Faculty of Computer & Informatics & Eng.

Department of Networks & Communication Eng.

Blockchain-based Cryptocurrency Exchange Platform: Design and Implementation

A report submitted in partial fulfillment of the requirements for the degree of Bachelor's Degree in Communication and Network Engineering

by

Baraa Saif **Abdulilah Khalif**

Supervised by

Dr. Eng. Talal Hammoud

SPU

2024-2025

Supervisor Certification

I certify that the preparation of this project entitled “Blockchain-based Cryptocurrency Exchange”, prepared by

Baraa Saif and Abdulilah Khalif, was made

under my supervision at Department of Networks & Communication Engineering / Faculty of Computer & Informatics & Eng.

of the requirements of the Degree of

Bachelors Networks & Communication Engineering.

Supervisor

Dr. Talal Hammoud

Signature

Acknowledgments

We take this opportunity to express gratitude to all of the Department faculty members for their help and support. We also thank our parents for the unceasing encouragement, support and attention. we are also grateful to my brothers and sisters who supported us through this venture. We place on record, our sincere thanks to Dr. Talal Hammoud for the continuous encouragement, supervision, patience and advices.

Abstract

This project represents a comprehensive exploration of blockchain technology and its application in creating secure and efficient systems for digital asset management. By leveraging cryptographic techniques such as hashing algorithms and digital signatures, the platform ensures the integrity and authenticity of transactions. The use of Python and Flask for the backend, combined with HTML and Bootstrap for the frontend, results in a robust and user-friendly interface. This project underscores the potential of blockchain technology to revolutionize the way we manage digital assets and conduct secure transactions. By combining theoretical knowledge with practical implementation, the project demonstrates the feasibility and benefits of blockchain-based systems in real-world applications.

Contents

Abstract i

General introduction ii

List of Figures vi

List of Abbreviations vii

Chapter 1: Cyber Security

1.1. Introduction 2

1.2. Cybersecurity Fundamentals 3

1.3. Cybercrime 4

1.3.1. Types of cybercrime 5

Chapter 2: Blockchain

2.1. Introduction 11

2.2. Blockchain architecture 12

2.3. Chain of blocks 13

2.4. Applications of Blockchains 16

Chapter 3: Digital Signature

3.1. Introduction 30

3.2. Standardization 30

3.3. Digital signature process 31

3.4. Secure Hash Algorithms 34

3.5. History of Hash Signature Standard 35

3.5.1. Secure Hash Algorithm (SHA-1) 38

3.5.2. Secure Hash Algorithms (SHA-224/ SHA-256) 42

3.5.3. Secure Hash Algorithms (SHA-384/ SHA-512) 46

Chapter 4: Cryptocurrency Platform Design

4.1. Introduction 54

4.2. Platform Structure 55

4.2.1. __init__.py 55

4.2.2. models.py 56

4.2.3. wallet.py 56

4.2.4. blockchain.py 57

4.2.5. auth.py 57

4.2.6. views.py 57

4.2.7. How the Files Work Together 58

4.3. Python code 59

4.3.1. Auth.py 59

4.3.2. Views.py 64

4.4. Front-end Design 71

4.4.1. Layout page 72

4.4.2. Index page 73

4.4.3. New_wallet page 76

4.4.4. Transaction page 78

4.4.5. Users page 80

4.4.6. User_info page 82

Conclusion 85

Future works 86

References 90

List of Figures

Figure 1-1: Types of cybercrime 5

Figure 1-2: Phishing phases 6

Figure 2-1: Centralized and decentralized architecture. 12

Figure 2-2: Simplified example of a blockchain 13

Figure 2-3: Architecture of a block in blockchain 15

Figure 2-4: Blockchain-based transaction process 16

Figure 2-5: Blockchain architecture with IoT 18

Figure 2-6: Blockchain-based RIM and membership management 19

Figure 2-7: LoC architecture 25

Figure 2-8: Healthcare transactions using blockchain 27

Figure 2-9: Finance transactions using blockchain 28

Figure 3-1: Digital signature block diagram 32

Figure 3-2: Hash function 34

Figure 3-3: The i^{th} cycle of SHA-1 calculation 42

Figure 3-4: The i^{th} cycle of SHA-224/SHA-256 calculation 46

Figure 3-5: The i^{th} cycle of SHA-384/SHA-512 calculation 51

Figure 4-1: Index Page 76

Figure 4-2: New_wallet Page 78

Figure 4-3: New_Transaction Page 80

Figure 4-4: Users Page 82

Figure 4-5: Users Page 84

List of Abbreviations

DoS	Denial of Service
OpSec	Operational Security
DDoS	Distributed DoS
MITM	Man in the Middle Attack
WAP	Wireless Access Point
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
SSL	Secure Sockets Layer
PKI	Public Key Infrastructure
ITU	International Telecommunication Union
CA	Certificate Authority
RCA	Root Certificate Authority
RA	Registration Authority
VA	Validation Authority
CRL	Certificate Revocation List
PoW	Proof of Work
PoS	Proof of Stake
PBFT	Practical Byzantine Fault Tolerance
DPoS	Delegated Proof of Stake
PoB	Proof of Bandwidth
PoET	Proof of Elapsed Time
PoA	Proof of Authority
NIST	National Institute of Standards and Technology
SHS	Secure Hash Standard
FIPS	Federal Information Processing Standards
SHA	Secure Hash Algorithm

General Introduction

In today's digital age, cybersecurity has become a cornerstone of modern technology. With the increasing reliance on digital systems for financial transactions, data storage, and communication, ensuring the security and integrity of these systems is paramount. Cyberattacks, such as data breaches, identity theft, and unauthorized access, pose significant threats to individuals, businesses, and governments. Among the various technologies that have emerged to address these challenges, blockchain stands out as a revolutionary innovation. Blockchain technology offers a decentralized, transparent, and tamper-proof system for storing and verifying data, making it an ideal solution for enhancing cybersecurity. Blockchain-based applications are rapidly growing in popularity across industries, from finance and healthcare to supply chain management and voting systems. The inherent characteristics of blockchain—such as immutability, transparency, and decentralization—make it particularly well-suited for securing sensitive information and preventing fraud. Cryptocurrency exchange platforms, which rely on blockchain technology, exemplify how this innovation can be leveraged to create secure and efficient systems for managing digital assets. This project aims to design and develop a secure platform for cryptocurrency transactions using blockchain technology. The platform will demonstrate the practical application of blockchain in creating a robust and user-friendly system for managing digital wallets, performing transactions, and maintaining a transparent ledger of all activities. The project is organized into four chapters, each focusing on a specific aspect of the study:

Chapter 1: provides an overview of cybersecurity, highlighting its significance in protecting digital systems and data. It explores common cyber threats and discusses how blockchain technology can mitigate these risks by providing a secure and decentralized framework for data management.

Chapter 2: delves into the fundamentals of blockchain technology, including its architecture, consensus mechanisms, and key features such as immutability and decentralization. It also examines the diverse applications of blockchain across various industries, emphasizing its role in enhancing security and transparency.

Chapter 3: focuses on the technical aspects of blockchain, specifically hashing algorithms and digital signatures. These cryptographic tools are essential for ensuring the integrity and authenticity of transactions within a blockchain network. The chapter explains how these mechanisms work and their importance in securing blockchain-based systems.

Chapter 4: is the final chapter presents the design and implementation of the cryptocurrency exchange platform. The backend of the platform is developed using Python and Flask, while the frontend is built using HTML and Bootstrap. This chapter provides a detailed explanation of the code, including the creation of user accounts, wallet management, transaction processing, and the display of blockchain content. The platform demonstrates the successful execution of cryptocurrency transfers between users, showcasing the practical application of blockchain technology.

Chapter 1

Cyber Security

1.1. Introduction

Cyber security refers to the strategies used to safeguard a user's cyber environment. This environment encompasses the person, their devices, networks, applications, and any software, among other things. Cyber security is a subset of computer security that deals with the internet. The main security goal is to project the device using various rules and to set up various safeguards to protect it from online attacks. The technique of protecting computers, servers, mobile devices, electronic systems, networks, and data from harmful technology security or electronic information security is known as cyber security [1]. The phrase is used in a range of contexts, ranging from business to mobile computing, and it may be broken down into a few categories. Since the advent of the Internet and the digital change that has occurred in recent years, the concept of cybersecurity has become a common topic in both our professional and personal life. For the last 50 years of technological advancement, cybersecurity and cyber threats have remained constant. Until the invention of the Internet in the 1970s and 1980s, computer security was primarily relegated to academia, where, with growing connectivity, computer viruses and network intrusions began to take off. The 2000s saw the institutionalization of cyber risks and cybersecurity, following the rise of viruses in the 1990s. Cloud computing, mobile computing, E-commerce, online banking, and other cutting-edge technologies all require a high level of security. Because these technologies contain sensitive information about a person, their security has become a priority. Enhancing cyber security and safeguarding important information infrastructures are critical to the security and economic well-being of any country. Making the Internet safer (and protecting Internet users) has become a key component of both new service development and government regulation. The fight against

cybercrime necessitates a more comprehensive and secure strategy. Given that technical measures alone cannot prevent any crime, it is vital that law enforcement agencies be given the tools they need to properly investigate and prosecute cybercrime. Many countries and governments are enacting severe cyber security legislation nowadays [1].

1.2. Cybersecurity Fundamentals

Cybersecurity refers to the practice of protecting systems, networks, and data from digital attacks. These cyberattacks aim to access, change, or destroy sensitive information, extort money from users, or disrupt normal business processes. The main goal of cybersecurity is to ensure the confidentiality, integrity, and availability of data and systems [2].

1. **Confidentiality**: Ensuring that sensitive information is only accessible to authorized individuals.
2. **Integrity**: Protecting data from being altered or tampered with by unauthorized parties.
3. **Availability**: Ensuring that systems, data, and services are available to authorized users when needed.

In today's digital age, every organization, government, and individual rely on technology for communication, commerce, and storage of sensitive information. Cybersecurity is essential for safeguarding this data from various threats. Without proper security measures, businesses and individuals are vulnerable to financial losses, reputational damage, and legal penalties. Cybersecurity can be applied in different fields such as [2]:

- **Network Security**: Protecting internal networks from unauthorized access or attacks.
- **Cloud Security**: Securing data stored in cloud environments.

- **Application Security**: Ensuring that applications are designed and developed with security in mind.
- **Endpoint Security**: Protecting devices such as laptops, smartphones, and desktops from cyber threats.
- **Operational Security (OpSec)**: Protecting internal operations, such as critical systems and processes.

1.3. Cybercrime

Cybercrime or computer crime is a criminal activity that involves unlawful access to computer systems. It is an illegal activity committed over the internet. The word "cyber" is slang word means anything relating to computers, Information Technology, and virtual reality. Therefore, is stans to reason that "cybercrime" are offences relating to computers, Information Technology, the internet, and virtual reality. Crime committed using a computer and the internet to steal data and information is a cybercrime. Such as [1-3]:

- Stealing credit card information.
- Breaking into the government website
- Email and Internet fraud.
- Identity fraud.
- Theft and sale of corporate data.
- Ransomware attacks.
- Cyberextortion (demanding money to prevent a threatened attack).
- Cyber Spying (where hackers access government or company data).
- Cryptojacking (where hackers mine cryptocurrency using the resources they do not own).

1.3.1. Types of cybercrime



Figure 1-1: Types of cybercrime

- **Malware Attacks:**

It is an attack where a computer system or network is infected with a computer virus or other type of malware. It is an all-encompassing term for a variety of cyberattacks including trojan viruses. It is defined as code with malicious intent that typically steals data or destroys something on the computer. A famous example of a malware attack is the WannaCry ransomware attack, a global cybercrime committed in May 2017. When the WannaCry ransomware attack hit, 230,000 computers were affected across 150 countries. Users were locked out of their files and sent a message demanding that they pay a BitCoin ransom to regain access. Many cyber criminals use computer viruses to gain unauthorized access to systems and steal data [4]. A computer virus is a malware (malicious software program) loaded into a computer without the knowledge of the user. Like its biological name says viruses attach themselves to clean files and infect other clean files and they can spread uncontrollably damaging a system's core functionality and deleting or corrupting files. They usually appear as an executable file that you might have downloaded from the

internet. Trojan malwares disguise itself as legitimate software or is included in legitimate software that can be tampered with. It tends to act discretely and creates backdoors in our security to let other malware enter our system. Worms infect entire networks of devices either local or across the internet by using the network interfaces. It uses each consecutive infected machine to infect more. Botnets are networks of infected computers that are made to work together under the controller of an attacker [1].

- **Phishing:**

It is a cybercrime where people are contacted through phone calls, email, or a message by cybercriminals posing as a person from a legitimate institution. A phishing campaign is when spam emails, or other forms of communication, are sent emails, to trick recipients into doing something that undermines the security or security of the organization they work for. These cybercriminals collect personal information like bank account details and passwords and then steal money [1].

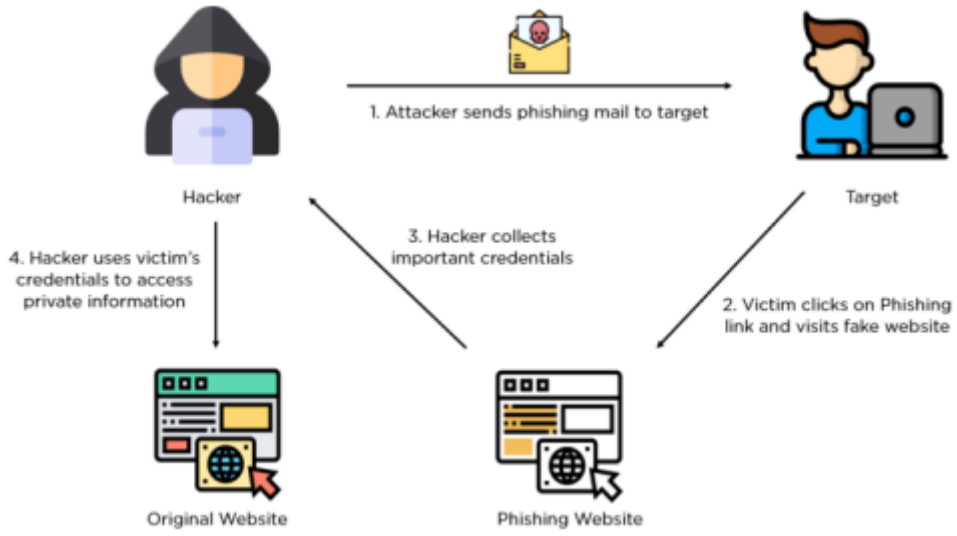


Figure 1-2: Phishing phases

Messages sent by phishing look authentic and attempt to get victims to reveal their information. The attacker must decide which business to target

and determine how to get the email address of the customers of that business. Then they go through a setup phase; once they know which business to spoof and who their victims are attackers create methods for delivering the messages and collecting the information; then they execute the attack. After that attacker records the information, the victims enter into the webpage or pop-up windows and in the last step which is identity theft and fraud the attacker uses the information they have gathered to make illegal purchases or commit fraud [1].

- **Password Attack:**

It is an attempt to obtain or decrypt a user's password for illegal use. Hackers can use cracking programs, dictionary attacks, and password sniffers in password attacks. Defense against password attacks is rather limited but usually consists of a password policy including a minimum length, unrecognizable words, and frequent changes. This attack can be done for several reasons but the most malicious reason is to gain unauthorized access to a computer without the computer's owner's awareness not being in place; so, this results in cybercrime such as stealing passwords to access bank information. There are three common methods used to break into a password-protected system [2].

1) **Brute-force attack:** In this, a hacker uses a computer program or script to try to log in with possible password combinations usually starting with the easiest to guess password.
2) **Dictionary attacks:** In this, a hacker uses a program or script and tries to log in by cycling through the combinations of common words. This attack tries only those possibilities which are most likely to succeed; typically derived from a list of words; for example, dictionary. These attacks are more successful because people tend to choose easy passwords like their names, birthdates, etc [1-2].

3) **Keylogger attacks:** In this, the hacker uses a program to track all of the user's keystrokes; so, at the end of the day, everything the user has typed including the login IDs and passwords has been recorded.

- **Distributed DOS Attack:**

Distributed DoS (DDoS) attacks are a type of cybercrime attack that cybercriminals use to bring down a system or network. Sometimes connected IoT devices are used to launch DDoS attacks. In this attacker sends a high volume of data of traffic through the network until the network becomes overloaded and can no longer function

- **Man in the Middle Attack (MITM):**

This attack can obtain information from the end-user and the entity he or she is communicating with. For example, if you are banking online the man in the middle would communicate with you by impersonating your bank and communicate with your bank by impersonating you. The man in the middle would receive all of the information transferred between both parties which could include sensitive data such as bank accounts and personal information. MITM can be prevented by [3]:

1) Use encrypted WAP (Wireless Access Point)
2) Always check the security of your connection. (HTTPS or HSTS)
3) Invest in VPN.

- **Drive-by Download:**

This attack occurs when vulnerable computers get infected by just visiting a website. Findings from the latest Microsoft Security Intelligence Report reveal that this attack has become the top web security threat to worry about [1].

- **Rogue software:**

This is also called rogue security. It is designed specifically to damage or disrupt a computer system. In this case, not only is the software going to disrupt your system, it's going to try and trick you into making a purchase using your credit card. Other ways to fool the people are as follows [2]:

1) **Hacking:** This is one form of cybercrime in which you get into somebody's system and then try to use information, disrupt the working, disrupt the network, etc.
2) **Credit card fraud:** The most common fraud performed under cybercrime is credit card fraud which happens over the internet, through the call centers, and lot many ways.
3) **Virus dissemination:** Installing, sending viruses via the network, vis emails, via messages, etc. is one of the more common types of cybercrime.
4) **computer vandalism:** is getting a lot of people involved nowadays.
5) **Software piracy:** The unauthorized copying, distribution, or use of the software is called software piracy. Cybercriminals distribute pirated software which causes loss to the software company. Southeast Asia is considered one of the bigger markets of software piracy.
6) **Identity theft:** It is a cybercrime where cybercriminals steal personal data like passwords or bank account details.
7) **Cyber bullying:** It is a form of online harassment of a person using smartphones or computers. It is also known as cyber harassment or online bullying. Cyberbullying generally occurs on platforms like social media and gaming platforms. It involves posting hate comments and sharing negative information about a person [1].

Chapter 2

Blockchain

2.1. Introduction

The blockchain has been around for over a decade as a proven technology for recording transactions in a decentralized, peer-to-peer network using a distributed database. It is considered a distributed computing paradigm that solves the confidence in a single entity problem. Therefore, several nodes work together in a blockchain network to securely and reliably maintain a distributed ledger of all past transactions. Satoshi Nakamoto launched Bitcoin in 2008, which was the first suggested cryptocurrency to present the blockchain as a distributed infrastructure platform. It made it possible for anyone to send and receive bitcoins, a kind of cryptocurrency, without needing to depend on any central authority. Other blockchain-based systems have also been suggested for usage with the coin, including Hyperledger Fabric and Ethereum. The blockchain is described as the underlying technology of Bitcoin and other cryptocurrencies - a shared digital ledger or a continuously updated record of all transactions. Blockchain may be considered both an economic and technological breakthrough. It provides a solution to any issue requiring a trustworthy ledger in a decentralized setting where not all participants, whether human or computer, can be trusted completely [5]. The blockchain is a collection of cryptographic techniques and protocols used by a network of nodes that cooperate to accomplish the safe recording of data inside a distributed database that consists of encrypted blocks encapsulating the data. There have been three major milestones in the evolution of blockchain technology so far: the use of digital money in the 1.0 stage, the use of smart contracts in the 2.0 stage, and the creation of programmable blockchains in the 3.0 stage. In its present phase of development, blockchain is mostly employed for small-scale local applications; there are very few industry-

or ecosystem-level uses for it. Nevertheless, the distinctive characteristics of blockchain have begun to extend across several sectors [5].

2.2. Blockchain architecture

The most common architectures for software systems are the centralized and the decentralized structure (Figure 2-1). In a centralized architecture, all components are connected to a central component. The central component can be thought of as an intermediary, like a bank or a lawyer. In decentralized architecture, which is equivalent to a distributed system, there is no central component. That means that, even though the components are connected to at least one other component, no component is directly connected to all the others and coordinates or controls the system. There is no intermediary. Peer-to-peer systems have a distributed architecture [6].

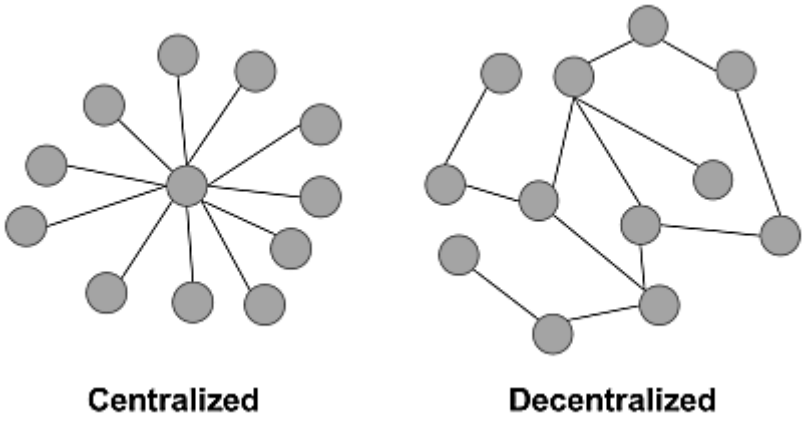


Figure 2-1: Centralized and decentralized architecture.

In a peer-to-peer network, there are nodes (individual computers) that make their computer's resources available to the network, and although there may be differences in the resources provided, all nodes have the same rights and responsibilities. As shown in the introductory example with the book, each node in the distributed ledger has an exact copy of the blockchain. Peer-to-peer interactions can replace intermediaries, called

disintermediation [5-6]. The blockchain can thus be described as a distributed ledger managed by the peer-to-peer network. Its potential of disintermediation of whole industries while preserving integrity is one of the main factors driving the hype around the technology. An advantage of the peer-to-peer network is that it can easily be modified, as new nodes can be added or eliminated without any great effort. Now that it has been clarified what a peer-to-peer network with distributed architecture is, the following section explains how the blockchain achieves and maintains integrity and trust in this system of peers, whose trustworthiness is unknown.

2.3. Chain of blocks

As the name suggests, blockchain is a chain of blocks. In one block, a number of transactions are collected and stored. The blocks are connected using cryptography (so-called hash functions). A block consists of a block header and a block body. The detailed structure of a block will be explained later; for now, it is sufficient to visualize the blockchain as in Figure 2-2. This chain of connected blocks is stored by every node in the distributed network [6].

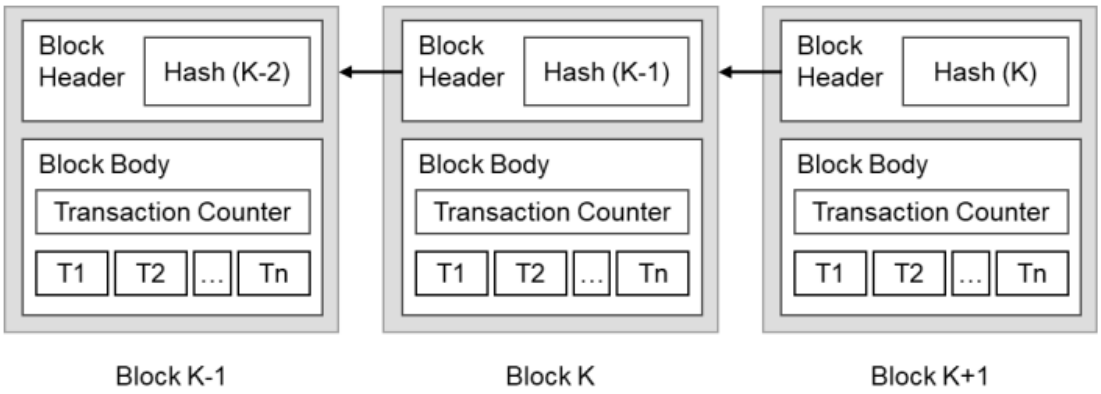


Figure 2-2: Simplified example of a blockchain [6]

The transactions can be much more than the transfer of money. Examples of transactions can be "Johanna sent Josefine 3 Bitcoin", but also "Company X78F received 3 tons of oranges from supplier T12Y", or "Pablo drove from point A to point B with the electric scooter." For a better understanding of the following explanations, it should already be mentioned that different types of blockchain have evolved over the years: Public blockchains, like Bitcoin, are permissionless and can be accessed by anyone, permissioned private blockchains that are used by a single organization and therefore the identities of the nodes are known, and permissioned consortium blockchains, a specialized private blockchain that is used by multiple organizations. Each block has a header and a body. The header contains a hash value and a hash reference that points to the hash of the previous block, and as each hash reference of each block points to the block generated before it, this sets up the chain between blocks. All transaction types of blocks are recorded in a ledger that is shared by all the connected nodes in the network [5]. A transaction is confirmed by the nodes only when a block is added. A consensus protocol needs to be verified and maintained for each block. Because many nodes or computers are connected as a chain and each node has a copy of the main chain, the information cannot beeasily accessed by hackers. If hackers want to break a block, they must break the hash reference that is pointing to the previous hash. Because of the protective processes embedded in blockchain technology, breaking the chain is impossible at present. Participants manage the blockchain using matching mechanisms such as Proof-of-Work (PoF), Proof-of-Elapsed Time (PoET), or Proof-of-Stake (PoS). Figure 2-3 shows the Blockchain architecture, and Figure 2-4 shows how transactions are managed in blockchain. There are two types of blockchain technologies [7]:

- **Public blockchain:** It is decentralized. Users can put data into it and view the data, and it is open to all. No permission is needed to access the blocks.
- **Private blockchain:** Only legal users can access the information in the blockchain, and they must maintaintheir authentication and control of the access.

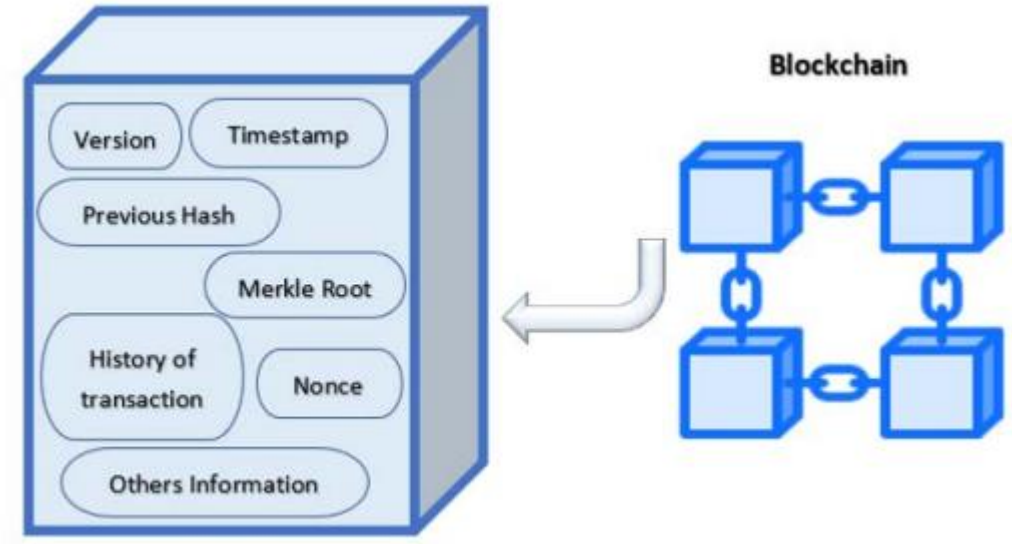


Figure 2-3: Architecture of a block in blockchain

There are three phases in creating a blockchain:

- **Blockchain 1.0**: Digital currency, such as Bitcoin, is the first production application of blockchain.
- **Blockchain 2.0**: Refers to economic and financial applications such as Ethereum.
- **Blockchain 3.0**: Refers to applications related to the digital society, such as education, healthcare, and gov-ernment, where money is not involved.

From the perspective of the value factor and maturity, there are four different phases of blockchain technologies, blockchain 1.0, blockchain 2.0, blockchain 3.0, and blockchain 4.0 [8]. The application area of

blockchain 1.0 is related to digital payment systems and currency transfer and remittance, which are mainly transaction oriented. Bitcoin is an example of blockchain 1.0. A smart contract is an example of blockchain 2.0; it provides value in the area of privacy. An opensource software platform called a decentralized application (dApp) uses blockchain 3.0 and is a platform on which application developers can conduct their transactions. Blockchain 4.0 is considered an emerging technology that uses a decentralized artificial intelligence (AI) system driven by autonomous decision making [8].

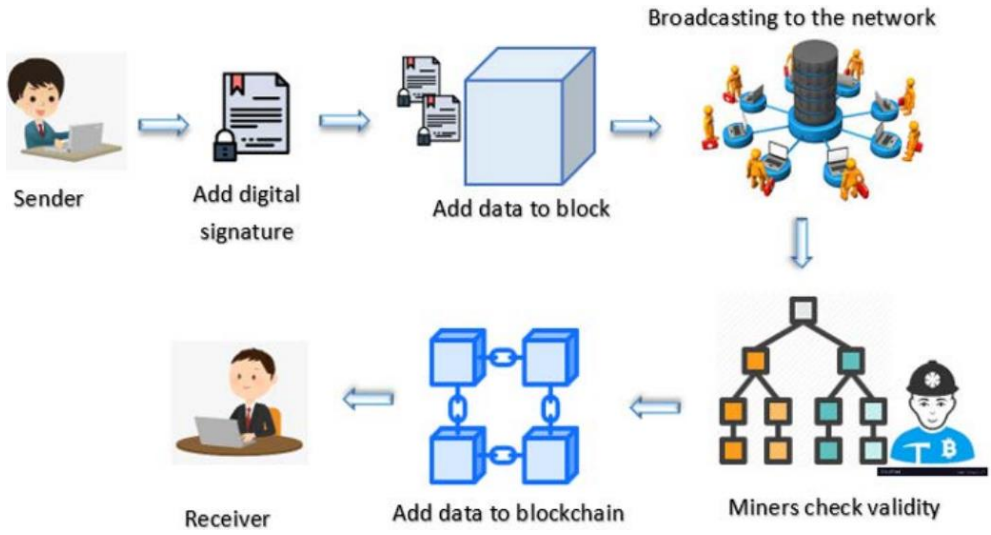


Figure 2-4: Blockchain-based transaction process

2.4. Applications of Blockchains

This section explains in detail the applications in which blockchain has been used in recent years and the areas in which blockchain can be used in the future [9].

A. Blockchain In Transportation

The intelligent transportation system (ITS) is a result of the growth and development of technologic innovations. The use of a smart framework in transportation can save human, financial, and material resources, and it can

also immensely improve the processes of automation and traffic management. Traditional vehicular communication systems are vulnerable to breaches of privacy, which can compromise security messages containing personal information, such as the identity of the driver of a vehicle, type of vehicle and area in which it is located. A well-known method for managing this problem was the use of a pseudonym that would conceal this information. Numerous articles have shown that the cost of maintaining these pseudonyms may increase and that the distribution of certificates will also be difficult. Blockchain's attributes of distribution and decen-tralization can help to minimize these limitations. In a blockchain-based ITS users were to share the traffic conditions for a specific location and then suggest a better route for other users. This sharing could become flawed if security issues arose. For some specific applications, mobile payment systems already exist in many countries. All transport-related payments, such as ticket purchases for buses or light rail transit, vehicle registrations, and parking fees, can be managed using blockchain technology. This reassures drivers and passengers so they do not have to worry much about trading their credits. This can open new opportunities for enhancing transportation scenarios based on a smart contract scheme. In supporting unmanned aerial vehicles and connected electric vehicles, an important issue is how to exchange energy [10]. A blockchain-based energy exchange method has been proposed to support a secure and energy-efficient transportation system. Blockchain is used to approve the energy requests of electronic vehicles in a delegated manner, inwhich themining node verifies the validity of each request. In addition, a software-defined backbone controller is used. For smart transportation, many tools and techniques have been considered for obtaining real-time responses, such as [9]:

route distribution, time management, environmental challenges, and safety management. Blockchain technology facilitates themanagement of these tasks. In vehicular communication systems (VCSs), the number of nodes in the network can be high. Thus, the security key management process is time consuming and requires a central administrator. By using blockchain with ITS, this key management proce-dure can be performed dynamically without a central author-ity. A secure key management procedure using blockchain technology is proposed for an ITSwith different performance results [11].

B. Blockchain in IoT

In recent years, IoT-based applications have gained popularity in various fields such as smart cities, healthcare, education, government, and social applications. In IoT-based networks, many datasets are available publicly for all users. Blockchain is used to guarantee the privacy and integrity of these shared data sets. The graph in Figure 2-5 shows an overview of how blockchain is combined with IoT [10].

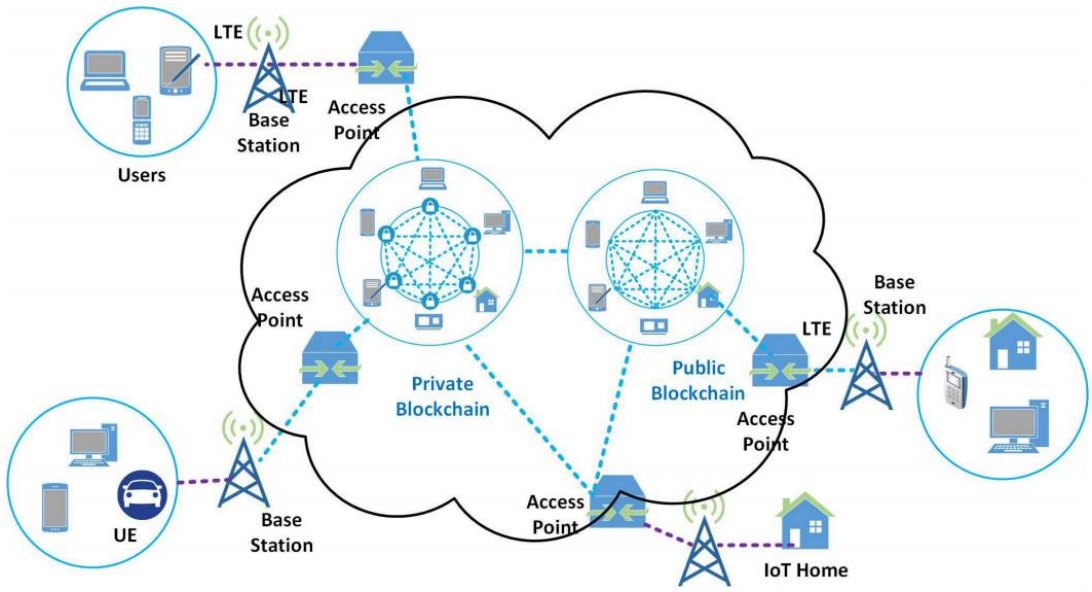


Figure 2-5: Blockchain architecture with IoT

Sensors are connected with many devices or nodes in the IoT network. A huge volume of data is captured on the IoT platform by a cloud service. Many nodes are connected with a gateway in small networks. In a large network, many nodes are also connected with gateways based on the cluster. Each node contains a pair of public and private keys. Every node in the network uses its public key at the time of registration and creates a digital profile record on the blockchain. When a node receives a transaction, the private key is used for creating a digital signature, which is verified by the gateway [9]. Different IoT devices are connected with the blockchain to synchronize and maintain a protocol for interacting with the blockchain network. There are two protocols for the synchronization and control of traffic between IoT and blockchain networks. They analyzed the bandwidth requirement and time needed to synchronize them. Using blockchain, a Reference Integrity Metric (RIM) is maintained to ensure the integrity of the dataset; the process is shown in Figure 2-6.

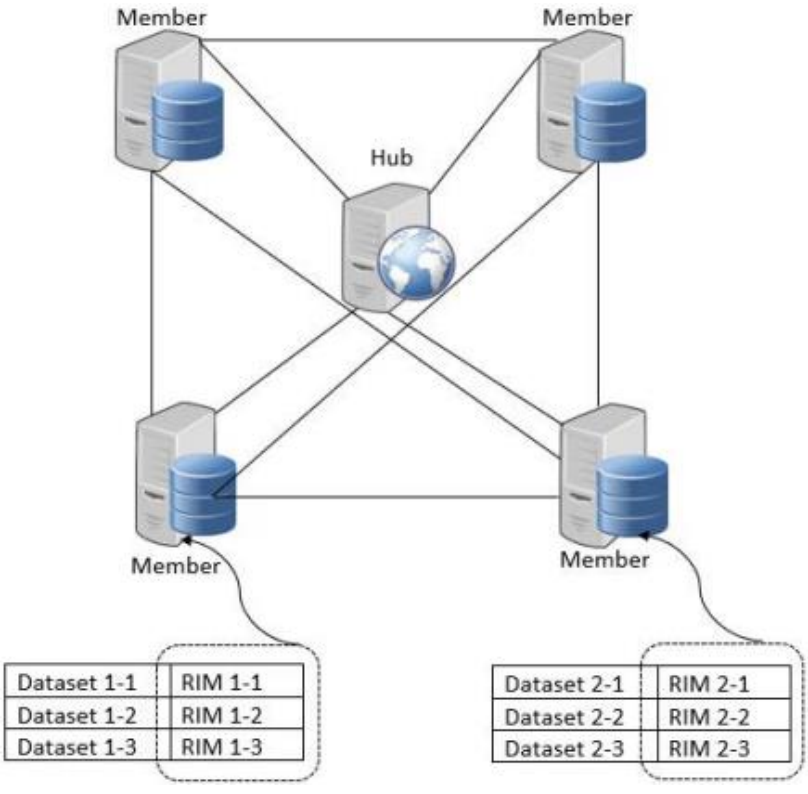


Figure 2-6: Blockchain-based RIM and membership management

a model for determining the optimal number of blocks has been proposed. For sending blocks in IoT-based applications, they developed an algorithm using hashmaps, which makes monitoring IoT networks faster and more reliable. For the proper implementation and use of blockchain technology, many questions need to be answered by managers, entrepreneurs, and business owners. A smart city is a diverse IoT-based network system that offers several applications and security solutions to citizens. Smart cities rely on the assembly, analysis, and digitalization of information. To solve single-point failures of IoT devices in smart city applications, Vivekanandan et al. proposed a blockchain-based authentication protocol. In their protocol, a private blockchain is used to register the IoT devices instead of a Registration Center Authority (RAC) [11].

C. Industrial Sector

Blockchain and IoT have opened numerous new opportunities and provided hope for improved productivity, efficiency, and transparency in the industrial sector. IoT provides real-time data by using sensors. Because the prices of sensors are falling day by day, sensors are becoming affordable for many industries. Blockchain is combinedwith IoT for sharing real-time data among users in a decentralized and distributed manner. The supply chain is an area in which many business problems occur, such as late deliveries, absent suppliers. There is a lot of paperwork involved in the shipping procedures for supplies. In addition, there are many losses of supplies and delays in deliveries of them. These problems can be resolved by using blockchain, which removes the dependency on an intermediary [10]. IoT devices can be connected to components or products, and the blockchain captures the data from these devices. Using blockchain, the location of the shipping container and the time stamp of the transaction can be captured. This elimi-nates the need for paperwork,

and delays can be minimized [12]. Digitization creates more opportunities for many companies and drives the supply chain. The use of technology creates not only opportunities but also manages complex procedures for following various rules. The transport providers receive the information and send a response to the company. The latter selects the best provider, enters into an agreement with that provider with the help of an agreement node, and thereby forms a smart contract with the provider. The company can enhance its product shipping platform by using packages. The autonomous vehicle is an attractive technology that may offer benefits for years to come. Sensors are attached to vehicles and all of their information is captured on the IoT platform, which is connected to a blockchain [9]. The data can show when a car needs to be refueled or repaired due to an engine breakdown or other problems. Because the blockchain keeps a permanent record of each transaction, the trust between the manufacturer and the consumer increases. To ensure data integrity and interconnectivity, a trust mechanism must be established for end users in IoT applications. The data layer consists of collected instances from IoT platforms and other sources. These data are hashed by a cryptographic function and stored in many blocks that are concatenated in the blockchain layer. This layer maintains the chain and records every transaction that interacts with the application layer. The responsibility of the application layer is to process the data and provide services to the end users. In a gateway, many sensors are connected and a sensor node can communicate with neighboring sensors for trustworthiness. When a node records a transaction, the gateway accepts it and creates a new block after authentication. Trustworthiness can be achieved by recalculating the value by verifying the data in the block. Smart agriculture scheme was proposed to improve the network's lifetime. Food traceability can help to ensure food safety; to do this, information is collected about the entire life cycle of food, such as

production, cultivation, and storage. In the smart agriculture ecosystem, a food traceability system based on blockchain and IoT has been proposed. The use of smart technologies reduces the human involvement in recording and verifying food production. Nowadays, there are many problems related to human health because of unhealthy food [10].

D. Blockchain In Financial Management

Each financial system can provide its services to thousands of users by regularly carrying out many transactions. Increasingly, however, these services have to face economic crime. In addition, their providers face increased regulation costs and their consumers incur additional costs. They have tomaintain all types of transactions using centralized databases and face many types of attacks [9]. Blockchain is considered a solution to this whole problem. This technology has some attractive features such as decentralized and distributed databases, P2P communication and transparency, and the keeping of a permanent record of each transaction. Explaining the working procedures of blockchain technologies and discussed two types of these technologies, public and private. They illustrated how blockchain affects the nine elements of an established framework of a business model. They collected information on this procedure from countries in North America and Europe and from South Africa, where blockchain technologies are implemented. They cited several obstacles, such as high costs due to the require-ment for dedicated developers and complex integration [12]. In a period of greater risks and uncertainties, blockchain architecture and potential remedies were proposed to overcome the challenges in the business sector and increase supply chain resilience. In this proposed architecture, five modules, data source, transaction, block creation, consensus, and connection and interface, were discussed. The difference between riskmanagement using

blockchain and conventional risk management was evaluated. A blockchain-based trading platform scenario was proposed for financial product lifecycle management. In this architecture, two financial institutions were responsible for product management and each institution had three departments where all nodes were connected to a P2P network. The proposed business network included product creation, modification, and processing and basic operations. After receiving a certificate from each institution, the institutions were allowed to create and modify the product. The verification and execution of transactions were handled by the other departments. The different types of blockchain application domains and IoT networks in the financial sectors are described as following [9]:

The banking sector can create several blockchain-based applications that can reduce transaction and intermediary costs by $20 billion. Several cloud-based startup compa-nies have been created to provide users with all the services of the banking system, such as access to regular and sav-ings accounts and to applications for loans and other financial products. After two years of research and development, another banking system based on the distributed ledger of blockchain called Corda has been created. This can provide banking services to users without intermediaries such as bank offices. Some global banks are using the blockchain system to provide secure, efficient operations and safe banking services to users [10].

The consulting firm Capgemini estimates that up to $16 billion in banking and insurance fees could be saved by using blockchain-based applications. Blockchain does not need an intermediary tomaintain the system. This can reduce the cost of intermediaries. Six blockchain-based services are being considered for financial management systems around the world. The available blockchain technology services in the world that reduce transaction costs are listed in Table 2-1.

Table 2-1: Established blockchain-based transaction services around the world

Name	Country	Service
Align Commerce	USA	Enabling payments to businesses in local currencies
Rebit	Philippines	Money transfer system for immigrants
Abra	USA	Mobile application-based money transfer system; can be downloaded from the Google Play Store.
Bitspark	Hong Kong	Remittance platform
BitPesa	Africa	Money transfer system
Slock	Italy	Universal Sharing Network (USN)
CoinRip	Singapore	Money transfer system

A decentralized supply chain management system as a blockchain-based token recipe is proposed for product source traceability and its revolutionary process, and a prototype implementation is evaluated in terms of smart contracts. Each token contains one flexible size of goods that can be evaluated in terms of weight, item, size, and volume. A new token can be created from multiple tokens. Contract owners can create, merge, split, transfer, and consume batches of tokens. During token production, the number of units and components for a given batch can be determined using a designed algorithm. Some found goods with similar qualities that should be used for production are identified by the certificate contract. The traceability process is described using the example of the wood industry. A set of solid contracts is used to achieve traceability in the supply chain, and the Ethereum Request for Comments (ERC) 721 interface is implemented to achieve compatibility. Tokens are created for all produced batches of wood, which are approved by resource suppliers or testifiers [10].

In the current loan management system, data protection for transparent transactions is not as strong. To minimize this problem, a financial loan management system called loan on blockchain (LoC) based on a smart contract has been proposed. The architecture and process of the LoC system are shown in Figure 2-7. To secure the transaction using LoC, the designers generated a digital account to automate the execution using the generated chain code. A digital signature is provided for the validity of the loan. The LoC system has different components, including the roles of the

participants (e.g., finance department, bank, customer). They are linked to the fabric SDK node in the system, which is connected to the participants by channels. Member-shipmanagement provides enrollment services for each node. The channels are also responsible for broadcasting services for themessages and the ordering platform that incrementally creates transaction blocks. There are several channels, each of which is associated with a single peer called a chaincode in the smart contract implementation. The ordering platform sends the order update status in the formof blocks (containing the hash value of the previous blocks and other information about the block) to the chaincode through the channels [11].

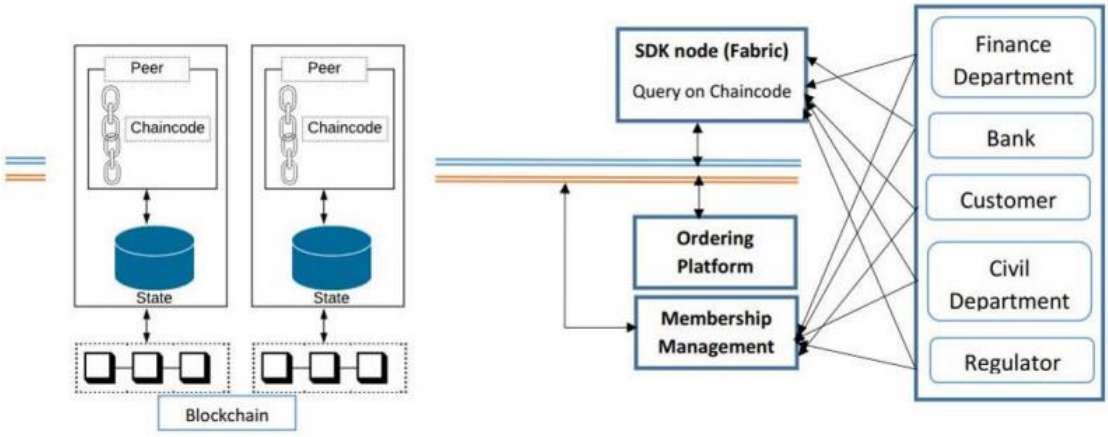


Figure 2-7: LoC architecture

Blockchain provides various benefits such as transparency of ownership, faster transactions, elimination of middlemen, and cheaper settlements when it comes to trading with businesses. The activities of shareholders, managers, and investors can be easily tracked. However, all participants may not agree to use this system because they may not want to disclose their trades to other managers and shareholders of small organizations. This problem can be solved by using smart contracts, in which visibility can be limited for some users. In this blockchain system, individual shareholders and shareowners can be identified [11].

Blockchain can also help the travel industry to maintain all its processes seamlessly using various tools and technolo-gies. In the traditional system, customer reviews have promoted tourism, but unfortunately, some of these reviews have been fraudulent. This problem can be curbed with the help of blockchain-based applications by verifying user profiles. Many researchers are trying to develop and ana-lyze blockchain applications in the tourism industry. There are many challenges to be faced in considering blockchain technology, and researchers are trying to mitigate these chal-lenges. For example, Airbnb and Uber are popular applica-tions that do not rely on a traditional businessmodel but rather are based on the consumer-to-consumer (C2C) model [12].

E. Blockchain In Security And Privacy

In this section, we address the security and privacy issues in the use of blockchain technology. A distributed cloud architecture based on this technology has been designed to improve privacy and security issues, which are the most challenging aspects in recent times. This model takes min-imal effort, is secure, and offers on-demand access to the most concentrated computing framework. It has somemetrics that enable cost-effective high-performance computing in IoT networks. A blockchain as a security factor after analyzing the challenges, feasibility, and effectiveness of IoT-related deployments. For a smart city hybrid system architecture with PoW scheme to achieve superior privacy and efficiency. The network was divided into two parts, the core network and the edge network. The nodes with extensive storage and computational resources belonged to the core network, which was responsible for cre-ating blocks and edge nodes. This network contained limited storage and computational power where the nodes acted as a centralized server. However, an effective deployment strategy and caching technique was not implemented in that work.

Blockchain can be used effectively in healthcare to store patient medical records, images, videos, and documents. All of these types of data are sensitive and need to be made very private and secure but must also be available for those who need them. Blockchain technology can be used to maintain the security and privacy of data. In a Bitcoin-based healthcare model, the health data from each user can be stored. However, the storage management system is not bandwidth intensive, and there is a massive dispersion of network resources via throughput optimization. Instead of using Bitcoin, a better solution is to use an access control manager and store the instances in a database management system. An authorized administrator can access these records using a unique identifier that locates them where they are encrypted and time-stamped in the storage device. All the data of the blockchain database are called data lakes, which are very essential for data analysis. Figure 2-8 shows the architecture of a healthcare system using blockchain [10].

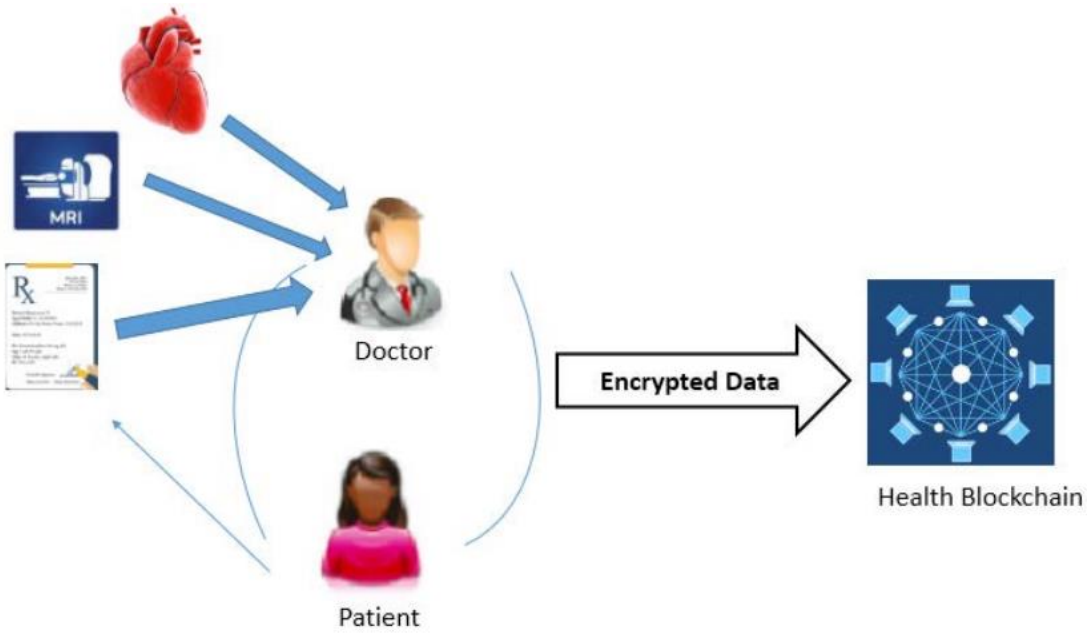


Figure 2-8: Healthcare transactions using blockchain

Blockchain is used as the backbone for Bitcoin, which has been a popular digital currency technology in recent years. Various technologies are used

for verification and validation in many methods. Figure 2-9 shows a transaction for blockchain in finance using smart contracts [10].

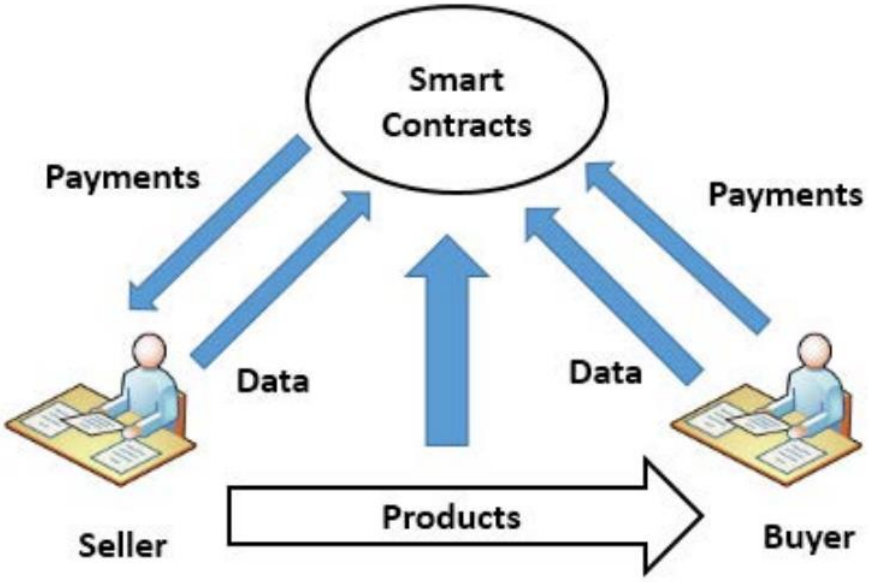


Figure 2-9: Finance transactions using blockchain

In this industry, data are encrypted from each block in the blockchain, and this provides security and confidentiality. In addition, private and public keys are used by financial organizations with high security needs. The consistency and integrity of data are confirmed in blockchain, which depends on the nature of traceability and immutability. There are diverse solutions for this kind of problem. In this system, user information is encrypted to ensure the security of records. Then hash values are included in each transaction. In addition to this, traceability is tracked over time and nonrepudiation records are used to ensure data security [10].

Chapter 3

Digital Signature

3.1. Introduction

In the world of communications, it is very important to ensure that messages are not modified by unauthorized persons. The mechanism for ensuring that data is not altered when transmitted from source to destination, or when it is stored, is called integrity. Secure Hash Standards (SHA-1, SHA-256, SHA-384, and SHA-512) are mechanisms that check the integrity of a message. Encryption provides intrinsic integrity because if a cipher-text block has been modified, the block will not be deciphered properly. Digital signature also provides integrity because it uses hash functions. Digital signatures let the recipient of information verify the authenticity of the information's origin, and also verify that the information was not altered while in transit. Thus, public key digital signatures provide authentication and data integrity. A digital signature also provides non-repudiation, which means that it pre-vents the sender from claiming that he or she did not actually send the information. These features are every bit as fundamental to cryptography as privacy, if not more [12].

3.2. Standardization

The FIPS Pub 186-2 (Federal Information Processing Standards (FIPS), 2000) issued the specification for the Digital Signature Standard (DSS), which prescribes the RSA digital signature algorithm. It described that for a digital signature to work, the signatory and the verifier need to have their public and private keys. The private key is used in the signature generation process, and the public key is used in the signature verification process. Also, the Secure Hash Algorithms (SHAs), specified in FIPS 180-2 are used for both signature generation and verification. If a message is signed with Alice's private key, then it is possible to assume that only Alice signed the message because she is the only one who has that private key.

A spoofer, who does not know Alice's private key, cannot generate Alice's correct signature. In other words, digital signatures cannot be forged. In the same way, if a message is enciphered with Bob's public key, only Bob will be able to decipher the message because it is assumed that only Bob has Bob's private key. Because there is the possibility that a spoofer may post his own public key as Alice's or Bob's, it is necessary to have a means of associating a user's identity and the user's public key [13]. A mutually trusted party, a certifying authority, could sign credentials containing the user's public key and identity by creating a digital certificate that binds the certificate to the user [13].

3.3. Digital signature process

A digital signature serves the same purpose as a handwritten signature. However, a handwritten signature is easy to counterfeit. A digital signature is superior to a hand-written signature in that it is nearly impossible to counterfeit, plus it attests to the contents of the information as well as to the identity of the signer. The basic manner in which digital signatures are created is shown in the following figure (3-1). Instead of encrypting information using someone else's public key, you encrypt it with your private key. If the information can be decrypted with your public key, then it must have originated with you [12].

A digital signature algorithm includes a signature generation process and a signature verification process. A signatory uses the generation process to generate a digital signature on data; a verifier uses the verification process to verify the authenticity of the signature. Each signatory has a public and private key and is the owner of that key pair. As shown in Figure 2.1, the private key is used in the signature generation process [12].

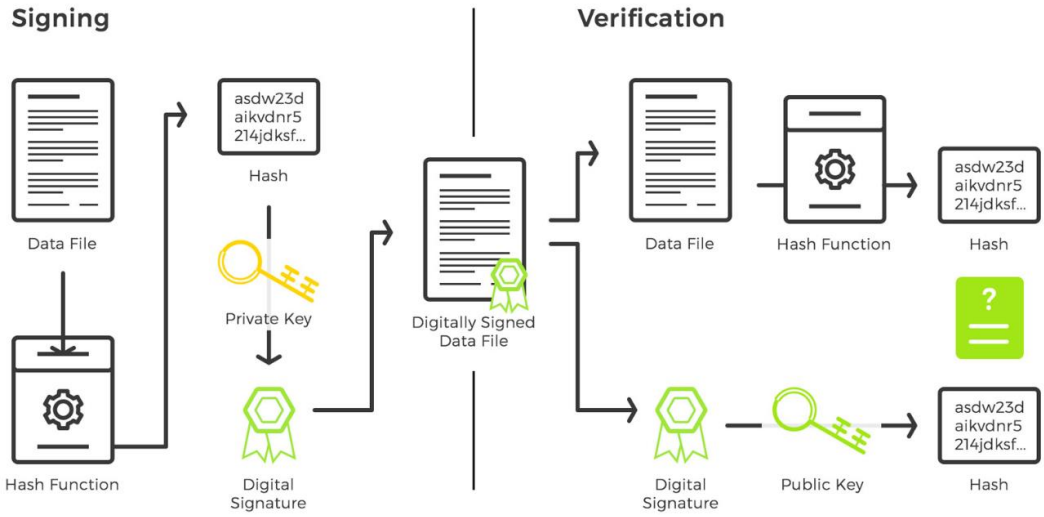


Figure 3-1: Digital signature block diagram

The key pair owner is the only entity that is authorized to use the private key to generate digital signatures. In order to prevent other entities from claiming to be the key pair owner and using the private key to generate fraudulent signatures, the private key must remain secret [12]. The approved digital signature algorithms are designed to prevent an adversary who does not know the signatory's private key from generating the same signature as the signatory on a different message. In other words, signatures are designed so that they cannot be forged. A number of alternative terms are used in this Standard to refer to the signatory or key pair owner. An entity that intends to generate digital signatures in the future may be referred to as the intended signatory. Prior to the verification of a signed message, the signatory is referred to as the claimed signatory until such time as adequate assurance can be obtained of the actual identity of the signatory. The public key is used in the signature verification process. The public key need not be kept secret, but its integrity must be maintained [13]. Anyone can verify a correctly signed message using the public key. For both the signature generation and verification processes, the message (i.e., the signed data) is converted to a fixed-length representation of the message by means of an approved hash function. Both the original message and the

digital signature are made available to a verifier [13]. A verifier requires assurance that the public key to be used to verify a signature belongs to the entity that claims to have generated a digital signature (i.e., the claimed signatory). That is, a verifier requires assurance that the signatory is the actual owner of the public/private key pair used to generate and verify a digital signature. A binding of an owner's identity and the owner's public key shall be effected in order to provide this assurance. A verifier also requires assurance that the key pair owner actually possesses the private key associated with the public key, and that the public key is a mathematically correct key. By obtaining these assurances, the verifier has assurance that if the digital signature can be correctly verified using the public key, the digital signature is valid (i.e., the key pair owner really signed the message). Digital signature validation includes both the (mathematical) verification of the digital signature and obtaining the appropriate assurances. The following are reasons why such assurances are required [12].

- If a verifier does not obtain assurance that a signatory is the actual owner of the key pair whose public component is used to verify a signature, the problem of forging a signature is reduced to the problem of falsely claiming an identity. For example, anyone in possession of a mathematically consistent key pair can sign a message and claim that the signatory was the President of the United States. If a verifier does not require assurance that the President is actually the owner of the public key that is used to mathematically verify the message's signature, then successful signature verification provides assurance that the message has not been altered since it was signed, but does not provide assurance that the message came from the President (i.e., the verifier has assurance of the data's integrity, but source authentication is lacking).

- If the public key used to verify a signature is not mathematically valid, the arguments used to establish the cryptographic strength of the signature algorithm may not apply. The owner may not be the only party who can generate signatures that can be verified with that public key.

- If a public key infrastructure cannot provide assurance to a verifier that the owner of a key pair has demonstrated knowledge of a private key that corresponds to the owner's public key, then it may be possible for an unscrupulous entity to have their identity (or an assumed identity) bound to a public key that is (or has been) used by another party. The unscrupulous entity may then claim to be the source of certain messages signed by that other party. Or, it may be possible that an unscrupulous entity has managed to obtain ownership of a public key that was chosen with the sole purpose of allowing for the verification of a signature on a specific message.

3.4. Secure Hash Algorithms

Hash functions are used to prove that transmitted data was not altered. A hash function ***H*** takes an input message ***m*** and transforms it to produce a hash value ***h*** that is a function of the message ***h = H (m)***; the input is a variable string and the output is a fixed-size string, as shown in figure 3-2.

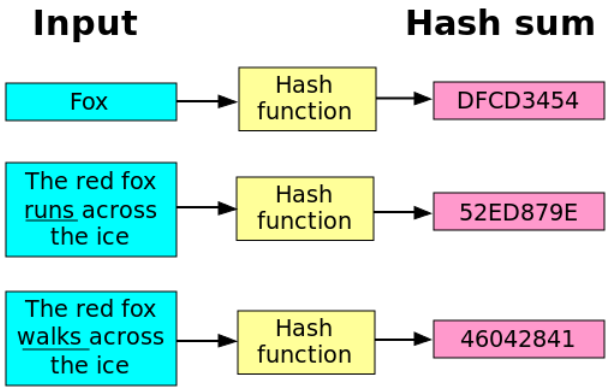


Figure 3-2: Hash function

The hash value is also called a ***message digest*** or a fingerprint of the message because there is a very low probability that two messages will produce the same hash value. Hash functions are hard to invert. Given the hash value, it is computationally infeasible to find the initial value ***m***. A hash function must have the following properties [12]:

- The message size can be of any length.
- The hash value has a fixed length.
- It is relatively easy to compute ***H(m)*** for any given message.
- It is computationally infeasible, virtually impossible, to:
  - Find the message m from ***H(m)*** (this is called a one-way function)
  - Have two messages, ***m1***and ***m2***, in which ***H(m1) = H(m2).***
  - Find two messages, ***m1***and ***m2***, such that ***H(m1) = H(m2).***

A hash function is a "strong hash function" if it satisfies all of the above properties.

3.5. History of Hash Signature Standard

On April 17, 1995, the National Institute of Standards and Technology, NIST, approved the Secure Hash Standard, Federal Information Processing Standards (FIPS) PUB 180-1, which included one secure hash algorithm, the SHA-1. On February 1, 2003, a new Secure Hash Signature Standard (SHS) (FIPS PUB 180-2) was approved; it added three hash algorithms that are capable of producing larger message digests, superseding FIPS 180-1. FIPS PUB 180-2 specifies four secure hash algorithms, SHA-1, SHA-256, SHA-384, and SHA-512. The message digests range in length from 160 to 512 bits, depending on the algorithm. The SHA-1 algorithm specified in the FIPS PUB 180-2 is the same algorithm that was specified previously in FIPS 180-1, although some of the notation was modified to be consistent with the notation used in the

SHA-256, SHA-384, and SHA-512 algorithms. On August, 2015, the latest FIPS PUP 180-4 specifies secure hash algorithms - SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 and SHA-512/256 - for computing a condensed representation of electronic data (message). When a message of any length less than 2^{64} bits (for SHA-1, SHA-224 and SHA-256) or less than 2^{128} bits (for SHA-384, SHA-512, SHA-512/224 and SHA-512/256) is input to a hash algorithm, the result is the message digest. All algorithms are iterative, i.e., one-way hash functions that can process a message in a condensed representation called a message digest. The next table shows the basic properties of all secure hash algorithms [12].

Algorithm	**Message Size (bits)**	**Block Size (bits)**	**Word Size (bits)**	**Message Digest Size (bits)**	**Security (bits)**
SHA-1	$< 2^{64}$	512	32	160	80
SHA-224	$< 2^{64}$	512	32	224	112
SHA-256	$< 2^{64}$	512	32	256	128
SHA-384	$< 2^{128}$	1024	64	384	192
SHA-512	$< 2^{128}$	1024	64	512	256

According to FIPS PUB 180-4, the following bitwise logical operator symbols are used to calculate the hash function:

- $\wedge$ Bitwise logical “and”.
- $\vee$ Bitwise logical “inclusive-or”.
- $\oplus$ Bitwise logical “exclusive-or”.
- $\neg$ Bitwise logical “complement”.
- $+$ Addition modulo 2.
- $\ll$ Left-shift operation, where $x \ll n$ is obtained by discarding the left-most nbits of the word ***x*** and then padding the result with ***n*** zeros on the right.

≫ Right-shift operation, where $x \gg n$ is obtained by discarding the rightmost nbits of the word ***x*** and then padding the result with ***n*** zeros on the left.

The following parameters are used in the secure hash algorithm specifications:

- ***a, b, c, …, h***: Working variables that are the *w*-bit words used in the computation of the hash values, $H^{(i)}$.
- $H^{(i)}$: The ***i^th^*** hash value. $H^{(0)}$is the initial hash value; $H^{(N)}$is the final hash value and is used to determine the message digest.
- $H_j^{(i)}$: The ***j^th^*** word of the ***i^th^*** hash value, where $H_0^{(i)}$ is the left- most word of hash value i.
- K_i: Constant value to be used for iteration t of the hash computation.
- k: Number of zeroes appended to a message during the padding step.
- l: Length of the message, ***M***, in bits.
- m: Number of bits in a message block, $M^{(i)}$.
- M: Message to be hashed.
- $M^{(i)}$: Message block ***i***, with a size of ***m*** bits.
- $M_j^{(i)}$: The ***j^th^*** word of the ***i^th^*** message block, where $M^{(i)}$ is the left-most word of message block ***i***.
- n: Number of bits to be rotated or shifted when a word is operated upon.
- N: Number of blocks in the padded message.
- T: Temporary *w*-bit word used in the hash computation.
- w: Number of bits in a word.

- W_t: The t^{th} w-bit word of the message schedule.

The following operations are used in the secure hash algorithm specifications:

$ROTL^n(x)$: The rotate left (circular left shift) operation.

$ROTR^n(x)$: The rotate right (circular right shift) operation.

$SHR^n(x)$: The right shift operation.

3.5.1. Secure Hash Algorithm (SHA-1)

The SHA-1 is designed so it is computationally infeasible to find a message that corresponds to a given message digest, or to find two different messages which produce the same message digest. The SHA-1 [12]:

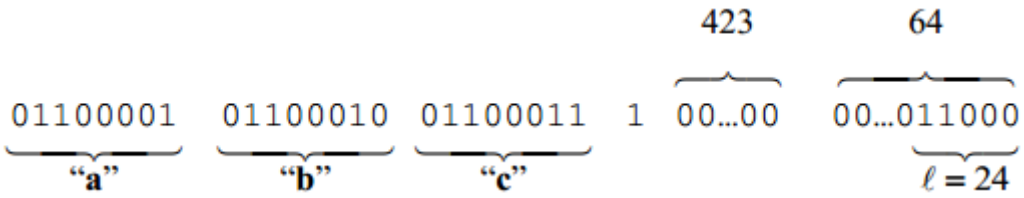
- Works on messages up to 2^{64} in length.
- Produces a 160-bit message digest.
- Pads a message to a multiple of 512 bits (length = 448 mod 512)
- Carries eighty operations in its main algorithm
- Performs a nonlinear operation on three of the five variables A, B, C, D, and E, in each operation.

SHA-1 uses a sequence of logical functions, $f_0, f_1, \ldots, f_{79}$. Each function f_t, where $0 \leq t \leq 79$, operates on three 32-bit words, x, y, and z, and produces a 32-bit word as output. The function $f_t(x, y, z)$ is defined as follows:

$$f_t(x,y,z) = \begin{cases} Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z) & 0 \le t \le 19 \\ Parity(x,y,z) = x \oplus y \oplus z & 20 \le t \le 39 \\ Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) & 40 \le t \le 59 \\ Parity(x,y,z) = x \oplus y \oplus z & 60 \le t \le 79 \end{cases}$$

Message padding and parsing

The message, M, shall be padded before hash computation begins. The purpose of this padding is to ensure that the padded message is a multiple of 512 for SHA-1 and SHA-256 or 1024 bits for SHA-384 and SHA-512. The SHA-1 sequentially processes blocks of 512 bits when computing the message digest and padding is required, even if the message is a multiple of 512. Suppose that the length of the message, ***M***, is ***l*** bits and ***k*** is the number of zero bit. Padding is done as follows [12]:



A 1 bit followed by 0 bits is appended until the length is 64 bits less than a multiple of 512, ($l + 1 + k = 448 \bmod 512$).

A 64-bit block that is equal to the number l expressed using a binary representation is appended. The padded message length is $512 \times n$. For example, the (8-bit ASCII) message "abc" has the length 8 x 3 = 24, so the message is padded with a one bit, then 448 – (24 + 1) = 423 zero bits. The length of the padded message is now a multiple of 512 bits.

After a message has been padded, it must be parsed into ***N*** m-bit blocks before the hash computation can begin. For SHA-1, the padded

message is parsed into N 512-bit blocks, $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$. Since the 512 bits of the input block may be expressed as sixteen 32-bit words, the first 32 bits of message block ***i*** are denoted M_0^i, the next 32 bits are M_1^i, and so on up to M_{15}^i.

Computing message digest

The SHA-1 computation uses two buffers, each consisting of five 32-bit words, and a sequence of eighty 32-bit words. The words of the first 5-word buffer are labeled H_0, H_1, H_2, H_3, H_4. The words of the second 5-word buffer are labeled a, b, c, d, and e. The words of the 80-word sequence are labeled $W_0, W_1, \ldots, W_{79}$. The following steps show the calculations to generate the message digest for a 512-bit message block for *i* =1 to *i* = ***N***. Nis the number of 512-bit message blocks.

- The initial hash value $H_j^{(0)}$, shall consist of the following five 32-bit words, in hex:

$$H_0^{(0)} = 67452301$$
$$H_1^{(0)} = efcdab89$$
$$H_2^{(0)} = 98badcfe$$
$$H_3^{(0)} = 10325476$$
$$H_4^{(0)} = c3d2e1f0$$

For *i* =1 to ***N*** do next steps:

- Prepare the message schedule, W_t

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ ROTL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) & 16 \le t \le 79 \end{cases}$$

- Initialize the five working variables, a, b, c, d, and e, with the $(i\text{-}1)^{st}$ hash value:

$$a = H_0^{(i-1)}$$
$$b = H_1^{(i-1)}$$
$$c = H_2^{(i-1)}$$
$$d = H_3^{(i-1)}$$
$$e = H_4^{(i-1)}$$

- For t = 0 to 79:

$$T = ROTL^5(a) + f_t(b,c,d) + e + K_t + W_t$$
$$e = d$$
$$d = c$$
$$c = ROTL^{30}(b)$$
$$b = a$$
$$a = T$$

Where:

$$K_t = \begin{cases} \texttt{5a827999} & 0 \le t \le 19 \\ \texttt{6ed9eba1} & 20 \le t \le 39 \\ \texttt{8f1bbcdc} & 40 \le t \le 59 \\ \texttt{ca62c1d6} & 60 \le t \le 79 \end{cases}$$

- Compute the ith intermediate hash value $H^{(i)}$:

$$H_0^{(i)} = a + H_0^{(i-1)}$$
$$H_1^{(i)} = b + H_1^{(i-1)}$$
$$H_2^{(i)} = c + H_2^{(i-1)}$$
$$H_3^{(i)} = d + H_3^{(i-1)}$$
$$H_4^{(i)} = e + H_4^{(i-1)}$$

After repeating steps one through four a total of ***N*** times (i.e., after processing $M^{(N)}$), the resulting 160-bit message digest of the message, ***M***, is:

$$H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)}$$

Figure 3-3 shows the block diagram for cycle *i* of computation.

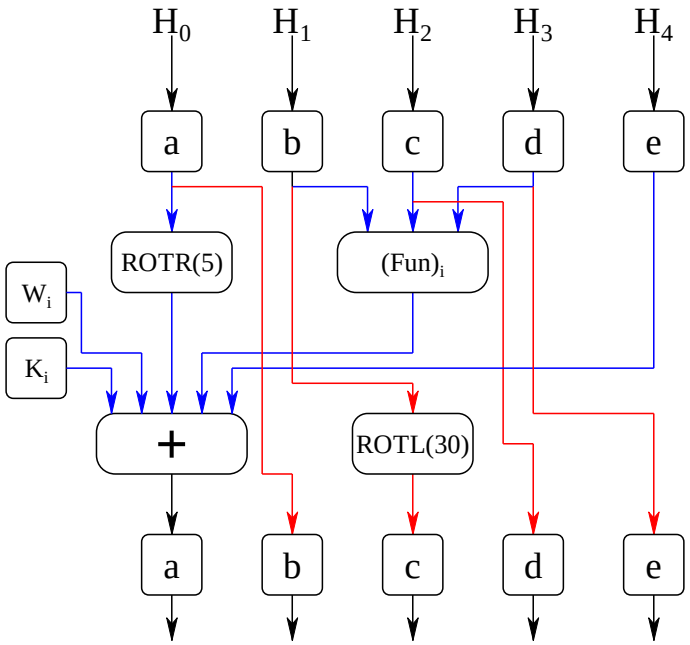


Figure 3-3: The ith cycle of SHA-1 calculation

3.5.2. Secure Hash Algorithms (SHA-224/ SHA-256)

SHA-224 and SHA-256 may be used to hash a message, ***M***, having a length of l bits, where $0 \leq l \leq 2^{64}$. The algorithms use a message schedule of sixty-four 32-bit words, with eight working variables of 32 bits each, and a hash value of eight 32-bit words. The words of the message schedule are labeled $W_0, W_1, \ldots, W_{79}$. The eight working variables are labeled ***a, b, c, d, e, f, g,*** and ***h***.

The final result of SHA-256 is a 256-bit message digest (224-bit for SHA-224). SHA-224/SHA-256 algorithms use six logical functions, where each

function operates on 32-bit words, which are represented as x, y, and z. The result of each function is a new 32-bit word.

$$Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z)$$
$$Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$

$$\sum_0^{\{256\}}(x) = ROTR^{2}(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x)$$
$$\sum_1^{\{256\}}(x) = ROTR^{6}(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x)$$
$$\sigma_0^{\{256\}}(x) = ROTR^{7}(x) \oplus ROTR^{18}(x) \oplus SHR^{3}(x)$$
$$\sigma_1^{\{256\}}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$$

SHA-224/SHA-256 message padding and parsing is the same as done in SHA-1 Algorithm.

Computing message digest

- The initial hash value $H_j^{(0)}$, shall consist of the following 32-bit words, in hex:

SHA-224:

$$H_0^{(0)} = \texttt{c1059ed8}$$
$$H_1^{(0)} = \texttt{367cd507}$$
$$H_2^{(0)} = \texttt{3070dd17}$$
$$H_3^{(0)} = \texttt{f70e5939}$$
$$H_4^{(0)} = \texttt{ffc00b31}$$
$$H_5^{(0)} = \texttt{68581511}$$
$$H_6^{(0)} = \texttt{64f98fa7}$$
$$H_7^{(0)} = \texttt{befa4fa4}$$

SHA-256:

$$H_0^{(0)} = \texttt{c1059ed8}$$
$$H_1^{(0)} = \texttt{367cd507}$$
$$H_2^{(0)} = \texttt{3070dd17}$$
$$H_3^{(0)} = \texttt{a54ff53a}$$
$$H_4^{(0)} = \texttt{510e527f}$$
$$H_5^{(0)} = \texttt{9b05688c}$$
$$H_6^{(0)} = \texttt{1f83d9ab}$$
$$H_7^{(0)} = \texttt{5be0cd19}$$

For i =1 to N do next steps:

- Prepare the message schedule, W_t

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16} & 16 \le t \le 63 \end{cases}$$

- Initialize the five working variables, a, b, c, d, e, f, g, and h, with the (i-1)st hash value:

$$a = H_0^{(i-1)}$$
$$b = H_1^{(i-1)}$$
$$c = H_2^{(i-1)}$$
$$d = H_3^{(i-1)}$$
$$e = H_4^{(i-1)}$$
$$f = H_5^{(i-1)}$$
$$g = H_6^{(i-1)}$$
$$h = H_7^{(i-1)}$$

- For t = 0 to 63 do next:

$$T_1 = h + \sum_1^{\{256\}}(e) + Ch(e,f,g) + K_t^{\{256\}} + W_t$$
$$T_2 = \sum_0^{\{256\}}(a) + Maj(a,b,c)$$
$$h = g$$
$$g = f$$
$$f = e$$
$$e = d + T_1$$
$$d = c$$
$$c = b$$
$$b = a$$
$$a = T_1 + T_2$$

Where $K_0^{\{256\}}, K_1^{\{256\}}, \ldots, K_{63}^{\{256\}}$:

```
428a2f98 71374491 b5c0fbcf e9b5dba5 3956c25b 59f111f1 923f82a4 ab1c5ed5
d807aa98 12835b01 243185be 550c7dc3 72be5d74 80deb1fe 9bdc06a7 c19bf174
e49b69c1 efbe4786 0fc19dc6 240ca1cc 2de92c6f 4a7484aa 5cb0a9dc 76f988da
983e5152 a831c66d b00327c8 bf597fc7 c6e00bf3 d5a79147 06ca6351 14292967
27b70a85 2e1b2138 4d2c6dfc 53380d13 650a7354 766a0abb 81c2c92e 92722c85
a2bfe8a1 a81a664b c24b8b70 c76c51a3 d192e819 d6990624 f40e3585 106aa070
19a4c116 1e376c08 2748774c 34b0bcb5 391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
748f82ee 78a5636f 84c87814 8cc70208 90befffa a4506ceb bef9a3f7 c67178f2
```

- Compute the i^{th} intermediate hash value $H^{(i)}$:

$$H_0^{(i)} = a + H_0^{(i-1)}$$
$$H_1^{(i)} = b + H_1^{(i-1)}$$
$$H_2^{(i)} = c + H_2^{(i-1)}$$
$$H_3^{(i)} = d + H_3^{(i-1)}$$
$$H_4^{(i)} = e + H_4^{(i-1)}$$
$$H_5^{(i)} = f + H_5^{(i-1)}$$
$$H_6^{(i)} = g + H_6^{(i-1)}$$
$$H_7^{(i)} = h + H_7^{(i-1)}$$

After repeating steps one through four a total of ***N*** times (i.e., after processing $M^{(N)}$), the resulting 256-bit message digest of the message, ***M***, is:

$$H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)} \| H_6^{(N)} \| H_7^{(N)}$$

For SHA-224, the initial hash value, $H^{(0)}$ should be specified differently from SHA-256, and The 224-bit message digest is obtained by truncating the final hash value, $H^{(N)}$, to its left-most 224 bits:

$$H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)} \| H_6^{(N)}$$

Figure 3-4 shows the block diagram for cycle *i* of computation of SHA-224/256.

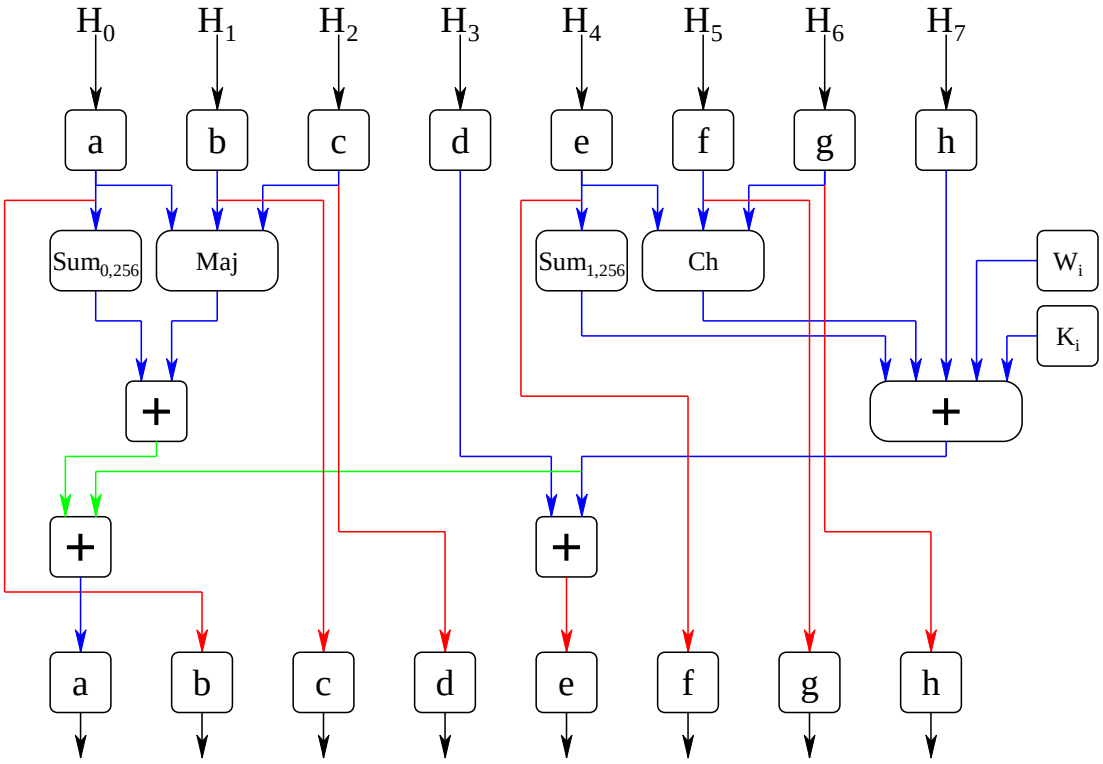


Figure 3-4: The ith cycle of SHA-224/SHA-256 calculation

3.5.3. Secure Hash Algorithms (SHA-384/ SHA-512)

SHA-384 and SHA-512 may be used to hash a message, ***M***, having a length of l bits, where $0 \leq l \leq 2^{128}$. The algorithms use a message schedule of eighty 64-bit words, with eight working variables of 64 bits

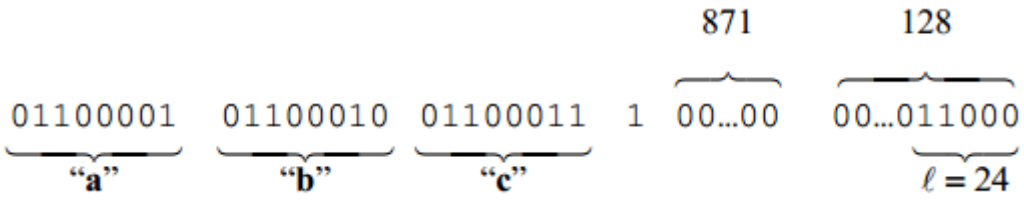
each, and a hash value of eight (6 for SHA-384) 64-bit words. The words of the message schedule are labeled $W_0, W_1, \ldots, W_{79}$. The eight working variables are labeled ***a, b, c, d, e, f, g,*** and ***h***. SHA-384/SHA-512 use six logical functions, where each function operates on 64-bit words, which are represented as x, y, and z. The result of each function is a new 64-bit word.

$$
\begin{aligned}
Ch(x,y,z) &= (x \wedge y) \oplus (\neg x \wedge z) \\
Maj(x,y,z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)
\end{aligned}
$$

$$
\begin{aligned}
\sum_0^{\{512\}}(x) &= ROTR^{28}(x) \oplus ROTR^{34}(x) \oplus ROTR^{39}(x) \\
\sum_1^{\{512\}}(x) &= ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x) \\
\sigma_0^{\{512\}}(x) &= ROTR^{1}(x) \oplus ROTR^{8}(x) \oplus SHR^{7}(x) \\
\sigma_1^{\{512\}}(x) &= ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus SHR^{6}(x)
\end{aligned}
$$

Message padding and parsing

The message, ***M***, shall be padded before hash computation begins. The SHA-384/SHA-512 sequentially processes blocks of 1024 bits when computing the message digest and padding is required, even if the message is a multiple of 1024. Suppose that the length of the message, ***M***, is ***l*** bits and ***k*** is the number of zero bit. Padding is done as follows [12]:



A 1 bit followed by 0 bits is appended until the length is 64 bits less than a multiple of 1024, ($l + 1 + k = 896 \bmod 1024$).

A 64-bit block that is equal to the number l expressed using a binary representation is appended. The padded message length is $1024 \times n$. For example, the (8-bit ASCII) message "abc" has the length 8 x 3 = 24, so the

message is padded with a one bit, then 896 – (24 + 1) = 871 zero bits. The length of the padded message is now a multiple of 1024 bits.

After a message has been padded, it must be parsed into ***N*** 1024-bit blocks before the hash computation can begin. the padded message is parsed into ***N*** 1024-bit blocks, $M^{(1)}, M^{(2)}, \dots, M^{(N)}$. Since the 1024 bits of the input block may be expressed as sixteen 64-bit words, the first 64 bits of message block ***i*** are denoted M_0^i, the next 64 bits are M_1^i, and so on up to M_{15}^i [12].

Computing message digest

- The initial hash value $H_j^{(0)}$, shall consist of the following 32-bit words, in hex:

SHA-384:

$$H_0^{(0)} = \texttt{cbbb9d5dc1059ed8}$$
$$H_1^{(0)} = \texttt{629a292a367cd507}$$
$$H_2^{(0)} = \texttt{9159015a3070dd17}$$
$$H_3^{(0)} = \texttt{152fecd8f70e5939}$$
$$H_4^{(0)} = \texttt{67332667ffc00b31}$$
$$H_5^{(0)} = \texttt{8eb44a8768581511}$$
$$H_6^{(0)} = \texttt{db0c2e0d64f98fa7}$$
$$H_7^{(0)} = \texttt{47b5481dbefa4fa4}$$

SHA-512:

$$H_0^{(0)} = \texttt{6a09e667f3bcc908}$$
$$H_1^{(0)} = \texttt{bb67ae8584caa73b}$$

$$H_2^{(0)} = \texttt{3c6ef372fe94f82b}$$
$$H_3^{(0)} = \texttt{a54ff53a5f1d36f1}$$
$$H_4^{(0)} = \texttt{510e527fade682d1}$$
$$H_5^{(0)} = \texttt{9b05688c2b3e6c1f}$$
$$H_6^{(0)} = \texttt{1f83d9abfb41bd6b}$$
$$H_7^{(0)} = \texttt{5be0cd19137e2179}$$

For i =1 to N do next steps:

- Prepare the message schedule, W_t

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ \sigma_1^{\{512\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{512\}}(W_{t-15}) + W_{t-16} & 16 \le t \le 79 \end{cases}$$

- Initialize the five working variables, ***a***, ***b***, ***c***, ***d***, ***e***, ***f***, ***g***, and ***h***, with the (i-1)st hash value:

$$a = H_0^{(i-1)}$$
$$b = H_1^{(i-1)}$$
$$c = H_2^{(i-1)}$$
$$d = H_3^{(i-1)}$$
$$e = H_4^{(i-1)}$$
$$f = H_5^{(i-1)}$$
$$g = H_6^{(i-1)}$$
$$h = H_7^{(i-1)}$$

- For t = 0 to 79 do next:

$$T_1 = h + \sum_1^{\{512\}}(e) + Ch(e,f,g) + K_t^{\{512\}} + W_t$$
$$T_2 = \sum_0^{\{512\}}(a) + Maj(a,b,c)$$
$$h = g$$
$$g = f$$
$$f = e$$
$$e = d + T_1$$
$$d = c$$
$$c = b$$
$$b = a$$
$$a = T_1 + T_2$$

Where $K_0^{\{512\}}, K_1^{\{512\}}, \ldots, K_{79}^{\{512\}}$:

```
428a2f98d728ae22 7137449123ef65cd b5c0fbcfec4d3b2f e9b5dba58189dbbc
3956c25bf348b538 59f111f1b605d019 923f82a4af194f9b ab1c5ed5da6d8118
d807aa98a3030242 12835b0145706fbe 243185be4ee4b28c 550c7dc3d5ffb4e2
72be5d74f27b896f 80deb1fe3b1696b1 9bdc06a725c71235 c19bf174cf692694
e49b69c19ef14ad2 efbe4786384f25e3 0fc19dc68b8cd5b5 240ca1cc77ac9c65
2de92c6f592b0275 4a7484aa6ea6e483 5cb0a9dcbd41fbd4 76f988da831153b5
983e5152ee66dfab a831c66d2db43210 b00327c898fb213f bf597fc7beef0ee4
c6e00bf33da88fc2 d5a79147930aa725 06ca6351e003826f 142929670a0e6e70
27b70a8546d22ffc 2e1b21385c26c926 4d2c6dfc5ac42aed 53380d139d95b3df
650a73548baf63de 766a0abb3c77b2a8 81c2c92e47edaee6 92722c851482353b
a2bfe8a14cf10364 a81a664bbc423001 c24b8b70d0f89791 c76c51a30654be30
d192e819d6ef5218 d69906245565a910 f40e35855771202a 106aa07032bbd1b8
19a4c116b8d2d0c8 1e376c085141ab53 2748774cdf8eeb99 34b0bcb5e19b48a8
391c0cb3c5c95a63 4ed8aa4ae3418acb 5b9cca4f7763e373 682e6ff3d6b2b8a3
748f82ee5defb2fc 78a5636f43172f60 84c87814a1f0ab72 8cc702081a6439ec
90befffa23631e28 a4506cebde82bde9 bef9a3f7b2c67915 c67178f2e372532b
ca273eceea26619c d186b8c721c0c207 eada7dd6cde0eb1e f57d4f7fee6ed178
06f067aa72176fba 0a637dc5a2c898a6 113f9804bef90dae 1b710b35131c471b
28db77f523047d84 32caab7b40c72493 3c9ebe0a15c9bebc 431d67c49c100d4c
4cc5d4becb3e42b6 597f299cfc657e2a 5fcb6fab3ad6faec 6c44198c4a475817
```

- Compute the i^{th} intermediate hash value $H^{(i)}$:

$$H_0^{(i)} = a + H_0^{(i-1)}$$
$$H_1^{(i)} = b + H_1^{(i-1)}$$

$$H_2^{(i)} = c + H_2^{(i-1)}$$
$$H_3^{(i)} = d + H_3^{(i-1)}$$
$$H_4^{(i)} = e + H_4^{(i-1)}$$
$$H_5^{(i)} = f + H_5^{(i-1)}$$
$$H_6^{(i)} = g + H_6^{(i-1)}$$
$$H_7^{(i)} = h + H_7^{(i-1)}$$

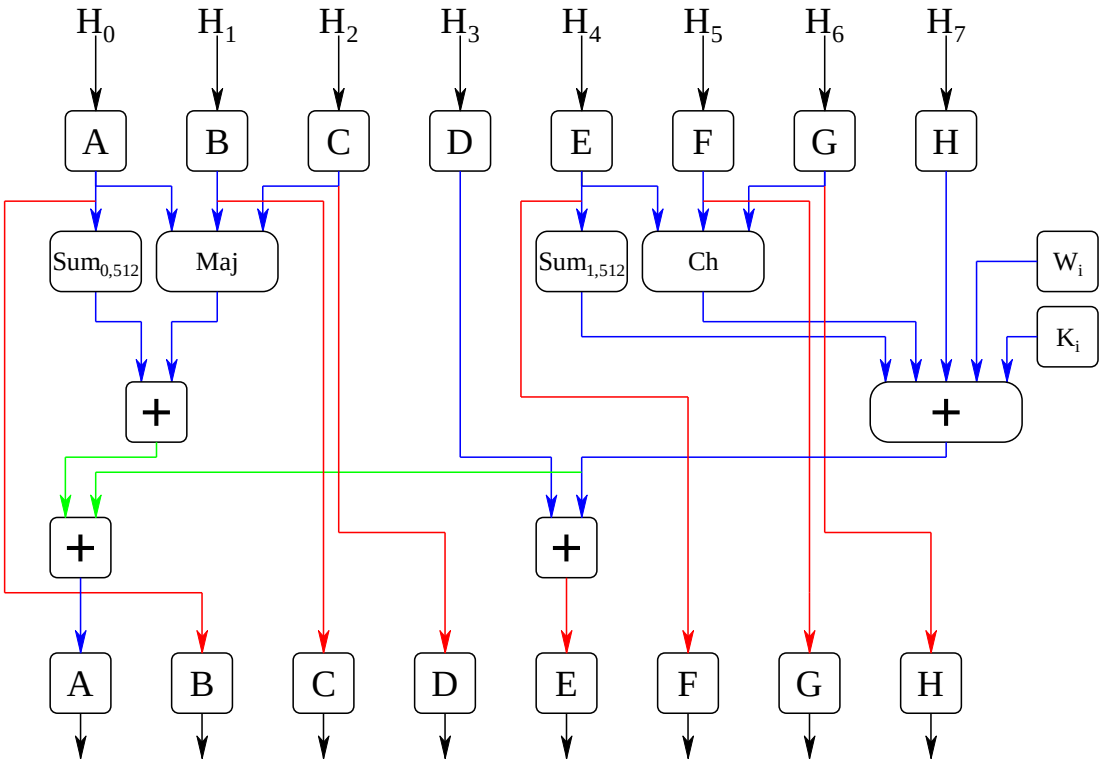


Figure 3-5: The i[th] cycle of SHA-384/SHA-512 calculation

After repeating steps one through four a total of N times (i.e., after processing $M^{(N)}$), the resulting 512-bit message digest of the message, M, is:

$$H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)} \| H_6^{(N)} \| H_7^{(N)}$$

For SHA-384, the initial hash value, $H^{(0)}$ should be specified differently from SHA-256, and The 384-bit message digest is obtained by truncating the final hash value, $H^{(N)}$, to its left-most 384 bits:

$$H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)}$$

Figure 3.4 shows the block diagram for cycle *i* of computation of SHA-384/512.

Chapter 4

Cryptocurrency Platform Design

4.1. Introduction

In this chapter, we explore the design and implementation of a **blockchain-based platform** that allows users to interact with a decentralized ledger system. The platform is designed to support user authentication, wallet management, transaction creation, and blockchain validation. It leverages **Python**, **Flask**, and related libraries for the back-end logic, while the front-end pages are rendered using Flask's templating system. The system is structured as a modular and scalable application, demonstrating key principles of blockchain technology, including **cryptographic hashing**, **proof-of-work**, and **blockchain integrity validation**. The primary goal of this platform is to provide a foundational design for a **concurrency platform**, enabling users to securely execute and validate transactions. This platform includes several key features:

- **User Authentication**: Secure user registration and login using hashed passwords.
- **Wallet Management**: Each user is assigned a unique wallet with public and private keys, along with an initial balance.
- **Transaction Processing**: Users can send digital funds to other users via transactions that are recorded in a blockchain.
- **Blockchain Construction**: Transactions are grouped into blocks, which are cryptographically linked to form a valid blockchain.
- **Blockchain Validation**: Ensures the integrity and security of the blockchain by validating hashes, proof-of-work, and block sequences.

4.2. Platform Structure

This platform is designed as a **concurrency system** that uses blockchain technology to manage secure transactions between users. By combining Python and Flask with a robust modular structure, the platform ensures scalability, security, and ease of use. To provide a foundational system for blockchain-based concurrency platforms with user authentication, wallet management, transaction processing, and blockchain validation. Below is a detailed overview of the files and their roles:

- **__init__.py**: Initializes the application.
- **models.py**: Defines the database schema.
- **wallet.py**: Manages wallet creation and hashing.
- **blockchain.py**: Implements blockchain logic.
- **auth.py**: Handles user authentication.
- **views.py**: Provides user-facing pages and transaction functionality.

4.2.1. __init__.py

Initializes the Flask application and configures essential components such as the database and login management. It acts as the central entry point for the application, bringing all components together

Key Features:

- Sets up the SQLite database and ensures all tables are created.
- Registers blueprints (views and auth) to modularize the application.
- Configures the LoginManager for session-based user authentication.

4.2.2. models.py

Defines the database schema and relationships for the application and provides the foundation for database interactions, ensuring a well-structured and relational data model.

Key Features:

- **User Model**: Represents users, including their credentials and a one-to-one relationship with their wallet.
- **Wallet Model**: Tracks wallet details such as public/private keys, balance, and associated transactions.
- **Transaction Model**: Records transactions between wallets, including sender, receiver, amount, and hash.
- **Block Model**: Represents individual blocks on the blockchain, linking transactions and maintaining cryptographic hashes.

4.2.3. wallet.py

Handles wallet creation and transaction hash generation, and Ensures the secure management of wallets and transactions, crucial for a blockchain system.

Key Features:

- **Wallet Creation**: Generates secure private keys, derives public keys, and initializes wallets with a starting balance.
- **Transaction Hashing**: Produces unique cryptographic hashes for transactions.

4.2.4. blockchain.py

Implements the core blockchain logic, including block creation, proof-of-work, and chain validation, and provides the technical backbone for blockchain operations, ensuring data integrity and security

Key Features:

- **Hash Calculation**: Computes cryptographic hashes for blocks using block data and proof-of-work.
- **Block Creation**: Constructs new blocks and links them to the blockchain.
- **Blockchain Validation**: Verifies the integrity of the entire blockchain, ensuring correct hash continuity and block order.

4.2.5. auth.py

Manages user authentication and wallet creation and Ensures secure access to the platform and links users to their wallets.

Key Features:

- **User Registration**: Allows new users to register, securely storing their credentials (hashed passwords).
- **Login/Logout**: Handles user login and logout functionality.
- **Wallet Assignment**: Automatically creates a wallet for new users during registration.

4.2.6. views.py

Defines the routes for user-facing pages and transaction creation and acts as the user interface layer, allowing interaction with the blockchain system through web pages

Key Features:

- **Home Page (/)**: Displays blockchain data and user-related transactions.
- **User Public Keys (/users_pk)**: Lists all users and their public keys for transaction purposes.
- **User Info (/user_info)**: Shows details about the logged-in user's wallet and outgoing transactions.
- **Transaction Creation (/transaction)**: Enables users to send funds, validates inputs, and updates the blockchain.

4.2.7. How the Files Work Together

The design of this platform is modular, with each file contributing to a specific aspect of the application. Here's how they interact:

1. **Application Initialization (__init__.py)**:
    - Sets up the Flask application and integrates the database and login system.
2. **Data Models (models.py)**:
    - Defines the database structure for users, wallets, transactions, and blocks.
3. **Wallet Management (wallet.py)**:
    - Handles the creation of wallets and transaction hashes, ensuring secure cryptographic operations.
4. **Blockchain Operations (blockchain.py)**:
    - Manages the creation, validation, and verification of blockchain blocks, ensuring data integrity.

5. **Authentication (auth.py)**:
    - Provides user authentication and links users to their wallets.
6. **Routes and Views (views.py)**:
    - Exposes the blockchain functionality to users through web pages and handles interactions like transaction creation.

4.3. Python code

4.3.1. Auth.py

The auth.py file implements user authentication and wallet creation functionality. It uses Flask's blueprint system to handle routes related to creating new users, logging users in, and associating wallets with users. Below is an in-depth explanation of each line and section.

Imports

```
from flask import Blueprint, render_template, request, flash, redirect, url_for
from .models import Wallet, User
from werkzeug.security import generate_password_hash, check_password_hash
from . import db
from flask_login import login_user, login_required, logout_user, current_user
from .wallet import create_wallet
```

1. **flask.Blueprint:** A way to organize routes (views) into modular components for a Flask application. Here, the auth blueprint is used for authentication-related routes.
2. **render_template**: Renders HTML templates for user-facing pages.

3. **request**: Accesses form data submitted via HTTP requests (e.g., user registration form).
4. **flash**: Displays one-time messages to users (e.g., success or error messages).
5. **redirect** and **url_for**: Redirects users to other routes dynamically.
6. **Wallet, User**: Imports the Wallet and User models from models.py to interact with the database.
7. **generate_password_hash, check_password_hash**: Used to securely hash passwords or verify hashed passwords.
8. **db**: The SQLAlchemy database object for performing database operations.
9. **flask_login**: Provides session management functions:
    - **login_user**: Logs in a user.
    - **login_required**: Ensures that a route is only accessible to logged-in users.
    - **logout_user**: Logs out the current user.
    - **current_user**: Provides access to the currently logged-in user's data.
10. **create_wallet**: A function from wallet.py that generates a public/private key pair and creates a new wallet for a user.

Blueprint for Authentication

```
auth = Blueprint('auth', __name__)
```

Creates a Blueprint object named auth for handling all authentication-related routes. This allows grouping and organizing routes related to user authentication and wallet management.

Route: /new_wallet

```
@auth.route('/new_wallet', methods=['GET', 'POST'])
def new_wallet():
```

- **@auth.route('/new_wallet')**:
  - Defines a route for creating a new user and wallet.
  - The route accepts both **GET** and **POST** requests.

Handling the POST Request

```
if request.method == 'POST':
    username = request.form.get('username')
    password = request.form.get('password')
    email = request.form.get('email')
```

1. **request.method == 'POST'**: Checks if the request is a POST request (indicating data submission).
2. **request.form.get():** Retrieves data from the submitted form:
   - username: The username of the new user.
   - password: The password of the new user.
   - email: The email address of the new user.

Check if the User Already Exists

```
user = User.query.filter_by(email=email).first()
if user:
    if user.password_hash == generate_password_hash(password, method='pbkdf2:sha256'):
        login_user(user, remember=True)
```

```
        return redirect(url_for('views.user_info'))
    else:
        flash('المستخدم موجود مسبقاً، كلمة سر خاطئة', category='error')
```

1. **User.query.filter_by(email=email).first()**: Queries the database to check if a user with the provided email already exists.
2. **If the User Exists:**
   - **Verify Password**: Compares the provided password to the stored hashed password using generate_password_hash.
   - **If Password is Correct**: Logs the user in using login_user(user, remember=True). Redirects the user to the user_info route.
   - **If Password is Incorrect**: Displays an error message using flash().
3. **Error Message:** If the user exists but the password is incorrect, a message is shown: ("The user already exists, incorrect password.").

Check Password Length

```
elif len(password) < 7:
    flash('كلمة السر يجب أن تكون مكونة على الأقل من 7 محارف', category='error')
```

Ensures that the provided password is at least 7 characters long.

- If not, displays an error message: "كلمة السر يجب أن تكون مكونة على الأقل من 7 محارف"

Create a New User and Wallet

```
else:
```

```
new_user = User(email=email, username=username, password_hash=generate_password_hash(password, method='pbkdf2:sha256'))

db.session.add(new_user)

db.session.commit()

new_w = create_wallet(new_user)

db.session.add(new_w)

db.session.commit()
```

1. **Create a New User**: Initializes a User object with the provided email, username, and hashed password.
    - **generate_password_hash()**: Hashes the user's password securely before storing it in the database. Adds the new user to the database using db.session.add() and commits the transaction.
2. **Create a Wallet**: Calls create_wallet(new_user) to generate a new wallet for the user. Adds the wallet to the database and commits the transaction.

Log In the New User

```
logout_user()

login_user(new_user, remember=True)

flash('تم إنشاء المحفظة بنجاح', category='success')

return redirect(url_for('views.user_info'))
```

1. **Log Out Any Existing Session**: Uses logout_user() to ensure no user is logged in before proceeding.

2. **Log In the New User**: Logs in the newly created user with login_user.
3. **Flash Success Message**: Displays a success message: "تم إنشاء المحفظة بنجاح" (Translation: "The wallet was successfully created.").
4. **Redirect to User Info**: Redirects the user to the user_info route to show wallet and transaction details.

Handling the GET Request

```
return render_template("new_wallet.html", user=current_user)
```

For GET requests, renders the new_wallet.html template. Passes the current_user object to the template to provide session-specific data.

4.3.2. Views.py

Views.py defines a set of routes for a Flask application using the Blueprint object views. These routes handle rendering templates, fetching data, and creating blockchain transactions.

Blueprint Creation

```
views = Blueprint('views', __name__)
```

A Flask Blueprint groups related routes (views) under a single object. In this case, routes related to blockchain and user interactions are grouped under views. The __name__ argument helps Flask locate templates and static files relative to this blueprint. This allows modular organization of the application, making it easier to manage routes.

Route: / (Home/Index)

```
@views.route('/', methods=['GET', 'POST'])
```

```
#@login_required
def index():
    ## Blockchain Home
    transactions = Transaction.query.filter_by(sender_wallet_id=current_user.wallet.id).all()
    blocks = Block.query.filter_by().all()
    return render_template('index.html', user=current_user, blocks=blocks)
```

1. **@views.route('/', methods=['GET', 'POST'])**: Defines the / route (home page of the application). Accepts both GET and POST requests, though this function primarily handles GET.
2. **#@login_required (Commented Out):** If uncommented, this decorator would ensure only logged-in users could access this route. Currently, any user can access this route.
3. **Fetch User Transactions:**

```
transactions = Transaction.query.filter_by(sender_wallet_id=current_user.wallet.id).all()
```

Retrieves all transactions where the logged-in user's wallet is the sender.

- **current_user**: Provided by Flask-Login, it represents the currently logged-in user.
- **current_user.wallet.id**: Refers to the wallet ID associated with the logged-in user.
- **Transaction.query.filter_by()**: Queries the Transaction table in the database, filtering by the sender's wallet ID.

4. **Fetch Blocks:**

```
blocks = Block.query.filter_by().all()
```

Fetches all blocks in the blockchain using the Block model.

5. Render Template:

```
return render_template('index.html', user=current_user, blocks=blocks)
```

Renders the index.html template, passing the current_user and the retrieved blocks. Displays the blockchain home page. Shows all blocks in the blockchain and the logged-in user's transactions.

Route: /users_pk (View Users and Public Keys)

```
@views.route('/users_pk', methods=['GET', 'POST'])
#@login_required
def users_pk():
    ## Blockchain Home
    users = User.query.all()
    return render_template('users_pk.html', users=users)
```

1. **@views.route('/users_pk', methods=['GET', 'POST'])**: Defines the /users_pk route for listing users and their public keys.
2. **Fetch All Users**:

```
users = User.query.all()
```

Retrieves all users from the User table in the database.

3. Render Template:

```
return render_template('users_pk.html', users=users)
```

Renders the users_pk.html template, passing the list of users. Displays a page where all users and their associated public keys are listed. Useful for identifying recipients for transactions.

Route: /user_info (User Wallet and Transaction Details)

```
@views.route('/user_info')
def user_info():
    transactions = Transaction.query.filter_by(sender_wallet_id=current_user.wallet.id).all()
    return render_template('user_info.html', user=current_user, transactions=transactions)
```

1. **@views.route('/user_info')**: Defines the /user_info route for viewing details about the logged-in user.
2. **Fetch User Transactions**:

```
transactions = Transaction.query.filter_by(sender_wallet_id=current_user.wallet.id).all()
```

Retrieves all transactions where the logged-in user's wallet is the sender.

3. Render Template:

```
return render_template('user_info.html', user=current_user, transactions=transactions)
```

Renders the user_info.html template, passing the current_user and their transactions. Displays detailed information about the logged-in user's wallet and outgoing transactions.

Route: /transaction (Create a Transaction)

```
@views.route('/transaction', methods=['GET', 'POST'])
```

```
def create_transaction():
```

1. Handling POST Request:

```
if request.method == 'POST':
    sender_wallet = current_user.wallet
    receiver_public_key = request.form['receiver_public_key']
    amount = float(request.form['amount'])
```

Form Data: Retrieves the recipient's public key (receiver_public_key) and the transaction amount from the submitted form.

2. Fetch Recipient Wallet:

```
receiver_wallet = Wallet.query.filter_by(public_key=receiver_public_key).first()
```

Queries the Wallet table to find the wallet associated with the recipient's public key.

3. Validation:

```
if not receiver_wallet or sender_wallet.balance < amount:
    flash('فشلت المناقلة', 'danger')
    return redirect(url_for('create_transaction'))
```

Ensures the recipient's wallet exists and that the sender has enough balance. If validation fails, flashes an error message and redirects back to the transaction page.

4. Create Transaction and Blockchain Block:

```
Trans = {
```

```
    'sender': current_user.username,
    'sender_public_key': sender_wallet.public_key,
    'receiver_public_key': receiver_public_key,
    'amount': amount,
}
validity = True  # Placeholder for blockchain validation
```

Transaction Data: Creates a dictionary containing the transaction details.

Blockchain Validation: The validity variable is a placeholder for blockchain validation (e.g., is_chain_valid()).

5. Block Creation:

```
if validity:
    new_block = create_block(Trans)
    print(new_block)
```

create_block(Trans): Calls the create_block function to create a new blockchain block for the transaction.

6. Add Transaction to Database:

```
transaction = Transaction(
    sender_wallet_id=sender_wallet.id,
    receiver_wallet_id=receiver_wallet.id,
    amount=amount,
    transaction_hash=generate_transaction_hash()
)
```

```
db.session.add(transaction)

db.session.commit()
```

Transaction Object: Creates a Transaction object with details including:

- Sender and receiver wallet IDs.
- Transaction amount.
- A unique transaction hash.

Database Commit: Adds the Transaction object to the database and commits the transaction.

Update Wallet Balances:

```
sender_wallet.balance -= amount

receiver_wallet.balance += amount
```

Deducts the transaction amount from the sender's wallet and adds it to the receiver's wallet.

7. Add Block to Database:

```
block = Block(
    block_number=new_block['block_number'],
    transaction_id=transaction.id,
    previous_hash=new_block['previous_hash'],
    current_hash=new_block['current_hash'],
    timestamp=transaction.timestamp,
    proof=new_block['proof']
)
```

```
db.session.add(block)

db.session.commit()
```

Block Object: Creates a Block object with details from the newly created block, linking it to the transaction.

Database Commit: Adds the Block object to the database and commits the transaction.

8. Success and Redirect:

```
flash('تمت المناقلة بنجاح', 'success')

return redirect(url_for('views.index'))
```

Flashes a success message and redirects to the home page.

Handling GET Request:

```
return render_template('transaction.html', user=current_user)
```

Renders the transaction.html template for GET requests.

- Allows users to create transactions.
- Updates the blockchain by adding a new block for each transaction.

4.4. Front-end Design

In the modern digital era, web pages serve as the primary interface between users and online services. A well-designed front-end not only enhances user experience but also ensures seamless interaction with the underlying functionality of a website. The front-end of a web application is crafted using HTML, CSS, and JavaScript, often complemented by powerful frameworks like Bootstrap. These tools enable developers to create responsive, visually appealing, and user-friendly interfaces that adapt to various devices and screen sizes. By leveraging the Bootstrap framework,

designers can implement elegant layouts, consistent styling, and interactive components that elevate the overall aesthetic and usability of a web page.

4.4.1. Layout page

This page represents a dynamic template for a web page, designed to integrate seamlessly with a back-end system. It employs a templating engine (likely Jinja2, based on the syntax) to render content dynamically, ensuring flexibility and personalization for each user.

1. **Title Block** : The **{% block title %}** section allows the inclusion of a dynamic title for the web page. This ensures that the title can be customized for different pages while maintaining a consistent structure across the application.

2. **User-Specific Content** : The **{% if current_user %}** block checks whether a user is logged in. If a user is authenticated, their username is displayed with a personalized greeting ("مرحبا {{ current_user.username }}"). This feature enhances user engagement by creating a tailored experience.

3. **Flash Messages** : The **{% with messages = get_flashed_messages(with_categories=true) %}** section handles feedback messages, such as success notifications or error alerts. These messages are categorized (e.g., "error" or other types), allowing the front-end to display them appropriately. For instance, error messages might be styled differently to grab the user's attention, ensuring clarity and effective communication.

4. **Dynamic Content Block** : The **{% block content %}** section serves as a placeholder for the main content of the web page. This design enables developers to inject specific content for each page while maintaining a consistent layout and structure across the application.

5. **Integration with Back-End** : The template demonstrates a strong connection between the front-end and back-end systems. By utilizing dynamic placeholders and conditional logic, the template adapts its behavior based on user authentication status and server-side data, ensuring a cohesive and interactive user experience.

6. **Responsive Design with Bootstrap** : Although the file does not explicitly include Bootstrap classes, the use of such frameworks is implied in the design philosophy. Bootstrap ensures that the web page is responsive, adapting gracefully to different devices and screen sizes. This responsiveness is crucial for delivering a consistent experience across desktops, tablets, and mobile devices.

4.4.2. Index page

It’s a template for the homepage of a web application. It extends the base layout defined in **layout.html** and fills in specific content blocks to create a dynamic and interactive user interface. Below is a detailed explanation of its behavior and structure:

1. Extending the Base Layout: The file begins with **{% extends 'layout.html' %}**, indicating that it inherits the structure and styling from the **layout.html** file. This inheritance allows the homepage to maintain consistency with the overall design and functionality of the application while adding or overriding specific sections.

2. Dynamic Title Block: The **{% block title %}** section is overridden to set the title of the page as "الصفحة الرئيسية" (Home Page). This ensures that the browser tab displays a meaningful and context-specific title for the homepage.

3. User-Specific Content: Within the **{% block content %}** section, the file provides personalized content for the logged-in user:

- **Welcome Message** : The user is greeted with "أهلاً بك، {{ user.username }}", dynamically displaying their username.
- **Wallet Balance** : The current balance of the user's wallet is displayed as "رصيد المحفظة الحالي: {{ user.wallet.balance }} وحدة". This information is fetched dynamically from the back-end, ensuring real-time accuracy.

4. Blockchain Data Display: The file includes a table-like structure to display the contents of a blockchain. Each block in the blockchain is represented with detailed information:

- **Block Number** : Displays the unique identifier of the block (**{{ block.block_number }}**).
- **Sender and Receiver** : Shows the usernames of the sender and receiver wallets (**{{ block.transaction.sender_wallet.user.username }}** and **{{ block.transaction.receiver_wallet.user.username }}**).
- **Transaction Amount** : Indicates the amount transferred in the transaction (**{{ block.transaction.amount }} وحدة**).
- **Hashes** :
  - **Previous Hash** : Displays the hash of the previous block (**{{ block.previous_hash }}**), ensuring the integrity of the blockchain.
  - **Current Hash** : Shows the hash of the current block (**{{ block.current_hash }}**), which can be toggled to reveal or hide the hash value.
  - **Proof** : Displays the proof-of-work value (**{{ block.proof }}**) associated with the block.

- **Timestamp** : Shows the date and time of the transaction (**{{ block.transaction.timestamp }}**).

This section iterates over a list of blocks (**{% for block in blocks %}**) provided by the back-end, dynamically rendering the data for each block in the blockchain.

5. Interactive Features: Show/Hide Hash : The phrase "إظهار التجزئة" (Show Hash) suggests an interactive feature where users can toggle the visibility of the current block's hash. This enhances user engagement and provides control over the level of detail displayed.

- **New Transaction Button** : The phrase "إجراء مناقلة جديدة" (Perform New Transfer) likely serves as a button or link to initiate a new blockchain transaction. This feature enables users to interact with the blockchain directly from the homepage.

6. Integration with Back-End: The file demonstrates strong integration with the back-end system:

- **Dynamic Data Rendering** : All content, including the user's wallet balance, blockchain data, and transaction details, is dynamically fetched and rendered based on the back-end data.
- **Scalability** : The use of loops (**{% for block in blocks %}**) ensures that the template can handle any number of blocks in the blockchain, making it scalable and adaptable to growing datasets.

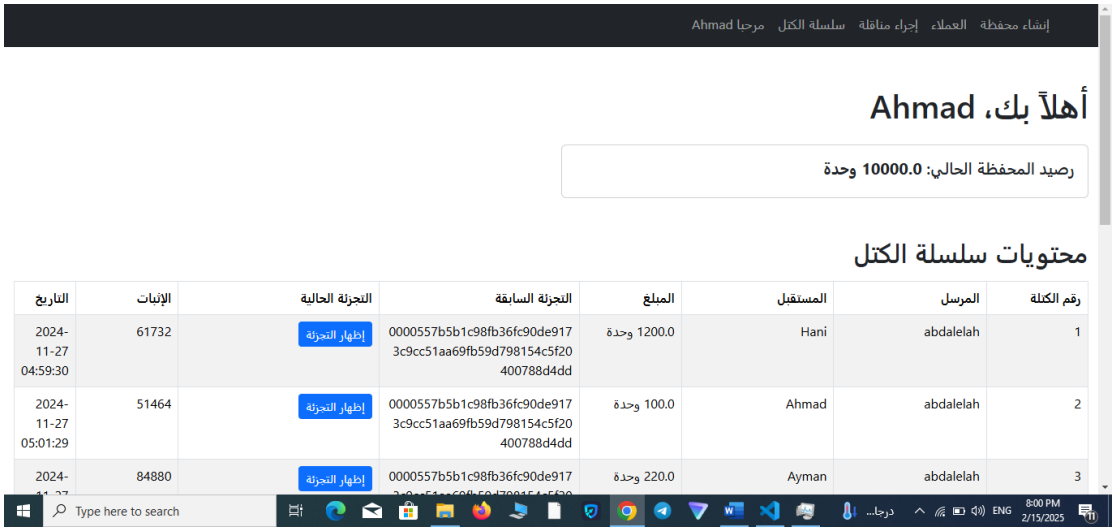


Figure 4-1: Index Page

4.4.3. New_wallet page

A template for a web page designed to allow users to create a new wallet within the application. It extends the base layout defined in **layout.html** and fills in specific content blocks to provide a focused and user-friendly interface for wallet creation. Below is a detailed explanation of its behavior and structure:

1. Extending the Base Layout: The file begins with **{% extends 'layout.html' %}**, indicating that it inherits the structure and styling from the **layout.html** file. This ensures consistency with the overall design and functionality of the application while adding or overriding specific sections.

2. Dynamic Title Block: The **{% block title %}** section is overridden to set the title of the page as "اسم المستخدم" (Username). This ensures that the browser tab displays a meaningful and context-specific title for the wallet creation page.

3. Content Block for Wallet Creation: Within the **{% block content %}** section, the file provides a form-like structure for users to input their details and create a new wallet:

- **Header** : The phrase "إنشاء محفظة جديدة" (Create New Wallet) serves as a clear and descriptive heading for the page.
- **Input Fields** :
  - **Username** : A field labeled "اسم المستخدم" (Username) allows users to enter their desired username.
  - **Email** : A field labeled "الإيميل" (Email) collects the user's email address.
  - **Password** : A field labeled "كلمة المرور" (Password) allows users to set a secure password for their account.
- **Submit Button** : The phrase "إنشاء المحفظة" (Create Wallet) likely serves as a button or link to submit the form and finalize the wallet creation process.

This structure ensures that users can easily provide the necessary information to create a new wallet, streamlining the onboarding process.

4. Integration with Back-End: The file demonstrates strong integration with the back-end system:

- **Form Submission** : While not explicitly shown, the input fields and submit button suggest that the data entered by the user will be sent to the back-end for processing. This includes validation of the inputs, creation of the wallet, and storage of the user's details in the database.

- **User Experience** : By providing clear labels and a straightforward layout, the template ensures a smooth and intuitive user experience during the wallet creation process.

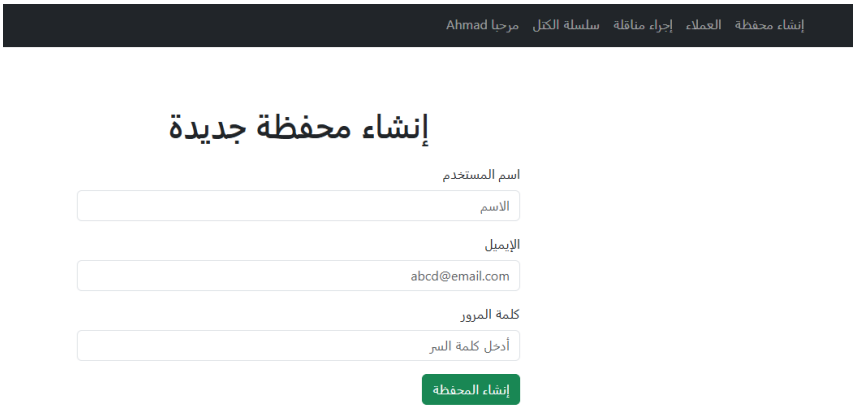


Figure 4-2: New_wallet Page

4.4.4. Transaction page

a template for a web page designed to allow users to initiate a new transaction within the application. It extends the base layout defined in **layout.html** and fills in specific content blocks to provide a focused and user-friendly interface for creating transactions. Below is a detailed explanation of its behavior and structure:

1. Extending the Base Layout: The file begins with **{% extends 'layout.html' %}**, indicating that it inherits the structure and styling from the **layout.html** file. This ensures consistency with the overall design and functionality of the application while adding or overriding specific sections.

2. Dynamic Title Block: The **{% block title %}** section is overridden to set the title of the page as "إجراء مناقلة" (Perform Transfer). This ensures that the browser tab displays a meaningful and context-specific title for the transaction page.

3. Content Block for Transaction Creation: Within the **{% block content %}** section, the file provides a form-like structure for users to input the details required to initiate a new transaction:

- **Header** : The phrase "إرسال مناقلة جديدة" (Send New Transfer) serves as a clear and descriptive heading for the page.
- **Input Fields** :
  - **Recipient's Public Key** : A field labeled "المفتاح العام للمستقبل" (Recipient's Public Key) allows users to enter the public key of the recipient's wallet. This ensures that the transaction is directed to the correct wallet.
  - **Amount** : A field labeled "المبلغ" (Amount) collects the amount of units the user wishes to transfer.
- **Submit Button** : The phrase "إرسال المعاملة" (Send Transaction) likely serves as a button or link to submit the form and finalize the transaction process.

This structure ensures that users can easily provide the necessary information to initiate a transaction, streamlining the process of transferring funds.

4. Integration with Back-End

The file demonstrates strong integration with the back-end system:

- **Form Submission** : While not explicitly shown, the input fields and submit button suggest that the data entered by the user will be sent to the back-end for processing. This includes validation of the inputs, verification of the recipient's public key, deduction of the specified amount from the sender's wallet, and updating the blockchain with the new transaction.

- **User Experience** : By providing clear labels and a straightforward layout, the template ensures a smooth and intuitive user experience during the transaction creation process.



Figure 4-3: New_Transaction Page

4.4.5. Users page

A template for a web page designed to display detailed information about a user, including their wallet balance and transaction history. It extends the base layout defined in **layout.html** and fills in specific content blocks to provide a focused and user-friendly interface for viewing user-specific data. Below is a detailed explanation of its behavior and structure:

1. Extending the Base Layout: The file begins with **{% extends 'layout.html' %}**, indicating that it inherits the structure and styling from the **layout.html** file. This ensures consistency with the overall design and functionality of the application while adding or overriding specific sections.

2. Dynamic Title Block: The **{% block title %}** section is overridden to set the title of the page as "معلومات المستخدم" (User Information). This ensures that the browser tab displays a meaningful and context-specific title for the user information page.

3. Content Block for User Information: Within the **{% block content %}** section, the file provides a structured display of the user's details and transaction history:

- **Welcome Message** : The user is greeted with "أهلاً بك، {{ user.username }}", dynamically displaying their username.
- **Wallet Balance** : The current balance of the user's wallet is displayed as "رصيد المحفظة: {{ user.wallet.balance }} وحدة". This information is fetched dynamically from the back-end, ensuring real-time accuracy.
- **Transaction History** :
  - A table-like structure is used to display the user's previous transactions. Each transaction includes:
    - **Transaction ID** : Displays the unique identifier of the transaction (**{{ transaction.id }}**).
    - **Sender and Receiver** : Shows the usernames of the sender and receiver wallets (**{{ transaction.sender_wallet.user.username }}** and **{{ transaction.receiver_wallet.user.username }}**).
    - **Transaction Amount** : Indicates the amount transferred in the transaction (**{{ transaction.amount }} وحدة**).
    - **Timestamp** : Shows the date and time of the transaction (**{{ transaction.timestamp }}**).
  - The file iterates over a list of transactions (**{% for transaction in transactions %}**) provided by the back-end, dynamically rendering the data for each transaction in the user's history.

4. Interactive Features: New Transaction Button : The phrase "إجراء مناقلة جديدة" (Perform New Transfer) likely serves as a button or link to initiate a new blockchain transaction. This feature enables users to interact with the blockchain directly from the user information page.

5. Integration with Back-End

The file demonstrates strong integration with the back-end system:

- **Dynamic Data Rendering** : All content, including the user's wallet balance and transaction history, is dynamically fetched and rendered based on the back-end data.
- **Scalability** : The use of loops (**{% for transaction in transactions %}**) ensures that the template can handle any number of transactions, making it scalable and adaptable to growing datasets.



Figure 4-4: Users Page

4.4.6. User_info page

A template for a web page designed to display a list of users along with their associated details, such as their username, email, and public key. It extends the base layout defined in **layout.html** and fills in specific content

blocks to provide a focused and user-friendly interface for viewing user information. Below is a detailed explanation of its behavior and structure:

1. Extending the Base Layout: The file begins with **{% extends 'layout.html' %}**, indicating that it inherits the structure and styling from the **layout.html** file. This ensures consistency with the overall design and functionality of the application while adding or overriding specific sections.

2. Dynamic Title Block: The **{% block title %}** section is overridden to set the title of the page as "معلومات المستخدمين" (User Information). This ensures that the browser tab displays a meaningful and context-specific title for the user information page.

3. Content Block for User Information: Within the **{% block content %}** section, the file provides a structured display of user details:

- **Header** : The table headers include "اسم المستخدم" (Username), "الإيميل" (Email), and "المفتاح العام" (Public Key), providing a clear and organized way to present user information.
- **User Details** :
    - A loop (**{% for user in users %}**) iterates over a list of users provided by the back-end, dynamically rendering the data for each user.
    - For each user, the following information is displayed:
        - **Username** : Displays the username of the user (**{{ user.username }}**).
        - **Email** : Shows the email address associated with the user (**{{ user.email }}**).

- **Public Key** : Displays the public key of the user's wallet (**{{ user.wallet.public_key }}**).

This structure ensures that users can easily view and access the details of all registered users in a clean and organized format.

4. Integration with Back-End

The file demonstrates strong integration with the back-end system:

- **Dynamic Data Rendering** : All content, including usernames, emails, and public keys, is dynamically fetched and rendered based on the back-end data.
- **Scalability** : The use of loops (**{% for user in users %}**) ensures that the template can handle any number of users, making it scalable and adaptable to growing datasets.

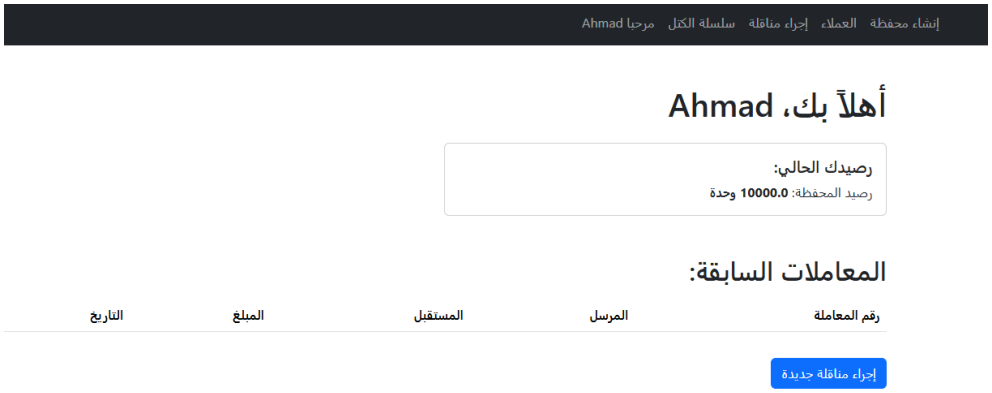


Figure 4-5: Users Page

Conclusion

The development of the Blockchain-based Cryptocurrency Exchange Platform has successfully demonstrated the practical application of blockchain technology in creating a secure and efficient system for managing digital assets. The platform achieves its objectives by providing a user-friendly interface for registering new users, displaying blockchain content, and facilitating cryptocurrency transactions. Key outcomes of the project include:

- The successful implementation of a blockchain ledger that records all transactions in a transparent and tamper-proof manner.
- The ability to perform secure cryptocurrency transfers between users, verified through cryptographic techniques.
- A responsive and intuitive user interface that enhances the overall user experience.

This project highlights the transformative potential of blockchain technology in addressing cybersecurity challenges and fostering trust in digital systems. By enabling secure and transparent transactions, the platform paves the way for broader adoption of blockchain-based solutions in various domains. The results of this project underscore the importance of continued research and innovation in blockchain technology, as it holds the promise of reshaping the future of digital interactions.

Future works

While the current implementation of the *Blockchain-based Cryptocurrency Exchange Platform* serves as a robust proof of concept, there are several areas for future work to enhance its functionality and prepare it for deployment in real-world networks. These enhancements will ensure that the platform is scalable, secure, and capable of handling the demands of real users and transactions. Below are some key areas for future development:

1. Scalability Improvements:

- **Sharding:** Implement sharding techniques to partition the blockchain into smaller, more manageable pieces. This will allow the platform to handle a larger number of transactions without compromising performance.
- **Layer-2 Solutions:** Integrate Layer-2 scaling solutions such as the Lightning Network or state channels to process off-chain transactions while maintaining the security of the main blockchain.
- **Optimized Consensus Mechanisms:** Explore alternative consensus mechanisms like Proof of Stake (PoS) or Delegated Proof of Stake (DPoS) to reduce energy consumption and improve transaction throughput.

2. Enhanced Security Measures

- **Multi-Factor Authentication (MFA):** Add multi-factor authentication for user logins to provide an additional layer of security against unauthorized access.

- **Smart Contract Audits:** If smart contracts are introduced in the future, conduct thorough audits to identify and mitigate vulnerabilities.
- **Encryption Enhancements:** Strengthen encryption protocols for data transmission and storage to protect sensitive information from cyberattacks.

3. Integration with Real-World Payment Systems

- **Fiat-to-Crypto Gateways:** Develop integration with fiat currencies to allow users to convert traditional money into cryptocurrency and vice versa. This will make the platform more accessible to a broader audience.
- **API Integration:** Provide APIs for third-party services, enabling seamless integration with external applications such as e-commerce platforms, banking systems, and payment gateways.

4. User Experience Enhancements

- **Mobile Application:** Develop a mobile application to provide users with on-the-go access to the platform. The app should include features like push notifications for transaction confirmations and biometric login options.
- **Advanced Analytics Dashboard:** Create a dashboard that provides users with insights into their transaction history, wallet balance trends, and blockchain activity.

5. Regulatory Compliance

- **KYC/AML Implementation:** Incorporate Know Your Customer (KYC) and Anti-Money Laundering (AML) procedures to comply

with regulatory requirements. This will involve verifying user identities and monitoring suspicious activities.

- **Data Privacy Compliance:** Ensure compliance with data privacy regulations such as GDPR or CCPA by implementing features like data anonymization and user consent management.

6. Network Deployment and Monitoring

- **Cloud Hosting:** Deploy the platform on cloud infrastructure such as AWS, Google Cloud, or Azure to ensure high availability, scalability, and reliability.
- **Real-Time Monitoring:** Implement real-time monitoring tools to track network performance, detect anomalies, and prevent downtime. Tools like Prometheus and Grafana can be used for this purpose.
- **Disaster Recovery Plan:** Develop a disaster recovery plan to ensure data integrity and system availability in case of unexpected failures or attacks.

7. Community and Ecosystem Development

- **Token Rewards:** Introduce a native token reward system to incentivize users for participating in the network, such as validating transactions or referring new users.
- **Decentralized Governance:** Enable decentralized governance by allowing users to vote on platform upgrades and policy changes using their tokens.

8. Cross-Chain Interoperability

- **Interoperability Protocols:** Implement cross-chain interoperability protocols like Polkadot or Cosmos to enable seamless transactions between different blockchain networks.
- **Atomic Swaps:** Support atomic swaps to allow users to exchange cryptocurrencies across different blockchains without relying on intermediaries.

9. Testing and Optimization

- **Stress Testing:** Conduct stress testing to evaluate the platform's performance under high transaction volumes and identify bottlenecks.
- **Bug Bounty Program:** Launch a bug bounty program to encourage ethical hackers to identify and report vulnerabilities in the platform.

References

[1] Diogenes & Ozkaya, “Cybersecurity--Attack and Defense Strategies”. Packt Publishing. 2018.

[2] Calder, Alan. “NIST Cybersecurity Framework: A Pocket Guide”. IT Governance Publishing Ltd. 2018.

[3] Alsmatti, Izzat. The NICE Cybersecurity Framework. Springer. 2019

[4] NIST. “Framework for Improving Critical Infrastructure Cybersecurity v1.1”. NIST. Retrieved 19 Aug 2019.

[5] X. Li, P. Jiang, T. Chen, X. Luo, and Q. Wen, “A survey on the security of blockchain systems,” Futur. Gener. Comput. Syst., no. Xiaoqi Li, pp. 1–25, 2017.

[6] C. Lin and T. C. Liao, “A survey of blockchain security issues and challenges,” Int. J. Netw. Secur., 2017.

[7] M. “Antonopoulos, Mastering Bitcoin: unlocking digital cryptocurrencies.” O’Reilly Media, Inc., 2015.

[8] S. Nakamoto and others, “Bitcoin: a peer-to-peer electronic cash system (2008).” 2008.

[9] L. Luu, D. H. Chu, H. Olickel, P. Saxena, and A. Hobor, “Making smart contracts smarter,” in Proceedings of the ACM Conference on Computer and Communications Security, 2016.

[10] Manuel Mogollon, “Cryptography and Security Services: Mechanisms and Applications”, CyberTech Publishing, University of Dallas, USA, 2007.

[11] “An Introduction to Cryptography”, version 8.0, PGP Corporation, Oct. 2002.

[12] R.L. Rivest, A. Shamir, and L. Adleman, “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”.

[13] William Stallings, “Cryptography and Network Security Principles and Practices”, Fourth Edition, Prentice Hall, November 16, 2005.

[14] Robert Plummer, “Number Theory: Theorems, Examples, and RSA”, CS103A, Handout #40, October 31, 2008.