

Authentication Using Blockchain

by

Anindita Bose

22341067

Raima Islam

22341060

Sikander Sofia Safrina

22341074

Dishari Bin Khurshid

21141082

Mashequr Rahman Khan

22341068

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
School of Data and Sciences
Brac University
January 2023

© 2023. Brac University
All rights reserved.

Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

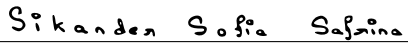
Student's Full Name & Signature:



Anindita Bose
22341067



Raima Islam
22341060



Sikander Sofia Safrina
22341074



Dishari Bin Khurshid
21141082



Mashequr Rahman Khan
22341068

Approval

The thesis/project titled “Authentication Using Blockchain” submitted by

1. Anindita Bose (22341067)
2. Raima Islam (22341060)
3. Sikander Sofia Safrina (22341074)
4. Dishari Bin Khurshid (21141082)
5. Mashequr Rahman Khan (22341068)

Of Fall, 2022 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on January 19, 2023.

Examining Committee:

Supervisor:
(Member)



Mobashir Monim
Lecturer

Department of Computer Science and Engineering
Brac University

Program Coordinator:
(Member)

Md. Golam Rabiul Alam, PhD
Professor

Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Sadia Hamid Kazi

Chairperson and Associate Professor
Department of Computer Science and Engineering
Brac University

Ethics Statement

We, hereby consciously declare that this report is based on the conclusions of our extensive research. The work accurately and completely represents the authors' own research and analysis. The paper properly credits the meaningful contributions of co-authors and co-researchers. All sources are correctly credited and the use of correct referencing has been used to denote a portion of the content. All of the authors were directly and actively involved in the extensive effort that led to the article, and are responsible for its content. This report has not been published or submitted by us or any other individual or institution before.

Abstract

While password-based authentication is widely used by many applications today, it has grave vulnerabilities that can make our devices and accounts prone to malicious attacks. 2-factor authentication (2FA) and Multi-factor authentication (MFA) were introduced to increase the security of the generic password-based authentication and they did enhance the security of the credential-based system by using second or multiple factors to provide additional security. However, even though the introduction of the second factor increased the security, this approach has weaknesses as well: there are many methods through which a hacker can get access to our domain, such as SIM swap hacking, phishing attacks, ambush attacks during password recoveries, or One Time Password (OTP)-based attacks. Instead of addressing the security problems of 2FA or MFA, we aim to strengthen the security of the generic password-based authentication system by adding a layer of security to the existing first factor. In this process, the user will log in using their credentials and on successful verification of the password, the blockchain-based authentication will begin. Lastly, the Diffie- Hellman algorithm will generate a NonceUnified and HashUnified on both the client and server sides. If the NonceUnified and HashUnified values match on both the client and server sides, the user is authenticated and granted access to the system. The Diffie-Hellman algorithm ensures that the user's private key is not transmitted over the network, further amplifying this model's security.

Keywords: Blockchain; Authentication; Security; Hash; Nonce; Diffie-Hellman

Acknowledgement

Firstly, we would like to thank the Almighty for this opportunity and for giving us patience and determination during this thesis. Secondly, we are beyond grateful to our supervisor, Mobashir Monim, and co-supervisor, Dr. Muhammad Iqbal Hos-sain, for their relentless support, invaluable advice, utmost kindness and profound patience throughout this thesis. Saving the best for the last. We would like to thank our parents for their constant encouragement, prayers and support.

Table of Contents

Declaration	i
Approval	ii
Ethics Statement	iii
Abstract	iv
Acknowledgment	v
Table of Contents	vi
List of Figures	viii
1 Introduction	2
1.1 Motivation	3
1.2 Problem Statement	3
1.3 Research Objective	4
2 Literature Review	5
3 Background Analysis	10
3.1 Diffie-Hellman Key Exchange	10
3.1.1 Cryptographic Explanation	10
3.2 Block Architecture	11
3.3 Hash Value	11
3.4 Hash Value Generation	12
3.5 Immutable Ledger	13
3.6 Distributed Peer-to-Peer Network	14
3.7 Block Mining	15
3.8 Consensus Protocol	16
3.9 Proof of Work	16
3.10 Proof of Stake	17
3.11 Delegated Proof of Stake	17
3.12 Ethereum	18
3.13 Encryption	18
3.14 Symmetric Key	19
3.15 Assymmetric Key	19
3.16 Public Key and Private Key	20
3.17 Smart Contracts	20

3.18	Public Key Infrastructure	20
3.19	Elliptic Curve	21
4	Proposed Model	22
4.1	Working Procedure of the Credentials-based Authentication	22
4.2	Working Procedure of the Blockchain-based Authentication	24
4.2.1	CreateNewData method on Client side	24
4.2.2	ProofOfWork method on Client Side	24
4.2.3	HashBlock Method on Client Side	25
4.2.4	Api '/block'	25
4.2.5	CreateNewData Method on Server Side	25
4.2.6	ProofOfWork Method on Server Side	25
4.2.7	HashBlock Method on Server Side	25
4.2.8	Diffie Hellman Algorithm for NonceUnified Generation	26
4.2.9	Api '/pks'	26
4.2.10	Api '/tkts'	27
4.2.11	Previous Hash Determination	27
4.2.12	Hash Unified Generation	28
4.2.13	Hash Storage	28
4.2.14	Logging in Again	28
4.2.15	Device Based Authentication	29
5	Implementation	31
5.1	Client-Server Model	31
5.2	User Login Application	32
5.3	API	39
5.3.1	Client Side	39
5.3.2	Server Side	40
5.4	Database	41
5.5	Blockchain	42
5.6	Scope For Performance Analysis	43
6	Results Analysis	45
7	Conclusion and Future Work	49
	Bibliography	52

List of Figures

3.1	Diffie-Hellman	11
3.2	Genesis Block	11
3.3	Unique Hash Value	12
3.4	Blockchain link created using previous hash	12
3.5	Hash Value Generation	13
3.6	Immutable Ledger	13
3.7	Distributed P2P Network part i	14
3.8	Distributed P2P Network part ii	15
3.9	Distributed P2P Network part iii	15
3.10	Block Mining	16
3.11	Consensus Protocol	16
3.12	Encryption	18
3.13	Symmetric key	19
3.14	Assymmetric key	20
4.1	UML Diagram	23
4.2	Use Case Diagram	24
4.3	Diffie-Hellman Implementation	26
4.4	Workflow Diagram	30
5.1	Home Page	32
5.2	Registration Page	32
5.3	Server side database users table	33
5.4	Client side nonce and hash	33
5.5	Server side nonce and hash	33
5.6	Public key client side	34
5.7	Public key server-side	34
5.8	TempKeyClient calculation	34
5.9	TempKeyServer calculation	34
5.10	NonceUnified calculation on the server side (python)	35
5.11	NonceUnified calculation on the client side (javascript)	35
5.12	NonceUnified and HashUnified on server side	35
5.13	NonceUnified and HashUnified on client side	35
5.14	Redirected to home after successful authentication	36
5.15	Client side local storage	36
5.16	Server side Hash table	36
5.17	User logging in through the Sign In page	37
5.18	Client Side previous hash retrieval	37
5.19	Server-side previous hash retrieval	37

5.20	New hash generated on the client side local storage	38
5.21	Server side database Hash table showing the same updated hash . . .	38
5.22	Redirected to the home after successful login authentication	38
5.23	The register and login API sends post requests to the server side to register and log in respectively	39
5.24	Authentication is successful	39
5.25	The '/block' API sends user data to the server side to generate server side's NonceServer and HashServer	39
5.26	The '/pks' receives the Public Key generated from the server side to be used for the Diffie-Hellman algorithm on the client side	39
5.27	Server Side Values	40
5.28	The '/register' and '/login' routes	40
5.29	The '/me' route	40
5.30	The '/pks' route	40
5.31	The '/block' route	41
5.32	The '/tks' route	41
5.33	Database Table Creation	42
5.34	create_new_data method	42
5.35	proofOfWork method	42
5.36	hashBlock method	43
5.37	create_block method	43
5.38	Implementation of DPoS	43

Chapter 1

Introduction

Daily duties are now completed online as a result of modern technological breakthroughs that have led to the rising usage of electronic devices. From stock markets to food carts on the street, the introduction of the internet in the world wide web has revolutionized every area of our lives. Web 2.0 and the world of e-commerce apps have fostered a technological boom. Because of this widespread reliance on the internet, cyber security is extremely important. The majority of websites require user verification, termed ‘user authentication’. For instance, online banking vividly reflects the sheer need for strong authentication systems by the providers of on-line banking services because otherwise, unfortunate incidents like the Bangladesh money heist will keep recurring around the world.

Password-based authentication is used by the vast majority of websites and applications that require access to verify a user’s identity. Despite serious security drawbacks, the general method of authenticating users with passwords and email credentials remains the primary method for verifying users’ digital identities. In today’s world, it is very unsafe to verify only with passwords as it can be easily cracked by brute force attacks. Moreover, users tend to sign up with easy passwords that they can remember and use this same password in most of their accounts. This makes them vulnerable to dictionary attacks or other guess attacks by hackers. To make this password-based authentication more secure, it was hashed and turned into a distorted version using an algorithm. Security is not guaranteed with only just a password, hence two-factor authentication was introduced and it was able to diminish the problem to some extent as it uses a centralized entity that sends one-time passwords or hardware tokens to the users each time the system is being attempted to be accessed by the users. Despite the additional security layer given by two-factor authentication, its colossal drawback is its full dependency upon the third party for providing the secret codes [1].

There are several problems associated with 2FA and MFA that can reduce their potency. A major problem with 2FA and MFA is that they can be vulnerable to phishing attacks, as the security token can be intercepted by attackers through phishing emails and websites [2]. The user may unknowingly reveal their code to the attacker, allowing them to gain access to the user’s account. Additionally, 2FA and MFA which are based on SMS messages are susceptible to SIM swapping attacks [3], where an attacker is able to take control of a user’s phone number, and in turn,

receive the 2FA or MFA code sent to that phone number. These methods also incur an extra cost for the service provider.

In recent years, Blockchain technology has become increasingly important with many specialists highlighting its possible uses across a wide range of organizations. Blockchain is a distributed ledger system at its core. Its architecture provides advanced security and its tamper-proof structure through distributed algorithms and hashing makes it well-suited for comprehensive applications and enhances security, as it is impossible to generate false records or tamper them [4].

Therefore, in this paper, we propose a device-based authentication system that will utilize blockchain to strengthen the security of credential-based authentication on the first factor. The hashes generated will be used to identify the particular device when trying to log in. This idea will also utilize the Diffie-Hellman Key exchange algorithm to unify keys and provide safer transmission.

1.1 Motivation

From a security standpoint, we believe that blockchain serves as an ideal data structure that can be integrated into an authentication system. The system is designed to be robust allowing it to be operated in highly contested systems where there is always the risk of malicious attacks being carried out. Blockchain's features (with regards to verifiability, immutability, and decentralization), when aided with other cryptographic tools, greatly improves the security of a system as well as enhancing its scalability [5]. It has already been explored in numerous contexts - data storage, data provenance, identity management, crowdsourcing, and much more [6][7][8][9][10].

1.2 Problem Statement

While 2FA and MFA systems provide several steps in verifying one's identity, it also ensures protection if a user has weak credentials, reduce chances of identity theft, shield companies from any breaches of security, save small businesses that are prone to easy malicious attacks and provides economical means of authentication using mobile devices that significantly enhance security without having to learn new methods of for every application [11][12]. Even though multi-factor authentication does provide all-around security for protecting users, organizations, and their sensitive information and objects, some of the downsides of authentication is having to own a secondary item such as another device or tokens. Mobile phone 2FA was developed to handle such problems but there is still the need to possess another device which can be a bit costly. Also, MFA systems require users to have some technical skills when using SMS or E-mail one-time passwords. Although multi-factor authentication reduces the risk of unauthorized access to a victim's information by requiring more than just a password, it is still susceptible to certain types of attacks such as man-in-the-browser and man-in-the-middle attacks [13]. Therefore, we proposed a device-based authentication security system using blockchain for a user login application which will make the process of authentication more enhanced

by only giving access to users if they are logging in from that particular registered device using device-specific hashes. This will make sure even if a hacker has the user credentials unless they are logging in from the registered device, access will not be granted.

1.3 Research Objective

With more techniques of security measure bypass being developed, it is critical that the authentication procedure be upgraded as well. While the generic email and password-based credentials system has been around for a while, it has yet to be considered the best security practice, and 2FA or MFA is also advised. Therefore, we have come up with a model, where we are strengthening the generic credential-based authentication by adding another layer of security in the same authentication factor. We are incorporating Blockchain technology to generate a hash that should be the same on both the client and server sides which will further confirm that the credentials that the user has given are indeed authentic. However, this step is not a 2FA/MFA replacement but rather an enhancement of the existing generic credentials system and 2FA/MFA can be used on this model where extra security is required. We plan to develop a blockchain-based authentication protocol that can be used as an added layer alongside a generic credentials system.

Chapter 2

Literature Review

Brasee's (2009) describes secure distributed SSO, a revolutionary single sign-on (SSO) system (SeDSSO) [13]. With a distributed authentication service, SeDSSO offers safe, fault-tolerant authentication utilizing threshold key encryption. The authentication service consists of numerous servers that work together to sign messages using a (t, n) threshold encryption technique. To sign a message, the service needs distinct server-signed messages. In order to increase security, the system also has a two-factor identification that employs both a username/password and a special USB device. The approach is designed to deal with the problem of maintaining several personal accounts and passwords, which may be difficult for people to remember and often leads to weak password security. The system has been evaluated via simulation, demonstrating its successful performance, and it intends to outperform current SSO solutions by being more secure and useful.

Gunson's (2011) study examines users' opinions on the security and usability of single-factor and two-factor authentication techniques used in automated telephone banking [14]. The experiment included 62 banking clients who were given the option of utilizing a knowledge-based, single-factor authentication technique or a two-factor strategy that included a hardware security token to produce a one-time passcode in addition to the knowledge-based step. The two-factor version was rated as having greater levels of security than the single-factor version, according to the results, but this advantage was countered by lower ratings for usability, convenience, and simplicity of use. The two-factor authentication process also required more time to finish. The study offers insightful factual proof of the trade-off between automated systems' usability and security.

Result of three different methods for generating a one-time access code for two-factor authentication: a portable keyfob, a smartcard and reader, and a mobile phone. The portable keyfob is an inexpensive option where the user presses a button on the device and an access code is displayed. However, it does require the user to carry around the hardware. The smartcard and reader option is a more expensive solution, but it is compatible with a Chip and PIN world. The user inserts the card into the reader and types in a PIN, and the access code is displayed. The last method is to use a mobile phone, where the user requests a text message containing the access code. This is also an inexpensive option but it relies on the user having a mobile phone and a good signal reception.

In 2017, Isler proposed a solution that provides security against attacks such as phishing, man-in-the-middle, honeypot, and offline dictionary attacks [15]. The paper emphasized how passwords are the most commonly used method of online user authentication with the aid of a login server. It also discusses how other solutions at the time are non-portable and are prone to many types of attacks. The solution model is known as the Threshold Single Password Authentication (Threshold SPA) and its schemes have been developed to define fitting and real-world indistinguishability, and thus can be applied by the means of a browser extension or mobile application. It makes use of multiple storage providers where the secret keys corresponding to the verification keys are kept with the verification keys being stored in the login server and the secret keys being manipulated by a function of the user passwords. Threshold SPA protocol aims to fix the issues by implementing three types of players, users, login servers, and storage providers and has two phases, the registration phase and authentication phase with each phase having multiple protocols.

This paper focuses on how blockchain is still not widely used in education systems but has great possibilities if implemented in Indonesia [16]. With education institutions wanting to employ the latest technologies, individual profiles can be created and their digital certificates can be issued with blockchain [17]. This can help reduce the manipulation of school or diploma certificates of educational organizations. While blockchain helps improve the measure of security, using ubiquitous technology can make security issues more complicated and harder to resolve [18]. By using this ubiquitous method, we can authenticate these digital educational certificates by using blockchain. These certificates can be used and retained for an extended period of time.

While the Internet of Things (IoT) saw the usage of revolutionary devices being used and implemented in various applications in different industries, such as smart cities, healthcare and communities, these instruments also produce a significant amount of sensitive and private data. These factors make protecting access to them a difficult task. The present methods for authentication and identification have significant drawbacks because of their widespread usage. As a result, the protection of such gadgets is critical to guarantee the program's safety and efficacy. Thus, Joshi's work proposed a novel blockchain based authentication system which maintains user privacy while maintaining security. This study provided an access control approach along with a decentralized authentication for IoT which is lightweight and for identification and secure communication with IoT devices, a blockchain system is utilized. This model is based on blockchain technology and fog computing, and the results of testing have shown that it outperforms other verification methods which use blockchain. The proposed method takes advantage of blockchain's existing advantageous properties and enhances existing authentication mediums [19].

Currently, information is a vital part of any institution that requires advanced shielding as to authenticate themselves: users need to verify their identity to execute any work [20]. Due to this, organizations want to control the roles of users in accordance with the central authority's rules. For this, the most popular one is Role-Based Ac-

cess Control (RBAC) mechanism which enables assigning permissions for users and resources based on the user's role within the facility [21][22]. Blockchain, with its decentralized characters has been incorporated in Kamboj's (2017, October) paper, where an RBAC model utilizes a smart contract based on blockchain to handle user-role permissions in a workplace. This model utilizes the Ethereum blockchain to manage interactions between assets and users. Before assigning roles and permissions, the framework includes authentication and verification for users. With the use of blockchain, this framework handles role-permission assignments and user-role assignments. Two smart contracts have been developed to facilitate communication within the organization between resource owners, users and role-issuers. The smart contract managed by the resource owner is utilized to give permission or decline access to the resource. To prevent attacks such as man-in-the-middle, a threat and security model has been implemented along with an algorithm to verify the authenticity of devices [23].

In the last ten years, smart home systems have become very popular because of their ability to improve life quality and comfort. By integrating Internet of Things (IoT) devices into actuators, these homes can securely communicate data. In contrast, smart home systems provide the convenience of requiring each device to authentication separately using credentials. In a trusted environment, this is not needed where sharing of same accounts can occur using various devices and if that happens, a consent basis procedure can be followed as the authentication medium. Therefore, with such a conventional authentication system, to evade redundant logins, there is no central authorization mechanism. In this matter, Mukherjee et al. proposed a system using Blockchain to give a decentralized framework for smart contract-based applications. Secured identification of people and devices can be provided by combining blockchain and public-key cryptography. For obtaining access in a similar service, this solution reduces communication latency and ignores duplicate authentication. The prominent security aims, such as confidentiality, authentication, integrity and authorization are satisfied by this model [24].

In Yazdinejad's (2020, August) paper discusses the importance of authentication with regards to establishing a secure channel of communication in hospital networks and proposes a blockchain-based authentication model as blockchain is able to provide a transparent and efficient communication platform [25]. Their system can record data in a secure manner in a rather geographically diversified hospital network. Peer to peer (P2P) communication can take place between all members of the scheme who are also able to move easily to other affiliated hospitals present in the network with the help of their distributed identity. The authentication system is not restricted to device based authentication which leads to increased throughput, reduced time overhead and energy consumption on the devices.

In 2021, Mubarakali suggested a solution: a blockchain based micro Wireless Sensor Network (WSN) identity authentication tool which integrates blockchain decentralization for decentralized delivery [26]. The paper raised the point that wireless networks make for environments that are prone to network attacks, they require relevant security measures, and that blockchain technology is the appropriate measure as it can protect data which are being handled by and in possession of wireless sen-

sensor networks. A private blockchain framework of sensor networks is created between cluster managers in an individual WSN, all WSN ground stations are connected to the public blockchain, and across the whole network a hybrid blockchain system is integrated. Within this proposed framework, the registration of a user takes place. Upon analysis, it can be seen that the model is protective and productive, and the experimental results reveal that the computational efficiency and security performance of the model are considerable.

Patwary's (2020, August) paper sheds light on fog computing which at the time was an emerging computing framework [27]. Fog Computing expands cloud based computing services close to the edge of the network. The distributed ownership of this system raises a few alarms in the security and privacy sectors. Regular forms of authentication, such as password based, certificate based, and biometric based, are not a good fit for the system due to the unique and complicated architecture and features of the system, and the authors emphasizes on the fact that regular authentication methods are subject to consuming larger computational resources and lead to high latency which does not meet the requirements of the Fog. To counter this, the author proposed a decentralised location based device to device (D2D) authentication system in which Fog devices can manually verify each other within the Fog layer with the aid of blockchain without depending on any intermediary, such as a trusted third party. Blockchain technology is used to carry out mutual authentication processes using Ethereum smart contracts. The Fog devices only have to store just a few keys for authentication, thus meeting the standard security requirements. The Fog devices mutually and securely authenticate themselves in an efficient and effective manner.

Li's (2020, November) paper focuses on strengthening password authentication [28]. The paper discusses Password Authenticated Key Exchange (PAKE) at length. PAKE is implemented to make password authentication more secure and prevent password-cracking, the system can be deployed between two peer participants. The issue with traditional PAKE is that when the remote server is compromised the passwords are unprotected and are in plaintext. The authors' goal is to integrate lattice-based password-hashing schemes (PHS) into the symmetric-PAKE scheme which requires the implementation of smooth projective hash functions (SPHF). In doing so they are able to construct an asymmetric PAKE system which is secure from quantum attacks. Non-interactive zero knowledge (NIZK) method proof is not needed for this model thus reducing time complexity and making the model not as costly. The model is able to bypass assumptions of the random oracle model as well allowing the model to achieve quantum resistance.

Lin (2018) proposed a hierarchical model consisting of four definite layers, and it is designed to vertically integrate inter-organizational value networks, engineering value chains, manufacturing factories, etc., a blockchain-based framework for secure mutual authentication, BSeIn, designed specifically for Industry 4.0 applications [5]. This is an attempt to better prepare organizations and industries for the 'Industry 4.0' era. The key focuses of this framework are privacy protection and anonymity, fine-grained access control, and integrity and confidentiality. It has been highlighted that a simple password authentication protocol is not sustainable in terms of protect-

ing the privacy of the clients' identities as a malicious or compromised cloud may be able to easily identify a client or user based on their daily routine. One common way to achieve fine-grain access control is by constructing a user-role mapping table and role authorization table (Sandhu et al., 1996), but these tables are prone to being attacked at the cloud-end due to the tables being mainly maintained by the semi-trusted cloud [29]. Access history is stored by the cloud in a local database, therefore posing a risk that the database can fall victim to unauthorized access and modification, even the traceability and auditability of access records will be difficult. The proposed system takes advantage of the notable characteristics of blockchain as well as numerous cryptographic materials to realize decentralized, privacy-preserving and auditable solutions, and used attribute-based signatures to anonymously authenticate terminals, message authentication code to expertly authenticate gateways, and a secure certificateless multi receiver encryption to eliminate single point of failure issue [30][31][5].

Chapter 3

Background Analysis

3.1 Diffie-Hellman Key Exchange

Diffie-Hellman (DH) was proposed by Whitfield Diffie and Martin Hellman in 1976, it is a mathematically reliable and secure way to exchange cryptographic keys over a public channel via the Diffie-Hellman key exchange algorithm. In order to encrypt future communications using a symmetric-key cipher, two parties who will be communicating together generate a shared secret key using an insecure channel. Diffie-Hellman ensures that the same secret message is generated between two parties without having to share anything private over a public channel.

3.1.1 Cryptographic Explanation

The protocol implements the multiplicative group of integers modulo p and g , where g is a primitive root modulo p , and p is a prime integer. The parameters are chosen so as to guarantee that the output secret key will always fall between 1 and $p-1$, where p is a prime number.

- Alice and Bob decide to use a modulus $p=13$ and base $g=7$
- Alice picks a secret integer $a=4$, and sends Bob $A = g^a \text{ mod } p$
- $A = 7^4 \text{ mod } 13 = 9$
- Bob picks a secret integer $b = 3$, and sends Alice $B = g^b \text{ mod } p$
- $B = 7^3 \text{ mod } 13 = 5$
- Alice calculates $s = B^a \text{ mod } p$
- $s = 5^4 \text{ mod } 13 = 1$
- Bob calculates $s = A^b \text{ mod } p$
- $s = 9^3 \text{ mod } 13 = 1$

Both Alice and Bob arrive at the same values under $\text{mod } p$

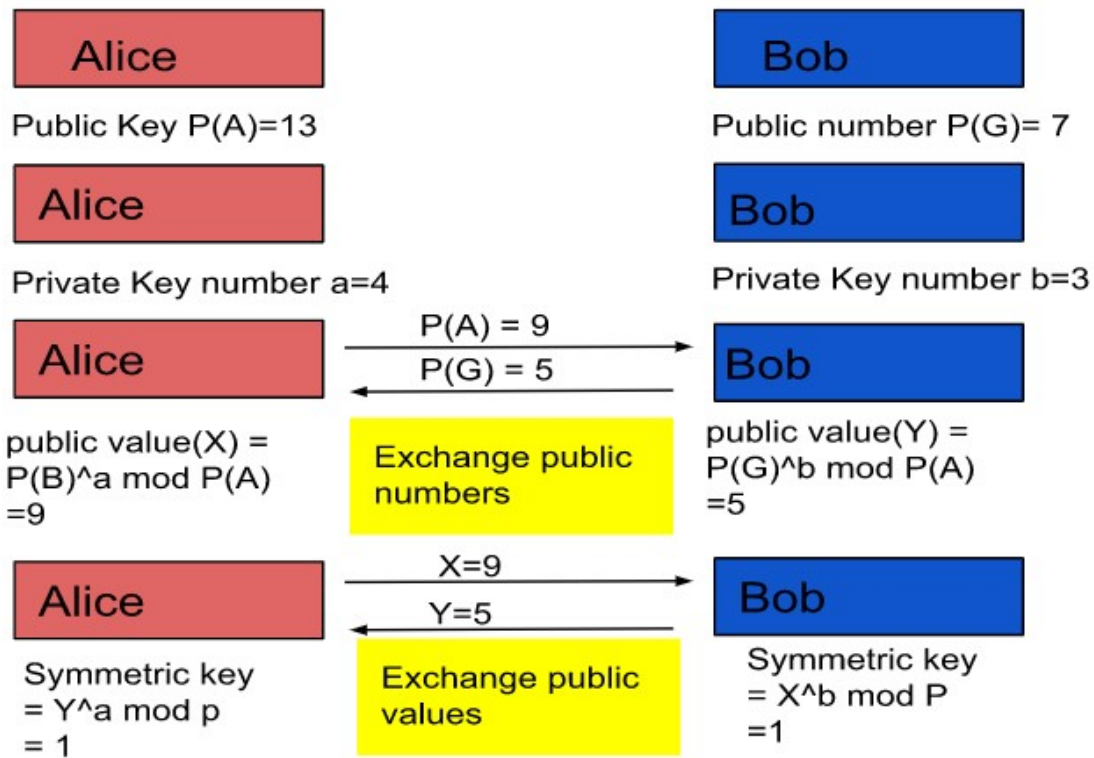


Figure 3.1: Diffie-Hellman

3.2 Block Architecture

Every blockchain has the following data stored in it: nonce, timestamp, data, previous hash and hash. However, the first block, also known as the genesis block, is the only block without the previous hash. The previous hash is set to be null in the genesis block and that is why the genesis block or the starting block is the only exception in the blockchain.

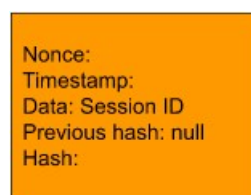


Figure 3.2: Genesis Block

3.3 Hash Value

Each block in a blockchain consists of several fields. To better understand what a block is, let us first take 3 fields of a block into consideration: data, hash and previous hash. The data field can contain anything; for instance, in bitcoin, the data consists of a list of transactions. The hash value is like an identification number of this block and the previous hash is the hash value of the previous block. Hence, each block is linked to the other one using hash cryptography because the previous hash value of this block is storing the hash value of its previous block.

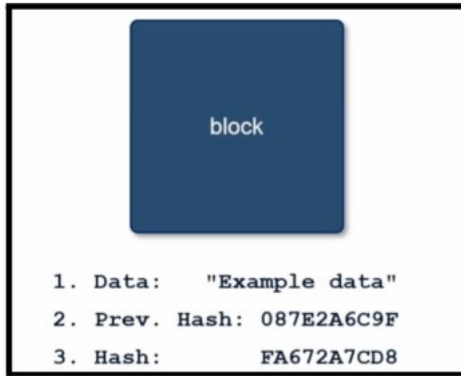


Figure 3.3: Unique Hash Value

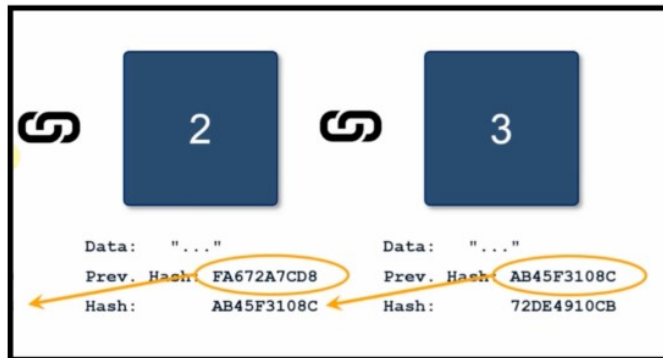


Figure 3.4: Blockchain link created using previous hash

3.4 Hash Value Generation

The data field can store any form of document as discussed earlier- it can be in txt, zip, obj, png, mp3, jpg, translation, etc format. The SHA256 algorithm takes any form of data as input and the algorithm hashes that data to form the 64-bit hexadecimal hash value of a block. The data that is being hashed is not just the data in the data field, rather the algorithm is hashing the entire data in that block. Therefore, data stored in each field is being used to calculate one hash value. Each block has a unique hash value because even if the data in all the fields match with another block, the previous hash value remains unique. Hence, when SHA256 is taking data in all the fields as an input (which includes the previous hash value), it is automatically generating a hash value that is strictly unique to that particular block only. Despite the variable length of the data that SHA256 is hashing depending on an individual block, the resulting hash value will always be a 64-bit hexadecimal value.

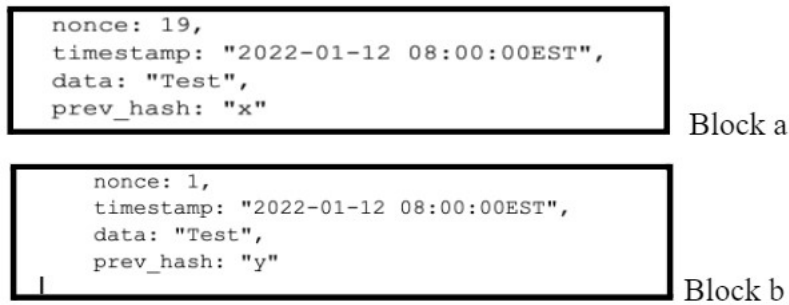


Figure 3.5: Hash Value Generation

3.5 Immutable Ledger

For simplicity, let us assume that there is a blockchain with 10 blocks. Let the hash value of the 5th block be “G”. If an attacker changes the data in the 5th block, its hash value will automatically change to a new one and this will ultimately break the 5th block’s link with the 6th, 7th, 8th, 9th and 10th blocks. This is because the 6th block’s previous hash value is still “G”; however, the attacker has altered the data in the 5th block, and the 5th block’s hash value has now changed. Therefore, the 6th block’s previous hash value is no longer pointing to the 5th block. This, in turn, has broken the chain and this blockchain is now invalid. Validating the chain using software is very quick and easy as it only checks if all the previous hash values correspond to the respective previous blocks. Furthermore, changing the data in the 5th block is very difficult for the attacker because he has to mine this block. Mining a block is very difficult due to the complicated nonce value calculation and on top of that, if he wants to effectively hack this blockchain, he will have to simultaneously mine the 6th, 7th, 8th, 9th and 10th blocks as well in order to maintain the cryptographic link from the 1st block to the last. Lastly, in reality, blockchains are very long which further increases the difficulty of hacking and data alteration.

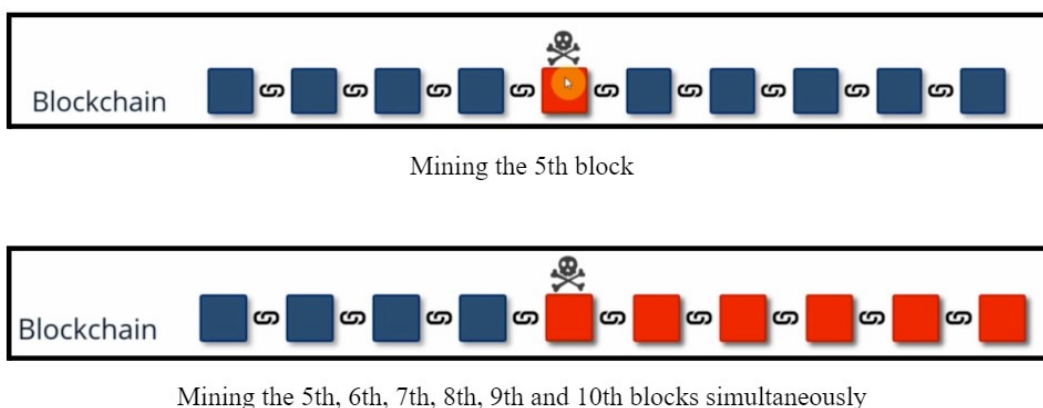


Figure 3.6: Immutable Ledger

3.6 Distributed Peer-to-Peer Network

In a permissioned blockchain network, it is not possible to join this network just by anyone because only a certain number of nodes with copies of the blockchain make up the network and new nodes are not allowed to join this network. However, in permissionless blockchain networks, for instance, bitcoin, new nodes are allowed to join the blockchain network and hence, those newly added nodes are allowed to keep a copy of the blockchain as well. To further explain the fundamental idea of distributed peer-to-peer networks (P2P), let us refer to the diagram below. The diagram consists of 7 nodes and it is a permissioned blockchain network. If an attacker wants to change the data in one of the nodes, as discussed earlier, the blockchain's immutable ledger principle makes it difficult to do so. Additionally, the distributed peer-to-peer network adds another layer of security because all the nodes in the blockchain network communicate with each other using protocols. Hence, even if the attacker simultaneously mines the 5th, 6th, 7th, 8th, 9th, and 10th blocks successfully (which is almost impossible because blockchains are usually very long and mining each block is computationally very difficult), this attacked node is connected with 4 other nodes. Through communicating with each other, the 4 directly connected nodes identify that the attacked node's blockchain copy does not match theirs and since blockchain follows the majority rule, the attacked node's affected blockchain is again updated with the original copy of the blockchain from its neighbors and the hacked blockchain is discarded. The attacker will only be able to change the blockchain if he can make the change in more than fifty percent of the nodes in the network at the same time. However, breaking into half the network at once is computationally almost impossible. Therefore, immutable ledger and distributed P2P network principles ensure very high security in blockchain technology.

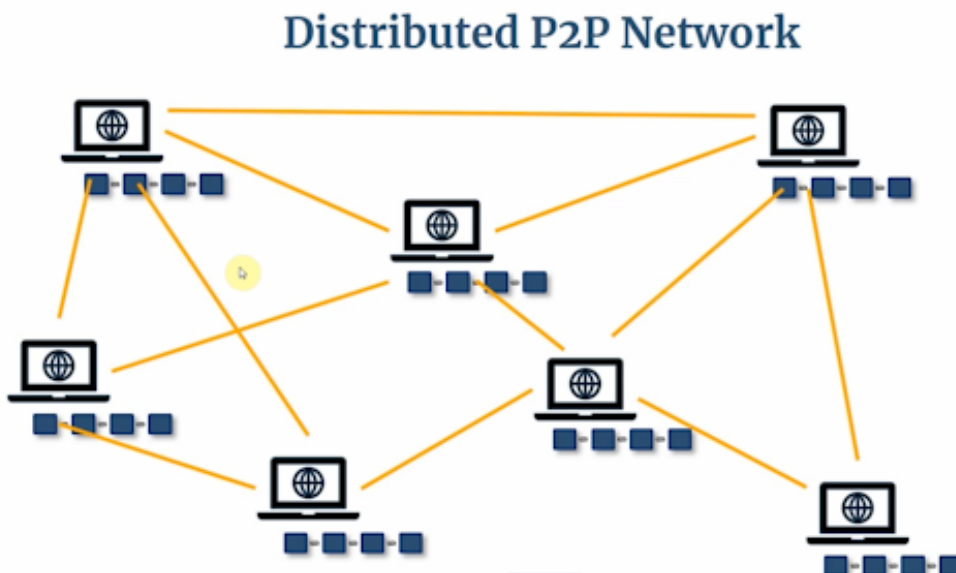


Figure 3.7: Distributed P2P Network part i

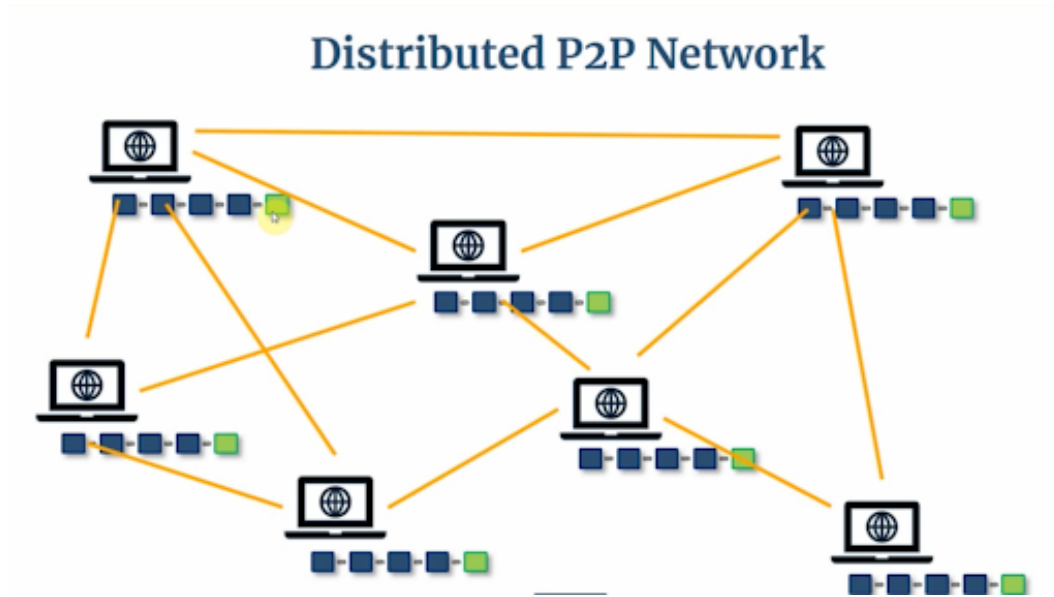


Figure 3.8: Distributed P2P Network part ii

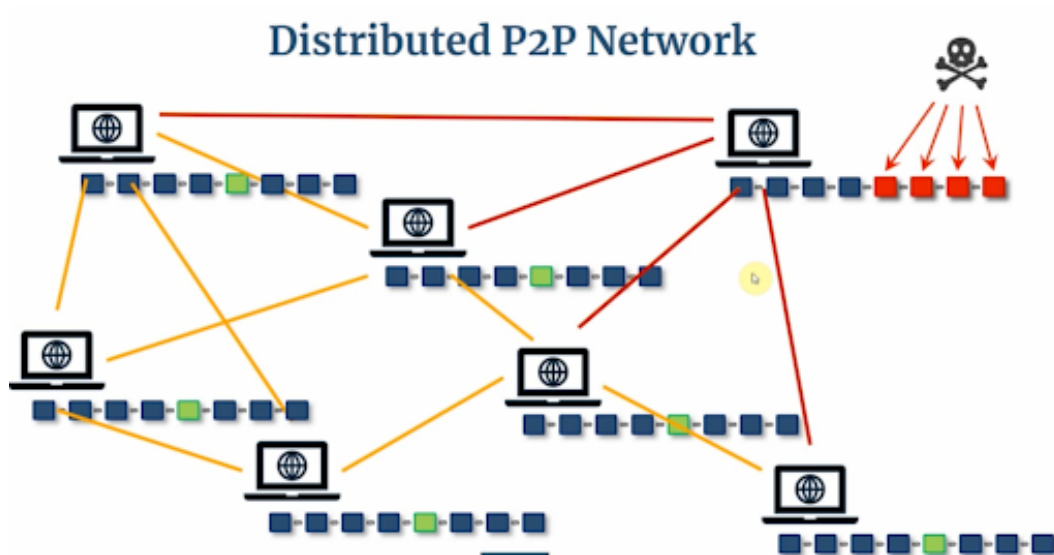


Figure 3.9: Distributed P2P Network part iii

3.7 Block Mining

A block may consist of several fields but it will always contain nonce, timestamp, data, previous hash and hash. Timestamp stores tiny data (serial number) that records the exact moment in which the block has been mined and validated by the network. The SHA256 algorithm takes nonce, timestamp, data and previous hash as input to generate the hash value. The hash value that has been generated must fall under the target value for the block to be mined. The diagram below demonstrates that only the last hash value has met the criteria. The timestamp, data and previous hash fields remained constant in all three but the value of the nonce was changed using the trial and error method to calculate the valid hash value. The avalanche effect makes it even harder to calculate the nonce value which makes it computationally even more difficult and expensive to generate the nonce

value.



Figure 3.10: Block Mining

3.8 Consensus Protocol

As discussed earlier, mining a block is computationally very difficult and the system rewards the miner with bitcoins. If an attacker is after the bitcoins and successfully mines a new malicious block at the end of the blockchain of a node, this new block will propagate through the network. All the other nodes in the network will now start verifying the content of the block which includes verification of the cryptographic link of this new block with the previous ones and then they run a list of other tests. Mining a new block is tough but verifying that block is fast and easy due to the process discussed above. Therefore, after the verification process is done, the other nodes will not accept that malicious block due to this process that is using consensus protocol.

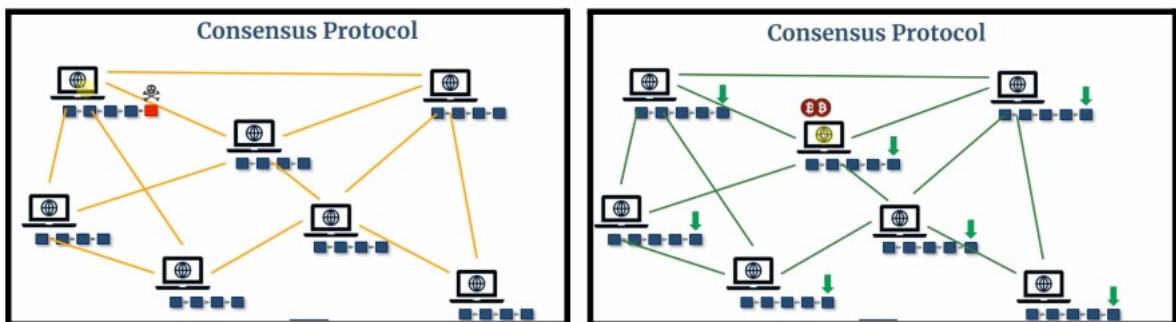


Figure 3.11: Consensus Protocol

3.9 Proof of Work

An example of Consensus Protocol is Proof of Work (PoW). It is one of the first of its kind to be used in blockchain based applications. It focuses on computing hash

values and validating transactions. It was designed for permissionless public ledgers and makes use of the computational resources available in the systems in the node. The transactions take the form of blocks which are fitted into a linear structure. Each and every transaction is validated and signed using the corresponding public and private keys that belong to the user. Bitcoin implements Proof of Work. PoW has a relatively large energy consumption rate and low speed when it comes to verification processes as compared to the likes of MasterCard or Visa making consumers look towards other consensus protocols. It uses up a lot of resources and requires lots of power and electricity to solve cryptographic puzzles.

3.10 Proof of Stake

When it comes to dealing with transactions in platforms like Bitcoin, a user has to validate the transactions by computing the hash value with a certain number of trailing zeros which helps with the allocation of the bitcoins. In Proof of Stake (PoS), a validator is picked and assigned a block in accordance to their economic stake in the network. The mission is to avoid the centralization of mining centers and to provide an opportunity to validate all miners. The miner holds the responsibility of allocating a particular part of their cryptocurrency in order to start the validation process, and if they succeed in invalidating the transaction, then they are rewarded with the stake they had pledged initially. This is a method by which penalties are given for bad behaviour and promotes fair behaviour. Proof of Stake is environmentally and economically friendly as it does not consume much power and saves electricity due to it not requiring to solve computational puzzles and special hardware isn't required for it as well. There are some drawbacks to PoS. An attacker would need to possess greater than 50% of the accuracy to gain control over the network compared to 51% in Proof of Work. One possible attack on a PoS based system network is a bribe attack where the attacker reverses the victim's transactions and offers a bribe(s) to the miners in an attempt to confirm the transactions.

3.11 Delegated Proof of Stake

A voting-based consensus technique called Delegated Proof of Stake (DPoS) was developed from Proof of Stake. For instance, there are ten nodes in the network. The nodes of the network will vote and elect the particular node who will have the authority to validate blocks. The chosen nodes are known delegates. The money invested in the protocol during the voting sessions is locked up in smart contracts. The group of elected delegates who are the block producers carry out transaction validation, block creation, network operations, and other maintenance. They are compensated according to the amount of work they have completed. Any unfavorable behavior, such as collusion or a skipped turn, may lead to the removal of a delegated node from the group of validated delegates.

When it comes to selecting who gets to verify the blocks, this consensus protocol has a democratic theme to it, which is how a more diverse group of people are able to participate in the process because it is based on the staker's earned reputation as a lawful staker rather than their overall wealth.

The basic flaws of the PoS system, like the "nothing-at-stake" issue, long-range attacks, and weak subjectivity, are clarified by the DPoS mechanism [32]. Due to its higher efficiency and high throughput, it uses less energy. Due to the low threshold, DPoS is recognized as the most decentralized method to consensus protocols. It does have some drawbacks, one of which is that it has a tendency to centralize, and participants with sufficiently substantial stakes in the network will be able to elect themselves as validators.

3.12 Ethereum

Every Ethereum node in the Ethereum network generates a copy of the Ethereum Virtual Machine (EVM). Users can request any computation from this computer. When a request is broadcast, other network nodes verify, validate, and compute, changing the EVM state and sending it to the entire network. All nodes store the blockchain's transaction history and EVM state [33]. Cryptographic mechanisms prevent tampering after a transaction is verified and added to the blockchain. It ensures all transactions are signed and executed legally.

3.13 Encryption

Suppose, Alice wants to say something to Bob. However, Trudy is trying to eavesdrop. How can Alice ensure that nobody but Bob gets Alice's message?

Alice's message is: Hello

Alice now uses a mathematical algorithm to rearrange the word. For simplicity let's say, Alice uses a simple technique called Caesar Cipher.

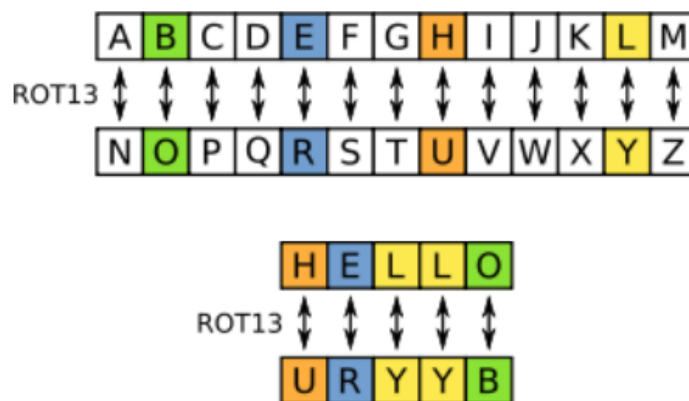


Figure 3.12: Encryption

Therefore:

H = U

E = R

L = Y

L = Y

O = B

Alice has now turned her plain text “HELLO” to some gibberish which is known as ciphertext. The ciphertext is “URYYB”. Alice has sent the key to Bob using which only Bob will be able to decode the ciphertext. Even if Trudy has access to it, he can’t decode it. This is basic encryption. However, encryptions like caesar-ciphers are way too simple in this age and can be broken down by brute force attack quite quickly and easily. Therefore, it is important to use strong encryption algorithms.

3.14 Symmetric Key

In Symmetric Key cryptography, the key that is used is the secret key. This key is used by both the sender and the receiver for encryption and decryption respectively. Therefore, only one key is used in symmetric key cryptography.

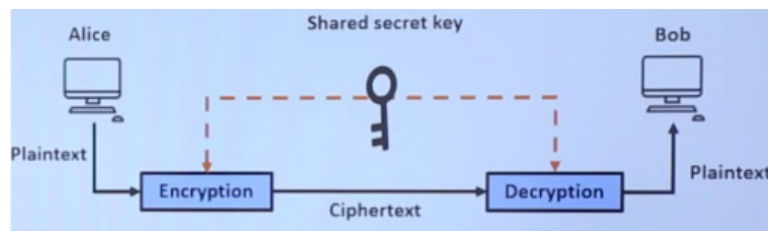


Figure 3.13: Symmetric key

Suppose, there are two users. Alice and Bob. Alice’s message (plaintext) is getting converted to ciphertext through encryption by a secret key. Bob uses the same key to decrypt the ciphertext into the original message sent by Alice. Other people who don’t have the secret key won’t be able to read the message as it would be encrypted.

3.15 Asymmetric Key

Asymmetric key cryptography provides better security than Symmetric cryptography. Here, two keys are used - a public key and a private key. The receiver’s public key is used by the user to encrypt, whilst the receiver uses their own private key to decrypt. Rivest Shamir Adleman (RSA), Digital Signature Standard (DSS), Digital Signature Algorithm (DSA), Elliptical Curve Cryptography (ECC) are some common examples of asymmetric encryption .

Consider those two users, Alice and Bob again. Here, Alice will use Bob’s public key to encrypt the plaintext. The ciphertext (encrypted text) will be decrypted back to the original text by Bob’s own private key. This private key is owned by only Bob. Not even Alice can decrypt her own text, as she doesn’t have Bob’s private key. However, Bob’s public key is accessible to the public, i.e. Bob’s public key will

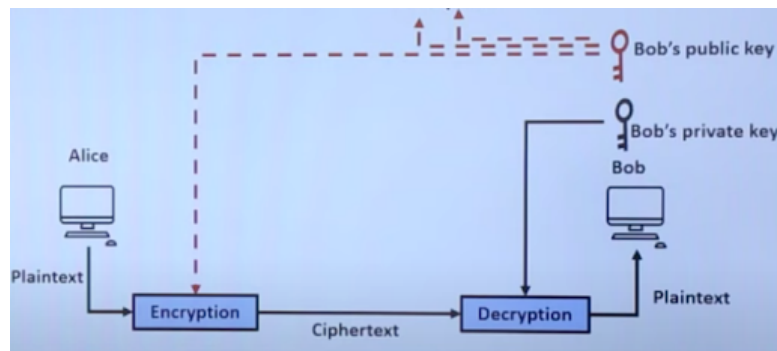


Figure 3.14: Asymmetric key

be available to everyone. In short, Bob's public key will be used to encrypt the text while Bob's private key will be used to decrypt the text. Example: Asymmetric encryption is also used in Bitcoin to ensure that only the owner of a money wallet may withdraw or move funds from it.

3.16 Public Key and Private Key

Public key encrypts, private key decrypts. Sender and recipient public keys are first shared. Suppose Alice and Bob are users who will be sharing messages between them. Bob will get this ciphertext encrypted by Alice using his public key. Only Bob's private key can decipher the ciphertext. Alice cannot decipher her text. Bob must encrypt a text using Alice's public key to transmit it now. Alice may decode the message using her private key. Suppose an attacker found Alice's private key. The attacker can read Alice's messages.

3.17 Smart Contracts

A smart contract extends the concept of storing data in a secure ledger to include computation. To put it another way, it's a consensus technique for executing a publicly defined program correctly. Users can use these smart-contract programs' functions, subject to the program's limits, and the function code is run in parallel by the miners. Users may rely on the results without having to redo the calculations, and they can develop their own programs to react to the results of other programs. Because the algorithms in issue can manage money—own it, transfer it, destroy it, and, in some circumstances, even print it—smart contracts are exceptionally powerful when linked with a cryptocurrency platform. For smart contracts, Bitcoin uses a restricted programming language. In this language, a "standard" transaction (one that transmits cash from one address to another) is stated as a brief script. Ethereum provides a more flexible and powerful programming language [34].

3.18 Public Key Infrastructure

In asymmetric cryptography, the user is provided with two keys. The two sorts of keys are the public and private keys. The issue of key management was solved

since the public key is not a secret, but an equivalent challenge, namely the problem of authentication or name management, was presented. To solve this problem, the public key infrastructure (PKI) was developed to facilitate public key cryptography. Authentication refers to the practice of employing all PKIs. Certification and validation are the most fundamental PKI operations. Certification, which legitimately associates the value of the public key with an entity, is the most fundamental function of all PKIs. The second operation is validation, which is the process of determining whether or not certificates are valid (still valid or not). A complete public key infrastructure [35] includes the registration authority (RA), certificate authority (CA), security policy, PKI-enabled applications, distribution mechanism, and certificate repository.

3.19 Elliptic Curve

ECC, or elliptical curve encryption, is a second form of asymmetric encryption. ECC is an alternative to RSA that can be utilized in its place. It is an effective cryptographic strategy. By applying the mathematics of elliptic curves, security is generated between key pairs for public key encryption. ECC generates keys that are more difficult to crack mathematically. Consequently, ECC is regarded as the future of public key cryptography and is regarded as more secure than RSA. ECC is a plane curve formed by points satisfying the equation $y^2 = x^3 + ax + b$ over a finite field. ECC employs a smaller key size and provides a high level of security. A 384-bit elliptic curve key provides the same level of security as a 7680-bit RSA key. ECC is one of the most popular methods for implementing digital signatures in cryptocurrencies. Bitcoin and Ethereum both sign transactions using the Elliptic Curve Digital Signature Algorithm (ECDSA). Due to its shorter key length and increased efficiency, ECC will be adopted by the majority of online applications in the coming years [36].

Chapter 4

Proposed Model

The process of authentication will begin from the client side, where users will try registering by inputting their credentials. After the credential-based registration is successful, the Blockchain-based authentication will begin to ensure the nonce and hash on both ends (client and server) will be the same. In order to achieve that, we used the Diffe-Hellman algorithm. We performed multiple mathematical operations to get our desired output and once the nonce and hash from both sides have been matched, the verification will be successful. When the user logs in again, they must use the correct credentials; the system then checks the user ID to verify the user and then a new block is mined for this log in on both the client and server sides. Since the user was already verified using the user ID, the previous hash of this new log in block will be the hash of the genesis block created during the registration process.

4.1 Working Procedure of the Credentials-based Authentication

Primarily our workflow will be following 2 specific paradigms. One side is the server side, the other is the client side. The server side is constructed using the Python flask micro web framework whilst the client side is constructed using React JavaScript.

For our implementation of our **server model**, we have implemented the following classes:

- App
- Model
- Config
- Blockchain

We will be utilizing SQLAlchemy and SQLite toolkits for creating the database in order to store the required information from the server side. We will be creating two tables on our database. The first table will be called the **user's table** and will be used to store the following:

- ID
- Name
- Email
- Password

The second table will be called **BlockHash** which will be used to store the following:

- ID
- User_ID (user ID from the User table)
- Nonce
- Hash
- Created_at

The User_ID of the BlockHash table will be the ID from the Users table and the two tables will have a foreign key relationship through the User_ID.

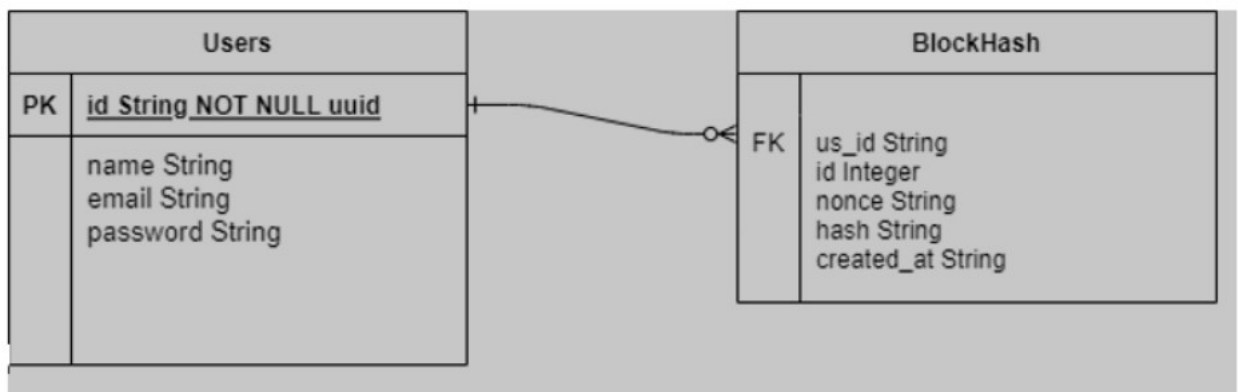


Figure 4.1: UML Diagram

We will utilize Redis to store values within the session in order to retrieve active users' directories using Redis session. The process will commence on the client side, with the user first registering themselves by entering their name, email address, and password. After which an API request will be sent to the server, which will collect the user input value from the client end and register it in the database table entitled user. For the request, a return value containing the id, name and email will be sent to the client and kept in a variable named data. The user id will be assigned using UUID, which stands for Universally Unique Identifier. It is a 128-bit label id that has 32 digits. This will lower the predictability of our user id. The password will be stored as a hash in the database.

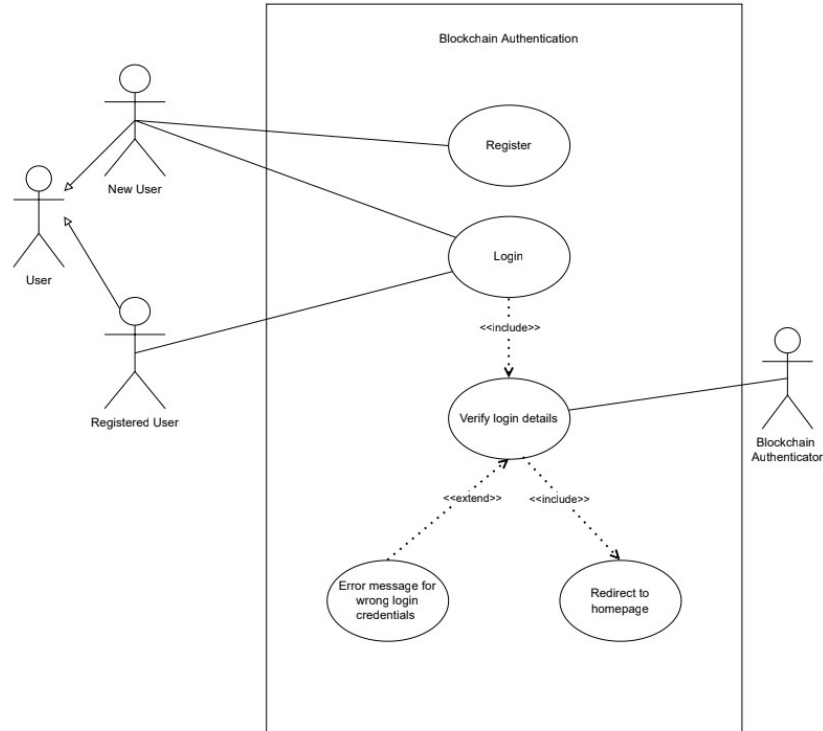


Figure 4.2: Use Case Diagram

4.2 Working Procedure of the Blockchain-based Authentication

For our implementation of the Blockchain class on both server side and client side, we have implemented the following method for the Blockchain class on both sides:

- CreateNewData
- ProofOfWork
- HashBlock
- CreateNewBlock

4.2.1 CreateNewData method on Client side

We will first begin with the registration process, which will create our genesis block. For this, our previous hash will be zero. Then, we will begin with CreateNewData and work our way down. We will provide the data (id, name, email, and password) we obtained from the server along with the timestamp to the CreateNewData method, which will produce an array of data (id, name, email, password and timestamp) that will be our block data.

4.2.2 ProofOfWork method on Client Side

Following on, we will begin the ProofOfWork method phase where the method will return a nonce value. In this method, we will produce a random number between 1-6

with the upper bound value being the generated random number after which we will enter a loop and begin creating hashes. The loop will be repeated until our hash value has the upper bound leading zeros. For instance, if our random number is 4, we will continue to generate hash values until they have four leading zeros, '0000'. Therefore, our hash should be 0000303030307261696d61726169333134676f6f676c6540676d6169-6c2e636f6d in hexadecimal, and the loop will continue until a hash with a leading four zero is formed, at which point the count value will grow. We will halt the loop when we have a hash with four leading zeros, and the count value will be our nonce.

4.2.3 HashBlock Method on Client Side

With the data and nonce calculated, we will pass these values to our HashBlock method on the Blockchain class. We will acquire a hash value created from our data and nonce using the above method. The calculated nonce will be our NonceClient and its subsequent hash will be our HashClient.

4.2.4 Api '/block'

Moving on to the next phase, we will send data from the client to the server using the api '/block'. The data will include id, name, email, password and the timestamp when the user first registered. Subsequently, we will carry out the same procedure as we did on the client side to generate NonceServer and HashServer on the server side using the Blockchain class on the server side.

4.2.5 CreateNewData Method on Server Side

First, we will send the data we received through the '/block' API to the create_new_data method of the Blockchain class, which will produce an array of data (id, name, email, password and timestamp) that will be our block data.

4.2.6 ProofOfWork Method on Server Side

Then we will generate the nonce using a random upper bound number between 1-6 that will be the leading zero. Using the same method of nonce generation, we will continue generating hash until we get a hash with a leading 'n' number of leading zeros, n being the random upper bound number between 1-6. As the hash is being generated while being on a loop, the count value will continue to increase until we find the hash with an 'n' number of leading zeros. The moment we get a hash with an 'n' number of leading zeros the loop will stop and the count value will be our Nonce. This will be our NonceServer.

4.2.7 HashBlock Method on Server Side

With the data and nonce server calculated, we will pass these values to our HashBlock method on the Blockchain class of the server side. We will obtain a hash value created from our data and nonce using the above method. The calculated nonce will be our NonceServer and its subsequent hash will be our HashServer.

4.2.8 Diffie Hellman Algorithm for NonceUnified Generation

We will use the Diffie-Hellman process for the same nonce on both the client and server sides. For the hash to be the same on both the client and server sides, all the parameters of the hash must be the same. The parameters of the hash method are:

- Previous Hash
- Data
- Nonce

The nonce is an integral part of the generation of hash. However, it is nearly impossible to generate the same nonce using a random upper bound on both the client and server sides. That's why, we will be using the Diffie-Hellman algorithm to generate the same nonce by using two pairs of keys, one, the Public Key, that will be exchanged between the client and server sides. Second, the private key will not be exchanged and will be kept private within the client and server side. The goal is to generate HashUnified on both ends- the server as well as the client; the NonceUnified and HashUnified on each side have to match for the user to be authenticated.

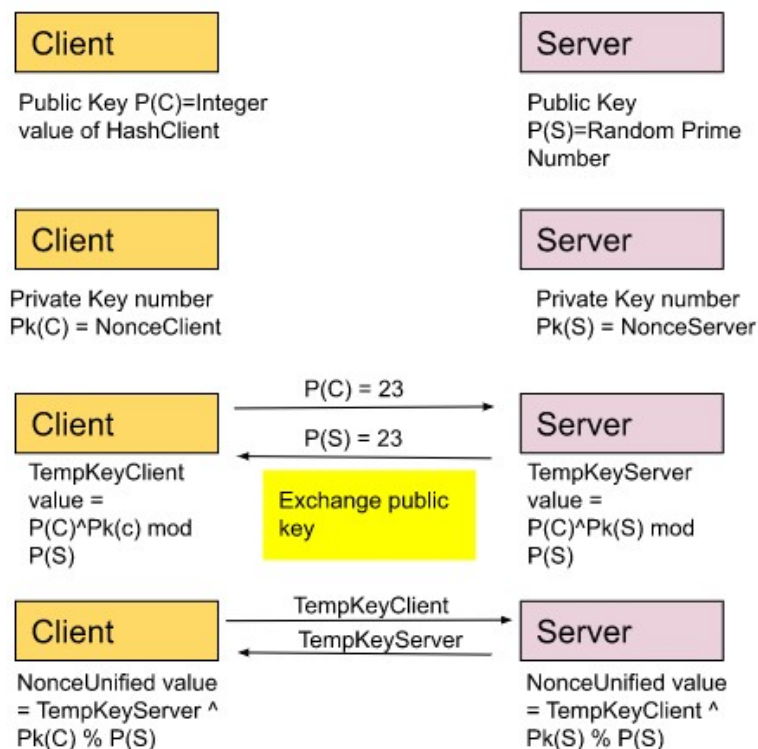


Figure 4.3: Diffie-Hellman Implementation

4.2.9 Api '/pks'

The server will generate a random prime number between 1 and 50 named PublicKeyServer and it will be sent to the client side using an API called '/pks'.

The HashClient will be converted to an integer (hexadecimal) value on the client side, which will be the PublicKeyClient.

The NonceClient that is generated using a random upper bound on the client side BlockChain class will be the private key for the client side. The NonceServer that is generated using a random upper bound on the server side BlockChain class will be the private key for the server side.

Now, the *TempKeyClient* will be calculated using the formula:

$$TempKeyClient = PublicKeyClient^{nonceClient} \text{ mod } PublicKeyServer \quad (4.1)$$

4.2.10 Api ‘/tks’

Following the successful calculation of TempKeyClient, we will create an API called ‘/tks,’ which will be used to send the following values through the API as requests to the server:

- UserID
- TempKeyClient
- HashClient
- PublicKeyServer

With every value ready, we can proceed to calculate the TempKeyServer on the server side using the formula:

$$TempKeyServer = PublicKeyClient^{nonceServer} \text{ mod } PublicKeyServer \quad (4.2)$$

Finally, we can calculate the NonceUnified value on the server side using the formula:

$$NonceUnified = TempKeyClient^{nonceServer} \text{ mod } PublicKeyServer \quad (4.3)$$

4.2.11 Previous Hash Determination

On the server side, we will be using the user_id to traverse through the database to check if the user_id already exists. If it does not exist, then on the server side, we will make the previous hash zero. However, if the id already exists, we will use the hash of the exact previous login as the previous hash from the database table. This can be verified by filtering through the BlockHash table user_id wise and we will use the subsequent hash of that particular user_id which had exactly the latest entry. On the client side we can locate if any previous hash exists by using ‘local-storage.getItem(Hash)’ command. In case of registration, the previous hash is set to be zero, however, in case of login, the previous hash will be retrieved from the local storage.

4.2.12 Hash Unified Generation

After determining our previous hash with `NonceUnified` being successfully calculated, we can generate our `HashUnified` by passing the following information to the `HashBlock` method on the `Blockchain` class on the server side:

- Previous Hash
- Data (id, name, email, password, timestamp)
- `NonceUnified`

As a return to the request of the API `/tks`, the server will send the value of `TempKeyServer` which we calculated previously, to the client side.

On the client side, the `NonceUnified` value will be calculated using the formula:

$$NonceUnified = TempKeyServer^{nonceClient} \text{ mod } PublicKeyServer \quad (4.4)$$

With `NonceUnified` being successfully calculated, we can generate our `HashUnified` by passing the following information to the `HashBlock` method on the `Blockchain` class on the client side:

- Previous Hash
- Data (id, name, email, password, timestamp)
- `NonceUnified`

Finally, if the `HashUnified` are the same on both the server side and client side, only then can we authenticate a user.

4.2.13 Hash Storage

On the client side, we are storing the `HashUnified` on the local storage with respect to the user id. And on the server side, the `HashUnified` along with the nonce, user id and data is stored on the database.

4.2.14 Logging in Again

On the next login, the user will have to authenticate in two steps. At first, the user will have to use the correct credentials to log in. Secondly, the entire procedure of `NonceUnified` and `HashUnified` will take place; although, this time the previous hash will not be zero. Only in the case of registration, the previous hash is set to be zero, however in the case of login, the previous hash will be the hash of the last login. On the client side, it is retrieved from the local storage. After a new log in the previous hash will be replaced with the new hash on the client side. However, on the server side, as the entire block is stored permanently in the database we will retrieve it by filtering out the exact last login's hash. If the `HashUnified` comes the same on both the server and client sides, then we can authenticate the user successfully and thus it fulfills the motive of strengthening password-based authentication.

4.2.15 Device Based Authentication

The idea is to build a device-based authentication system rather than a user-based one. As discussed before, each login will create one block and each time the user logs in, a chain of blocks is maintained in that particular device for that specific user. The login process was a success since it was done by a registered user of this device. We ensure that the device is a registered device by looking for the previous hash in the local storage with respect to the user id of the client's device. If there remains any hash in the local storage we will be able to deduce the device as a registered user, otherwise, the user will have to register first to be able to log in.

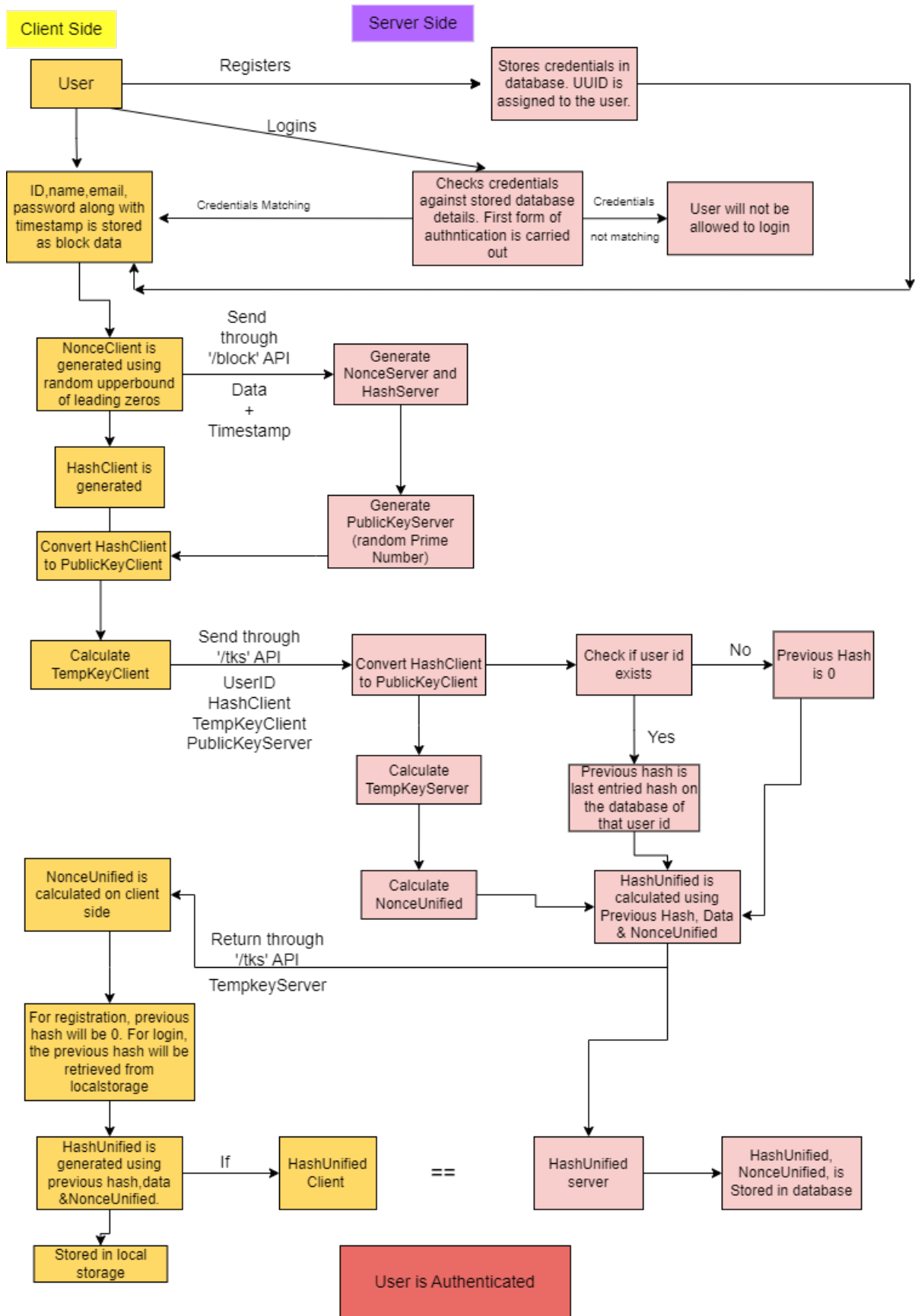


Figure 4.4: Workflow Diagram

Chapter 5

Implementation

In this paper, we created our own device-based blockchain authentication system where a login application is implemented along with its various functionalities where it is simultaneously communicating with the customized server end. The client-side has been coded using Javascript and its framework React as it is a web application. The server side has been programmed using Python and its framework Flask to enable smooth operation between API requests from both ends.

5.1 Client-Server Model

There are two parts to our model - one is the client, another is the server which can be further broken down into:

1. User Login Application
 - Registration
 - Login
 - Home Page
 - Home
2. API
 - Client
 - Server
3. Database
 - User Table
 - BlockHash Table
4. Blockchain Class
 - Client
 - Server

5.2 User Login Application

This is a web application on port 3000 where a user will be taken to a home page where they can either register or login.

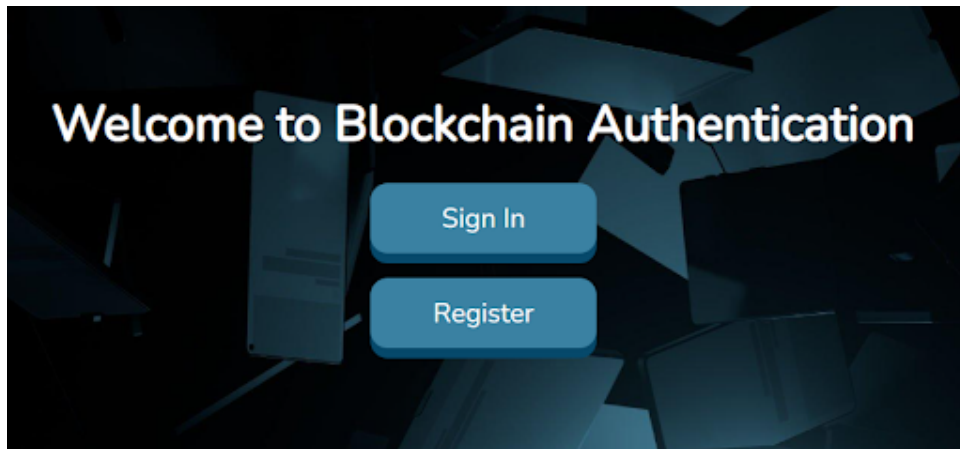


Figure 5.1: Home Page

When the user clicks on the 'Register' button, they are redirected to the registration page where they have to give their credentials: name, email, and password.

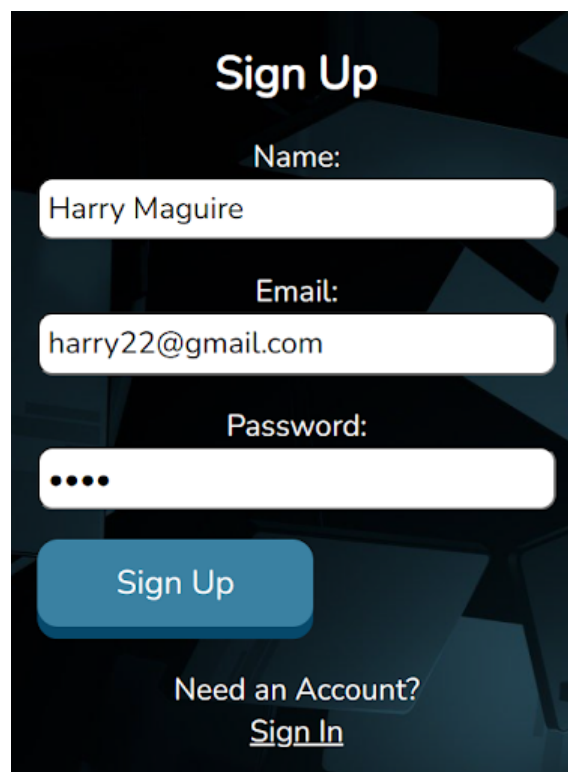
The image shows a registration form titled "Sign Up" in a white, sans-serif font. The form consists of three input fields, each with a label above it: "Name:" with the value "Harry Maguire", "Email:" with the value "harry22@gmail.com", and "Password:" with four dots indicating a masked password. Below the input fields is a blue, rounded rectangular button labeled "Sign Up". At the bottom of the form, there is a link that says "Need an Account? [Sign In](#)".

Figure 5.2: Registration Page

After the registration has been successfully completed, an API request will be sent to the server side where it will fetch the user information, and store it in its allocated database. An 'ID' will be generated, along with it, while storing the registration

	id	name	email	password
4e4	Filter	Filter	Filter	
1	4e400dab38354d9e8f3a96aea2dd3c98	Harry Maguire	harry22@gmail.com	\$2b\$12\$ozZmARTEc9KrrhxfFcF2e2j...

Figure 5.3: Server side database users table

credentials on the database, however, it will not store passwords in plain text but rather in hashed form using Bcrypt.

The entire registration data along with the timestamp will be sent to the method CreateNewData on the Blockchain class on both the server and client side to form an array of data.

Afterward, the nonce using a random upper bound will be generated along with its subsequent hash. We will call the generated nonce on the client side NonceClient and the generated hash HashClient. Similarly, we will call the generated nonce on the server side NonceServer and the generated hash HashServer.

```

The data is: {"id": "4e400dab38354d9e8f3a96aea2dd3c98", "name": "Harry Maguire", "email": "harry22@gmail.com", "password": "1234", "timestamp": 1673359525109}
The NonceClient using random upper bound is 1235
The HashClient using random upper bound nonce is 00d7d006ebf2c3f68efe2ce7557f5099e9c0daa9e3802e21062c4a1fa1af88af

```

Figure 5.4: Client side nonce and hash

```

{'id': '4e400dab38354d9e8f3a96aea2dd3c98', 'name': 'Harry Maguire', 'email': 'harry22@gmail.com', 'password': '1234', 'timestamp': 1673359525109}
NonceServer using random upper bound is 55
HashServer using random upper bound is 00f971eee6f02b1cff1ff34820a1b3cb6f744cfce2f2bb2c6bd9260ed8c79bec
127.0.0.1 - - [10/Jan/2023 20:05:25] "POST /block HTTP/1.1" 200 -
127.0.0.1 - - [10/Jan/2023 20:05:25] "OPTIONS /pks HTTP/1.1" 200 -
127.0.0.1 - - [10/Jan/2023 20:05:25] "GET /pks HTTP/1.1" 200 -
127.0.0.1 - - [10/Jan/2023 20:07:01] "OPTIONS /tk HTTP/1.1" 200 -
PublicKeyClient is 381307868775421539452738443677452373796954710163328988031409231938601322671
Nonce is 55
PublicKeyServer is 29
This is TempKeyClient dict {'number': [2], 'sign': 1, 'rest': 0}
[2]
TempKeyClient is 2
TempKeyServer is 19
NonceUnified on server side is 15
HashUnified on server side is 4b08b719ee23f38e80d8d2661a0697c2f5e0036ec48155e0519f8cd055fbadb

```

Figure 5.5: Server side nonce and hash

Now we can initiate our Diffie Hellman Key Exchange for NonceUnified generation on both the server and client side. Diffie hellman requires two pairs of keys on both sides, in our case the client side and the server side. On the client side, the public

key is found by finding the integer value of HashClient and the private key will be the NonceClient. For the server side, the public key is a random prime number (between 1-50) and the private key will be NonceServer.

```
46 PublicKeyClienthex = int(HashClient,16)
47 print("PublicKeyClient is ",PublicKeyClienthex)
```

Figure 5.6: Public key client side

```
108 @cross_origin
109 @app.route("/pks",methods =["GET"])
110 def get_public_key_server():
111     PublicKeyServer =sympy.randprime(1, 50)
```

Figure 5.7: Public key server-side

With two pairs of public key and private key, the TempKeyClient and TempKeyServer are to be calculated.

```
158
159     var TempKeyClient = BigInteger(NonceClient).mod(PublicKeyServer)
160
```

Figure 5.8: TempKeyClient calculation

```
53
54     TempKeyServer = PublicKeyClienthex ** nonceServer % PublicKeyServer
55
```

Figure 5.9: TempKeyServer calculation

The TempkeyClient and TempKeyServer are to be exchanged on both sides through the API '/tks', which eventually will be used to calculate the NonceUnified on both the client and server sides.

```

64
65     NonceUnified = TempKeyClient ** nonce % PublicKeyServer
66

```

Figure 5.10: NonceUnified calculation on the server side (python)

```

171
172     var NonceUnified = BigNumber(nonceone).mod(PublicKeyServer)
173

```

Figure 5.11: NonceUnified calculation on the client side (javascript)

The NonceUnified on the server side should match with the client side, which in turn will generate the same hash on both the client and server side, known as HashUnified. During registration, the previous hash is used as 0 on both the client and server sides.

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
{ 'id': '4e400dab38354d9e8f3a96aea2dd3c98', 'name': 'Harry Maguire', 'email': 'harry22@gmail.com', 'password': '1234', 'timestamp': 1673359525109}
NonceServer using random upper bound is 55
HashServer using random upper bound is 00f971eee6f02b1cff1ff34820a1b3cb6f744cfce2f2bb2c6bd9260ed8c79bec
127.0.0.1 - - [10/Jan/2023 20:05:25] "POST /block HTTP/1.1" 200 -
127.0.0.1 - - [10/Jan/2023 20:05:25] "OPTIONS /pks HTTP/1.1" 200 -
127.0.0.1 - - [10/Jan/2023 20:05:25] "GET /pks HTTP/1.1" 200 -
127.0.0.1 - - [10/Jan/2023 20:07:01] "OPTIONS /tkc HTTP/1.1" 200 -
PublicKeyClient is 381307868775421539452738443677452373796954710163328988031409231938601322671
Nonce is 55
PublicKeyServer is 29
This is TempKeyClient dict {'number': [2], 'sign': 1, 'rest': 0}
[2]
TempKeyClient is 2
TempKeyServer is 19
NonceUnified on server side is 15
HashUnified on server side is 4b08b719ee23f38e80d8d2661a0697c2f5e0036ec48155e0519f8cd055fbadbb

```

Figure 5.12: NonceUnified and HashUnified on server side

```

Welcome  Elements  Console  Sources  Network  >> +
top  Filter  Default levels  10  3 hidden
Public Key Client is 381307868775421539452738443677452373796954710163328988031409231938601322671  Api.js:54
Private key client is 1235  Api.js:57
(92108) [7, 5, 0, 8, 8, 4, 2, 0, 9, 0, 1, 9, 6, 3, 9, 4, 0, 4, 6, 0, 1, 8, 7, 5, 4, 8, 0, 5, 1, 5, 3, 9, 4, 2, 7, 8, 3, 3, 9, 4, 8, 9, 0, 8, 2, 3, 8, 3, 5, 8, 3, 0, 2, 5, 6, 5, 2, 1, 5, 7, 7, 9, 1, 8, 9, 2, 7, 5, 9, 3, 2, 6, 2, 6, 7, 6, 4, 8, 7, 6, 7, 8, 4, 3, 8, 9, 1, 7, 3, 3, 8, 5, 8, 2, 0, 3, 5, 7, 4, 0, ...]  Api.js:60
TempKeyclient is 2  Api.js:66
TempKeyServer on client is 19  Api.js:70
NonceUnified on client side is 15  Api.js:73
HashUnified on client side is 4b08b719ee23f38e80d8d2661a0697c2f5e0036ec48155e0519f8cd055fbadbb  Api.js:76

```

Figure 5.13: NonceUnified and HashUnified on client side

As it can be observed from the server side, the nonce and hash generated are the same as the ones on the client side after applying the Diffie-Hellman algorithm, we can successfully conclude that authentication is successful.

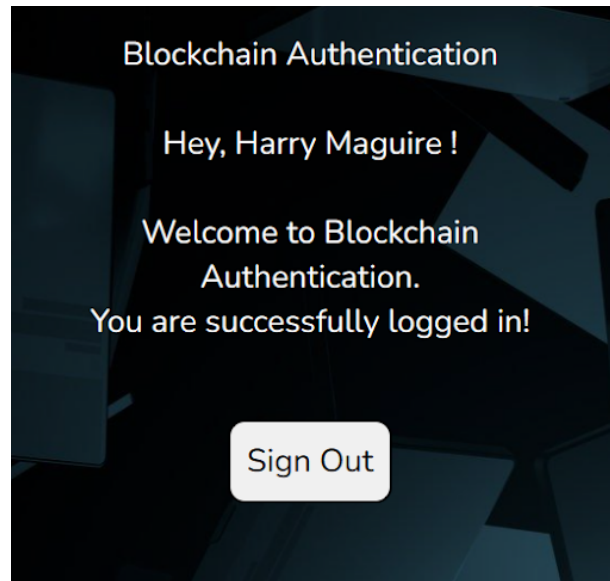


Figure 5.14: Redirected to home after successful authentication

The HashUnified generated from registration will be temporarily stored in the local storage of the client side's device, user-id wise. However, on the server side, the HashUnified along with NonceUnified, User-id and its time of entry will be stored in the database.

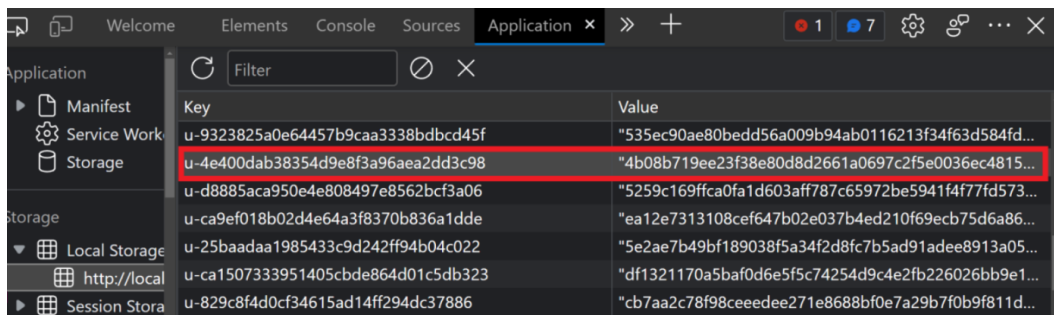


Figure 5.15: Client side local storage

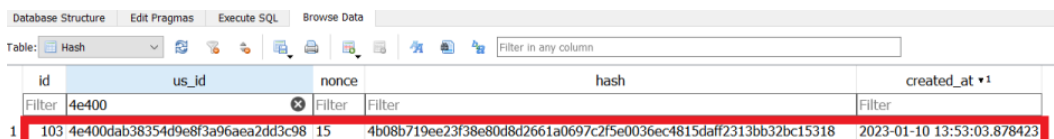


Figure 5.16: Server side Hash table

Our model focuses on per device per blockchain. So that particular user will be able to log in again using that same particular device, using his successfully authenticated registration credentials.

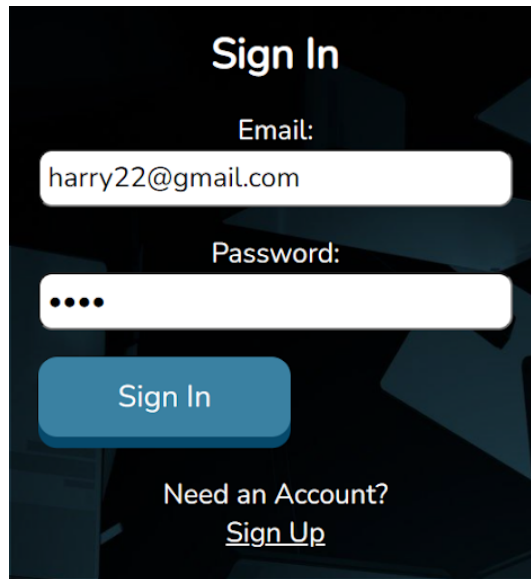


Figure 5.17: User logging in through the Sign In page

When logging in, the exact process of authentication, one using his credentials and another using blockchain to generate NonceUnified and HashUnified using the process of the Diffie-Hellman algorithm will occur. However, unlike in registration, the previous hash will not be zero. On the client side, the hash of that particular user id will be fetched from the local storage and on the server side, the last entry of that user will be filtered out from the database to be used as the previous hash.

```
let prevhash = JSON.parse(localStorage.getItem('u-${res.data.id}'))
```

Figure 5.18: Client Side previous hash retrieval

```
prevhash =  
BlockHash.query.filter_by(us_id=us_id).order_by(BlockHash.created_at.desc()).first()
```

Figure 5.19: Server-side previous hash retrieval

Upon successful login the newly generated HashUnified will replace the previous hash on the client side, however, on the server side, the newly generated hash along with Nonceunified and user id will be stored on a new store not replacing any previous entry.

Again, if the HashUnified and NonceUnified are the same on both the server and client side, the login will be successful allowing the user to go to the home.

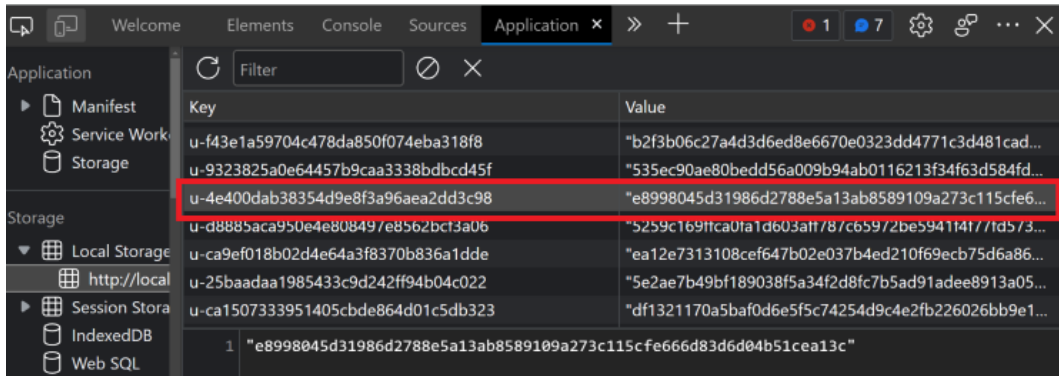


Figure 5.20: New hash generated on the client side local storage

The screenshot shows a database table named 'Hash'. The table has columns: id, us_id, nonce, hash, and created_at. A new entry is highlighted in red:

id	us_id	nonce	hash	created_at	
1	103	4e400dab38354d9e8f3a96aea2dd3c98	15	4b08b719ee23f38e80d8d2661a0697c2f5e0036ec4815daff2313bb32bc15318	2023-01-10 13:53:03.878423
2	136	4e400dab38354d9e8f3a96aea2dd3c98	1	e8998045d31986d2788e5a13ab8589109a273c115cfe666d83d6d04b51cea13c	2023-01-11 07:29:25.800555

Figure 5.21: Server side database Hash table showing the same updated hash

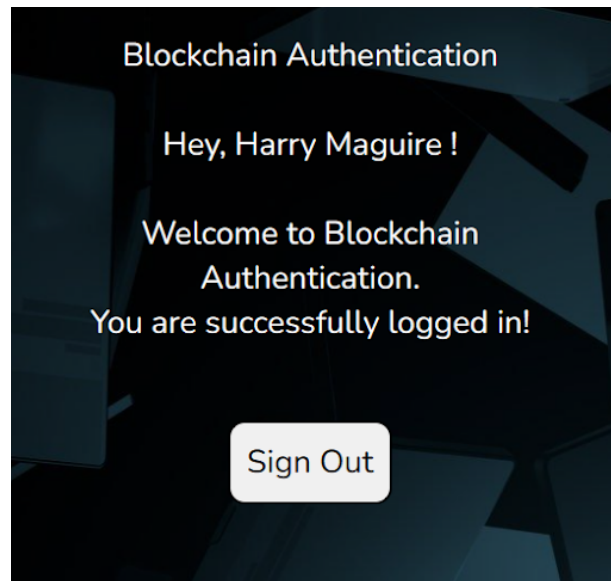


Figure 5.22: Redirected to the home after successful login authentication

5.3 API

5.3.1 Client Side

This is an integral part of our model which actively maintains the proper flow of work with the server side. It handles login and registration requests, pop-up messages, the blockchain mining process and the Diffie-Hellman algorithm.

```
export const register = async (user) => {
  try {
    const res = await axios.post(`${baseUrl}/register`, user, {withCredentials: true,
  });
}
export const login = async (user) => {
  try {
    const res = await axios.post(`${baseUrl}/login`, user, {
      withCredentials: true,
    });
  }
}
```

Figure 5.23: The register and login API sends post requests to the server side to register and log in respectively

During registration, the entire user data is sent as a request from the client side to the server side to be stored in the database and as a return, a user ID is assigned to that user. During login, the user input details are sent to the server side to verify the credentials. If it matches, the server returns the user_id along with the name to the client side. So that after successful authentication, we can welcome the user on the homepage by calling their name.

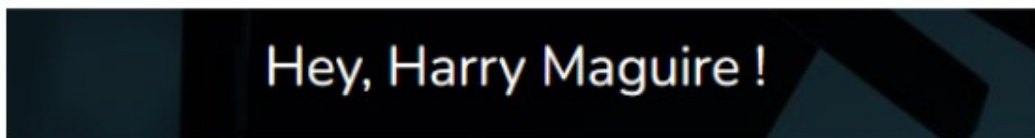


Figure 5.24: Authentication is successful

```
const datapass = await axios.post(`${baseUrl}/block`, {"data": data})
```

Figure 5.25: The '/block' API sends user data to the server side to generate server side's NonceServer and HashServer

```
var PublicKeyServer = await axios.get(`${baseUrl}/pks`)
```

Figure 5.26: The '/pks' receives the Public Key generated from the server side to be used for the Diffie-Hellman algorithm on the client side


```
var serversidevalue = await axios.post(`${baseUrl}/tkc`, {"tempkeyclient": TempKeyClient,
"hashclient": hash, "PublicKeyServer":PublicKeyServer, "us_id": res.data.id  })
```

Figure 5.27: Server Side Values

Finally, the ‘/tkc’ API sends the TempKeyClient it calculated along with the Hash-Client which is the Public Key for client-side, as well as the server’s Public Key so that the Diffie-Hellman algorithm can start on the server side. In return, the server returns its own TempKeyServer which is used to calculate the NonceUnified on the client side and finally the HashUnified on the client side.

5.3.2 Server Side

Similar to the client API, the server side handles cases such as registering new users to the server, not allowing anyone to log in with the wrong credentials or with no accounts, verifying login sessions, storing proper information in the database, and performing server-side Diffie-Hellman process along with blockchain mining.

The ‘/register’ and ‘/login’ routes registers and logs in users respectively according to their credentials. The register route stores new user credentials on the User database and the login route checks user credentials by iterating through the User table and verifies their credentials as a form of the first layer of authentication.

The ‘/me’ route shows currently logged-in users using the redis session.

The ‘/pks’ route generates a random prime number to be used as Public Key for the server side which is exchanged to the client side for the Diffie-Hellman algorithm.

```
@cross_origin
@app.route("/register", methods=["POST"])
@cross_origin
@app.route("/login",methods=["POST"])
```

Figure 5.28: The ‘/register’ and ‘/login’ routes

```
@cross_origin
@app.route("/me",methods=["GET"])
```

Figure 5.29: The ‘/me’ route

```
@cross_origin
@app.route("/pks",methods=["GET"])
```

Figure 5.30: The ‘/pks’ route

The ‘/block’ route takes user information from the client side and sends it to the CreateNewData method on the server side Blockchain class to form an array of data, as well as calls the method (ProofOfWork) for generating Nonce using random upper bound from Blockchain class. Additionally, it also calls the method (Hash-Block) to generate a HashServer using NonceServer.

```
@cross_origin
@app.route("/block",methods=["POST"])
```

Figure 5.31: The ‘/block’ route

The ‘/tks’ route is the most vital route for the server side as it conducts the Diffie-Hellman algorithm on the server side. This route accepts the Public Key from the client side and using the server-side private key and public key it generates Temp-KeyServer which in turn is used to calculate the NonceUnified on the server side and finally, the HashUnified. Finally, it stores the entire Blockchain data on the BlockHash table of the database.

```
@cross_origin
@app.route("/tks",methods=["POST"])
```

Figure 5.32: The ‘/tks’ route

5.4 Database

This is made on the server side where all the necessary information about users and their corresponding blocks are stored.

In the table Users, the columns are id, name, email and password. The id is auto-generated upon successful registration and a unique hexadecimal id is assigned to the user. Followed by successful registration, the user’s id, name, email and password are stored in the “**Users**” table.

After generating the unified nonce and hash on both sides, comes the part of storing them on the server and client side. On the server side, the foreign key relationship will be established using the ‘id’ of the ‘Users’ table. The same user id will be used on the BlockHash table for the column us_id.

The id of the BlockHash table is an auto-increment id for every entry followed by the columns nonce, hash and created_at, which will be the nonce unified, hash unified value for that correspondent us_id. The created_at is the timestamp to keep track of the very last entry of a particular user so that we can use the immediate last hash of the user as the previous hash for their current login session.

```

CREATE TABLE "users" (
  "id"    VARCHAR(32) NOT NULL,
  "name"  VARCHAR(345),
  "email" VARCHAR(345),
  "password" TEXT NOT NULL,
  UNIQUE("id"),
  PRIMARY KEY("id")
);

CREATE TABLE "Hash" (
  "id"    INTEGER NOT NULL,
  "us_id" TEXT,
  "nonce" TEXT NOT NULL,
  "hash"  TEXT NOT NULL,
  "created_at" TEXT NOT NULL,
  PRIMARY KEY("id" AUTOINCREMENT),
  FOREIGN KEY("us_id") REFERENCES "users"("id")
);

```

Figure 5.33: Database Table Creation

5.5 Blockchain

In order to ensure the chain of information is maintained, we have used the concepts of blockchain to make our system contain all login details of the device-used usage.

On both the client and server side, blockchain is utilized to generate the hash and nonce. The Blockchain class contains the following four methods:

- createNewData: We pass the data (id, name, password, email and password, timestamp) to this method which will return us its array which in turn is our block data.

```

1  method create_new_data(id,name,email,password,timestamp):
2      Form id,name,email,password,timestamp into an array of data
3      Append data into a variable called newData
4      return newData

```

Figure 5.34: create_new_data method

- proofOfWork: this is what we are using to get the nonce value. Here, a leading zero value is randomly generated where a hash generation loop will start. If we get a hash that matches the upper bound value, the loop stops and the nonce is the number of times the loop runs to find that particular hash.

```

1  method proofOfWork(self,prevhash,currentBlockData):
2      Nonce variable is initialized
3      Find limit by generating a random number from 0-6
4
5      method upperbound(digit,limit)
6          Run loop according to size
7          Concatenate digit the no.of times loop run
8          Return the concatenated digit
9      while loop:
10         Keep on generating hash until a hash is found with
11         Concatenated digit upper bound
12         Nonce incremented with each generation of hash
13
14     return Nonce

```

Figure 5.35: proofOfWork method

- hashBlock: With the data and nonce calculated, this is passed to the hash-Block method where the previous hash, nonce and data are concatenated and converted to sha256 code to produce the hash.

```

1  method hashBlock(self,prevhash,currentBlockData,nonce) :
2      Variable dataAsString = make prevhash,currentBlockData,nonce into
      string
3      Encode dataAsString into sha256 hash
4      return hash

```

Figure 5.36: hashBlock method

- createNewBlock: Now that we have the nonce, hash, previous hash, and data, this method creates a block every time a user logs in.

```

1  method create_block(self,nonce,prevhash,hash) :
2      newBlock= append index,data,nonce,prevhash,hash into the variable
3      append newBlock to previous block
4      return newBlock

```

Figure 5.37: create_block method

5.6 Scope For Performance Analysis

Currently, we are using our server to mine a new block as a user logs in with a registered device. Therefore, it does not extensively showcase the usage of Blockchain's P2P network distribution for mining a new block. On the server side, a Blockchain networking protocol, preferably Delegated Proof of Stake (DPoS) can be used. Using DPoS, we can choose which node(s) in the network can mine or verify the next block, for additional security on the server side.

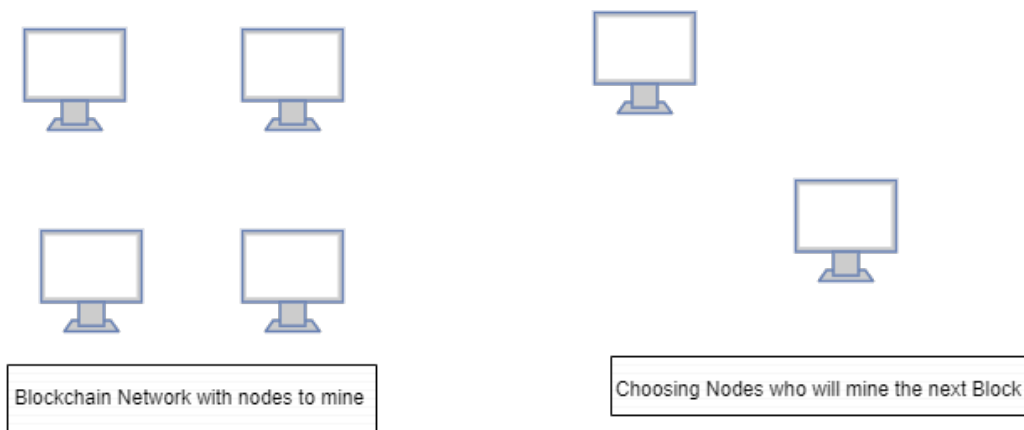


Figure 5.38: Implementation of DPoS

Furthermore, we are storing the block hash in the localstorage of the client's device. This can give rise to various phishing attacks or any form of malicious attacks. Therefore, the block hash in further encrypted form is to be stored in an offline database, that can be decrypted using a key. Thus it will lower the chances of exposing the entire hash in its raw format to any attacker, while verifying the device successfully.

Lastly, if a user forgets his password, each user has a chain of blocks in their respective registered device and the information of the entire chain can be backtracked to traverse to a specific block's hash, which can be used as an OTP to validate the user who forgot their password.

Chapter 6

Results Analysis

The past few decades have encountered numerous authentication mechanisms that can be classified into three main categories based on the fundamental factors of authentication- knowledge factor, inherence factor and possession factor. In knowledge factor authentication, the user is required to answer some question(s) and the system is designed in such a way, it assumes that only the valid user can answer the question correctly. The most prevalent use of this instantiation is passwords and Personal Identification Numbers (PINs). However, the crucial weaknesses of this approach include the use of awfully simple passwords that are easy to guess using brute force attack and not only has it shown success with encrypted but with decrypted passwords well. The use of strong passwords has diminished the success rate of brute force attack by 10% [37][38]. On the contrary, even well-crafted passwords can be hacked through malware (Trojan), keylogging, social engineering, packet sniffing, remote administrative tools, etc. Another major issue is the replication of passwords across widespread accounts and one single compromised account can make all the other accounts susceptible as well. The use of secret patterns is not too difficult to crack either as it leaves oily smudges on the screen which can be retrieved using high-resolution imagery. Biometrics is one of the implementations of inherence factor authentication as it is dependent upon behavioral or physical biometrics. This approach usually requires an enrollment process where multiple samples of the user's behavioral or physical traits are recorded. Later, when the user tries to authenticate himself, he has to provide the respective trait again and this data will be matched with the one saved during the enrollment procedure. Face recognition and fingerprint are the most widely used examples of biometrics. However, the main vulnerability of the fingerprint method is that this can be subverted by collecting the victim's fingerprint from any service that also requires his touch. A profound shortcoming of face recognition is, it can easily be broken using the victim's photograph which is easy to find on the internet through a basic social media search. Security tokens and MFA (multi-factor authentication) are the most customary examples of possession factor authentication; this type of authentication relies on some sort of hardware that will be in the possession of the legitimate user and the authentication will only be a success if it is confirmed by the user through some mechanism that he has possession of that physical hardware. OTP (one-time password) is an excellent example of 2FA; OTP SMS sent through the network can be intercepted and this process also increases the cost for the service provider [39].

Our model uses knowledge factor authentication in the first step as the user initially authenticates himself using a password. Even though we have taken the knowledge factor approach in our model, there is flexibility to either replace this initial step with an inherence factor or even add an inherence factor or possession factor authentication as the second step for added security. Hence, our goal is to strengthen first-factor authentication with an added layer of security. 2FA uses a second channel where the problem of network interception (middleman attacks) and a bad network with no access to OTP still exists. Each of the authentication methods discussed above either requires input from the user or dependence on a third party which is why we wanted to come up with a device-based secure method of authentication that will take place automatically right after initial verification. This method allows the device to recognize the authenticated user using the previous hash stored on the client side; hence, even if the attacker knows the user's credentials, he won't be able to log in from any other device. To sum up, by using a combination of email and password credentials along with device information, the proposed solution aims to provide a more secure and robust authentication process without involving input from the user or third parties. The user would be granted access once the hash of the device matches the hash stored in the blockchain server. Therefore, the traditional first-factor authentication system is strengthened. This ensures that even if the hacker has the ability to crack the MFA like biometrics and security tokens, they first need to break the more secure first line of defense, which is the traditional password authentication system based on blockchain [39].

Blockchain implemented for authentication purposes has shown great potential for user identification, smart home systems, health-care records, educational certificates and IoT devices. One prominent usage of blockchain that has been employed for authentication of IoT devices is Hammi et al.'s (2018, September) proposed model, where the authors designed a novel system that is decentralized, named 'Bubbles of Trust' [40]. The model addresses device authentication and identification and also utilizes the security benefits of blockchain to establish specific zones called "bubbles" where only trusted devices can authenticate each other. It uses smart contracts and public blockchain to let only authenticated devices operate in these zones. However, in our proposed model we are utilizing a permissioned blockchain where only authorized users will be allowed to use our device-based authentication system as our model was created to facilitate the increased security of governmental organizations. We are also not using any smart contracts here; that is because in smart contracts, once a process has been validated, it is very difficult to change it, and the procedure is very time-consuming and costly as well. We have created our own environment as in cases where a user needs to change their credentials, they can do so easily without the hassle of smart contracts. The model only allows devices to operate in specific hubs, which to an extent reduces accessibility but our model will allow devices to operate in any place as long as it is a registered authenticated device.

Secondly, another effective implementation of blockchain has been seen in two-factor authentication (2FA). The generic two-factor authentication uses a centralized entity that sends secret information (tokens) to the users each time the users attempt to access the system. Despite the additional security layer given by two-factor authentication, its colossal drawback is its full dependency upon the third party for

providing the secret codes. However, in terms of security, this type of authentication system is always at risk of being compromised. In this paper, the writers have proposed a novel approach to strengthening the security of the traditional 2FA system by building 2FA over a blockchain platform; the platform is distributed in nature and fully eliminates the dependency on third parties. The model uses the concept of a Blockchain wallet for user information encryption. When users create an account, they will be offered a public and a private key. Their public key may be shared among other users to communicate, however, the private key is to be kept safe and private. When the user logs in after they are authenticated, their user ID, i.e, the public key, along with their private key will be used to authorize what they will be able to access through their login information. To generate a pair of private keys, they used the ECC algorithm. They will be using ECC because key generation from ECC is mathematically more difficult and thus becomes more difficult to decrypt. ECC uses less space and generates keys faster so we can manage large amounts of key generation for users at a faster time and the key length being necessarily smaller will help save more data, which is why ECC is a faster and more secure approach. After the authentication, comes authorization. The categorization of entities has been done in 3 different parts: user, staff and servers.

They all have a private and public key pair; these keys contain corresponding special Ethereum addresses that are related to them and serve as a distinctive identifier representing these categories. First and foremost, the staff will register all the servers and users in the smart contract on the basis of their respective Ethereum addresses. As the user is registered, that person can form a request which will give him permission to access the smart contract's registered servers. According to the staff's decision, these requests can be accepted or neglected. A user can generate an arbitrary six-digit numerical OTP which will be kept in the blockchain system if his/her request is confirmed and will be hashed with the SHA3 algorithm. This piece of information is available in all nodes as the blockchain system is synced with every single node, if a user later attempts to connect to a certain server via SSH and launch into the identical OTP, the server's PAM module queries this blockchain for every corresponding hash tokens and searches the hash of input OTP. If identified, user entrance is granted, otherwise, the connection is closed [1]. In our model, when users create an account, a genesis block is mined on both the server and client sides which contrasts with this model; that is because, in this model, the users will be offered a public and private key during the registration process. Therefore, they have taken an encryption-based approach in the registration process while we have used a core blockchain-based approach. Furthermore, their login system uses generic credential-based authentication in the first factor and the encryption keys are used in the authorization phase, unlike our model. Our login process is much more secure because when the registered user tries to log in again, in the first step of the credential-based authentication system the registered user is verified using the user ID and hence the system knows that the HashUnified on the client side for this user will be the previous hash value for the new block that is being mined for this user upon the login. This is how a blockchain is maintained in our model where the genesis block is the registration or the first block and the first login block is the second block, the second login is the third block and so on. On the other hand, our use of the Diffie-Hellman algorithm eliminated the need for sending

sensitive information over the network but in their model, they have used the ECC algorithm in the authentication phase but the key is being sent over the network which makes it prone to middle-man attacks. Moreover, the implementation of blockchain technology does not start until the authorization process in their model; our model uses core blockchain concepts in the authentication phase instead of the authorization. We have implemented blockchain in the first factor whereas, they have used it in the second factor. As mentioned earlier, we have created our own blockchain environment instead of using smart contracts and Ethereum. We have not used Ethereum due to its issues with scaling since it has a ledger, a platform for smart contracts, etc, all of these can lead to malfunctions, hacks and errors. Furthermore, it is risky to invest in Ethereum as the price of ether has significantly changed over the past. Lastly, our model offers scalability because a per device per chain approach allows each device to have its own unique chain, which can help to prevent issues with scalability as the number of users and devices increases. Our model is also very flexible as it allows the creation of custom chains for each device, rather than having a single chain for all users.

Chapter 7

Conclusion and Future Work

In this paper, we devised a model that enables a user logging into any system to be authenticated using Blockchain methodology with full-proof security. A lot of secured login systems exist today, but those are still prone to cyber-attacks due to dependence on third-party applications. In order to ensure credentials are not stolen and reduce reliance on mediators, we have developed a model where a user login session is verified if and only if the nonce and hash from the client side and Blockchain server side are the same. We extensively discussed how this method works and have shown how our techniques add an enhanced layer of authentication to an existing system in terms of reliability of proper security. We have included fully labeled diagrams to demonstrate the flow of the procedure on how the client and server sides are concurrently communicating with one another.

Although our model implements per device per blockchain, so one user can have only one registered device under their credentials. However, in the future, we are aiming to build a model where one user will be able to login on multiple devices. As a result our model will still be per device per blockchain, but it will allow one user to have multiple blockchains. The user data of the devices' block will be the same to validate the users, however, their previous hash will not be the same to maintain separate Blockchains per device.

Bibliography

- [1] V. Amrutiya, S. Jhamb, P. Priyadarshi, and A. Bhatia, “Trustless two-factor authentication using smart contracts in blockchains,” in *2019 international conference on information networking (ICOIN)*, IEEE, 2019, pp. 66–71.
- [2] M. Jakobsson, “Two-factor inauthentication—the rise in sms phishing attacks,” *Computer Fraud & Security*, vol. 2018, no. 6, pp. 6–8, 2018.
- [3] V. Papaspirou, M. Papathanasaki, L. Maglaras, *et al.*, “Cybersecurity revisited: Honeytokens meet google authenticator,” *arXiv preprint arXiv:2112.08431*, 2021.
- [4] D. Berdik, S. Otoum, N. Schmidt, D. Porter, and Y. Jararweh, “A survey on blockchain for information systems management and security,” *Information Processing & Management*, vol. 58, no. 1, p. 102397, 2021.
- [5] C. Lin, D. He, X. Huang, K.-K. R. Choo, and A. V. Vasilakos, “Bsein: A blockchain-based secure mutual authentication with fine-grained access control system for industry 4.0,” *Journal of network and computer applications*, vol. 116, pp. 42–52, 2018.
- [6] H. Kopp, D. Mödinger, F. Hauck, F. Kargl, and C. Bösch, “Design of a privacy-preserving decentralized file storage with financial incentives,” in *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, IEEE, 2017, pp. 14–22.
- [7] X. Liang, S. Shetty, D. Tosh, C. Kamhoua, K. Kwiat, and L. Njilla, “Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability,” in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, IEEE, 2017, pp. 468–477.
- [8] G. Zyskind, O. Nathan, *et al.*, “Decentralizing privacy: Using blockchain to protect personal data,” in *2015 IEEE Security and Privacy Workshops*, IEEE, 2015, pp. 180–184.
- [9] S. Azouvi, M. Al-Bassam, and S. Meiklejohn, “Who am i? secure identity registration on distributed ledgers,” in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, Springer, 2017, pp. 373–389.
- [10] N. More and D. Motwani, “A blockchain-based decentralized framework for crowdsourcing,” in *Image Processing and Capsule Networks: ICIPCN 2020*, Springer, 2021, pp. 448–460.
- [11] E. E. Nwabueze, I. Obioha, O. Onuoha, *et al.*, “Enhancing multi-factor authentication in modern computing,” *Communications and Network*, vol. 6, no. 03, p. 172, 2017.

- [12] J. Mann, *Mobile two factor authentication*, 2016.
- [13] X. Boyen, “Hidden credential retrieval from a reusable password,” in *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, 2009, pp. 228–238.
- [14] N. Gunson, D. Marshall, H. Morton, and M. Jack, “User perceptions of security and usability of single-factor and two-factor authentication in automated telephone banking,” *Computers & Security*, vol. 30, no. 4, pp. 208–220, 2011.
- [15] D. İşler and A. Küpçü, “Threshold single password authentication,” in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, Springer, 2017, pp. 143–162.
- [16] U. Rahardja, A. N. Hidayanto, P. O. H. Putra, and M. Hardini, “Immutable ubiquitous digital certificate authentication using blockchain protocol,” *Journal of applied research and technology*, vol. 19, no. 4, pp. 308–321, 2021.
- [17] U. Rahardja, Q. Aini, Y. Graha, and A. Khoirunisa, “Implementation of gamification into management of education for motivating learners,” in *Proceeding Interuniversity Forum for Strengthening Academic Competency*, vol. 1, 2019, pp. 209–209.
- [18] L. Sellami, K. Sellami, and P. F. Tiako, “Efficient management of security for supporting intrusion detection in ubiquitous and pervasive environments,” *Procedia Computer Science*, vol. 155, pp. 402–409, 2019.
- [19] S. Joshi, S. Stalin, P. K. Shukla, *et al.*, “Unified authentication and access control for future mobile communication-based lightweight iot systems using blockchain,” *Wireless Communications and Mobile Computing*, vol. 2021, 2021.
- [20] Y. Chen and C. Bellavitis, “Blockchain disruption and decentralized finance: The rise of decentralized business models,” *Journal of Business Venturing Insights*, vol. 13, e00151, 2020.
- [21] P. Kamboj, M. C. Trivedi, V. K. Yadav, and V. K. Singh, “Detection techniques of ddos attacks: A survey,” in *2017 4th IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics (UPCON)*, IEEE, 2017, pp. 675–679.
- [22] K. Scholer, “An introduction to bitcoin and blockchain technology,” *Kaye Scholer LLP*, pp. 3–22, 2016.
- [23] P. Kamboj, S. Khare, and S. Pal, “User authentication using blockchain based smart contract in role-based access control,” *Peer-to-Peer Networking and Applications*, vol. 14, no. 5, pp. 2961–2976, 2021.
- [24] A. Mukherjee, M. Balachandra, C. Pujari, S. Tiwari, A. Nayar, and S. R. Payyavula, “Unified smart home resource access along with authentication using blockchain technology,” *Global Transitions Proceedings*, vol. 2, no. 1, pp. 29–34, 2021.
- [25] A. Yazdinejad, G. Srivastava, R. M. Parizi, A. Dehghantanha, K.-K. R. Choo, and M. Aledhari, “Decentralized authentication of distributed patients in hospital networks using blockchain,” *IEEE journal of biomedical and health informatics*, vol. 24, no. 8, pp. 2146–2156, 2020.

- [26] A. Mubarakali, “An efficient authentication scheme using blockchain technology for wireless sensor networks,” *Wireless Personal Communications*, vol. 127, no. 1, pp. 255–269, 2022.
- [27] A. A.-N. Patwary, A. Fu, S. K. Battula, R. K. Naha, S. Garg, and A. Mahanti, “Fogauthchain: A secure location-based authentication scheme in fog computing environments using blockchain,” *Computer Communications*, vol. 162, pp. 212–224, 2020.
- [28] Z. Li, D. Wang, and E. Morais, “Quantum-safe round-optimal password authentication for mobile devices,” *IEEE transactions on dependable and secure computing*, 2020.
- [29] R. Sandhu and P. Samarati, “Authentication, access control, and audit,” *ACM Computing Surveys (CSUR)*, vol. 28, no. 1, pp. 241–243, 1996.
- [30] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-An, and H. Ye, “Significant permission identification for machine-learning-based android malware detection,” *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3216–3225, 2018.
- [31] S. H. Islam, M. K. Khan, and A. M. Al-Khourri, “Anonymous and provably secure certificateless multireceiver encryption without bilinear pairing,” *Security and communication networks*, vol. 8, no. 13, pp. 2214–2231, 2015.
- [32] D. P. Oyinloye, J. S. Teh, N. Jamil, and M. Alawida, “Blockchain consensus: An overview of alternative protocols,” *Symmetry*, vol. 13, no. 8, p. 1363, 2021.
- [33] *Intro to ethereum — ethereum.org*, <https://ethereum.org/en/developers/docs/intro-to-ethereum/>, (Accessed on 01/23/2023).
- [34] A. Narayanan and J. Clark, “Bitcoin’s academic pedigree,” *Communications of the ACM*, vol. 60, no. 12, pp. 36–45, 2017.
- [35] A. Albarqi, E. Alzaid, F. Al Ghamdi, S. Asiri, J. Kar, *et al.*, “Public key infrastructure: A survey,” *Journal of Information Security*, vol. 6, no. 01, p. 31, 2014.
- [36] *What is elliptic curve cryptography? definition & faqs — avi networks*, <https://avinetworks.com/glossary/elliptic-curve-cryptography/>, (Accessed on 01/23/2023).
- [37] N. L. Clarke and S. M. Furnell, “Authentication of users on mobile telephones—a survey of attitudes and practices,” *Computers & Security*, vol. 24, no. 7, pp. 519–527, 2005.
- [38] A. Vance, “If your password is 123456, just make it hackme,” *The New York Times*, vol. 20, A1, 2010.
- [39] S. W. Shah and S. S. Kanhere, “Recent trends in user authentication—a survey,” *IEEE access*, vol. 7, pp. 112 505–112 519, 2019.
- [40] M. T. Hammi, B. Hammi, P. Bellot, and A. Serhrouchni, “Bubbles of trust: A decentralized blockchain-based authentication system for iot,” *Computers & Security*, vol. 78, pp. 126–142, 2018.