VFAST Transactions on Software Engineering
https://vfast.org/journals/index.php/VTSE@ 2026, ISSN(e): 2309-3978, ISSN(p): 2411-6246
Volume 14, Number 2, April-June 2026 *pp:62-80*

Refine Security Control Protocols for Block chain in Textile Industry Supply Chain Management

Asif Raza[1*], **Salahuddin**[2], **Sadia Latif**[1], **Ghazanfar Ali**[3]

[1]Department of Computer Science, Bahauddin Zakariya University Multan, Pakistan; [2]Department of Computer Science, NFC Institute of Engineering and Technology: Multan, Pakistan; [3]Department of Computer Science, University of South Asia, Lahore Cantt, Pakistan

Keywords:*Block chain Technology, Supply Chain Management, Security, Decentralize.*

Journal Info:
Submitted:
March 18, 2026
Accepted:
April 17, 2026
Published:
April 22, 2026

Abstract Cotton is a vital crop with numerous environmentally friendly applications in our daily lives, making it a globally significant agricultural commodity. Pakistan ranks as the world's third-largest producer and consumer of cotton, utilizing 15% of its land for cultivation. However, challenges like supply chain malpractice and opacity lead to economic losses for farmers, textile industries, and governments. To tackle these issues, a block chain-based framework for supply chain traceability and reliability in the cotton and textile industry has been developed. This innovative solution aims to create transparency and trust among supply chain participants. This solution introduces Cotton Coin (CC) for transaction tracking and utilizes smart contracts for monitoring trading and currency conversion. The Inter Planetary File System (IPFS) securely stores encrypted data of supply chain participants. The experimental results demonstrate the effectiveness of the suggested framework in comparison to existing supply chain projects. Key performance metrics, such as new block latency, transactions per minute, average gas charges, and transaction verification times, have shown significant improvements. These findings highlight the potential of this block chain-based solution to enhance supply chain transparency, security, and efficiency in the cotton and textile industry.

***Correspondence author email address:** asifraza.raza14@gmail.com
DOI: 10.21015/vtse.v14i2.2370

1 Introduction

Cotton is a flowering plant of the genus Gossypium, and is grown to produce fibers which are important to the textile industry worldwide. A significant part of the apparel production in the world is 75 percent of cotton. At the moment, the global yield of cotton is an impressive 25 million tons, which is a precious agricultural crop. China, India, the United States, Brazil, and Pakistan are the five biggest cotton-producing nations which represents the importance of the crop in other parts of the world. Other than being used in the textile industry, cotton has been found to be a multi-purpose resource with environmentally friendly byproducts. These include the production of cardboard, paper, fertilizers, cooking oil, animal feed, as well as contributing to personal hygiene and cosmetics products.

Cotton is a luxurious and soft natural material, normally refined into thread or strand and is appreciated because of its potential to create breathable and posh fabrics. It is a vital part of keeping the textile industry alive and developing, as well as a highly profitable cash crop to many farmers and other agricultural businesses. Over

100 countries have over the past 100 years succeeded in cotton production with China, India, the United States, and Pakistan all contributing to about 75 percent of the total global cotton production [1]. In 2013-14, world cotton production was 26.23 million tones, which greatly decreased to 22.03 million tones in the year 2015-16, according to [2].

This massive reduction in cotton production across the world has been mainly credited to two key issues; climate change and uncontrolled use of resources. The definition of sustainability is the continuous process of making sure that changes are made in the way that best suits the capability to serve the current and future human needs and aspirations. This complex idea includes a wise use of resources, distribution of investments, the direction of technological progress and the reorganization of institutions. Generally, the concept of sustainability is associated with the integration of three main paradigms which include social, economic and environmental paradigms which are essential in the quest to achieve a sustainable future as expounded by Yang. The aim of sustainability objectives is to drive progress in the realms of societal improvement, economic expansion, safeguarding the environment, and responsible utilization of natural resources, as indicated by Vasileiou. Sustainable development, in its turn, is aimed at satisfying the current needs and protecting the possibilities of the future generations to satisfy theirs [3].

The efficient and effective transportation of raw materials plays a crucial role in sustainable production and supply chain management in the textile industry. Sustainable cotton supply chain is very important in conserving both the social, economic and environmental health of local communities and also in maintaining a safe, efficient and competitive production process [4]. Notably, farm-level production contributes to 56% of the total emissions in the cotton supply chain cycle, making it a pivotal aspect of sustainability and its impact on the environment, society, and health. The rest of the emissions are shared between other supply chain players, such as weaving, dyeing, transportation, and ginning, with all these activities contributing to the rest two-thirds of the emissions [5].

Block chain is a convenient and unchangeable register in the context of the supply chain management, provides a decentralized and transparent method of supply chain management that lacks central entities. Information processing on the block chain is very efficient and promotes cooperative activities. Comparing block chain technology to the traditional centralized systems, a multitude of benefits can be identified such as an increased degree of efficiency, improved security, and increased transparency [6–9].

This study, as referenced in [10–16], examines the potential of block chain technology to enhance sustainability and traceability within the textile supply chain management. The focus is on evaluating the specific requirements essential for implementing such a system that involves multiple stakeholders. In various industries, block chain technology has been increasingly adopted as a supply chain solution, offering a multitude of benefits. These advantages encompass enhanced data and information transparency, increased trustworthiness, improved traceability, and cost-effectiveness. Pakistan ranks sixth globally in cotton farming, but the fluidity of cotton production, processing, and consumption poses significant challenges for accurate information management and traceability. The presence of intermediaries serving as intermediaries between farmers and consumers results in the final cotton and textile products being offered to consumers at unexpectedly high prices [17–20].

Kumar et al. [19] emphasize the importance of ethical sourcing in the textile industry and also shed light on key aspects like transparency and traceability. They argue that when it comes to achieving these goals, block chain technology stands out as the sole reliable and decentralized platform. However, it is noteworthy that they do not provide a specific decentralized approach for securely managing data transactions. Saleh K. and colleagues [3] proposed eliminating the centralized authority as a means to enhance the traceability of soybeans. They stored the transaction ledger on the IPFS file system. Nevertheless, it's worth noting that no experiments were carried out to showcase the framework's performance and system integrity.

Musamih et al. [12] proposed an Ethereum block chain-based approach to enhance the healthcare supply chain. Their strategy incorporated smart contracts to

establish a robust framework for managing medication within the supply chain. However, it is important to note that the block chain technology has its limitations, leading to challenges in terms of efficiency, scalability, data privacy, immutability, and interoperability, which become increasingly prominent in this context or other AI domain such as medical imaging.

Villalva-Catano et al. [16] in their research carried out an analysis of the Peruvian coffee industry to examine the reasons behind the high logistic costs. Their strategy was to make use of ABC costing analysis, enabling them not only to identify the root causes but also suggest possible solutions. However, it is worth mentioning that the issues presented and the ways to address them are majorly associated with centralized processes and organizations, which can result in trust and confidence concerns. Alternatively, a decentralized approach can be a more reliable and better way of getting desired outcomes.

Pandey et al. [17] conducted a thorough analysis of the Additive Manufacturing supply chain and subsequently devised three models aimed at minimizing potential vulnerabilities and introducing a novel category of cyber-physical hazards. These proposed models have all been rigorously tested and incorporate robust safety protocols. However, it is noteworthy that none of these systems embrace decentralization or employ cryptographic techniques to safeguard the integrity and security of the data, rendering them less trustworthy. On a different note, Ferdousi et al. [18] introduced a distributed supply chain paradigm for the management of the US beef cattle supply chain. Under this paradigm, user interactions, communications, and data management are intricately coordinated, particularly when ownership changes hands.

1.1 Problem Formulation

Some companies also engage in the cotton farming season, where they purchase affordable cotton products from textile mills and later sell them at a premium. During the agricultural season, certain unscrupulous individuals stockpile cotton products, which they subsequently sell at inflated prices during the off-season. To address the shortage of cotton products during this period, the government resorts to importing cotton from other nations at high costs.

Despite some governments having clear laws and regulations in place, the current system proves challenging to regulate due to widespread malpractice and a lack of transparency in the supply chain, as illustrated in Figure 1.

The primary issues stemming from the insufficient transparency in the cotton management process include:

- **Disheartened Farmers:** Farmers become discouraged when they receive very low compensation for their hard work.
- **Expensive Cotton Products:** Consumers are forced to purchase expensive cotton goods, including clothing, which is a basic necessity.
- **Economic Impact:** The national economy suffers because the government must subsidize and pay for imported cotton.

The workflow of the current system, as depicted in Figure 1, is as follows:

The foundation of the system consists [21–25] rices to farmers for their products. This hinders farmers from making a profit when selling their crops to textile mills. During the season, textile mills offer cotton to intermediaries or middlemen and export it to other countries. However, during the off-season, they supply the same cotton to distributors at significantly higher prices. Cotton is then sold to retailers after price hikes, and at times, shortages necessitate importing cotton, which is acquired at a steep cost. In the end, after all the price increases, consumers bear the brunt of the expenses, while farmers and customers both suffer losses. The middlemen or stockers are the ones reaping the sole benefits of this current structure. Figure 2, present the whole production procedure of cotton making.

2 Method and Materials

This study aims to leverage cutting-edge block chain technology to establish a robust and transparent cotton traceability system for the textile industry. This system will meticulously track the journey of cotton, ensuring its safety and efficiency from the farm to the final product. By harnessing this innovative technology, retailers will have the opportunity to procure top-quality merchandise at competitive prices, encompassing clothing, threads, and textiles, all of which can be seamlessly

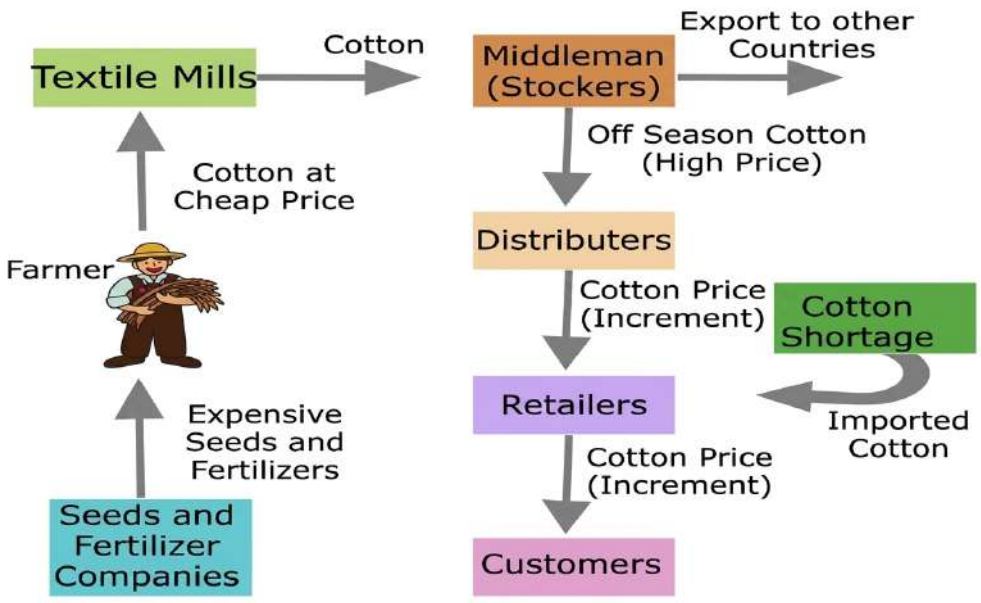

Figure 1. Traditional System

traced back to their inception in the manufacturing process. Once a farmer acquired cotton crop seeds from a Seed Company, all pertinent information, including payment records, was securely documented within a block chain.

Furthermore, the operational protocols were set in motion through the utilization of a smart contract, which employs a tamper-resistant proof-of-work consensus mechanism within the block chain. Transactions and data recorded through these smart contracts were impervious to unauthorized access, modifications, or deletions. Retailers were granted effortless access to the block chain's distributed ledger, enabling them to trace the origins of the cotton seeds, the farms involved in production, dates of crop yields, and intricate details of the distribution network before making any procurement decisions. This comprehensive system also allowed for transparency in monitoring the production processes of textile mills, distribution networks for clothing, fabrics, or thread, retailer acquisitions, pricing information, and all other aspects of the supply chain [25–28].

To regulate the pricing of cotton and its associated products, payments made for cotton purchases are held in a pending status until they receive approval from designated authorities. These authorized entities have the power to oversee and manage transactions by leveraging data from public block chains. In case any irregular or suspicious events occur, legal actions can be taken against individuals involved in fraudulent activities. Figure 7 illustrates the potential application of block chain technology to enhance the existing cotton production system. The proposed system's architecture, depicted in Figure 3, distinguishes between two types of farmers: those equipped with digital wallets and those without. Both groups are depicted as receiving their payments after selling their agricultural produce.

2.1 Proposed CC Block Chain Framework Details

The proposed system serves a wide spectrum of consumers such as government, distributors, retailers, textile mills, farmers, and seed firms. The core element of this framework is based on a decentralized application (DApp), which is tightly connected with the block chain technology, user interface web platform, digital storage, IPFS, and smart contracts. Each user is given a unique public address when registered and uploading data via the Cotton Management System using the web or DApp. This is a public address that is critical in determining all entities involved in the network. IPFS is a mediator that helps the data to flow smoothly between the digital store and the decentralized application [29, 30]. Three major actors or organizations have been presented in the proposed architectural solution: the

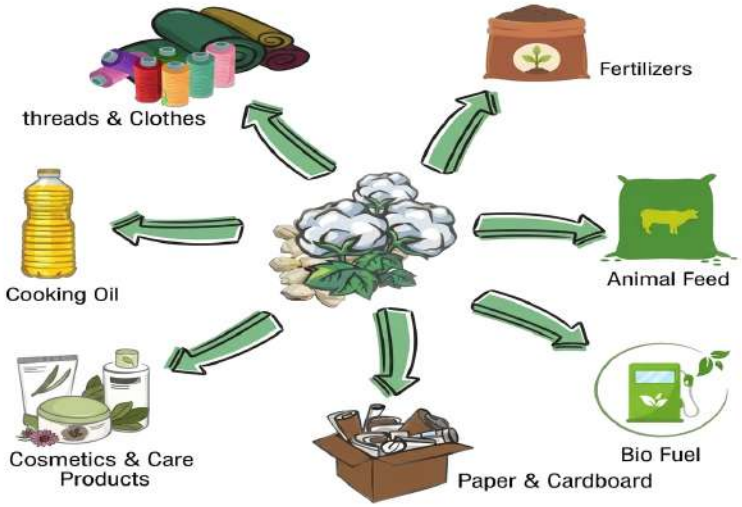

Figure 2. Production Procedure of Cotton making

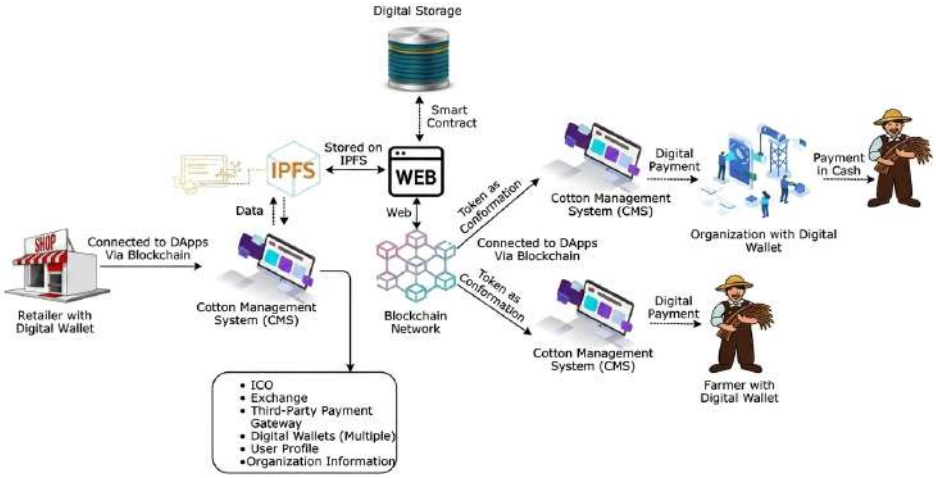

Figure 3. Overview of Security Protocols of Block chain within the Framework of Managing Supply Chains in the Textile Industry Based on CC novel block chain

store, the farmer, and the organization. These parties may communicate with the Cotton Management System (CMS) in a number of ways, such as a mobile application or site powered by a block chain. The CMS also involves many administrators who can be said to be users and are expected to oversee and authorise procedures depending on their given levels of authority.

The development of a DApp of the Cotton Management platform is based on the smooth connection of various payment gateways, where cryptocurrency and traditional fiat currencies can be used. Such an extensive integration will include a broad scope of features that will include support of local and digital currencies, ICOs, the ability to facilitate token-based exchanges, and a variety of digital wallets to effectively manage different tokens and coins. In the Cotton Management Platform, IPFS is used to store and share data and user information by applying smart contracts and utilizing IPFS. To manage its digital storage processes, IPFS uses both public and private keys. In an attempt to give a pictorial illustration, Figure 4 illustrates the architectural structure of this proposed system 8. Overview of CC with ICO via IPFS through DApp Payment verification workflow.

Figure 4 illustrates the functionality of our proposed framework. In this system, farmers equipped with digital wallets receive direct payments from organizations, while those without digital wallets receive payments through intermediaries who possess wallets.

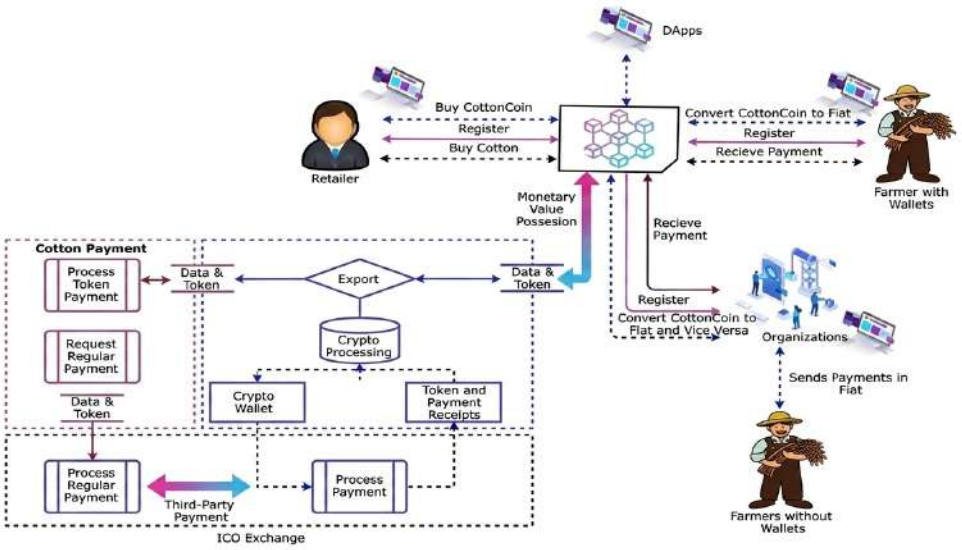

Figure 4. Overview of CC with ICO via IPFS through DApp Payment verification workflow

Explanation of the Figure: Retailers can purchase cotton and its related products using their decentralized application (DApp) accounts, which are connected to digital wallets. Once a transaction is initiated, the payment is transferred to the farmers. Each participant involved in the supply chain, such as textile mills and retailers, incurs a predefined transaction fee. The blockchain ledger automatically records all relevant information whenever a retailer acquires cotton or related products. This includes traceability data covering the entire process from the farm to the final product. In addition, pricing information for cotton sold to retailers is reported to the relevant authorities for transparency and monitoring.

It is important to note that not all farmers possess digital wallets. In such situations, farmer groups that have access to digital wallets can assist other farmers by helping them convert their digital earnings into cash.

2.2 Proposed CC Economic CC Block Chain Framework Details

A novel cryptocurrency named Cotton Coin has been created to revolutionize the transparent, auditable, and secure management of cotton while also controlling its pricing, eliminating intermediaries. With a total circulation of 500 million Cotton Coins, each coin has a fixed value of 100 USD. However, tokens are divisible, allowing users to acquire fractions such as 0.1CC for 10 USD, without any specific purchase limit.

Cotton Coin (CC) can be acquired using various digital currencies like BTC, ETH, and BNB, or with local currencies such as PKR, USD, EUR, and more. Being part of a decentralized block chain platform, businesses can obtain Cotton Coin without incurring taxes or exchange fees, receiving an equivalent quantity of tokens for their payment instantly. Retailers' wallets are credited with the corresponding balance immediately after their Cotton Coin purchase. With a seamless 100% currency conversion rate between Cotton Coin and other tokens, retailers can use these tokens to procure cotton directly from distributors, bypassing the need for third-party applications or exchanges and avoiding additional fees. Initially distributed through an ICO, Cotton Coin was established as an ERC20 token on the Ethereum network.

Explanation of the figure 5 is following:

- Farmers who possess digital wallets have the opportunity to receive direct payments in *Cotton Coin* and subsequently convert these tokens into traditional currency through an ICO exchange. Alternatively, farmers can access funds by engaging with government-approved entities. These organizations can accept payments in *Cotton Coin* through their digital wallets, convert them into fiat currency via an ICO, and subsequently distribute the funds to the farmers. Each time a farmer receives payment, the blockchain records proof of the transaction, often in the form of an image or scanned receipt. This record serves as reliable

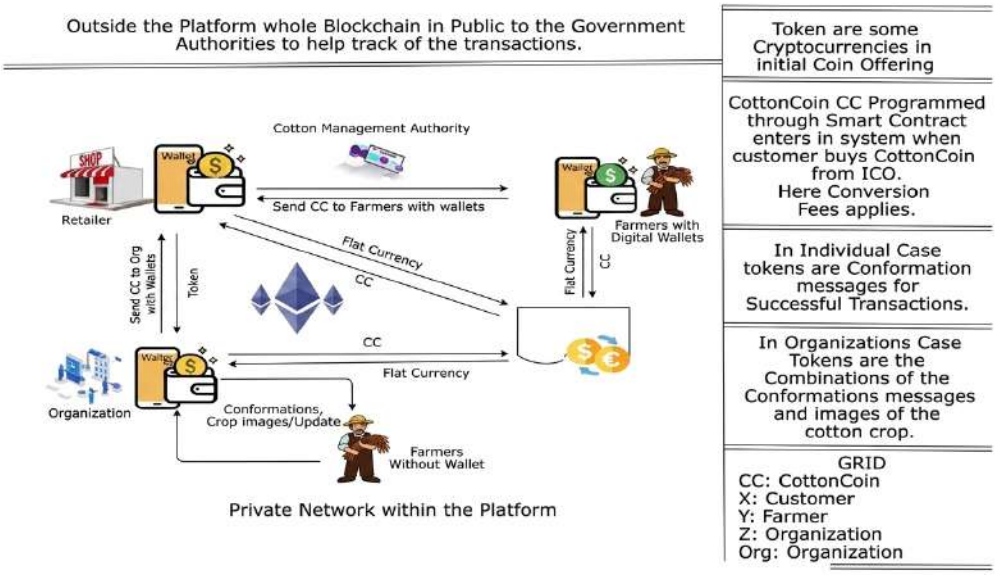

Figure 5. Represents that there are two ways for farmers to receive payments in return of their Cotton crop yield sold.

confirmation of the precise amount disbursed to the farmer.

- The blockchain is a multifaceted system that engages various stakeholders. It allows public access to blockchain data to facilitate government verification, authentication, and auditing processes. At the same time, it preserves the privacy of key stakeholders such as farmers, retailers, textile mills, and distributors. Figure 5 illustrates the proposed economic model and token distribution.

2.3 Distribution of Token

2.4 Fee Conversion to Secure Payment

Different charges have been implemented to ensure long term sustainability of Cotton Coin. These include exchange costs, miscellaneous costs and the running of Cotton Coin itself. It is worth mentioning that these charges can differ, depending on the exchange, even such popular ones as Binance. The rules that have been laid down determine how these fixed charges are charged. Each time CottonCoin is bought or traded, a fee is incurred. Also, there will be a small mining charge on any exchange of Cotton Coin between one party and another, and the aim will be to maintain the stability of the system in the long run.

The Cotton Management System and its associated DApp serve as direct access points for users. Every user within the system possesses a DApp profile that includes a public address, a wallet, and settings for managing their profile. The platform's backend is responsible for hosting an ICO as well as an exchange. The purchase of CC within the platform can be made through various payment methods, such as bank transfers and credit card payments. Subsequently, the funds are converted to CC through the exchanges linked to the ICO. Upon completing this cycle, the retailer receives Unspent Transaction Outputs (UTXOs) in their wallet, which can then be sent directly to textile factories and distributors.

The exchange functionality is not limited to retailers; textile distributors and mills can also utilize it to convert CC to fiat currency and vice versa. Moreover, organizations have the option to compensate farmers who may not have wallets by converting CC to traditional currency through the exchanges. Admin users are responsible for managing token distribution policies, user authentication, and smart contract administration. The connections between the exchange and the ICO are governed by smart contracts, which also define the pricing rates of CC on the exchanges in relation to various currencies.

2.5 Layers Proposed CC Block Chain Proposed Model

Proposed Framework Architecture is based on different Layered structure. It includes eight different layers including application layer, infrastructure layer, interface

layer, business logic layer, security and administration layer, block chain layer, transaction layer and trust layer. Each layer details are as follows: (see Figure 6)

The interface layer encompasses web applications, DApps, and user interaction tools, serving as the bridge for communication with the cotton management system. It provides an interface accessible to farmers, retailers, textile mills, consumers, and various stakeholders, facilitating the initiation of all processes within the system. This layer encompasses transaction metadata, identity verification, payment details, and record-keeping. Its primary function is to facilitate secure communication between the business logic layer and the interface levels through the use of smart contracts.

This layer serves as the central hub for managing various roles, user interactions, stipulated rules, and term compliance within a smart contract. It functions as the dynamic core of the smart contract, encompassing all aspects of invocation, communication, and execution rules.

3 Results and Implementation

In this section, we will present performance outcomes, practical applications of our proposed framework in real-world settings, and simulated scenarios. We operate under several assumptions: The distribution of hash power in the system ensures that no single miner or coalition of miners possesses more than 51% of the total hashing power. Access to product purchases and sales will be restricted to registered users.

3.1 Setup Systems

To facilitate the deployment of our framework, we leverage the open-source MetaMask wallet, which is compatible with Web 3.0 technology, to establish a connection with the Ethereum software within the Hardhat development environment. For the development of smart contracts, we have chosen the Remix IDE. To interact with the bloc kchain's HTTP requests, we utilize Postman. We assess the block chain's performance through the Access Chain and Validate Block functions, employing a similar approach for additional performance evaluations. Additionally, we utilize a customized Python script and the Web 3.0 Python library to send 50,000 transactions to the network for testing purposes, (see Figure 7).

We have implemented a contract function named buy Cotton from Farmer, as depicted in Figure 8. This functionality is designed to facilitate the purchase of cotton by the buyer from a specific farmer. It captures and stores essential information, including the addresses of both the buyer and the farmer, the price of the cotton, and the quantity being purchased. Whenever a successful cotton purchase occurs, the contract triggers an event named Cotton Purchased. This event is initiated by the contract's constructor, which initializes crucial parameters such as the buyer's details, cotton price, and the quantity to be purchased. To ensure the integrity of the transaction, the contract incorporates multiple require statements. These statements serve as checks to ensure proper authorization and valid inputs. If all the conditions specified in the require statements are met, the contract proceeds to update the farmer's address and adjust the quantity and price of the cotton. Finally, if all prerequisites are satisfied, the contract emits the Cotton Purchased event to signal a successful transaction.

This functionality of the buy Thread from Warehouse method is captured in our contract as discussed in Figure 9. This is an attribute, which can be used to control vital data, including the address of the buyer and the warehouse, the price and quantity of the threads being purchased. Each time a purchase is executed successfully by the buyer of thread, it raises an event called Thread Purchased. It is the duty of the builder of this feature to initialize the various required variables, such as buyer identity, thread price and amount to be bought. This feature will mainly be used to make it easier to buy thread within a particular warehouse. The system has a number of require statements to assure integrity of the system. Such statements are checks that verify that the authorization is authentic, that the inputs given are valid. Within the functions of this feature, various actions take place. These are the updating of variables related to the price and quantity of the thread along with the address of the warehouse. Moreover, upon fulfillment of all the given requirements, the feature sends the Thread Purchased event, indicating a successful transaction.

The functionality of the buyClothesFromFactories function is described in a contract that is represented in Figure 10. This feature is meant to simplify the process of purchasing clothes, in which important details like the address of the buyer and factory, the price of clothes

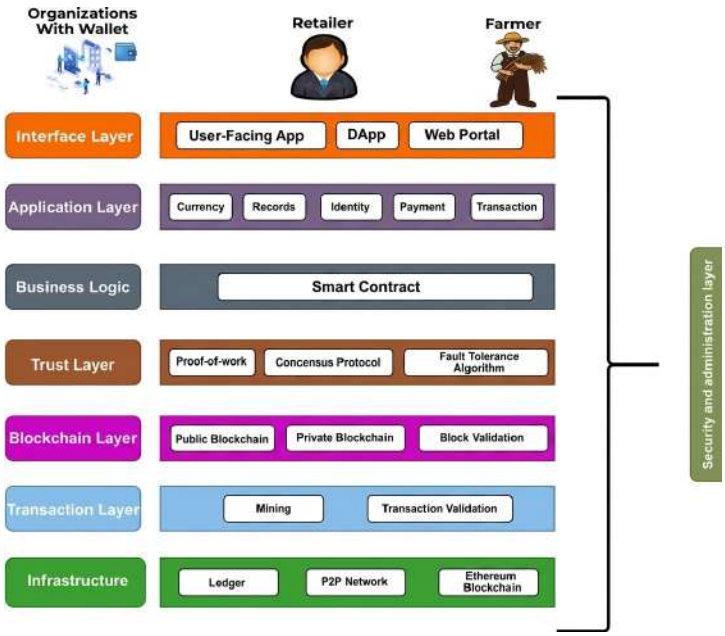

Figure 6. Layers Proposed CC Block Chain Proposed Model

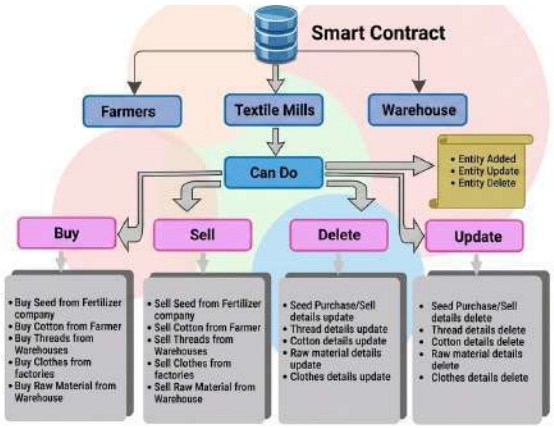

Figure 7. Construct of Systems

```
function buyCottonFromFarmer(address _farmer, uint256 _price, uint256 _quantity) external {
    require(msg.sender == buyer, "Only the buyer can purchase cotton.");
    require(_farmer != address(0), "Invalid farmer address.");
    require(_price > 0, "Price should be greater than 0.");
    require(_quantity > 0, "Quantity should be greater than 0.");

    // Perform the cotton purchase
    farmer = _farmer;
    cottonPrice = _price;
    cottonQuantity = _quantity;

    emit CottonPurchased(buyer, farmer, cottonPrice, cottonQuantity);
}
```

Figure 8. Farmer Buy Cotton

and the overall number of clothes purchased are captured. An event called ClothesPurchased is fired once a successful purchase has been made. The first values of

```
function buyClothesFromFactories(address _factory, uint256 _price, uint256 _quantity) external {
    require(msg.sender == buyer, "Only the buyer can purchase clothes.");
    require(_factory != address(0), "Invalid factory address.");
    require(_price > 0, "Price should be greater than 0.");
    require(_quantity > 0, "Quantity should be greater than 0.");

    // Perform the clothes purchase
    factory = _factory;
    clothesPrice = _price;
    clothesQuantity = _quantity;

    emit ClothesPurchased(buyer, factory, clothesPrice, clothesQuantity);
}
```

Figure 9. Warehouse Thread for Buy Cotton

the buyer, the cost of the clothing, and the quantity are determined in the process of the contract construction.

```
function buyThreadFromWarehouse(address _warehouse, uint256 _price, uint256 _quantity) external {
    require(msg.sender == buyer, "Only the buyer can purchase thread.");
    require(_warehouse != address(0), "Invalid warehouse address.");
    require(_price > 0, "Price should be greater than 0.");
    require(_quantity > 0, "Quantity should be greater than 0.");

    // Perform the thread purchase
    warehouse = _warehouse;
    threadPrice = _price;
    threadQuantity = _quantity;

    emit ThreadPurchased(buyer, warehouse, threadPrice, threadQuantity);
}
```

Figure 10. Factory Clothes for Buy Former

The feature has been coded with several require statements available in order to guarantee the correct authorization and the correctness of input parameters. The functions behind this contract can update the location of the factory, quantity of clothing and pricing variables as well. Providing all the required conditions,

the contract sends out the ClothesPurchased event, thereby simplifying the customer process of obtaining clothing in a particular factory.

```
function buyRawMaterialFromWarehouse(address _warehouse, uint256 _price, uint256 _quantity) external {
    require(msg.sender == buyer, "Only the buyer can purchase raw materials.");
    require(_warehouse != address(0), "Invalid warehouse address.");
    require(_price > 0, "Price should be greater than 0.");
    require(_quantity > 0, "Quantity should be greater than 0.");

    // Perform the raw material purchase
    warehouse = _warehouse;
    rawMaterialPrice = _price;
    rawMaterialQuantity = _quantity;

    emit RawMaterialPurchased(buyer, warehouse, rawMaterialPrice, rawMaterialQuantity);
}
```

Figure 11. Warehouse Raw Clothes for Buy

Figure 11 shows the contract of the buy Raw Material From Warehouse function, which is a vital aspect and helps in acquiring raw materials within a warehouse. This role carefully takes care of addresses of buyers and the warehouse, cost and quantity of the raw materials being acquired. When it is successfully executed, it causes an event called Raw Material Purchased. The initiator of this contract determines the initial values of the buyer, quantity and price of the raw material they wish to buy. To guarantee the integrity of the operation, the contract has a series of require statements which are used to validate authorization and input data. The roles in this contract execute a number of important activities among them updating the address of the warehouse, changing the quantity and price variables of the raw materials and issuing the Raw Material Purchased event under the condition that all the prerequisite conditions are satisfied. This strong strategy makes the process of acquiring raw materials by customers in the warehouse of their preference seamless and secure.

```
function sellCottonByFarmer(uint256 _price, uint256 _quantity) external {
    require(farmer == address(0), "Cotton can only be sold once by the farmer.");
    require(msg.sender != buyer, "Buyer cannot be the seller.");
    require(_price > 0, "Price should be greater than 0.");
    require(_quantity > 0, "Quantity should be greater than 0.");

    // Perform the cotton sale
    farmer = msg.sender;
    cottonPrice = _price;
    cottonQuantity = _quantity;

    emit CottonSold(farmer, buyer, cottonPrice, cottonQuantity);
}
```

Figure 12. Cotton Clothes for Sell

We have implemented a contract called sell Cotton By Farmer which is outlined in Figure 12. This functionality is supposed to facilitate the process of cotton sales between a buyer and a farmer. It entails the storage of the addresses of the buyer and farmer and also the recording of price and quantity of cotton being sold. An event called CottonSold is raised when a successful cotton sale is made. The address, price of cotton, and quantity of cotton that should be purchased are all initiated by the builder of this contract.

To ensure the correctness and security of the contract, it includes multiple require statements. These statements serve to enforce proper authorization and validate input parameters. One of the functions within the contract restricts the farmer's ability to sell cotton only once, and additionally prevents the buyer from taking on the role of the seller. The functions within this contract also handle the updating of the farmer's address, as well as the variables related to the amount and price of cotton. If all the necessary conditions are met, the contract emits the CottonSold event to indicate a successful transaction.

```
function sellThreadsByWarehouse(uint256 _price, uint256 _quantity) external {
    require(warehouse == address(0), "Threads can only be sold once by the warehouse.");
    require(msg.sender != buyer, "Buyer cannot be the seller.");
    require(_price > 0, "Price should be greater than 0.");
    require(_quantity > 0, "Quantity should be greater than 0.");

    // Perform the thread sale
    warehouse = msg.sender;
    threadPrice = _price;
    threadQuantity = _quantity;

    emit ThreadsSold(warehouse, buyer, threadPrice, threadQuantity);
}
```

Figure 13. Warehouse Clothes Thread for Sell

We have a contract in place called sellThreadByWarehouse (Figure 13) that serves as a critical component of our system. This functionality is responsible for managing the sale of threads, keeping track of essential information such as the buyer's and warehouse's addresses, as well as details regarding the thread's price and the quantity sold. When a seller successfully completes a thread sale, this contract emits an event named ThreadSold. This event includes crucial information about the buyer, the thread's price, and the quantity sold. The initial values for these parameters are set during the contract's constructor execution. To ensure the integrity and security of the system, the sell ThreadByWarehouse contract incorporates several input data. Among the various conditions checked, there is one specific condition that restricts the number of sales the warehouse can execute

to just one.

Furthermore, the contract ensures that the buyer cannot require statements. These statements play a crucial role in confirming proper authorization and validating input data. Among the various conditions checked, there is one specific condition that restricts the number of sales the warehouse can execute to just one. Furthermore, the contract ensures that the buyer cannot assume the role of the seller. Upon fulfilling all the necessary conditions, the contract proceeds to update the variables related to thread price and quantity. Additionally, it emits the ThreadSold event, capturing all pertinent details of the transaction. If required, the contract may also update the address of the warehouse as part of its operations. This robust set of checks and actions ensures the smooth and secure operation of the thread-selling process within the system.

```
function sellClothesByFactories(uint256 _price, uint256 _quantity) external {
    require(factory == address(0), "Clothes can only be sold once by the factory.");
    require(msg.sender != buyer, "Buyer cannot be the seller.");
    require(_price > 0, "Price should be greater than 0.");
    require(_quantity > 0, "Quantity should be greater than 0.");

    // Perform the clothes sale
    factory = msg.sender;
    clothesPrice = _price;
    clothesQuantity = _quantity;

    emit ClothesSold(factory, buyer, clothesPrice, clothesQuantity);
}
```

Figure 14. Factories Clothes Thread for Sell

We have a contract that includes the sellClothesByFactories function, illustrated in the Figure 14. This feature is designed to manage the sale of clothing items, tracking important details such as the buyer's and factory's addresses, the cost of the clothing, and the quantity being sold. Whenever a successful clothing sale occurs, an event named ClothesSold is triggered. The initial values for the buyer, clothing cost, and quantity are set during the contract's constructor execution. This functionality plays a crucial role in facilitating clothing sales within the warehouse. It incorporates several require statements to ensure proper authorization and valid inputs. Among its functions is a constraint that restricts a factory's ability to make multiple transactions, and it also enforces the rule that the buyer cannot act as the seller. These functions are responsible for updating the factory's address, as well as adjusting the quantity and price variables for the clothing. If all the necessary conditions are met, they emit the ClothesSold event to indicate a successful transaction.

```
function sellRawMaterialsByWarehouse(uint256 _price, uint256 _quantity) external {
    require(warehouse == address(0), "Raw materials can only be sold once by the warehouse.");
    require(msg.sender != buyer, "Buyer cannot be the seller.");
    require(_price > 0, "Price should be greater than 0.");
    require(_quantity > 0, "Quantity should be greater than 0.");

    // Perform the raw materials sale
    warehouse = msg.sender;
    rawMaterialPrice = _price;
    rawMaterialQuantity = _quantity;

    emit RawMaterialsSold(warehouse, buyer, rawMaterialPrice, rawMaterialQuantity);
}
```

Figure 15. Warehouse Raw Clothes for Sell

Our contract contains sellRawMaterialsByWarehouse (Figure 15) function, which is represented in the diagram. This operation has the mandate of supporting the sale of raw materials by a warehouse to a buyer. It includes important information like the address of the buyer and warehouse, price of the raw material, and the quantity of raw materials sold. An event named RawMaterialsSold is emitted on the successful completion of a sale of a raw material by a seller. The values of the buyer, the cost of the raw material and the quantity are initially established during the course of the execution of the constructor. This is a useful feature in tracking the sales of raw materials in a warehouse. It uses several require statements to facilitate appropriate authorization and valid inputs. It is worth noting that one of the functions limits the number of sales that a warehouse can make to one, and it also precludes the buyer to play the role of a seller. When all the required conditions and prerequisites are provided, these functions can also update the price of the raw material, and the address of the warehouse and offer a complete solution to manage the sales of the raw material in a safe and systematic way.

```
struct CottonProduct {
    uint256 quantity;
    string qualityType;
    uint256 amount;
}

mapping(uint256 => CottonProduct) public cottonProducts;
uint256 public nextProductId;

event CottonProductUpdated(uint256 productId, uint256 quantity, string qualityType, uint256 amount);

function updateCottonProduct(uint256 _quantity, string memory _qualityType, uint256 _amount) public {
    uint256 productId = nextProductId;
    cottonProducts[productId] = CottonProduct(_quantity, _qualityType, _amount);
    nextProductId++;

    emit CottonProductUpdated(productId, _quantity, _qualityType, _amount);
}
```

Figure 16. Structure of Cotton

Attributes of Cotton Product (Figure 16) structure include quantity of cotton, type of quality and quantity.

Each distinct cotton product is linked to these attributes in a mapping where the product ID is the key. Under the Cotton Product function, we have a variable of next Product Id which is used as a tracker of next available product ID. Additionally, this function triggers a Cotton Product Updated event whenever updates are made to cotton products, capturing all the necessary information. The main aim of this feature is to allow users to make changes to the properties of a cotton product. This is done by establishing a new cotton Product struct, a new product ID in the cotton Products mapping and then the nextProductId variable is incremented. This mechanism allows the user to change the quantity, type of quality, and the amount of the cotton product. CottonProductUpdated is an event that is fired every time any of the following changes are applied to the product: Id, quantity, quality type, and amount.

Figure 17. Structure of Thread

The contract includes a function called updateThread (Figure 17) which is responsible for managing threads. This function takes into account attributes such as thread quantity, thread quality type, and thread the updateThread function, there's a variable called next-ThreadId which plays a crucial role in monitoring the next available thread ID. Additionally, this function triggers a ThreadUpdated event whenever threads amount. To keep track of these threads, a mapping is utilized, with the product ID serving as the key. Within are updated, collecting the necessary information in the process. Users have the capability to modify various properties of a thread through this function.

A new instance of the Threads struct is created and assigned to the threads mapping, with the next available thread ID being used as the key. Furthermore, the next-ThreadId variable is incremented based on the input values for quantity, quality type, and amount. Once these updates have taken place, the ThreadUpdated event is emitted, providing information about the thread ID, quantity, quality type, and amount that have been modified.

Figure 18. Structure of Clothes

Figure 18 illustrates a contract named updateClothes with a set of properties designed to manage information about clothes. To store instances of the Clothes, struct, a mapping called clothes is employed, utilizing the clothes ID as the key.

Under this function exists a variable, nextClothesId, which holds the next available clothing ID, and an event called ClothesUpdated, which occurs when changes are made to the clothing data, and the required information is captured. The main aim of this functionality is to enable users to adjust the properties of a piece of clothing. It takes three input parameters: quantity, type of quality and amount. Then a new Clothes struct is created and this is added to the clothes mapping. Also, nextClothesId variable is made to go through to give a new clothing ID. After the records of the clothes have been updated in terms of their ID, quantity, quality type and quantity, the Clothes Updated event is emitted. Same as other technologies [31–36].

Figure 19. Structure of Raw Materials

The updateRawMaterial functions in Figure19 are key

elements of our system, which handle the raw material information of our contract. Attributes used in each update operation include the number of raw material, the type of raw material quality and the quantity of the raw material. To effectively store and retrieve the data of RawMaterials, we use a mapping structure whose unique key is the raw material ID. Among these functions, the nextRawMaterialId is a critical variable. It continuously checks the available raw material ID next to be tracked. When updates are made, there is an emission of an event named RawMaterialUpdated which gives the needed information of the change made to the raw materials.

This activity assists in keeping the transparency and capturing significant changes in our raw material data. This operation enables one to modify the properties of a raw material. It creates a new RawMaterial struct that has its inputs, the amount, number, and type of quality. It also updates the value of the nextRawMaterialId variable to make sure that the next available raw material ID is assigned in the mapping of the raw materials. Once the raw materials ID, quantity, quality type and amount are successfully updated, the RawMaterialUpdated event will be triggered and shown.

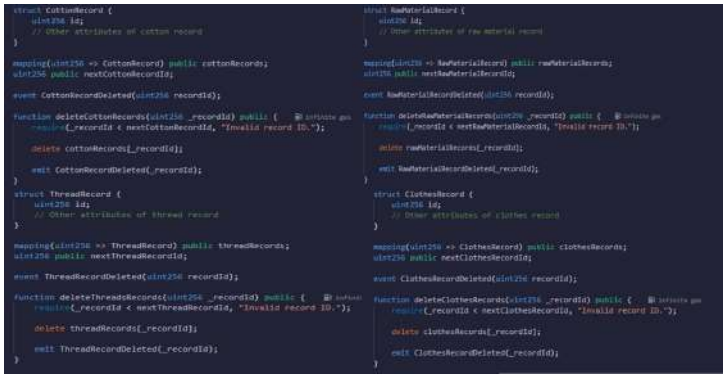

Figure 20. Structure of Record of Cotton, Threads, Clothes, and Raw Materials

Figure 20 illustrates the Record contract, which serves to create and manage cotton records. The contract employs a mapping named Records with the ID serving as the key to store instances of the Record struct. The Record struct encapsulates the data related to a cotton, threads, clothes, and Raw Materials record. The contract also maintains the tracking of the next available record ID via the nextRecordId variable. Whenever a cotton, threads, clothes, and Raw Materials record are successfully removed, the contract emits the RecordDeleted event, capturing the ID of the deleted record.

Users have the ability to delete records using the deleteRecords function by specifying the ID as a parameter. If the provided ID is valid, the record associated with that ID is removed from the mapping using the delete keyword. After the removal of a record, an event is triggered with the identifier RecordDeleted. Overall, this smart contract seems to be a basic system for managing raw material records with the ability to create, delete, and emit events for record management operations. Users can interact with the contract to maintain and keep track of their raw material data.

EntityAdded Function function (Figure 21) allows users to add entities with various entity types through a dropdown menu. It releases an EntityAdded event when an entity is successfully added, providing information about the entity, including its ID, name, and entity type. EntityType Enum available entity kinds are represented by an EntityType enum, which includes the following values: None, Farmer, Factory, Warehouse, and Stakeholder. This enum likely defines the different types of entities that can be added. Entities Mapping There is a mapping named entities that uses the entity ID as the key and stores instances of the Entity struct.

```
contract TextileSupplyChain {
    enum EntityType {None, Farmer, Factory, Warehouse, Stakeholder}

    struct Entity {
        string name;
        EntityType entityType;
    }

    mapping(uint256 => Entity) public entities;
    uint256 public nextEntityId;

    event EntityAdded(uint256 entityId, string name, EntityType entityType);

    function addEntity(string memory _name, EntityType _entityType) public {
        uint256 entityId = nextEntityId;
        entities[entityId] = Entity(_name, _entityType);
        nextEntityId++;

        emit EntityAdded(entityId, _name, _entityType);
    }
}
```

Figure 21. Structure of New Entity

This mapping is used to keep track of added entities. Entity Struct represents an entity and has at least two properties: name and entity type. These properties are used to describe the entity.nextEntityId Variable is used to keep track of the next available entity ID. When a new entity is added, this variable is incremented to assign a unique ID to the newly added entity. addEntity

Function is responsible for adding new instances of the Entity struct. It increases the nextEntityId variable to generate a new ID for the entity, assigns the provided parameters (name and entity type) to the entity, and stores it in the entities mapping with the next available entity ID.EntityAdded Event: After adding the entity to the mapping, the addEntity function broadcasts the EntityAdded event, providing the ID, name, and entity type of the newly added entity as part of the event data.

Dropdown Menu: The function EntityUpdated allows users to interact with a dropdown menu. This menu enables them to select from various entity kinds. These selections seem to represent different types or categories of entities. EntityType: Each selected entity kind is represented by the EntityType. It implies that the function uses some sort of data structure or enumeration to categorize entities into these types.

Entity Storage: The function utilizes a data structure called entities mapping to store instances of an Entity struct. The Entity struct appears to contain at least two properties: the entity's name and its entity type. The struct instances are stored using the entity ID as the key in this mapping.

Event Triggering: An event named EntityUpdated is triggered or emitted when a new entity is successfully added. This event carries information about the newly added entity, including its ID, name, and entity type. This event likely allows other parts of the system to react to the addition of new entities. Updating Existing Entities: Users can also update existing entities using a function called updateEntity.

```
contract TextileSupplyChain {
    enum EntityType {None, Farmer, Factory, Warehouse, Stakeholder}

    struct Entity {
        string name;
        EntityType entityType;
    }

    mapping(uint256 => Entity) public entities;

    event EntityUpdated(uint256 entityId, string newName, EntityType newEntityType);

    function updateEntity(uint256 _entityId, string memory _newName, EntityType _newEntityType) public {
        require(_entityId < nextEntityId, "Invalid entity ID.");

        entities[_entityId].name = _newName;
        entities[_entityId].entityType = _newEntityType;

        emit EntityUpdated(_entityId, _newName, _newEntityType);
    }
}
```

Figure 22. Structure of New Entity Update

To do this, they need to provide the entity's ID, a new name, and a new entity type as parameters. This function performs validation to ensure that the supplied ID is valid and less than nextEntityId. If successful, it updates the entity's name and entity type in the entities mapping. Broadcasting Event for Updates: After successfully updating an existing entity, the updateEntity function broadcasts the EntityUpdated event again. This time, it contains information about the changed entity, including its ID, the new name, and the new entity type. This allows other parts of the system to react to changes made to existing entities (Figure 22). Similarly work has been done for delete entity (Figure 23).

```
contract TextileSupplyChain {
    enum EntityType {None, Farmer, Factory, Warehouse, Stakeholder}

    struct Entity {
        string name;
        EntityType entityType;
    }

    mapping(uint256 => Entity) public entities;

    event EntityDeleted(uint256 entityId);

    function deleteEntity(uint256 _entityId) public {
        require(_entityId < nextEntityId, "Invalid entity ID.");

        delete entities[_entityId];

        emit EntityDeleted(_entityId);
    }
}
```

Figure 23. Structure of New Entity Delete

We utilized a selection of transactions as a representative sample for evaluating the data. To assess the proposed framework against alternative solutions, we conducted the following five tests and comparisons:

- **Calculation of response time:** Measurement of response time in milliseconds using the proposed framework to obtain the longest chain of data.
- **Delay analysis:** Analysis of delays in adding new blocks to the blockchain, along with other statistical comparisons.
- **Block and transaction evaluation:** Evaluation of block confirmations, transaction processing time, block addition latency, and data retrieval latency.
- **Transaction fee comparison:** Comparison of average transaction fees between the proposed framework and other alternatives.
- **Performance assessment:** Assessment of overall performance ratings across various blockchain technologies.

The implementation of the Access Chain functional to various blocks to get blockchain data. This operation

supported the process of timestamping to determine the amount of time required in accessing data and helped in extracting particular data among different blocks in the chain. Latency varied, beginning at about 412 milliseconds/block, dropping to 94 milliseconds and finally returning to 963 milliseconds. This variability in latency across blocks is due to the decentralized nature of the block chain. New block latency, in which the numbers of the blocks, and the measurements of the delay in milliseconds, which are performed. The data of different latency rates: 412 milliseconds block 1, 368 milliseconds block 53, 57 milliseconds block 107, 643 milliseconds block 168 etc. The time of retrieval improves significantly as the block size is larger because of the need to access several nodes to retrieve the data.

The response time to retrieve the longest chain, shown in Figure 24, shows response times in pairs with different blocks. The time to retrieve the longest chain data is represented in Figure 8 graph. The number of blocks is plotted on the horizontal axis, whereas the time is in milliseconds, plotted on the vertical axis. The graph line shows the time needed to take each block: Block 1 (412 milliseconds), Block 53 (368 milliseconds), Block 107 (57 milliseconds), Block 168 (643 milliseconds), etc. The approach that we have proposed shows considerably quicker block addition to the block chain, in comparison to other networks. As an example, Ethereum network requires 15 seconds on average to add a new block when compared to the Bitcoin network which requires 10 minutes.

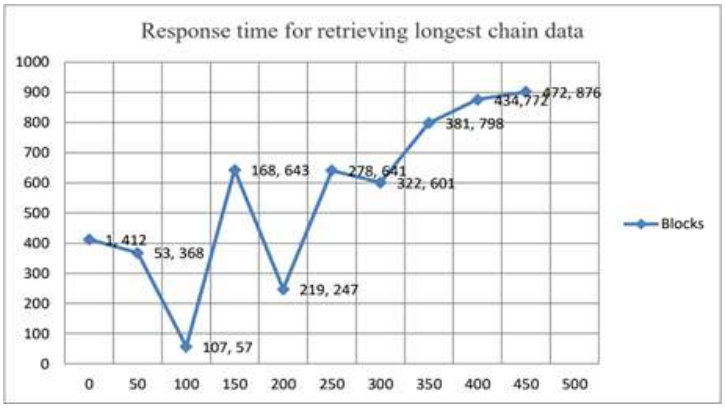

Figure 24. Verification Response Time

Comparing the transaction fees of Ethereum and Bitcoin networks with our suggested method, as depicted in Figure 25, highlights the cost-effectiveness of our approach. Our architecture loads a new block every three seconds, as illustrated in Figure 25, showcasing superior latency performance in incorporating new blocks. With this expedited block insertion, the suggested method proves more efficient in verifying transactions compared to traditional networks. Show casing superior latency performance in incorporating new blocks. With this expedited block insertion, the suggested method proves more efficient in verifying transactions compared to traditional networks. Due to increased block delays, Ethereum and Bitcoin networks handle daily transaction volumes of 1.13 million and 0.213 million, respectively.

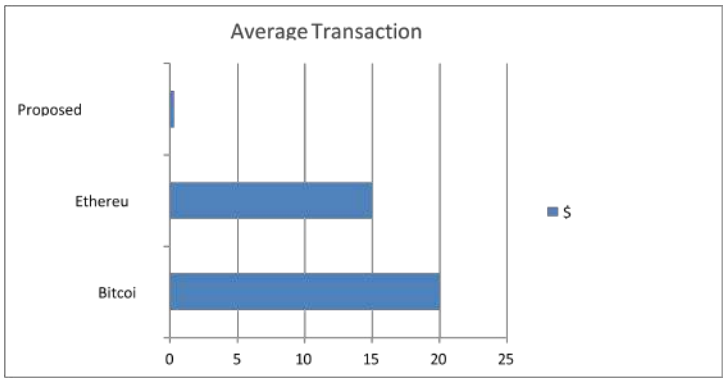

Figure 25. Fee per Transections

In contrast, the suggested framework boasts the capacity to accommodate an average of 15.28 million transactions per day onto the block chain. Figure 26 elucidates the time required for twelve network confirmations, crucial for accepting financial payments, and presents new block statistics across various block chains. Remarkably, our suggested system verifies new transactions 200 times faster than Bitcoin and 6 times swifter than Ethereum. While the average transaction price in our suggested system stands at 0.3 USD, congestion periods on Bitcoin and Ethereum networks can escalate fees up to 100 USD. A comparative analysis of the suggested framework's average fee against current systems is delineated in Figure 26. Moreover, Figures 8-23 presents the code scripts of proposed method. Table 3 lists the outcome.

Table 1 furnishes a comprehensive juxtaposition between the proposed framework and the existing block chain. It facilitates a swift comparison of transactions

Table 1. Comparison with Existing Studies

References	Transaction/Block	Block Time (min)	Time for 12 Verify (KS)	Fee	Transactions/Day (Million)
Nakamoto et al. [13]	2.7	10	7.2	$10	0.213
Salah et al. [14]	0.07	0.25	0.18	$20	1.136
Kumar et al. [15]	0.07	0.25	0.18	$20	1.136
Proposed System	**0.57**	**0.03**	**0.0362**	**$0.30**	**15.28**

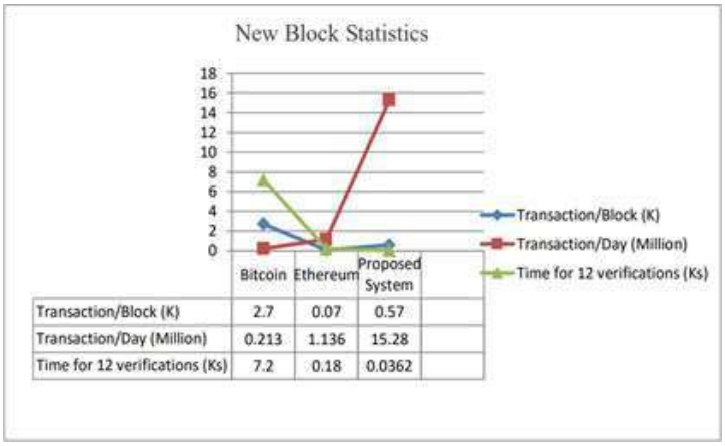

Figure 26. Statistics of New Block

per block, total block time in minutes, time required for 12 verifications in KS, fee per transaction, and number of transactions per day in million between the proposed and conventional contract systems.

Performance Evaluation Result

Table 3 presents the performance evaluation results for blocks 1 to 100, including latency and size measurements.

4 Conclusion

Cotton farming has remained a pillar of world agriculture because of its centrality in manufacturing environmentally friendly products, as well as ordinary cotton clothes. Pakistan becomes one of the strongest countries in the world in cotton crop production. When it comes to resolving the problems of its supply chain, a research project suggests the use of a block chain-based architecture designed specifically to solve the problems of the cotton and textiles industry. This new system provides a full traceability and better price control on cotton products.

The proposed structure, which is implemented on the Binance Smart Chain (BSC), has fast transactions and lower costs than the popular block chains such as Ethereum and Bitcoin. At the core of this solution is the proposal to introduce Cotton Coin, a cryptocurrency that is supported by smart contracts. The use of the CC token has brought in significant monetary gains to all the parties concerned. The system also provides good control over the export of cotton, which allows the government to simplify activities, and save resources by avoiding the cost of importing cotton at the off-season.

Table 2. Performance Evaluation Result

Total Number of Blocks Added	Latency (ms)	Size (KB)
1	87	1.00000000
2	50	1.81519183
3	90	2.36487979
4	79	2.57874558
5	65	2.64587879
6	85	3.24556657
7	53	3.71321515
8	60	3.82684682
9	57	4.48265641
10	68	5.32451265
11	52	6.19254927
12	99	6.35985658
13	66	7.26512554
14	73	8.15584682
15	55	8.52820551
16	81	8.92248214
17	100	9.25884684
18	100	9.82264566
19	96	10.42558438
20	52	10.88698952
21	87	11.28225268
22	91	11.98822225
23	65	12.05516888
24	79	12.32659419
25	77	12.50515944
26	69	13.11558449
27	51	13.35959929
28	89	14.35461646
29	61	14.50461694
30	93	14.88264646
31	83	15.66664643
32	100	15.76946565
33	51	16.36466822
34	93	17.22698946
35	69	17.96666164
36	89	18.26943694
37	80	18.46565161
38	52	18.71665862
39	67	19.21695146
40	69	19.46816655
41	70	19.92654664
42	57	20.46161944
43	67	20.76639556
44	83	21.23564169
45	100	22.41644322
46	97	23.11549499
47	67	23.46598946
48	55	24.56993599
49	83	25.14464299
50	77	25.35646946

Table 3. Performance Evaluation Result

Total Number of Blocks Added	Latency (ms)	Size (KB)
51	88	25.76323265
52	225	25.96231649
53	221	26.16494916
54	91	26.31466414
55	221	26.81196622
56	65	27.25965223
57	219	27.46166929
58	74	27.65656356
59	256	27.95231352
60	71	28.11245656
61	88	28.23463594
62	108	28.46679946
63	106	29.63594926
64	126	30.13265946
65	68	30.99492399
66	259	31.16494926
67	175	31.26659944
68	217	31.29646662
69	68	31.30161616
70	232	31.72649164
71	215	32.11616918
72	256	32.48964694
73	158	32.80661495
74	52	33.10995269
75	51	33.46589462
76	255	33.82616135
77	280	33.62669916
78	291	34.25649497
79	179	34.59598166
80	272	35.16655999
81	256	36.49268166
82	158	36.76299952
83	52	36.81696299
84	151	37.49997595
85	240	37.99169499
86	95	38.14659933
87	205	38.44662699
88	84	38.76199412
89	234	39.13649466
90	202	39.35649962
91	136	39.46689260
92	96	39.86594359
93	79	40.66593594
94	84	41.49688943
95	296	41.53294644
96	229	42.12546997
97	198	42.65639466
98	171	43.64692546
99	143	44.14554680
100	140	44.93005160

To enhance the level of data safety and inspire trust in stakeholders, the Inter Planetary File System (IPFS) is the storage of encrypted information related to farmers, textile companies, warehouses, and merchants.

In the future, the next possible direction of improvement is to customize smart contracts and maximize performance and reduce transactional costs by optimizing transactional functions. Also the further improvements may include creating mobile traceability applications as well as special smart wallets to handle Cotton Coin transactions in a flawless manner.

Data Availability

Data and explanations related to this study are available upon reasonable request by contacting the first or corresponding author.

Author Contributions

Asif Raza: Conceptualization, Methodology, Software **Salahuddin**: Data curation, Writing- Original draft preparation. **Ghazanfar Ali**: Visualization, Investigation.**Sadia Latif**: Software, Validation, Writing- Reviewing and Editing

Compliance with Ethical Standards

It is declare that all authors don't have any conflict of interest. It is also declare that this article does not contain any studies with human participants or animals performed by any of the authors. Furthermore, informed consent was obtained from all individual participants included in the study.

Funding Information

No Funding

References

[1] H. R. Sarma et al., "Effect of carbonization behaviour of cotton biomass in electrodes for sodium-ion batteries," *ChemElectroChem*, Art. no. e202300127, 2023.

[2] M. A. ReHabib, M. Ahmad, S. Jabbar, S. Khalid, J. Chaudhry, K. Saleem, and M. S. Khalil, "Security and privacy based access control model for internet of connected vehicles," *Future Generation Computer Systems*, vol. 97, pp. 687–696, 2019.

[3] M. U. Farooq and M. O. Beg, "Big data analysis of Stack Overflow for energy consumption of Android framework," in *Proc. International Conference on Innovative Computing (ICIC)*, Lahore, Pakistan, Nov. 2019.

[4] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *Proc. IEEE International Congress on Big Data*, pp. 557–564, 2017.

[5] X. Li, J. Chen, and Y. Wu, "Blockchain for textile supply chain management: A review," *Journal of Textile Science and Technology*, vol. 6, no. 3, pp. 1–9, 2020.

[6] X. Bai, R. Tao, X. Du, and J. Wu, "A blockchain-based textile supply chain traceability system," *Journal of Industrial Information Integration*, vol. 17, Art. no. 100127, 2020.

[7] A. Zeb and G. Shabir, "Blockchain technology for supply chain management: A systematic literature review and future research directions," *Journal of Cleaner Production*, vol. 259, Art. no. 120785, 2020.

[8] R. Richero and S. Ferrigno, "A background analysis on transparency and traceability in the garment value chain," European Commission, International Cooperation and Development Report.

[9] P. Rogaway and T. Shrimpton, "Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance and collision resistance," Springer, Heidelberg, 2004.

[10] K. Toyoda, P. T. Mathiopoulos, I. Sasase, and T. Ohtsuki, "A novel blockchain-based product ownership management system (POMS) for anti-counterfeits in the post supply chain," *IEEE Access*, vol. 5, pp. 17465–17477, 2017.

[11] A. Hekmat, Z. Zhang, S. U. R. Khan, I. Shad, and O. Bilal, "An attention-fused architecture for brain tumor diagnosis," *Biomedical Signal Processing and Control*, vol. 101, Art. no. 107221, 2025.

[12] S. U. R. Khan, A. Raza, M. Waqas, and M. A. R. Zia, "Efficient and accurate image classification via spatial pyramid matching and SURF sparse coding," *Lahore Garrison University Research Journal of Computer Science and Information Technology*, vol. 7, pp. 10–23, 2023.

[13] A. Hekmat, Z. Zhang, O. Bilal, and S. U. R. Khan, "Differential evolution-driven optimized ensemble network for brain tumor detection," *International Journal of Machine Learning and Cybernetics*, pp. 1–26, 2025.

[14] P. K. Sharma, N. Kumar, and J. H. Park, "Blockchain-based distributed framework for automotive industry in a smart city," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 7, pp. 4197–4205, 2018.

[15] S. U. R. Khan, "Multi-level feature fusion network for kidney disease detection," *Computers in Biology and Medicine*, vol. 191, Art. no. 110214, 2025.

[16] A. Villalva-Cataño, E. Ramos-Palomino, K. Provost, and E. Casal, "A model in agri-food supply chain costing using ABC costing: An empirical research for Peruvian coffee supply chain," in *Proc. 7th International Engineering, Sciences and Technology Conference (IESTEC)*, pp. 1–6, 2019.

[17] S. U. R. Khan and S. Asif, "Oral cancer detection using feature-level fusion and novel self-attention mechanisms," *Biomedical Signal Processing and Control*, vol. 95, Art. no. 106437, 2024.

[18] T. Ferdousi, D. Gruenbacher, and C. M. Scoglio, "A permissioned distributed ledger for the US beef cattle supply chain," *IEEE Access*, vol. 8, pp. 154833–154847, 2020.

[19] A. Kumar and A. Mishra, "Blockchain based security and privacy solution for textile supply chain management," *Journal of Cleaner Production*, 2021.

[20] S. U. R. Khan and Z. Khan, "Detection of abnormal cardiac rhythms using feature fusion technique with heart sound spectrograms," *Journal of Bionic Engineering*, pp. 1–20, 2025.

[21] U. Rahardja, A. N. Hidayanto, N. Lutfiani, D. A. Febiani, and Q. Aini, "Immutability of distributed hash model on blockchain node storage," *Scientific Journal of Informatics*, vol. 8, no. 1, pp. 137–143, 2021.

[22] M. A. Al-Khasawneh, A. Raza, S. U. R. Khan, and Z. Khan, "Stock market trend prediction using deep learning approach," *Computational Economics*, pp. 1–32, 2024.

[23] U. S. Khan, M. Ishfaque, S. U. R. Khan, F. Xu, L. Chen, and Y. Lei, "Comparative analysis of twelve transfer learning models for crack detection in concrete dams using borehole images," *Frontiers of Structural and Civil Engineering*, pp. 1–17, 2024.

[24] S. U. R. Khan et al., "Robust and precise knowledge distillation-based context-aware predictor for disease detection in brain and gastrointestinal systems," 2025.

[25] Q. Dai, M. Ishfaque, S. U. R. Khan, Y. Luo, Y. Lei, B. Zhang, and W. Zhou, "Image classification for subsurface crack identification in concrete dams using borehole CCTV images with deep dense hybrid model," *Stochastic Environmental Research and Risk Assessment*, pp. 1–18, 2024.

[26] Y. P. Tsang, K. L. Choy, C. H. Wu, G. T. S. Ho, and H. Y. Lam, "Blockchain-driven IoT for food traceability with an integrated consensus mechanism," *IEEE Access*, vol. 7, pp. 129000–129017, 2019.

[27] H. Treiblmaier, "The impact of blockchain on the supply chain: A theory-based research framework and a call for action," *Supply Chain Management: An International Journal*, vol. 23, pp. 545–559, 2018.

[28] U. S. Khan and S. U. R. Khan, "Boosting diagnostic performance in retinal disease classification utilizing deep ensemble classifiers based on OCT," *Multimedia Tools and Applications*, pp. 1–21, 2024.

[29] S. U. R. Khan, S. Asif, O. Bilal, et al., "LEAD-CNN: Lightweight enhanced dimension reduction convolutional neural network for brain tumor classification," *International Journal of Machine Learning and Cybernetics*, 2025.

[30] A. Raza, M. T. Meeran, and U. Bilhaj, "Enhancing breast cancer detection through thermal imaging and customized 2D CNN classifiers," *VFAST Transactions on Software Engineering*, vol. 11, pp. 80–92, 2023.

[31] S. Inzamam, J. Ouyang, and S. Khan, " FedVC-ADDiM: a federated learning framework for diagnosis of alzheimer disease using deep learning," *Multimedia Systems*, 32.3 (2026): 161.

[32] S. Khan, M. N. Asim, S. Vollmer, and A. Dengel, " FloraSyntropy-net: scalable deep learning with novel FloraSyntropy archive for large-scale plant disease diagnosis," *Plant Methods*, (2026).

[33] M. Hasaan, S.U.R Khan, S. Vollmer, A. Dengel, and M. N. Asim, " Automated Diabetic Screening via Anterior Segment Ocular Imaging: A Deep Learning and Explainable AI Approach," *arXiv preprint arXiv:2603.14727*, (2026).

[34] I. Misbah, S.U.R.Khan, A.U. Rehman, S. Vollmer, A. Dengel, and M. N. Asim, " StructDamage: A Large Scale Unified Crack and Surface Defect Dataset for Robust Structural Damage Detection," *arXiv preprint arXiv:2603.10484*, (2026).

[35] H. Arash, O. Bilal, Z. Zhang, S.U.R Khan, and S. Asif, " FRE-Net: A Fuzzy Richards Functions-Based Ensemble Network for Brain Tumor Detection," *Journal of Bionic Engineering*, (2026): 1-23.

[36] B. Omair, A. Hekmat, S.U.R Khan, A. Raza, and G. Ali, " MS-STO-Net: A Multi-Scale State Transition Optimization-Based Ensemble Network for Accurate White Blood Cell Classification," *In 2025 27th International Multitopic Conference (INMIC)*, pp. 1-6. IEEE, 2025.