEZm3p7Da5W72u
My/ErN9wzsspiaW7DNlb580oZryIh9Bk8nHVCgzgZVVke51vX3ytfTaUTuuWurYDXD5mH+oSvABlCgEqVL+8lLcEw4Xml
SwszLtDa+rp3/+LOkxnNqYYlhcsiNBWTc+8ur0yNMR6VqvxykK+nkX8wdwKB8oSvwGdpW2XWv6K4QbmM6N93c655q/EVy
pHD2QfiB/KCTSgDE0ttKBA4YxVBemQDMiCypqJ2YuI5uDC7X7dqVY8AoNxKS3g==
```

(10.2) Console output for created onion cipher text



```
CURRENT BLOCK  GAS PRICE  GAS LIMIT  HARDFORK     NETWORK ID  RPC SERVER             MINING STATUS   WORKSPACE
151            2          6721975    MUIRGLACIER  5777        HTTP://127.0.0.1:7545  AUTOMINING      VIOLENT-QUICKSAND

CONTRACT

CONTRACT                                                  ADDRESS
ChatApp                                                   0×42B97C4e30601A2a323C8B757e86007a685686fC

FUNCTION
sendMsg(to: address, message: string)

INPUTS
0×7f1351d2ddfe5de8abb82c25315dd78a96cb37dc, ijrKVUDryj17pclAIu6ERMGpSxh1X/TkhySMSw02XuCnDF102L6l46vzgaOeODwRR+9MSnj
XKS19EFoKGk/gth75Ao0SUM/56olD+AXwVYM8I/fVSjJJd9RVvSyOxOr1AAlqUvS21yW2BdZnggYzExzH+YeS5Ir+T8TmFpkHK3wElEZm3p7Da5W72u
My/ErN9wzsspiaW7DNlb580oZryIh9Bk8nHVCgzgZVVke51vX3ytfTaUTuuWurYDXD5mH+oSvABlCgEqVL+8lLcEw4XmlSwszLtDa+rp3/+LOkxnNqY
YlhcsiNBWTc+8ur0yNMR6VqvxykK+nkX8wdwKB8oSvwGdpW2XWv6K4QbmM6N93c655q/EVypHD2QfiB/KCTSgDE0ttKBA4YxVBemQDMiCypqJ2YuI5u
DC7X7dqVY8AoNxKS3g==

EVENTS

EVENT NAME
messageSentEvent

CONTRACT          TX HASH                                                   LOG INDEX   BLOCK TIME
ChatApp           0×e735ca00c11c4ffcd96f9b848b95f871e63175d4424448b9b2ed    0           2023-02-12 22:50:57
                  4a0f785b7204
```

(10.3) Back-end Ganache framework recorded transaction and data published on the blockchain

Figure 10: Shows front-end, back-end and console logs for creation of onion packet.

Now let us take a look at some of the statistical analysis for the creation of our onion cipherText in terms of time taken vs number of intermediate relay nodes.

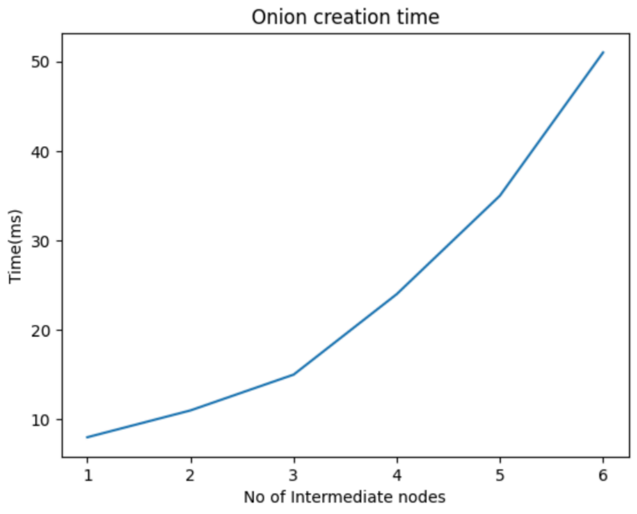


Figure 11: Shows a slight exponential growth in time to create onion vs the number of intermediate nodes chosen

- **Message Propagation**

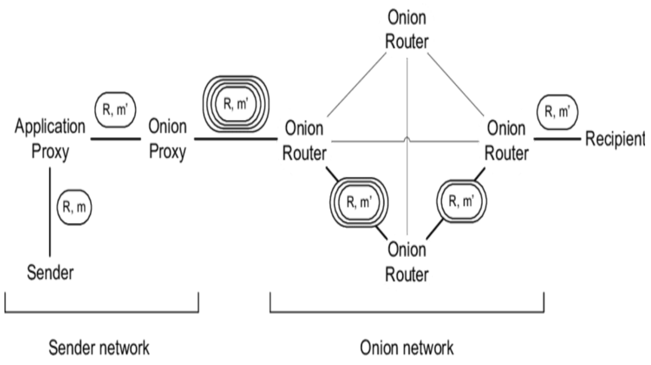


Figure 12: Represents [14] the message propagation in an onion routing protocol

We will continue our previously mentioned example with three intermediate relay nodes $\mathbf{N_A}$, $\mathbf{N_B}$ and $\mathbf{N_C}$ and look into detail how the created onion cipher 'C' is propagated after the sender 'S' sends the encrypted cipherText. The next steps of the process are:

1) As soon as the sender sends the cipherText 'C', the implemented function ***sendMessage(M, isUserMessage)*** checks and identifies whether the message sent is by the sender or an intermediate relay node. Once verified it also checks for the function ***sendEther(M, isUserMessage)*** where we check whether the requested message is an ethereum transaction request or not.

2) The cipherText ‘C’ received by the first intermediate relay node N_A is represented by equation (iii). As we can see the cipher is encrypted using $K^{4000}_{S\text{-}A} \Rightarrow Pub_A^{4000}$ or public key of N_A. Now the node N_A can use its own private key $Priv_A^{4000}$ to decrypt the received cipher message using the function ***decryptMessage(C,N$_A$,b)***. The output of the mentioned decryptMessage() function is a string which consists of next-hop information for $N_A \rightarrow N_B$ and the remaining cipherText encrypted using the public key of next-node i.e $Pub_B^{\ 2000}$. Let the decrypted message by N_A represented as IM_A consist of next-node and remaining cipherText ‘C_1’ shown in equation (iv):

$$IM_A \Rightarrow N_A \rightarrow N_B\ ,\ ENC(K^{2000}_{A\text{-}B},\ (N_B \rightarrow N_C\ \|\ ENC(K^{1000}_{B\text{-}C},\ (N_C \rightarrow N_R\ \ \|\ ENC(K^{500}_{R},\ m)))))\quad ...(iv)$$

3) The remaining ciphertext ‘C_1’ represented in equation (v) will be a result of intermediate layer encryption of size 2000 bits generated due to encrypting the next-hop info for receiving node B telling it to forward the remaining onion to node C.

$$C_1 \Rightarrow ENC(K^{2000}_{A\text{-}B},\ (N_B \rightarrow N_C\ \|\ ENC(K^{1000}_{B\text{-}C},\ (N_C \rightarrow N_R\ \|\ ENC(K^{500}_{R},\ m)))))\quad ...(v)$$

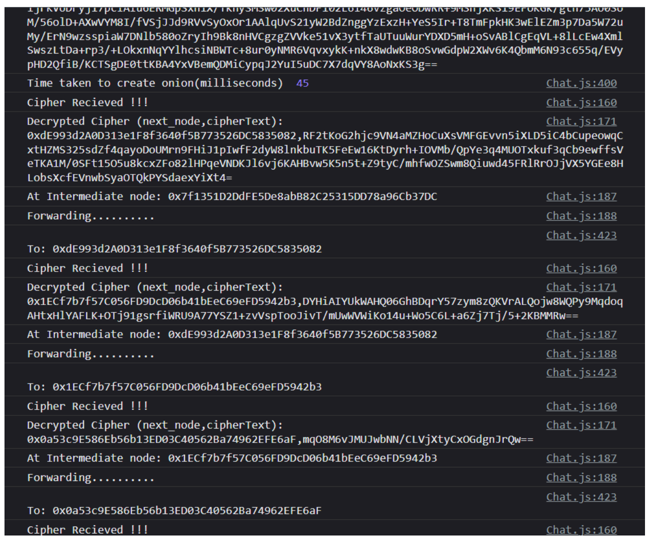
4) Similarly, at the second layer N_B decrypts the message ‘C_1’ to find out the next-node ‘N_C’ and remaining cipherText ‘C_2’ which is represented in equation (vi).

$$C_2 \Rightarrow ENC(K^{1000}_{B\text{-}C},\ (N_C \rightarrow N_R\ \|\ ENC(K^{500}_{R},\ m)))\quad …(vi)$$

5) At the penultimate layer in the onion, we decrypt the message ‘C_2’ to obtain the receiving node N_R. Although an advantage we have is that ‘N_C’ has no idea whether ‘N_R’ is just another node in the onion or the destination node. The remaining cipherText ‘C_3’ is shown in equation (vii):

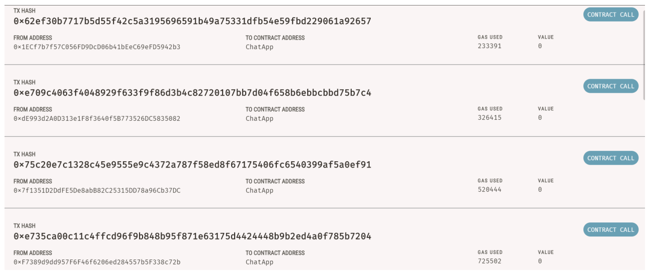
$$C_3 \Rightarrow ENC(K^{500}_{R},\ m)\quad ...(vii)$$

6) When the node ‘N_R’ receives the encrypted message C3, it is encrypted using a 500 bit public key of N_R. The receiver then uses $Priv_R^{500}$ to decrypt and finally obtain the plain text message **‘m’**



```
M/56olD+AXwVYM8I/fVSjJJd9RVvSyOxOr1AAlqUvS21yW2BdZnggYzExzH+YeS5Ir+T8TmFpkHK3wElEZm3p7Da5W72u
My/ErN9wzsspiaW7DNlb580oZryIh9Bk8nHVCgzgZVVke51vX3ytfTaUTuuWurYDXD5mH+oSvABlCgEqVL+8lLcEw4Xml
SwszLtDa+rp3/+LOkxnNqYYlhcsiNBWTc+8ur0yNMR6VqvxykK+nkX8wdwKB8oSvwGdpW2XWv6K4QbmM6N93c655q/EVy
pHD2QfiB/KCTSgDE0ttKBA4YxVBemQDMiCypqJ2YuI5uDC7X7dqVY8AoNxKS3g==
Time taken to create onion(milliseconds)  45                                     Chat.js:400
Cipher Recieved !!!                                                               Chat.js:160
Decrypted Cipher (next_node,cipherText):                                          Chat.js:171
0xdE993d2A0D313e1F8f3640f5B773526DC5835082,RF2tKoG2hjc9VN4aMZHoCuXsVMFGEvvn5iXLD5iC4bCupeowqC
xtHZMS325sdZf4qayoDoUMrn9FHiJ1pIwfF2dyW8lnkbuTK5FeEw16KtDyrh+IOVMb/QpYe3q4MUOTxkuf3qCb9ewffsV
eTKA1M/0SFt15O5u8kcxZFo82lHPqeVNDKJl6vj6KAHBvw5K5n5t+Z9tyC/mhfwOZSwm8Qiuwd45FRlRrOJjVX5YGEe8H
LobsXcfEVnwbSyaOTQkPYSdaexYiXt4=
At Intermediate node: 0x7f1351D2DdFE5De8abB82C25315DD78a96Cb37DC                  Chat.js:187
Forwarding..........                                                              Chat.js:188
                                                                                  Chat.js:423
To: 0xdE993d2A0D313e1F8f3640f5B773526DC5835082
Cipher Recieved !!!                                                               Chat.js:160
Decrypted Cipher (next_node,cipherText):                                          Chat.js:171
0x1ECf7b7f57C056FD9DcD06b41bEeC69eFD5942b3,DYHiAIYUkWAHQ06GhBDqrY57zym8zQKVrALQojw8WQPy9Mqdoq
AHtxHlYAFLK+OTj91gsrfiWRU9A77YSZ1+zvVspTooJivT/mUwWVWiKo14u+Wo5C6L+a6Zj7Tj/5+2KBMMRw==
At Intermediate node: 0xdE993d2A0D313e1F8f3640f5B773526DC5835082                  Chat.js:187
Forwarding..........                                                              Chat.js:188
                                                                                  Chat.js:423
To: 0x1ECf7b7f57C056FD9DcD06b41bEeC69eFD5942b3
Cipher Recieved !!!                                                               Chat.js:160
Decrypted Cipher (next_node,cipherText):                                          Chat.js:171
0x0a53c9E586Eb56b13ED03C40562Ba74962EFE6aF,mqO8M6vJMUJwbNN/CLVjXtyCxOGdgnJrQw==
At Intermediate node: 0x1ECf7b7f57C056FD9DcD06b41bEeC69eFD5942b3                  Chat.js:187
Forwarding..........                                                              Chat.js:188
                                                                                  Chat.js:423
To: 0x0a53c9E586Eb56b13ED03C40562Ba74962EFE6aF
Cipher Recieved !!!                                                               Chat.js:160
```

(13.1) Console output showing message propagation in onion routing protocol



```
TX HASH                                                                                        CONTRACT CALL
0×62ef30b7717b5d55f42c5a3195696591b49a75331dfb54e59fbd229061a92657
FROM ADDRESS                                 TO CONTRACT ADDRESS          GAS USED    VALUE
0×1ECf7b7f57C056FD9DcD06b41bEeC69eFD5942b3   ChatApp                      233391      0

TX HASH                                                                                        CONTRACT CALL
0×e709c4063f4048929f633f9f86d3b4c82720107bb7d04f658b6ebbcbbd75b7c4
FROM ADDRESS                                 TO CONTRACT ADDRESS          GAS USED    VALUE
0×dE993d2A0D313e1F8f3640f5B773526DC5835082   ChatApp                      326415      0

TX HASH                                                                                        CONTRACT CALL
0×75c20e7c1328c45e9555e9c4372a787f58ed8f67175406fc6540399af5a0ef91
FROM ADDRESS                                 TO CONTRACT ADDRESS          GAS USED    VALUE
0×7f1351D2DdFE5De8abB82C25315DD78a96Cb37DC   ChatApp                      520444      0

TX HASH                                                                                        CONTRACT CALL
0×e735ca00c11c4ffcd96f9b848b95f871e63175d4424448b9b2ed4a0f785b7204
FROM ADDRESS                                 TO CONTRACT ADDRESS          GAS USED    VALUE
0×F7389d9dd957F6F46f6206ed284557b5F338c72b   ChatApp                      725502      0
```

(13.2) Transactions recorded on the blockchain in backend ganache server

Figure 13: Console logs and backend transaction history by propagating onion cipher.

7) Figure 13 shows the front-end web application console where we log the entire process of propagating the ciphertext, as well as the backend ganache transaction logged by the blockchain when the intermediate nodes relay the message

Now let us take a look at some of the statistical analysis for the propagation of our onion cipherText in terms of time taken vs number of intermediate relay nodes as well as security and anonymity analysis of the implemented function.

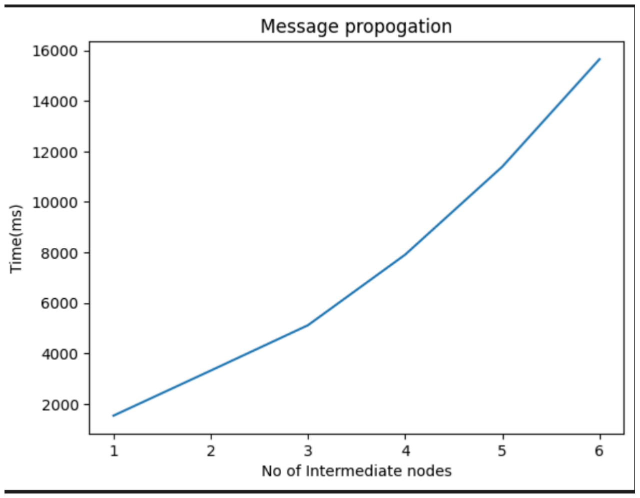


Figure 14: Shows the growth in time vs the number of intermediate nodes chosen

Considering figure 11. and figure 14. the exponential increase in time taken to create the onion and to propagate the message, we consider using ‘3’ intermediate nodes as the sweet spot, to achieve anonymity while still keeping the time for onion creation and propagation low.

- **Decryption Layer**

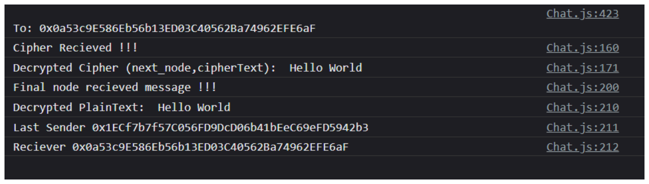
Now let us take a look at what happens to the message at the last layer and how it is decrypted and converted back to plainText which can only be seen by the intended receiver. It is important to securely and safely decrypt the message for the receiver without exposing the original sender. Let us take a look at the steps we take in order to properly implement this part of the protocol:

1) After the last relay node ‘N_C’ has forwarded the remaining onion message ‘C_3’ to receiver node ‘N_R’. If we remember, we have created the onion in a reverse order and have encrypted the plainText message with the receiver node’s public key of size 500 bits i.e Pub_R^{500}.

$$C_3 \Rightarrow ENC(K^{500}_R, m) \qquad \text{...(viii)}$$

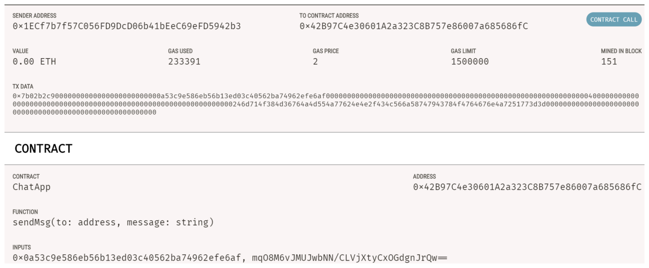
2) Once N_R receives the message C_3 it calls the **DEC(C_3 , N_R, 500)** function where it uses $Priv_R^{500}$ to decrypt C_3 to readable plainText message ‘m’ sent by the original sender.

3) Since the message ‘m’ can only be decrypted by N_R, no intermediate relays can decrypt this without using private key $Priv_R^{500}$. We also guarantee freshness of the message and verify and drop the stale messages based on certain threshold time delay for each hop. If tampered the entire message is dropped and the entire process is repeated with a new set of nodes.
4) The receiver N_R also has no idea about any previous node except for the last relay node N_C which appears as the sender of the message to N_R thereby protecting the identity of the original sender N_S successfully.
5) Identically, using the above steps we can securely and anonymously send not only text messages but also transact ethereum tokens.



```
                                                        Chat.js:423
To: 0x0a53c9E586Eb56b13ED03C40562Ba74962EFE6aF
Cipher Recieved !!!                                     Chat.js:160
Decrypted Cipher (next_node,cipherText):  Hello World   Chat.js:171
Final node recieved message !!!                         Chat.js:200
Decrypted PlainText:  Hello World                       Chat.js:210
Last Sender 0x1ECf7b7f57C056FD9DcD06b41bEeC69eFD5942b3  Chat.js:211
Reciever 0x0a53c9E586Eb56b13ED03C40562Ba74962EFE6aF     Chat.js:212
```

(15.1) Console log for message decryption and last node propagation



```
SENDER ADDRESS                               TO CONTRACT ADDRESS                           CONTRACT CALL
0×1ECf7b7f57C056FD9DcD06b41bEeC69eFD5942b3   0×42B97C4e30601A2a323C8B757e86007a685686fC

VALUE          GAS USED          GAS PRICE          GAS LIMIT          MINED IN BLOCK
0.00 ETH       233391            2                  1500000            151

TX DATA
0×7b02b2c90000000000000000000000000a53c9e586eb56b13ed03c40562ba74962efe6af00000000000000000000000000000000000000000000000000000000000000400000000000
0000000000000000000000000000000000000000000000000000246d714f384d36764a4d554a77624e4e2f434c566a58747943784f4764676e4a7251773d3d000000000000000000000
00000000000000000000000000000000000

CONTRACT

CONTRACT                                     ADDRESS
ChatApp                                      0×42B97C4e30601A2a323C8B757e86007a685686fC

FUNCTION
sendMsg(to: address, message: string)

INPUTS
0×0a53c9e586eb56b13ed03c40562ba74962efe6af, mqO8M6vJMUJwbNN/CLVjXtyCxOGdgnJrQw==
```

(15.2) Receiver node transaction showing it receives encrypted message

Figure 15: Console information and blockchain transaction record for decryption of cipherText.

Figure 15.1 shows us the final stage decryption as seen on the front end react web3 application. It is important to pay attention that the receiver N_R can see the message coming from N_C as the sender instead of N_R which shows us that we have successfully hidden the original sender identity. This can also be verified from the logs of transactions on our backend ganache server as well as seen in figure 15.2.

We can now move on to analyzing the security of our system in terms of decryption time and message size as well as the effectiveness and correctness of our system in detail. The time to decrypt any given message is dependent on the size of the message and the size of the encryption we choose.

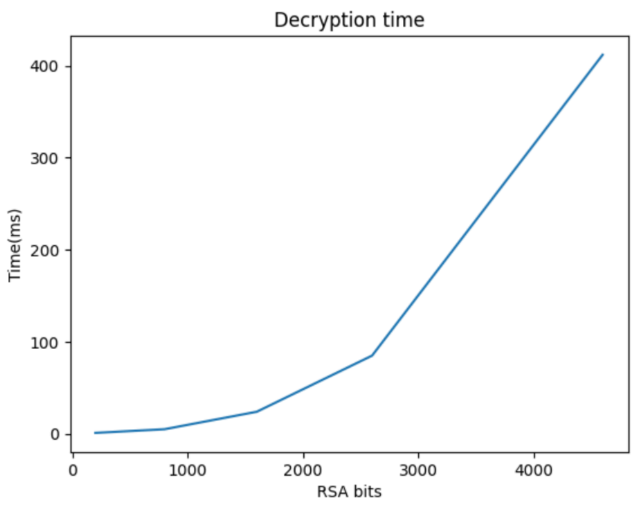


Figure 16: Shows the exponential growth in time taken to decrypt the message vs the RSA bits length of 200, 800, 1600, 2600, 4600

- Verification Protocol

Unlike traditional messaging systems, we do not plan on implementing a signature and validation scheme where at each stage the sender receives acknowledgement of the message. This is mainly done to protect the path the message is transmitted through from getting exposed. Also the selection of relay nodes is randomized to improve privacy of nodes on the blockchain. The transactions logged on the public ledger are immutable i.e cannot be changed after it is written on the ledger making it impossible for a transaction to be compromised and tampered with.

On top of that, similar to traditional validation schemes, we propose to use a time based approach to verify and detect stale transactions and drop them before the receiver receives them. In order to achieve this, we implement a **VERIFY(T_R , T_S , τ)** function where T_R is the time at which a node receives a message, T_S is the time at which the node send the message '**m**' to the next node and 'τ' is a predetermined threshold time . At each step in the message transmission process will check and verify using steps mentioned below in Algorithm 1. The verification protocol uses the above input parameters to compare and verify the time taken by the message was not exceeded at any stage in the process.

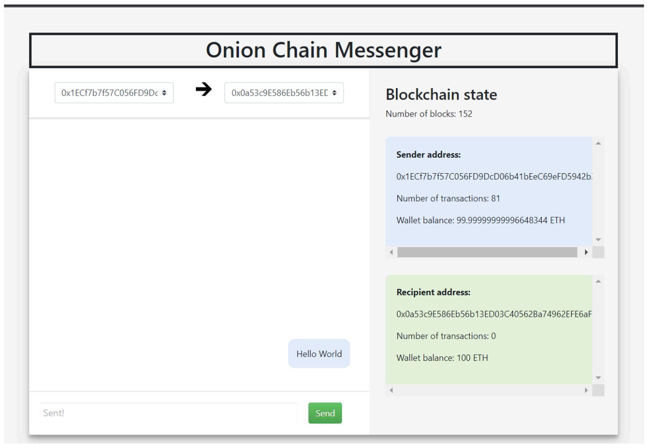
Algorithm 1: Verification Protocol

```
Input Data:
$T_R$ - Time at which node $N_i$ receives the onion cipher '$C_i$'
$T_S$ - Time at which node $N_i$, send the onion cipherText '$C_{i+1}$'
$\tau$ - Threshold timer

Output: isStale → Boolean

Initialization: isStale = False;

$T_R$ = time();
$C_{i+1}$ = DEC($C_i$ , $N_i$ , b)
$T_S$ = time();
if $T_S - T_R > \tau$ then
    isStale = True;
    RETURN isStale ;
end
else
    RETURN isStale ;
end
if isStale == True then
    dropMessage();
    sendMessage( $C_{i+1}$ , True);
end
else
    sendMessage( $C_{i+1}$, False );
end
```

1) Whenever a node 'N_i' receives an intermediate cipherText 'C_i', the time at which the message is

received by N_i is denoted by T_R. We just store the time using an inbuilt time() function.

2) Next the received cipher C_i is decrypted using the ***DEC(C$_i$, N$_i$, b)*** function to obtain the next-node information and encrypted cipherText C_{i+1} is obtained to propagate to the next-node.
3) Just before C_{i+1} is sent using the ***sendMessage(C$_{i+1}$, isUserMessage = False)*** function, we log the time at which N_i tries to send the message.
4) Next step is to verify whether the difference between these time intervals is more than our expected threshold value.
5) By calculating the difference between the time the message arrived to the N_i and the time message was requested to be sent by N_i helps us analyze whether the received message has been tampered with or modified during the propagation. Crossing the threshold would not guarantee the freshness of the message sent and hence we set the isStale flag to be True.
6) If a stale message is detected, it is not propagated forward and the receiver will never receive the message. Instead, the stale message will be dropped and we will re attempt to start the entire onion routing process again.
7) Once verified that the message is still fresh/ untampered by the intermediate node, only then will the message be allowed to propagate to the next node.


Onion Chain Messenger

0x1ECf7b7f57C056FD9Dc → 0x0a53c9E586Eb56b13ED

Blockchain state

Number of blocks: 152

Sender address:

0x1ECf7b7f57C056FD9DcD06b41bEeC69eFD5942b

Number of transactions: 81

Wallet balance: 99.99999999996648344 ETH

Recipient address:

0x0a53c9E586Eb56b13ED03C40562Ba74962EFE6aF

Number of transactions: 0

Wallet balance: 100 ETH

Hello World

Sent!

Figure 17: Front-end message displays to receiver after verification by the last relay node N_3 .

The verification protocol will help us propagate messages without being vulnerable to common attacks like man in the middle and stop propagation of malicious messages. Hence with this, we do not require a traditional signature and validation scheme since we intend to keep the identity of the nodes private. Some major advantages of a time-based stale message verification are:

A. Reliability: Timestamps are reliable sources of information as they are generated by the system itself and can't be easily altered. This provides a secure way of verifying if a message has taken too long to be forwarded.

B. Easy Implementation: Implementing a time-based verification method is relatively simple, as it just involves checking the difference between the current time and the time the message was sent.

C. Real-time Monitoring: A time-based verification method allows real-time monitoring of messages, as it can be constantly checking if a message has taken too long to be forwarded. This is important in cases where timeliness is critical, such as in emergency situations.

D. Transparency: Time-based verification provides transparency in the message forwarding process, as it allows the sender and receiver to easily see how long the message took to be forwarded.

E. Cost Effective: Unlike other methods, a time-based verification method does not require any additional hardware or software to be implemented, making it a cost-effective solution for identifying stale messages.

In conclusion, time-based stale message verification is a simple, reliable, and cost-effective way of ensuring that messages are being forwarded in a timely manner, and can be especially useful in real-time critical situations.

4 Security Analysis

There are several known security flaws in existing systems using blockchain, smart contracts and onion routing. We go over each of them and address how our proposed system handles each of them.

- Vulnerability of Blockchain technology: Despite being touted as a secure and tamper-proof technology, blockchain technology is not immune to security threats such as 51% attacks, smart contract vulnerabilities, and other types of hacking incidents. However reaching 51% majority ownership on the block chain is a challenging task, so the probability of such attacks is low.

- Complexity: The implementation of onion routing and blockchain technology could make the system complex, which could lead to bugs and vulnerabilities that could be exploited by attackers.

- Smart Contract Security: Smart contracts are self-executing code that run on the blockchain. If the smart contracts used to manage access control of the messages contain vulnerabilities or are not properly secured, the system could be compromised.

- Man in middle attack: The attacker could be an intermediate node on the chain that tries to decrypt or alter the message. We introduce a VERIFY function that drops the message if the message is found to be stale.

- Cracking: Although RSA encryption is safe, there are several ways used to crack and brute force the original message from the cipher text, our proposed method uses multiple layers of encryption on top of the original message. Each layer has uses different bits of RSA encryption, which takes a very long time to crack and get through all of them.

- Traffic Analysis: Protecting against traffic analysis is challenging, in general, the more nodes an adversary can compromise, the greater their ability to perform traffic analysis and potentially identify the receiver of a message. However, our proposed onion routing system is designed with multiple layers of encryption and a large number of randomly selected nodes, it can become very difficult for an adversary to identify the receiver even if they compromise a significant number of nodes.

- User Adoption: The security of the system also depends on the users adopting secure practices such as properly protecting their encryption keys and being cautious of phishing attacks.

5 Evaluation of proposed system against existing systems

Comparing the proposed system with existing systems, we have identified a few major points that are advantageous to have in an anonymous immutable messaging system.

SSH tunnel: Existing systems form an SSH tunnel to propagate messages, our proposed system uses smart contracts to write the message and propagation data, encrypted and written to the blockchain, leaving no need for SSH and to protect against SSH based vulnerabilities

Anonymity between sender and receiver: Existing systems do not provide anonymity between sender and receiver, both of them are aware of each other. In our proposed system, the receiver does not know who the sender is. The intermediate nodes do not know either the sender or receiver.

Immutability: Existing systems use blockchain to only store the message request for immutability and verification. Our proposed system stores the message itself on the blockchain and is permanent.

Cracking the encryption: Existing systems only encrypt the message once, which makes it faster and easier to crack by the attacker. Our proposed system encrypts the message in multiple layers of RSA, significantly increasing the time it would take to crack the encryption.

Features	**Proposed system**	**Existing systems**
Requires SSH tunneling	╳	✔
True Anonymity	✔	╳
Message immutability	✔	╳
Eth transaction support	✔	╳
High time to crack the encryption	✔	╳

Table 5: Shows the major differences in proposed system vs existing systems

6 Discussion

The significance of the proposed system can be viewed individually for both its use cases i.e Anonymous Text & Anonymous Transactions. Let's discuss the significance of the Anonymous Text feature:

Privacy: Onion routing combined with blockchain technology provides a high level of privacy for users, as it makes it very difficult for anyone to trace the source of a message. The combination of these two technologies provides a secure and anonymous platform for communication.

Decentralization: Blockchain technology is decentralized, meaning there is no central authority controlling it. This makes it an ideal platform for anonymous communication, as it eliminates the need for a central entity that could potentially access user information.

Immutability: The immutable nature of blockchain means that once a message is recorded on the blockchain, it cannot be altered or deleted. This provides an additional layer of security, as it makes it nearly impossible for messages to be tampered with.

Resistance to Censorship: Anonymous chat messengers using onion routing on blockchain are resistant to censorship, as there is no central point of control that can be used to censor the communication. This makes it a useful tool for individuals in countries with restrictive governments, as it provides a secure and anonymous way to communicate.

Transparency: Despite the anonymity of the users, the transactions on the blockchain are public and can be viewed by anyone. This provides a level of transparency to the system, as it allows for the detection of any malicious activity.

Encrypted Communication: The use of onion routing provides encrypted communication, as each layer of the routing process adds an additional layer of encryption to the message. This provides a high level of security for sensitive information that is being transmitted.

Anonymity for Political Activism: In countries with restrictive governments, the ability to communicate anonymously can be crucial for political activists who want to express their views without fear of retaliation. An anonymous chat messenger using onion routing on blockchain provides a secure platform for political discourse.

Cross-border Communication: Anonymous chat messengers using onion routing on blockchain can be used for cross-border communication, as there are no geographical restrictions to their use. This makes it possible for individuals in different countries to communicate securely and anonymously.

All in all, anonymous chat messengers using onion routing on blockchain provide a secure and private platform for communication, while also being resistant to censorship and providing a level of transparency. This makes it a valuable tool for individuals seeking privacy and security in their online communications. Now let us look at the significance of Anonymous Transactions:

Privacy: An anonymous financial transaction application using onion routing on blockchain can provide a high level of privacy for users, as it makes it very difficult for anyone to trace the source of the transaction. This can be particularly important for those who want to keep their financial information private.

Security: The use of onion routing provides encrypted communication, which can help to prevent malicious actors from accessing sensitive financial information. In addition, the decentralized and immutable nature of blockchain provides a secure platform for financial transactions.

Decentralized and Borderless: Blockchain is decentralized and operates without geographical restrictions, which makes it possible for individuals in different countries to conduct secure financial transactions without the need for a central authority or intermediary.

Low Transaction Fees: As there is no central authority controlling the network, transaction fees are typically lower than those associated with traditional financial transactions. This can make it a more cost-effective solution for those who need to make financial transactions regularly.

Improved Financial Inclusion: An anonymous financial transaction application has the potential to improve financial inclusion for individuals who are currently unbanked or underbanked. By providing a secure and accessible platform for financial transactions, it can help to bring financial services to those who would not otherwise have access to them.

Resistance to Fraud: As all transactions are recorded on the blockchain, it makes it difficult for anyone to commit fraud. The decentralized nature of the platform makes it more difficult for malicious actors to compromise the system, providing an additional layer of security.

Smart Contract Functionality: The use of smart contracts on blockchain technology provides the ability to automate financial transactions and enforce the terms of a contract. This can provide a more efficient and secure platform for financial transactions, reducing the need for intermediaries and improving the overall efficiency of the financial system.

We have discussed the significance of the proposed system but now let's take a look at some of the practical applications and issues where the proposed system can be used. The following are some of the example scenarios where a Anonymous text and transaction application can be useful:

Secure Communication: One of the main practical applications of an anonymous messenger using onion routing on blockchain is to provide secure communication. It can be used to transmit sensitive information or to communicate with individuals who are located in countries with restrictive governments.

Privacy-Preserving Transactions: The use of onion routing on blockchain can provide a high level of privacy for users who want to conduct secure financial transactions. This can be useful for individuals who want to keep their financial information private or for businesses who need to conduct secure transactions with customers.

Whistleblowing Platforms: Anonymous chat messengers using onion routing on blockchain can be used as a secure platform for whistleblowers to share sensitive information without fear of retaliation. The encrypted and decentralized nature of the platform makes it difficult for anyone to trace the source of the information.

Decentralized Marketplaces: An anonymous transactions application using onion routing on blockchain can be used to facilitate decentralized marketplaces, allowing users to buy and sell goods and services securely and privately.

Secure Payment Processing: An anonymous transactions application using onion routing on blockchain can be used for secure payment processing, reducing the risk of fraud and improving the overall security of online transactions.

Remittances: An anonymous transactions application using onion routing on blockchain can be used for secure and private cross-border remittances, reducing the need for intermediaries and lowering the cost of these transactions.

Supply Chain Management: An anonymous transactions application using onion routing on blockchain can be used for secure and transparent supply chain management, allowing businesses to track the movement of goods and ensure that they are being sourced ethically and sustainably.

Charitable Donations: An anonymous transactions application using onion routing on blockchain can be used for secure and transparent charitable donations, allowing individuals to donate money to their favorite causes without revealing their identity.

Political Campaigns: An anonymous chat messenger using onion routing on blockchain can be used by political campaigns to communicate securely with supporters and volunteers without fear of surveillance or censorship.

Secure Data Sharing: An anonymous chat messenger using onion routing on blockchain can be used to securely share sensitive information, such as medical or legal records, between individuals or organizations.

Crowdfunding: An anonymous transactions application using onion routing on blockchain can be used for secure and transparent crowdfunding, allowing individuals to raise funds for their projects without revealing their identity.

7 Drawbacks

Like any technology, an anonymous messenger and transactions application using onion routing on blockchain also has some drawbacks, including:

Technical Complexity: The technical complexity of onion routing and blockchain technology can make it difficult for some users to use the system, especially those who are not familiar with these technologies. This can limit the adoption of the system.

Slow Transactions: The encryption and routing processes involved in onion routing can slow down transactions, making it less suitable for applications that require fast processing times.

Potential for Misuse: The anonymity provided by onion routing can also make it easier for individuals to engage in illegal activities, such as money laundering, fraud, or cybercrime. This raises concerns about the potential misuse of the system.

Limited Scalability: The current scalability limitations of blockchain technology can also impact the performance of an anonymous transactions application using onion routing on blockchain, especially if the number of users grows rapidly.

Limited Interoperability: The lack of standardization between different implementations of onion routing and blockchain technologies can limit interoperability between different systems, making it difficult for users to move their assets between different platforms.

Dependence on the Network: The security and privacy of an anonymous transactions application using onion routing on blockchain is dependent on the size and robustness of the network. If the network is not large enough, it can be vulnerable to attacks, and if the network is not secure, user data can be exposed.

In conclusion, while anonymous messengers and transactions applications using onion routing on blockchain offer many benefits, they also have some drawbacks that need to be considered, including technical complexity, slow transactions, potential for misuse, limited scalability, limited interoperability, and dependence on the network.

8 Future Work

There are several areas of future work for an anonymous messenger and transactions application using onion routing on blockchain, including:

Improving Scalability: One of the key challenges in the current implementations of blockchain technology is scalability, and this is also a challenge for anonymous messengers and transactions applications using onion routing on blockchain. Future work in this area should focus on improving the scalability of the system to handle larger volumes of transactions and users.

Enhancing Security: Improving the security of the system is also an important area of future work. This can be achieved by implementing stronger encryption algorithms, improving the robustness of the network, and developing methods to detect and prevent cyberattacks.

Interoperability: Developing methods to improve interoperability between different implementations of onion routing and blockchain technologies will be important for the future of anonymous messengers and transactions applications using onion routing on blockchain. This will make it easier for users to move their assets between different platforms and will help to increase the adoption of the technology.

User Experience: Improving the user experience is also an important area of future work. This can be achieved by developing user-friendly interfaces and tools, making the system more accessible to a wider range of users, and improving the overall performance of the system.

Regulation: As the use of anonymous messengers and transactions applications using onion routing on blockchain continues to grow, it will be important to develop regulatory frameworks that can help to ensure the safety and security of the system, while also preserving its privacy and anonymity features.

9 Conclusion

In conclusion, anonymous messengers and transactions applications using onion routing on blockchain offer several benefits, including privacy, security, and decentralization. The use of onion routing provides a high level of anonymity by routing messages through multiple layers of encryption, making it difficult for third parties to track the origin and destination of the message. Meanwhile, the use of blockchain technology provides a secure and decentralized platform for transactions, which makes it resistant to censorship and tampering.

Additionally, the use of anonymous messengers and transactions applications using onion routing on blockchain can have significant implications for the freedom of speech and personal privacy. In countries where government surveillance is a concern, or where individuals may be at risk for expressing their opinions, an anonymous messenger and transactions application can provide a safe and secure platform for communication and transactions.

However, it's important to acknowledge the limitations and potential risks associated with this technology. The technical complexity of onion routing and blockchain can make it difficult for some users to use the system, and the anonymity provided by onion routing can also make it easier for individuals to engage in illegal activities. Despite these limitations, the use of anonymous messengers and transactions applications using onion routing on blockchain is growing and has the potential to revolutionize the way we communicate and transact.

Nevertheless, the potential benefits of anonymous messengers and transactions applications using onion routing on blockchain cannot be ignored and continued efforts to improve the technology will be important for the continued growth and success of the system. In the future, continued efforts to improve the scalability, security, interoperability, and user experience of the system will be important for the continued growth and success of the technology and its applications.

10 Acknowledgement

We would like to express our sincere gratitude to you, Professor Michail Faloutsos, for your invaluable guidance and support throughout the execution of this project. Your extensive knowledge and expertise in the field provided us with invaluable insights that helped shape the project and refine our research methodology. Your willingness to engage in discussions and offer constructive feedback was instrumental in overcoming the challenges we encountered during the course of our research. We would like to thank you for your unwavering commitment to our project and for helping us produce work that we are proud of.

11 References

[1] Fran Casino, Thomas K. Dasaklis, Constantinos Patsakis, A systematic literature review of blockchain-based applications: Current status, classification and open issues, Telematics and Informatics, Volume 36, 2019, Pages 55-81, ISSN 0736-5853, https://doi.org/10.1016/j.tele.2018.11.006

[2] Nakamoto, S. (2008) Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf

[3] Garay, Juan & Kiayias, Aggelos & Leonardos, Nikos. (2015). The Bitcoin Backbone Protocol: Analysis and Applications. 281-310. 10.1007/978-3-662-46803-6_10.

[4] V Buterin, (2014) A next-generation smart contract and decentralized application platform. https://ethereum.org/en/whitepaper

[5] Khan, S.N., Loukil, F., Ghedira-Guegan, C. *et al.* Blockchain smart contracts: Applications, challenges, and future trends. *Peer-to-Peer Netw. Appl.* 14, 2901–2925 (2021). https://doi.org/10.1007/s12083-021-01127-0

[6] M., Niranjanamurthy & Nithya, B. & Sree, Jagannath. (2019). Analysis of Blockchain technology: pros, cons and SWOT. 22. 10.1007/s10586-018-2387-5.

[7] Sheikh, H.H., Azmathullah, R.M., & Haque, F.R. (2018). Proof-of-Work Vs Proof-of-Stake: A Comparative Analysis and an Approach to Blockchain Consensus Mechanism.

[8] Buterin, V., Hernandez, D., Kamphefner, T., Pham, K., Qiao, Z., Ryan, D., Sin, J., Wang, Y., & Zhang, Y.X. (2020). Combining GHOST and Casper. *ArXiv, abs/2003.03052*.

[9] M. G. Reed, P. F. Syverson and D. M. Goldschlag, "Anonymous connections and onion routing," in IEEE

Journal on Selected Areas in Communications, vol. 16, no. 4, pp. 482-494, May 1998, doi: 10.1109/49.668972.

[10] P.Syverson, A. Johnson, J.Feigenbaum, “Probabilistic Analysis on Onion Routing in a black-box model” , 2005.

[11] M.Wright, G.Danezis, J.Podalanko, “Lilac:Light weight low-latency anonymous chat”, 2017

[12] Y.Zhang, J. Weng, M.Li, W.Luo, “Onionchain: Towards balancing privacy and traceability of blockchain applications”, 2021

[13] Rajesh Gupta, Nilesh Kumar Jadav, Harsh Mankodiya, Sudeep Tanwar, “Blockchain and Onion-Routing-Based Secure Message Exchange System for Edge-Enabled IIoT”, 2023

[14] Ruben Rios, Javier Lopez , “ (Un)Suitability of Anonymous communication systems”, 2013.

[15] Infosec Article by Pierluigi Pagani, “Hacking the Tor Network: Follow up”, 2020

[16] “Symmetric Vs Asymmetric Encryption” blog by ClickSSL in 2022.