Hindawi
Security and Communication Networks
Volume 2021, Article ID 5592119, 20 pages
https://doi.org/10.1155/2021/5592119

Research Article

A Blockchain-Based Hierarchical Authentication Scheme for Multiserver Architecture

Miqi Wu, Lin You, Gengran Hu, Liang Li, and Chengtang Cao

College of Cybersecurity, Hangzhou Dianzi University, Hangzhou 310018, China

Correspondence should be addressed to Lin You; mryoulin@gmail.com

Received 9 February 2021; Revised 11 March 2021; Accepted 31 March 2021; Published 23 April 2021

Academic Editor: Omar Cheikhrouhou

Copyright © 2021 Miqi Wu et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

In a multiserver architecture, authentication schemes play an important role in the secure communication of the system. In many multiserver authentication schemes, the security of the mutual authentications among the participants is based on the security of the registration center's private key. This centralized architecture can create security risks due to the leakage of the registration center's private key. Blockchain technology, with its decentralized, tamper-proof, and distributed features, can provide a new solution for multiserver authentication schemes. In a lot of multiserver authentication schemes, users' permission is generally controlled by the registration center (RC), but these permission control methods cannot be applied in the decentralized blockchain system. In this paper, a blockchain-based authentication scheme for multiserver architecture is proposed. Our scheme provides a hierarchical authentication method to solve the problems of user permission control and user revocation caused by no registration center. The security of our scheme is formally proved under the random oracle model. According to our analysis, our scheme is resistant to attacks such as impersonation attacks and man-in-the-middle attacks. In addition, our performance analysis shows that the proposed scheme has less computation overhead.

1. Introduction

With the developments of the Internet economy and the network application scale, it is difficult for a single server to provide complete services for users. To address this problem, multiserver architectures have emerged. In multiserver systems, the authentication and the corresponding secret key agreement between the servers and the users are important issues to ensure service efficiency and communication security. However, the traditional password-based authentication methods [1] applicable to single client/server architectures are not applicable to multiserver architectures. In recent years, many kinds of new multiserver authentication schemes [2–8] have been proposed, including various improvements to scheme security and performance. The common feature of these schemes is the existence of three participants: the user, the registration center, and the server. The private key of the registration center is used in the process of participant registration and participant mutual authentication. If the private key of the registration center is leaked, the whole system will suffer from significant security risks.

The emergence of blockchain technology has provided a new solution to the authentication and key agreement scheme in multiserver architecture. Blockchain technology is decentralized, tamper-proof, and jointly maintained by multiple parties. Using blockchain technology can solve the trust problem between the users and the servers [9] and give better robustness to multiserver systems and avoid the security risks mentioned above. The use of blockchain for enforcing the security of the network systems has proved to be significantly beneficial. For example, blockchain can provide access control, maintain data integrity, and improve availability [10]. Xiong et al. [11] proposed the first blockchain-based two-factor multiserver authentication scheme. Their scheme was claimed to be able to make mutual authentication of the participants without online RC and have effective revocation. However, in fact, the scheme did not take into account the impact of the introduction of decentralized blockchain networks on the design of multiserver authentication schemes. The registration process and revocation process of the users in their scheme were done

through only one server, without the supervision of the nodes or other servers, which allows the servers to tamper with the revocation and reregistration messages of users and thus steal their identities. In addition, their scheme lacks user permission control, as each user only needs to register with one of the servers to have unrestricted access to all other servers. In practical applications, access control in multiserver systems is important, and we give an example of a multiserver system for a school league in Figure 1. The creation of the multiserver system can facilitate the students and the teachers to access the resources of other schools' servers, while the hierarchical access control function can prohibit the users with low-level permission (such as students) from accessing high-level servers (such as confidential servers). Kou et al. [2] proposed a scheme that have a hierarchical approach where the users of different levels could access servers of different levels. The permission level in their scheme construction is bound to the user's public-private key pair, which requires the user to change his public-private key pair when the permission level is changed. Besides, they used the bilinear pairing operation in their scheme, and it will make Kou et al.'s scheme be of low usability and have high computation overhead.

In this paper, a blockchain-based hierarchical authentication scheme for a multiserver architecture is proposed. The main contributions are as follows:

(1) A blockchain-based decentralized multiserver authentication scheme is proposed to deal with the problem of centralization under the original multiserver architecture. Our scheme has three security factors and has a lower computation overhead shown by performance analysis. Compared with Kou et al.'s scheme [2], our scheme does not use the bilinear pairing operation, so the computation overhead in our scheme can be reduced by 82%. Compared with Xiong et al.'s scheme [11], the computation overhead in our scheme can be reduced by 17%.

(2) A blockchain-based hierarchical access control method for a multiserver architecture is proposed in this paper. It makes up for the lack of user permission control in several blockchain-based multiserver authentication schemes (such as Kou et al.'s scheme [2] and Ren et al.'s scheme [12]). This method can adapt to the decentralized operation model of blockchain systems, control user access to the servers, and achieve secure revocation of the user accounts in the blockchain environment.

(3) We provide a security model for blockchain-based multiserver authentication scheme and prove its security under the random oracle model. Also, a security analysis is provided to illustrate that our scheme is secure against impersonation attack, etc.

The remainder of the paper is organized as follows. Section 2, respectively, reviews the related work on the traditional multiserver authentication schemes and the blockchain-based identity authentication schemes. Section 3 introduces the background knowledge of blockchain technology, fuzzy extractor, and elliptic curve cryptographic theory. Section 4 describes the system in our scheme. Section 5 elaborates the phases in our scheme in detail. Section 6 makes a formal proof of the security of our scheme. Section 7 shows the advantages of our scheme in terms of safety and performance through the comparison with other schemes. Section 8 summarizes this paper and enumerates future application.

2. Related Work

In 2001, Li et al. [13] proposed a neural network-based multiserver authentication system. In 2014, Chuang et al. [14] proposed an authenticated key agreement scheme for anonymous multiserver architecture based on trusted computing by combining smart cards, user passwords, and biometrics and claimed that their scheme could meet all the key requirements for the multiserver architectures. Wan et al. [3] observed that Chuang et al.'s scheme could not guarantee anonymity and could not resist smart card loss attacks, so they proposed an improved scheme that not only solved the problems in the above scheme but also inherited the advantages of the original scheme. In 2015, He and Wang [4] proposed a robust biometric-based multiserver authentication scheme which was the first truly three-factor authentication scheme for multiserver environment. They used the fuzzy extractor to extract biometric key from biometric information. Later, Odelu et al. [15] pointed out that He and Wang's scheme was vulnerable to the known session-specific temporary information attack and impersonation attack. To address these issues, they proposed a secure biometric-based multiserver authentication scheme [5] using smart cards that resisted various attacks. In 2015, Tsai and Lo [16] proposed an efficient authentication scheme for distributed mobile cloud computing services. Later in 2017, Odelu et al. [6] pointed out that Tsai et al.'s scheme did not provide both the session key security and strong user credentials' privacy. To address these issues, they designed a provably secure authentication scheme for the distributed mobile cloud computing services. In 2016, He et al. proposed a new scheme [7] that reduced the computation and communication costs.

Because of the decentralized and tamper-proof features of blockchain, some blockchain-based identity authentication schemes have been proposed. In 2019, Hei et al. [17] proposed an identity information sharing authentication scheme which used smart contract technology to share and update identity information, and it could solve the problem of multiple user registrations on multiple servers. In addition, it used attribute-based cryptography to protect the privacy of users. Zhou et al. [18] proposed a blockchain-based two-factor authentication scheme for cross-domain authentication. Mohsin et al. [19] proposed a biometric-based patient identity authentication scheme that solved the problem of storing biometric information on the blockchain. Ren et al. [12] designed a IoMT node authentication and key agreement scheme for across to the trust domain based on blockchain, but their scheme cannot provide user anonymity and untraceability and do not have user permission control.

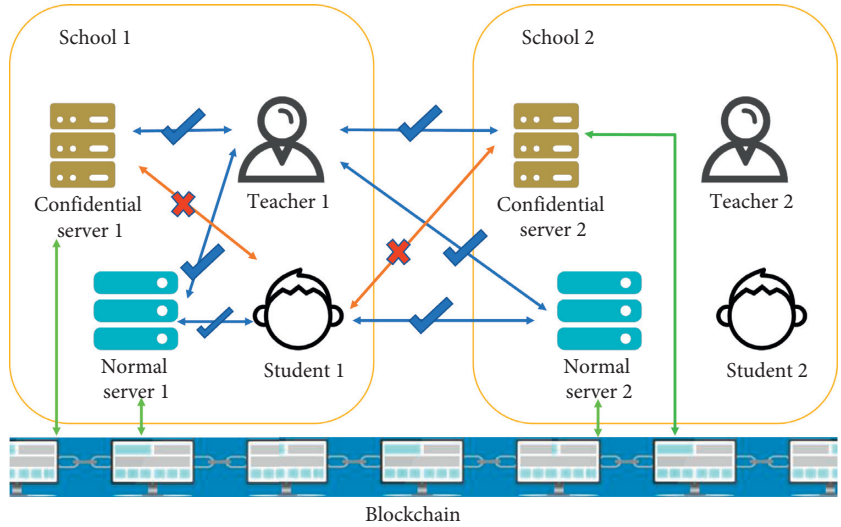

Figure 1: A blockchain-based hierarchical multiserver authentication system in a school league. Each school has its own confidential servers and normal servers. These servers can identify the user's identity with the help of the blockchain system, regardless of the school they come from. And the confidential server is only available to the teachers who have access to it, while the students do not have access to it.

3. Background and Notations

3.1. Blockchain Technology. Blockchain technology originated from a paper "A Peer-to-Peer Electronic Cash System" published in 2008 by an academic with the pseudonym Nakamoto [20]. Blockchain can be applied in various fields owing to its good properties such as auditability, transparency, immutability, and decentralization. In recent years, blockchain has helped improve security and privacy in a variety of application scenarios such as smart cities [21], IoT [22], healthcare [23], and smart vehicles [24].

There are three main concepts in blockchain technology, that is, nodes, consensus, and blocks. The node is an entity in a blockchain network that is responsible for recording transactions in the blockchain network and packaging them into a block, which is then broadcasted to other nodes or reach consensus with other nodes. The consensus algorithm is the most important part of blockchain technology. Different blockchain networks use different consensus algorithms. For example, Bitcoin uses PoW (proof of work). The node broadcasts the block after a block is generated. The block is considered to be recorded on the chain when most of the nodes reach consensus about the block and work based on this block.

The consortium blockchain is a kind of blockchain, which is suitable for building a system of cooperation among multiple institutions. Each node of the consortium blockchain usually has its corresponding entity organization, which can join and exit the network only after authorization, and each organization forms an alliance with related interests to maintain the healthy operation of the blockchain. Nowadays, Hyperledger Fabric is one of the most mainstream consortium blockchain applications.

3.2. Fuzzy Extractor. Fuzzy extractor is an algorithm first proposed by Dodis et al. [25] which can extract a uniformly distributed random secret key R and an auxiliary information BP from input biometric information ω. When another biometric information $\omega\prime$ similar to ω is input, R can be recovered with the help of BP. According to [25], we define the following two algorithms:

(i) $(R, BP) \leftarrow \mathrm{Gen}(\omega)$: fuzzy-extractor generation algorithm. Upon receiving the user's biometric information ω, the algorithm will output the random string secret key R and public information BP corresponding to the user's biometric information.

(ii) $R \leftarrow \mathrm{Rep}(\omega\prime, BP)$: fuzzy-extractor recovery algorithm. Upon receiving the user's biometric information ω and the corresponding public information BP, the algorithm will output the string R corresponding to the user's biometric information ω if the error between the two input biometric features is satisfied within the tolerance range, i.e., $\mathrm{dis}(\omega\prime, \omega) \le r$, where r is the tolerance error.

3.3. Elliptic Curve Cryptography. Let p be a large prime number and $\mathbb{F}_p$ be a finite field of order p, then the elliptic curve E over the finite field $\mathbb{F}_p$ is defined. The point P is an element with large prime order q on the additive group G on E. We introduce the discrete logarithm problem and the computational Diffie–Hellman problem in the elliptic curve cryptography:

(i) Elliptic Curve Discrete Logarithm (ECDL) Problem: it is easy to calculate A by given P and a, but it is hard to determine a by given A and P, where $A = a \cdot P$, and $A \in G$.

(ii) Elliptic Curve Computational Diffie–Hellman (ECCDH) Problem: it is hard to compute $(a \cdot b) \cdot P$ as a, b are unknown elements in Z_q^*, where $A = a \cdot P$, $B = b \cdot P$, and $P, A, B \in G$.

Based on the elliptic curve cryptography, we represent the elliptic curve digital signature algorithm ECDSA as the following three functions:

(i) Keygen $(1^k) \longrightarrow (SK, PK)$: Key Generation Function. The function generates a random private key SK and the corresponding public key PK.

(ii) Sig $(SK, m) \longrightarrow$ SIG: Signature Function. The function uses the private key SK to calculate the signature SIG of massage m.

(iii) Ver $(PK, \text{SIG}, m) \longrightarrow \text{bool} \in \{\text{true}, \text{false}\}$ Verify Function. The function uses the public key PK to verify whether SIG is the correct signature of the message m. If SIG is the correct signature of the message m, the function will output true, otherwise output false.

3.4. Notations. We provide the definition of notations appearing in our scheme in Table 1.

4. System in Our Scheme

4.1. Multiserver Architecture. Our scheme mainly contains the following roles: a blockchain network consisting of several nodes, some registration terminals, some clients, some application servers, and some permission servers. The architecture of the system is shown in Figure 2. Now, we introduce the functions of each role:

(i) Blockchain network: the main function of the blockchain network is to receive and process transaction requests from the registered terminals and the permission servers and then ensures that the legitimate transactions are stored at each node through consensus algorithms. After comparing various blockchains [26], we build the blockchain network with reference to Hyperledger Fabric [27,28] in our scheme. We will describe the architecture of the blockchain system in detail in Section 4.2.

(ii) Registration terminal: we assume that the user registers offline by using the registration terminal and then gets a smart card. The registration terminal has the power to commit transaction proposal to the blockchain network.

(iii) Client: the remote user accesses the application server through the client. We assume that the client can learn the public key of each server before communicating.

(iv) Application server: the application server is responsible for verifying the user's identity and permission and providing services to the users who pass the verification. Based on the distributed feature of the blockchain, we reasonably assume that at any time the application server can establish a secure channel with at least one blockchain node, and then it can query the node for information on the blockchain.

(v) Permission server: the main function of the permission server is to accept requests from the users, verify their entity information, and then grant or revoke their permissions. The permission server has the authority to commit transaction proposal to the blockchain network. The permission server is usually managed by some authority in reality. In our scheme, there can exist several different permission servers to achieve flexible permission control.

4.2. Blockchain. The blockchain system structure in our scheme is referred to the Hyperledger Fabric system [27,28]. In Hyperledger Fabric system, there are four main types of nodes: the client node, the peer node, the order node, and the CA node. The peer nodes are responsible for verifying the transactions in the blocks sent by the sorting service nodes and storing copies of the blockchain. Some peer nodes also execute transactions and endorse the results, acting as the endorser node.

4.2.1. Transaction Process. Referring to the Hyperledger Fabric system and the specifics of our scheme, the transaction process on the blockchain is designed as follows: the client node (registration terminal or permission server) commits a transaction proposal to the blockchain network. The endorser node verifies the legality of transaction, endorses the transaction, and sends the result to the client node. The client node sends the result to the orderer node. The orderer node packages transactions together to generate a new block and sends it to the committer node. The committer node checks the block and appends the block to the local blockchain, and then the transaction is completed.

4.2.2. World State. In the Hyperledger Fabric system, a ledger consists of two parts: a world state and a blockchain [29].

The world state is a database that holds a cache of the current values of a set of ledger states. When a transaction occurs and causes some data to change, it is immediately reflected in the world state. The world state makes it easy for a program to directly access the current value of a state rather than having to calculate it by traversing the entire transaction log. World states are expressed as key-value pairs, and it is shown in Figure 3.

5. The Proposed Blockchain-Based Hierarchical Authentication Scheme for Multiserver Architecture

5.1. Initialization Phase. At the initialization of the system, a blockchain network is established. The blockchain network chooses an elliptic curve E over the finite field $\mathbb{F}_p$: $y^2 = x^3 + ax + b$, where $p(>2^{160})$ is a large prime number. The point P is an element with large prime order $q(>2^{160})$ on the additive group G on E. Then, it chooses the following hash function h_1: $\{0,1\}^* \longrightarrow \{0,1\}^{l_1}$,$h_2\{0,1\}^* \longrightarrow \{0,1\}^{l_2}$, where l_1, l_2 are the bit length of the hash function's output. Then, it exposes $\{\mathbb{F}_p, E, G, P, p, q, h_1, h_2\}$.

Table 1: Definition of notations.

Notation	Definition
p, q	Prime numbers
$\mathbb{F}_p$	A finite field of order p
E	An elliptic curve over the finite field $\mathbb{F}_p$
G	An additive group consisting of points on E
P	A generator of the group G with order q
U_i	i_{th} user
S_j	j_{th} server
ID_{U_i}	i_{th} user's identity
ID_{S_j}	j_{th} server's identity
PW_i	i_{th} user's password
ω_i	i_{th} user's biometrics information
R_i	i_{th} user's biometrics key
L_n	Access permission level
PK_{U_i},SK_{U_i}	i_{th} user's public-private key pair
PK_{S_j},SK_{S_j}	i_{th} application server's public-private key pair
PK_{P_r},SK_{P_r}	r_{th} permission server's public-private key pair
$a\|b$	Encoding a and b as strings from which the constituent objects are easily recoverable
$a \oplus b$	Encoding a and b as a binary bit string for an XOR operation
$s \xleftarrow{\$} S$	Operation of assigning to s an element of S chosen at random
h_1, h_2	Two secure hash functions
$\mathcal{H}_1$, $\mathcal{H}_2$	The range of the output of the hash functions h_1, h_2
$\mathcal{A}$	Adversary in the security model
$\mathcal{B}$	Challenger in the security model
$\prod_{i,j}^{t}$	t_{th} session between user U_i and server S_j

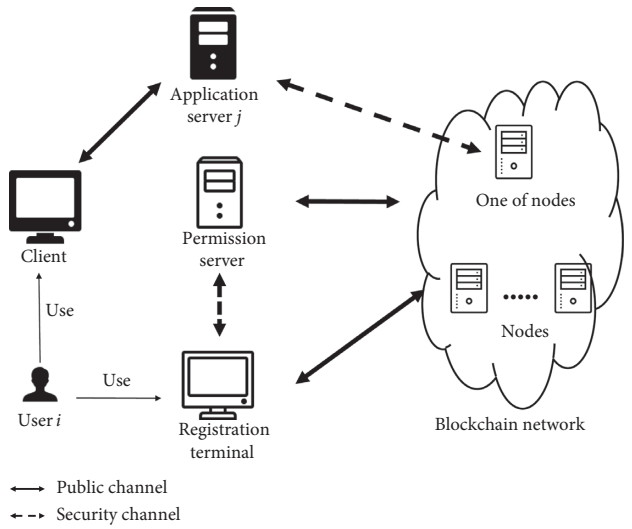

Figure 2: The architecture of the multiserver system.

5.2. Server Registration Phase. The server registration phase is as follows, and the main steps are provided in Table 2.

Step 1. The server S_j chooses the identity ID_{S_j}, selects a random number $SK_{s_j} \xleftarrow{\$} \mathbb{Z}_q^*$ as the private key, and then calculates its public key $PK_{s_j} = SK_{s_j} \cdot P$. The server uses the private key to sign the message $\left\{ID_{S_j} \| PK_{s_j}\right\}$: $\text{SIG}_{s_j} = \text{Sig}(SK_{s_j}, ID_{s_j} \| PK_{s_j})$. Then, the server commits a transaction proposal to the blockchain network. The transaction content is $\left\{\text{SIG}_{s_j}, ID_{S_j}, PK_{s_j}\right\}$.

Step 2. The blockchain network processes the transaction proposal: the endorser node verifies whether the submitter has the right for the operation and rejects this transaction if it does not; if it does, the endorser node verifies the signature $\text{Ver}(PK_{s_j}, \text{SIG}_{s_j}, ID_{s_j} \| PK_{s_j}) \stackrel{?}{=} \text{true}$; if it does not hold, the endorser node refuses to endorse this transaction; otherwise, the transaction is processed according to the transaction process we described in Section 4.2.1. Finally, the data $\left\{ID_{S_j}, PK_{s_j}\right\}$ are stored in the blockchain.

Step 3. After receiving that the notification of the transaction is successful, the server saves the SK_{s_j}. The server registration phase is complete.

5.3. User Registration Phase. Users need to register at the registration terminal offline and get his or her smart card. The user registration phase is as follows, and the main steps are provided in Table 3:

(i) *Step 1.* User U_i enters the identity ID_{U_i}. The registration terminal selects a random number $\text{SK}_{U_i} \xleftarrow{\$} \mathbb{Z}_q^*$ as the user's private key. Then, it calculates the public key $PK_{U_i} = SK_{U_i} \cdot P$, signs the message $\left\{ID_{U_i} \| PK_{U_i}\right\}$: $\text{SIG}_{U_i} = \text{Sig}(SK_{U_i}, ID_{U_i} \| PK_{U_i})$. The registration terminal packages $\left\{\text{SIG}_{U_i}, ID_{U_i}, PK_{U_i}\right\}$ as the transaction content and commits the transaction proposal to the blockchain network.

(ii) *Step 2.* The blockchain network processes the transaction proposal: the endorser node verifies

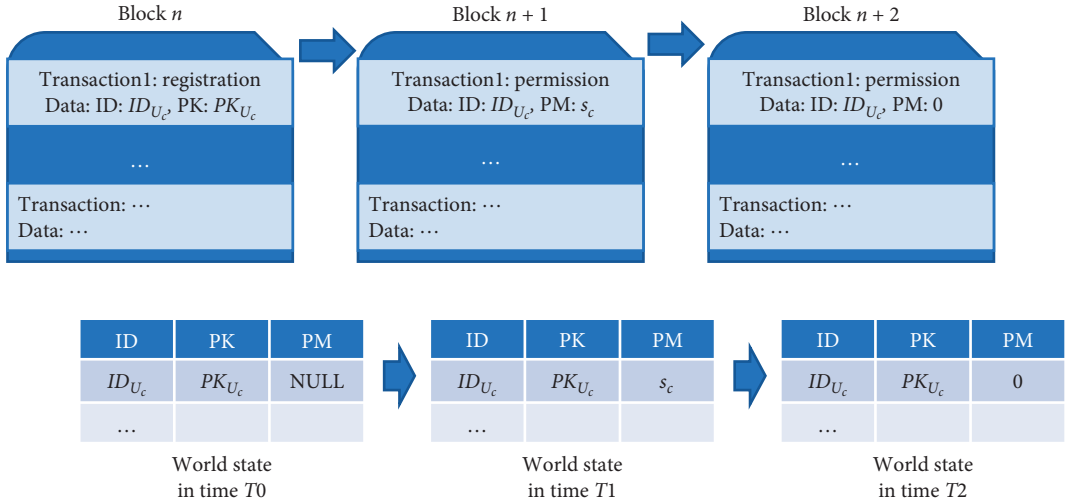

Figure 3: Blockchain and world state. The result of the transaction will be reflected in the world state. The example in the figure shows the blockchain and the change of the world state. The transaction recorded in the blockchain is the transaction of the user's registration, the user's getting access permission, and the user's revocation. The world state first records the user's identity ID_{U_c} and public key PK_{U_c}, then records the user's permission value s_c, and finally sets the permission value to 0.

Table 2: Server registration phase.

S_j		Blockchain network
$SK_{s_j} \xleftarrow{\$} \mathbb{Z}_q^*, PK_{s_j} = SK_{s_j} \cdot P$ $\mathrm{SIG}_{s_j} = \mathrm{Sig}(\mathrm{SK}_{s_j}, \mathrm{ID}_{s_j} \Vert \mathrm{PK}_{s_j})$	$\xrightarrow{\{\mathrm{SIG}_{s_j}, ID_{s_j}, PK_{s_j}\}}$	
		Verifies $\mathrm{Ver}(PK_{s_j}, \mathrm{SIG}_{s_j}, ID_{s_j} \Vert PK_{s_j}) \stackrel{?}{=}$ true if holds, stores: $\{ID_{S_j}, PK_{s_j}\}$
	$\xleftarrow{\text{Success}}$	
Stores: $\{SK_{s_j}\}$		

Table 3: User registration phase.

U_i		Blockchain network
$SK_{U_i} \xleftarrow{\$} \mathbb{Z}_q^*, PK_{U_i} = SK_{U_i} \cdot P$ $\mathrm{SIG}_{U_i} = \mathrm{Sig}(SK_{U_i}, ID_{U_i} \Vert PK_{U_i})$	$\xrightarrow{\{\mathrm{SIG}_{U_i}, ID_{U_i}, PK_{U_i}\}}$	
		Verifies $\mathrm{Ver}(PK_{U_i}, \mathrm{SIG}_{U_i}, ID_{U_i} \Vert PK_{U_i}) \stackrel{?}{=}$ true if holds, stores: $\{ID_{U_i}, PK_{U_i}\}$
	$\xleftarrow{\text{Success}}$	
$(R_i, BP_i) \leftarrow \mathrm{Gen}(\omega_i)$ $A_i = SK_{U_i} \oplus h_1(PW_i \Vert R_i)$ $B_i = h_1(h_1(ID_{U_i}) + h_1(PW_i \Vert R_i) \bmod n_0)$ Stores: $\{A_i, B_i, BP_i\}$ in the smart card		

whether the submitter has the right for the operation and rejects this transaction if it does not; if it does, the endorser node verifies $\mathrm{Ver}(PK_{U_i}, \mathrm{SIG}_{U_i}, ID_{U_i} \Vert PK_{U_i}) \stackrel{?}{=}$ true; if it does not hold, the endorser node refuses to endorse this transaction; otherwise, the transaction is processed according to the transaction process we described in Section 4.2.1. Finally, the data $\{ID_{U_i}, PK_{U_i}\}$ are stored in the blockchain and the registration terminal receives a notification of successful transaction.

(iii) *Step 3*. After the registration terminal prompted that the registration is succeed, the user U_i enters the password PW_i, the biometric information ω_i. The registration terminal runs the fuzzy-extractor generation algorithm on the input biometric information ω_i to get $(R_i, \mathrm{BP}_i) \leftarrow \mathrm{Gen}(\omega_i)$. Then, it

calculates $A_i = SK_{U_i} \oplus h_1(PW_i \| R_i)$, $B_i = h_1(h_1(ID_{U_i}) + h_1(PW_i \| R_i) \bmod n_0)$, where n_0 is a medium integer (as defined in [30]) which determines the capacity of the pool of the pair (ID_{U_i}, PW_i, R_i) against online guessing. Finally, the registration terminal saves $\{A_i, B_i, BP_i\}$ in the smart card. The user registration phase is complete.

5.4. Hierarchical Access Control Method. In order to provide hierarchical access control function and reduce the consumption of storage on the server side, we propose a lightweight hash-based hierarchical access control method based on the blockchain. After a user completing registration, the user needs to request the corresponding permission from the permission server. Only the user's permission level is higher than the permission level of the application server can access that application server. In our scheme, it is the authority agency that manages the permission server. In practical applications, the authority agency can be the controller of a multiserver system.

The proposed hierarchical access control method is as follows.

5.4.1. Initialization of Permission

(i) *Step 1*. The permission server sets a total of max levels of hierarchical access permission.

(ii) *Step 2*. The permission server takes a random number d and then publishes d.

(iii) *Step 3*. The permission server takes max random numbers $\{a_1, a_2, \ldots, a_{\max}\}$ as the permission keys.

(iv) *Step 4*. The permission server sets the access permission of the application server S_j is level L_j $(1 \le L_j \le \max)$.

(v) *Step 5*. The permission server sends $\{a_{L_j}, a_{L_j+1}, \ldots, a_{\max}\}$ to the application server S_j which access permission level is L_j in a secure channel. The permission key array in the application server of each level is listed in Table 4.

5.4.2. Granting Access Permission Phase. The user presents his or her personal information to the permission server through the registration terminal to request an access permission level, which is performed in a secure channel. The main steps are provided in Table 5:

(i) *Step 1*. The user U_i inserts the smart card and enters ID_{U_i}, PW_i, and biometric information ω_i' on the registration terminal. The registration terminal uses the fuzzy-extractor recovery algorithm to get $R_i' \leftarrow \mathrm{Rep}(\omega_i', BP_i)$ and calculate $B_i' = h_1(h_1(ID_{U_i}) + h_1(PW_i \| R_i') \bmod n_0)$. If $B_i' = B_i$, the preliminary verification passes and the smart card outputs A_i; otherwise, the smart card rejects the current login request. Then, the user inputs personal information I_i. The registration terminal calculates $SK_{U_i} = A_i \oplus h_1(PW_i \| R_i')$, $PK_{U_i} = SK_{U_i} \cdot P$, signs personal information I_i: $\mathrm{SIGI}_{U_i} = \mathrm{Sig}(SK_{U_i}, I_i)$, and sends the message $\{\mathrm{SIGI}_{U_i}, ID_{U_i}, I_i\}$ to the permission server.

Table 4: The permission key array in the application server of each level.

The level of the application server	Permission key array
max	$\{a_{\max}\}$
max − 1	$\{a_{\max-1}, a_{\max}\}$
max − 2	$\{a_{\max-2}, a_{\max-1}, a_{\max}\}$
...	...
2	$\{a_2, \ldots, a_{\max-2}, a_{\max-1}, a_{\max}\}$
1	$\{a_1, a_2, \ldots, a_{\max-2}, a_{\max-1}, a_{\max}\}$

(ii) *Step 2*. After receiving the message, the permission server queries the user's public key PK_{U_i} according to the user's identity ID_{U_i} from blockchain network, verifies the signature $\mathrm{Ver}(PK_{U_i}, \mathrm{SIGI}_{U_i}, I_i) \stackrel{?}{=} \text{true}$, and rejects the session if the verification fails; otherwise, the permission server gives the user permission level L_i according to the user's personal information I_i and then takes a random number e_i to calculate $s_i = h_1(d \| h_1(e_i \| a_{L_i}))$. Finally, the permission server signs $\{ID_{U_i} \| s_i\}$ with its own private key SK_{P_r}: $\mathrm{SIG}_{P_r} = \mathrm{Sig}(SK_{P_r}, ID_{U_i} \| s_i)$. The permission server packages $\{PK_{P_r}, \mathrm{SIG}_{P_r}, ID_{U_i}, s_i\}$ as the transaction content and commits the transaction proposal to the blockchain network.

(iii) *Step 3*. The blockchain network processes the transaction proposal: the endorser node verifies whether the submitter has the right for the operation and rejects this transaction if it does not; if it does, the endorser node verifies the signature SIG_{P_r}: $\mathrm{Ver}(PK_{P_r}, \mathrm{SIG}_{P_r}, ID_{U_i} \| s_i) \stackrel{?}{=} \text{true}$, and it refuses to endorse this transaction if the signature is not legal; otherwise, the transaction is processed according to the transaction process we described in Section 4.2.1. Finally, the blockchain stores $\{ID_{U_i}, s_i\}$.

(iv) *Step 4*. After receiving a notification that the transaction is successful, the permission server saves $\{ID_{U_i}, I_i\}$ and sends $\{L_i, e_i\}$ to the registration terminal.

(v) *Step 5*. After receiving $\{L_i, e_i\}$, the registration terminal writes $\{L_i, e_i\}$ into the smart card.

5.5. Authentication Phase. In this phase, a remote user and an application server have reached mutual authentication and key agreement. The main steps are provided in Table 6:

(i) *Step 1*. The user U_i inserts the smart card and enters ID_{U_i}, PW_i, and biometric information ω_i' on the

Table 5: Granting access permission phase.

U_i	Permission server	Blockchain network
$R_i' \leftarrow \mathrm{Rep}(\omega_i', BP_i)$ $B_i' = h_1(h_1(ID_{U_i}) + h_1(PW_i \Vert R_i') \bmod n_0)$ Verifies $B_i' \stackrel{?}{=} B_i$, if holds, outputs A_i $SK_{U_i} = A_i \oplus h_1(PW_i \Vert R_i')$ $PK_{U_i} = SK_{U_i} \cdot P$ $\mathrm{SIGI}_{U_i} = \mathrm{Sig}(SK_{U_i}, I_i)$		
$\xrightarrow{\{\mathrm{SIGI}_{U_i}, ID_{U_i}, I_i\}}$		
	Verifies $\mathrm{Ver}(PK_{U_i}, \mathrm{SIGI}_{U_i}, I_i) \stackrel{?}{=} \text{true}$ if holds: $s_i = h_1(d \Vert h_1(e_i \Vert a_{L_i}))$ $\mathrm{SIG}_{P_r} = \mathrm{Sig}(SK_{P_r}, ID_{U_i} \Vert s_i)$	
	$\xrightarrow{\{\mathrm{PK}_{P_r}, SIG_{P_r}, ID_{U_i}, s_i\}}$	
		Verifies $\mathrm{Ver}(PK_{P_r}, \mathrm{SIG}_{P_r}, ID_{U_i} \Vert s_i) \stackrel{?}{=} \text{true}$ if holds: stores: $\{ID_{U_i}, s_i\}$
	$\xleftarrow{\text{Success}}$	
	Stores: $\{ID_{U_i}, I_i\}$	
$\xleftarrow{(L_i, e_i)}$		
Stores: $\{L_i, e_i\}$ in the smart card		

Table 6: Authentication phase.

U_i	S_j
$R_i' \leftarrow \mathrm{Rep}(\omega_i', BP_i)$ $B_i' = h_1(h_1(ID_{U_i}) + h_1(PW_i \Vert R_i') \bmod n_0)$ verifies $B_i' \stackrel{?}{=} B_i$, if holds, outputs A_i, L_i, e_i $SK_{U_i} = A_i \oplus h_1(PW_i \Vert R_i')$ $PK_{U_i} = SK_{U_i} \cdot P x \stackrel{\$}{\leftarrow} \mathbb{Z}_q^*$, $X = x \cdot P$ $\mathrm{SID} = (X \Vert ID_{U_i} \Vert e_i \Vert L_i) \oplus h_2(x \cdot SK_{S_j})$	
$\xrightarrow{\{X, \mathrm{SID}\}}$	
	$(X \Vert ID_{U_i} \Vert e_i \Vert L_i) = \mathrm{SID} \oplus h_2(SK_{S_j} \cdot X)$ $s_i' = h_1(d \Vert h_1(e_i \Vert a_{L_i}))$ verifies $s_i' \stackrel{?}{=} s_i$ if holds, $y \stackrel{\$}{\leftarrow} \mathbb{Z}_q^*$, $Y = y \cdot P$ $TK_{S_j} = (SK_{S_j} + y) \cdot (PK_{U_i} + X)$ $\mathrm{AUTH}_{S_j} = h_1(TK_{S_j} \Vert X \Vert Y \Vert ID_{U_i} \Vert ID_{S_j})$
$\xleftarrow{\{Y, \mathrm{AUTH}_{S_j}\}}$	
$TK_{U_i} = (SK_{U_i} + x) \cdot (PK_{S_j} + Y)$ $\mathrm{AUTH}_{U_i} = h_1(TK_{U_i} \Vert X \Vert Y \Vert ID_{U_i} ID_{S_j})$ Verifies $\mathrm{AUTH}_{U_i} \stackrel{?}{=} \mathrm{AUTH}_{S_j}$ if holds: $\mathrm{KEY} = h_1(TK_{U_i} \Vert X \Vert Y \Vert \mathrm{AUTH}_{U_i})$ $M_{U_i} = h_1(\mathrm{KEY} \Vert X \Vert Y \Vert ID_{U_i} \Vert ID_{S_j})$	
$\longrightarrow (M_{U_i})$	
	$\mathrm{KEY} = h_1(TK_{S_j} \Vert X \Vert Y \Vert \mathrm{AUTH}_{S_j})$ $M_{S_j} = h_1(\mathrm{KEY} \Vert X \Vert Y \Vert ID_{U_i} \Vert ID_{S_j})$ verifies $M_{U_i} \stackrel{?}{=} M_{S_j}$ if it holds, accepts the session key KEY

client. The client uses the fuzzy-extractor recovery algorithm to get $R_i' \leftarrow \mathrm{Rep}(\omega_i', BP_i)$ to calculate $B_i' = h_1(h_1(ID_{U_i}) + h_1(PW_i \Vert R_i') \bmod n_0)$, and if $B_i' = B_i$, the preliminary verification passes and the smart card outputs A_i, L_i, e_i; otherwise, the smart card rejects the current login request. Then, the client calculates $SK_{U_i} = A_i \oplus h_1(PW_i \Vert R_i')$ and $PK_{U_i} = SK_{U_i} \cdot P$ and takes a random number $x \stackrel{\$}{\leftarrow} \mathbb{Z}_q^*$. The client calculates $X = x \cdot P$, $\mathrm{SID} = (X \Vert ID_{U_i} \Vert e_i \Vert L_i) \oplus h_2(x \cdot SK_{S_j})$, and then sends $\{X, \mathrm{SID}\}$ to the application server.

(ii) *Step 2.* The application server S_j calculates $(X \Vert ID_{U_i} \Vert e_i \Vert L_i) = \mathrm{SID} \oplus h_2(SK_{S_j} \cdot X)$ to get $\{X, ID_{U_i}, e_i, L_i\}$ and queries the blockchain node to get the information $\{PK_{U_i}, s_i\}$ corresponding to ID_{U_i}. The operation of querying data on the blockchain can be performed through a smart contract [31], which is called the chaincode in

Fabric. Then, the access permission verification process begins as follows:

(1) The application server finds whether the a_{L_i} exists in its permission key array, and if it does not exist, the application server rejects the user's authentication request.

(2) The application server calculates $s_i' = h_1(d \| h_1(e_i \| a_{L_i}))$ and then verifies whether s_i' is equal to s_i. If not, the application server rejects the user's authentication request.

(3) The application server completes the access permission verification process and continues the authentication phase.

The application server takes a random number $y \xleftarrow{\$} \mathbb{Z}_q^*$ and calculate $Y = y \cdot P$, $TK_{S_j} = (SK_{S_j} + y) \cdot (PK_{U_i} + X)$, $\mathrm{AUTH}_{S_j} = h_1(TK_{S_j} \| X \| Y \| ID_{U_i} \| ID_{S_j})$. After that, the application server sends $\{Y, \mathrm{AUTH}_{S_j}\}$ to the client.

(iii) *Step 3.* After receiving the massages, the client calculates $TK_{U_i} = (SK_{U_i} + x) \cdot (PK_{S_j} + Y)$, $\mathrm{AUTH}_{U_i} = h_1(TK_{U_i} \| X \| Y \| ID_{U_i} \| ID_{S_j})$ and checks whether AUTH_{U_i} and the received AUTH_{S_j} are equal. If they are equal, U_i confirms that S_j is legitimate. If not equal, the client stops the session. The client calculates $\mathrm{KEY} = h_1(TK_{U_i} \| X \| Y \| \mathrm{AUTH}_{U_i})$ and $M_{U_i} = h_1(\mathrm{KEY} \| X \| Y \| ID_{U_i} \| ID_{S_j})$. The client sends $\{M_{U_i}\}$ to the application server.

(iv) *Step 4.* The application server calculates $\mathrm{KEY} = h_1(TK_{S_j} \| X \| Y \| \mathrm{AUTH}_{S_j})$ and $M_{S_j} = h_1(\mathrm{KEY} \| X \| Y \| ID_{U_i} \| ID_{S_j})$. The application server checks whether M_{S_j} and the received M_{U_i} are equal. If they are equal, S_j confirms that U_i is legitimate and a session key KEY is negotiated by U_i and S_j. If they are not equal, the application server stops the session.

5.6. User Revocation Phase. When the smart card is lost or stolen, in order to prevent fraudulent use of the original account, the user needs to cancel all permissions of his account and then applies for a new account. The user reregistration phase is the same as the initial user registration phase. The user revocation phase is described as follows:

(i) *Step 1.* The user sends a revocation request to the permission server offline with his or her identity ID_{U_i} and personal information I_i.

(ii) *Step 2.* After receiving the message, the permission server verifies the user's personal information. If the user is confirmed to be the owner of the ID_{U_i}, the permission server signs message $\{ID_{U_i}, s_i'\}$ where $s_i' = 0$, $\mathrm{SIG}_{P_r} = \mathrm{Sig}(SK_{P_r}, ID_{U_i} \| s_i')$. Then, the permission server packages $\{\mathrm{SIG}_{P_r}, ID_{U_i}, s_i'\}$ as the transaction content and commits the transaction proposal to the blockchain network.

(iii) *Step 3.* The blockchain network processes the transaction proposal: the endorser node verifies whether the submitter has the right for the operation or not. The endorser node rejects this transaction if it does not. Otherwise, the endorser node verifies the signature SIG_{P_r}: $\mathrm{Ver}(PK_{P_r}, \mathrm{SIG}_{P_r}, ID_{U_i} \| s_i') \stackrel{?}{=}$ true, and the endorser node will refuse to endorse this transaction if the signature is not legal; otherwise, the transaction will be processed as we described in Section 4.2.1. Finally, the blockchain stores $\{ID_{U_i}, s_i'\}$, where $s_i' = 0$

6. Security Analysis

In our scheme, the user revocation phase is carried out in a secure channel, and the registration phase uses the ECDSA algorithm to submit the identity and public key $\{ID, PK\}$ to the blockchain, and we assume that the security of the blockchain system is guaranteed by the consensus algorithm; thus, we mainly analyze the security of the authentication phase. In this section, we demonstrate the security of our scheme from three aspects: the security of client-to-server authentication, the security of server-to-client authentication, and the security of session key agreement. In this section, first we construct a security model for authentication and key agreement in a multiserver environment without the registration center, then we make a formal proof of the security of our scheme, and finally we make some discussions on the security of our scheme.

6.1. Security Model. Based on the literature studies [2, 8, 11, 32], we construct a security model for the authentication and key agreement scheme in the multiserver environment without the registration center. In our security model, the game of polynomial-time adversary $\mathcal{A}$ and polynomial-time challenger $\mathcal{B}$ defines the security of our scheme. Referring to the literature [32], we define that $\mathcal{A}$ has the following capabilities:

(i) The adversary $\mathcal{A}$ can enumerate offline all the items in the Cartesian product $\mathcal{D}_{id} * \mathcal{D}_{pw} * \mathcal{D}_b$ within polynomial time, where $\mathcal{D}_{\mathrm{id}}$ denotes the identity space, $\mathcal{D}_{pw}$ denotes the password space, and $\mathcal{D}_b$ denotes the biometric key space.

(ii) The adversary $\mathcal{A}$ has the capability of somehow learning the victim's identity when evaluating security strength (but not privacy provisions) of our scheme.

(iii) The adversary $\mathcal{A}$ has full control over the communication channels between participants.

(iv) The adversary $\mathcal{A}$ can intercept the password of a legitimate user, or extract secret parameters from a smart card by side-channel attacks, or obtain biometric information of a legitimate user through a special device, but it cannot do all three at the same time.

(v) The adversary $\mathscr{A}$ can learn session keys of previous sessions.

(vi) When evaluating the security of mutual authentication, the adversary $\mathscr{A}$ can obtain the private key of the authenticating party but cannot obtain its temporary session secret in the current session, while $\mathscr{A}$ cannot obtain the private key of the authenticated party but can arbitrarily select its temporary session secret in the current session.

(vii) The adversary $\mathscr{A}$ is able to obtain the private keys of both the user and the server only when evaluating the resistance to eventual failing of the system (e.g., forward secrecy).

The adversary $\mathscr{A}$ can issue queries to the challenger $\mathscr{B}$ and get answers from $\mathscr{B}$. We define the following rules of the game:

System building phase: $\mathscr{B}$ runs the system building algorithm, initializes the public parameters and hash functions, and then exposes them to the public. Meanwhile, $\mathscr{B}$ maintains some lists during the entire process of interacting with $\mathscr{A}$ which are empty at the beginning.

(1) Hash(str): $\mathscr{A}$ issues query any string and $\mathscr{B}$ returns the corresponding hash values and guarantees that the same queries get the same values.

(2) ExtractU (ID_{U_i}): $\mathscr{B}$ maintains a list L_U. When $\mathscr{A}$ issues queries of the user's identity ID_{U_i}: if ID_{U_i} exists in L_U, $\mathscr{B}$ returns the corresponding public key; otherwise, $\mathscr{B}$ randomly generates the user's public and private key pairs and stores them in L_U, and then it returns the public key.

(3) ExtractS (ID_{S_j}): $\mathscr{B}$ maintains a list L_S. When $\mathscr{A}$ issues queries of application server's identity ID_{S_j}: if ID_{S_j} exists in L_S, $\mathscr{B}$ returns the corresponding public key; otherwise, $\mathscr{B}$ randomly generates application server's public and private key pairs and stores them in L_S, and then it returns the public key.

(4) Send ($\prod_{i,j}^{t}, m$): $\mathscr{A}$ sends a message to $\mathscr{B}$, which belongs to the t_{th} session of the user U_i and the server S_j. $\mathscr{B}$ receives the message and processes the message according to our scheme and returns the corresponding result.

(5) CorruptU (ID_{U_i}): if $\mathscr{A}$ has already sent ExtractU (ID_{U_i}) query to $\mathscr{B}$, $\mathscr{A}$ can query the user U_i's private key. $\mathscr{B}$ will return U_i's private key SK_{U_i} to $\mathscr{A}$ from the list L_U.

(6) CorruptS (ID_{S_j}): if $\mathscr{A}$ has already sent ExtractS (ID_{S_j}) query to $\mathscr{B}$, $\mathscr{A}$ can query the application server S_j's private key. $\mathscr{B}$ will return S_j's private key SK_{S_j} to $\mathscr{A}$ from the list L_S.

(7) Reveal ($\prod_{i,j}^{t}$): $\mathscr{A}$ issues query for the session key of the session $\prod_{i,j}^{t}$, and $\mathscr{B}$ returns the session key generated by the session $\prod_{i,j}^{t}$.

6.1.1. Issue Query Phase. After issuing the queries above, the adversary $\mathscr{A}$ begins the challenge phase: $\mathscr{A}$ initiates a session key guess for session $\prod_{i,j}^{t}$. If $\mathscr{A}$ has never sent Reveal ($\prod_{i,j}^{t}$) query, challenger $\mathscr{B}$ randomly flips a coin to obtain $b \in \{0, 1\}$. If $b = 1$, the challenger $\mathscr{B}$ outputs the session key of the session $\prod_{i,j}^{t}$. If $b = 0$, $\mathscr{B}$ outputs a random number which length is equal to that of the session key of the session $\prod_{i,j}^{t}$. $\mathscr{A}$ guesses the value of b and outputs $b\prime$. If $b = b\prime$, we consider that $\mathscr{A}$ wins the game. We define the advantage of $\mathscr{A}$ winning the game as

$$\mathrm{Adv}^{\mathrm{AKA}}(\mathscr{A}) = \left|\Pr\left(\mathscr{A}_{b=b\prime}\right) - \frac{1}{2}\right|. \tag{1}$$

Definition 1. AKA-Secure: our scheme is authentication key agreement secure (AKA-Secure) if $\mathrm{Adv}^{\mathrm{AKA}}(\mathscr{A})$ is negligible for any polynomial-time adversary $\mathscr{A}$.

If $\mathscr{A}$ can forge the authentication message sent by the legitimate user to the legitimate server, or the response message sent by the legitimate server to the legitimate user, then our scheme is considered insecure. Defining the event $\mathscr{A}_{U_i \longrightarrow S_j}$ as $\mathscr{A}$ can successfully forge the authentication message sent by the legitimate user U_i to the legitimate server S_j. Defining the event $\mathscr{A}_{S_j \longrightarrow U_i}$ as $\mathscr{A}$ can successfully forge the response message sent by the legitimate server S_j to the legitimate user U_i, then we can define the advantage of $\mathscr{A}$ attacking the mutual authentication game as

$$\mathrm{Adv}^{\mathrm{MA}}(\mathscr{A}) = \Pr\left(\mathscr{A}_{U_i \longrightarrow S_j}\right) + \Pr\left(\mathscr{A}_{S_j \longrightarrow U_i}\right). \tag{2}$$

Definition 2. MA-Secure: our scheme is mutual authentication secure (MA-Secure) if $\mathrm{Adv}^{\mathrm{MA}}(\mathscr{A})$ is negligible for any polynomial-time adversary $\mathscr{A}$.

6.2. Proof of Security. In this part, we will show our scheme is AKA-secure and MA-secure in the security model we described above.

Lemma 1. *No polynomial-time adversary $\mathscr{A}$ can forge an authentication message sent by the legitimate user to the legitimate server with a nonnegligible probability if the ECCDH problem is hard.*

Proof

Suppose that a polynomial-time adversary $\mathscr{A}$ can forge an authentication message sent by the legitimate user to the legitimate server with a nonnegligible probability, then the challenger $\mathscr{B}$ can solve an instance of the ECCDH problem by a nonnegligible advantage. Given an instance of the ECCDH problem, we assume that $\mathscr{A}$ eventually chooses to forge the authentication message sent by the legitimate user U_c to the legitimate server S_d as a challenge, and the objective of challenger $\mathscr{B}$ is to compute $SK_{U_c} \cdot y \cdot P$ as SK_{U_c}, y are unknown elements in $\mathbb{Z}_q^*$, where $PK_{U_c} = SK_{U_c} \cdot P$, $Y = y \cdot P$, and $PK_{U_c}, Y, P \in G$. In the following, we will demonstrate how $\mathscr{B}$ can solve the ECCDH problem by exploiting the adversary $\mathscr{A}$.

$\mathscr{B}$ runs the system building algorithm: $\mathscr{B}$ initializes the public parameters and hash function params = $\{\mathbb{F}_p, E, G, P, p, q, h_1, h_2\}$ and sends them to $\mathscr{A}$. Meanwhile,

$\mathcal{B}$ maintains several lists L_H, L_U, L_S, which are empty at the beginning. Then, $\mathcal{A}$ makes the following queries:

(1) $\text{Hash}_n(\text{str})$: $\mathcal{B}$ maintains a list $L_{Hn} = \{\text{str}, h_n(\text{str})\}$, which is empty at the beginning. After receiving the query $\text{Hash}_n(\text{str})$ from $\mathcal{A}$, $\mathcal{B}$ checks whether $\{\text{str}, h_n(\text{str})\}$ exists in the list L_{Hn}. If it exists, $\mathcal{B}$ returns the corresponding value $h_n(\text{str})$; otherwise, $\mathcal{B}$ takes a random number $h_n(\text{str})$, stores $\{\text{str}, h_n(\text{str})\}$ in L_{Hn}, and finally returns $h_n(\text{str})$ to $\mathcal{A}$.

(2) Extract U (ID_{U_i}): $\mathcal{B}$ maintains a list $L_U = \{ID_{U_i}, SK_{U_i}, PK_{U_i}\}$. After receiving the query ExtractU (ID_{U_i}) from $\mathcal{A}$, $\mathcal{B}$ checks whether $\{ID_{U_i}, SK_{U_i}, PK_{U_i}\}$ exists in L_U or not. If it exists, $\mathcal{B}$ returns PK_{U_i} to $\mathcal{A}$. Otherwise, $\mathcal{B}$ executes the operations as follows:

(i) If $i \neq c$, $\mathcal{B}$ takes $SK_{U_i} \xleftarrow{\$} \mathbb{Z}_q^*$ as the private key of U_i and calculates the public key $PK_{U_i} = SK_{U_i} \cdot P$, then stores $\{ID_{U_i}, SK_{U_i}, PK_{U_i}\}$ into the list L_U, finally returns PK_{U_i} to $\mathcal{A}$.

(ii) If $i = c$, $\mathcal{B}$ takes $PK_{U_c} \xleftarrow{\$} G$ as the public key of U_c and sets the private key SK_{U_c} to $\perp$ (which means an unknown value), then stores $\{ID_{U_c}, SK_{U_c}, PK_{U_c}\}$ into the list L_U, finally returns PK_{U_c} to $\mathcal{A}$.

(3) ExtractS (ID_{S_j}): $\mathcal{B}$ maintains a list $L_S = \{ID_{S_j}, SK_{S_j}, PK_{S_j}\}$. After receiving the query ExtractS (ID_{S_j}) from $\mathcal{A}$, $\mathcal{B}$ checks whether $\{ID_{S_j}, SK_{S_j}, PK_{S_j}\}$ exists in L_S, if it exists, $\mathcal{B}$ returns PK_{S_j} to $\mathcal{A}$. Otherwise, $\mathcal{B}$ takes $SK_{S_j} \xleftarrow{\$} \mathbb{Z}_q^*$ as the private key of S_j, then calculates the public key $PK_{S_j} = SK_{S_j} \cdot P$, then stores $\{ID_{S_j}, SK_{S_j}, PK_{S_j}\}$ into the list L_S, and returns PK_{S_j} to $\mathcal{A}$.

(4) Send $(\prod_{i,j}^t, m)$: $\mathcal{A}$ sends a message m belonging to the session $\prod_{i,j}^t$ to $\mathcal{B}$. $\mathcal{B}$ processes the message m according to the following rules:

(i) If $\prod_{i,j}^t \neq \prod_{c,d}^t$, $\mathcal{B}$ processes the received message according to our scheme and sends the result to $\mathcal{A}$.

(ii) If $\prod_{i,j}^t = \prod_{c,d}^t$ and $m =$ 'begin', $\mathcal{B}$ picks a random number $x \xleftarrow{\$} \mathbb{Z}_q^*$, calculates $X = x \cdot P$, $\text{SID} = (X \| ID_{U_c} \| e_i \| L_i) \oplus h_2(x \cdot PK_{S_d})$, and sends $\{x, X, \text{SID}\}$ to $\mathcal{A}$.

(iii) If $\prod_{i,j}^t = \prod_{c,d}^t$ and $m = \{X, \text{SID}\}$, $\mathcal{B}$ calculates $(X \| ID_{U_c} \| e_i \| L_i) = \text{SID} \oplus h_2(SK_{S_d} \cdot X)$, picks a random point $Y \xleftarrow{\$} G$, sets y to $\perp$ (which means an unknown value), and sends Y to $\mathcal{A}$.

(iv) If $\prod_{i,j}^t = \prod_{c,d}^t$ and $m = \{Y, \text{AUTH}_{S_d}\}$, $\mathcal{B}$ aborts the game.

(v) If $\prod_{i,j}^t = \prod_{c,d}^t$ and $m = \{M_{U_c}\}$, $\mathcal{B}$ aborts the game.

(5) CorruptU (ID_{U_i}): if $\mathcal{A}$ has already sent ExtractU (ID_{U_i}) query to $\mathcal{B}$, $\mathcal{A}$ can query the user U_i's private key. If $i = c$, $\mathcal{B}$ aborts the game; otherwise $\mathcal{B}$ searches for U_i's private key SK_{U_i} in the list L_U and returns SK_{U_i} to $\mathcal{A}$.

(6) CorruptS (ID_{S_j}): if $\mathcal{A}$ has already sent ExtractS (ID_{S_j}) query to $\mathcal{B}$, $\mathcal{A}$ can query the application. $\mathcal{B}$ searches for S_j's private key SK_{S_j} in the list L_S and returns SK_{S_j} to $\mathcal{A}$.

(7) Reveal $(\prod_{i,j}^t)$: $\mathcal{A}$ issues query for the session key of the session $\prod_{i,j}^t$. If $\prod_{i,j}^t = \prod_{c,d}^t$, $\mathcal{B}$ aborts the game; otherwise, $\mathcal{B}$ returns the session key generated by the session $\prod_{i,j}^t$.

According to the above queries, if $\mathcal{A}$ can successfully forge an authentication message sent by the legitimate user U_c to the legitimate server S_d, it means that $\mathcal{A}$ can output M_{U_c} belonging to the session $\prod_{c,d}^t$ where $M_{U_c} = h_1(\text{KEY} \| X \| Y \| ID_{U_c} \| ID_{S_d})$ and $\text{KEY} = h_1(\text{TK}_{U_c} \| X \| Y \| \text{AUTH}_{U_c})$. According to the given instance of the ECCDH problem, $\mathcal{B}$ can transform the equation

$$TK_{U_c} = (SK_{U_c} + x) \cdot (PK_{S_d} + Y), \tag{3}$$

to get

$$SK_{U_c} \cdot y \cdot P = TK_{U_c} - SK_{S_d} \cdot PK_{U_c} - SK_{S_d} \cdot X - x \cdot Y, \tag{4}$$

and $\mathcal{B}$ gets $SK_{S_d}, PK_{U_c}, X, x, Y$ by searching the query records and gets TK_{U_c} by search the hash-query records with the probability of $(1/q_h)$ (q_h denotes the number of hash-queries). Then, $\mathcal{B}$ outputs $TK_{U_c} - SK_{S_d} \cdot PK_{U_c} - SK_{S_d} \cdot X - x \cdot Y$ as the solution to the given instance of the ECCDH problem. The probability that $\mathcal{B}$ can solve the ECCDH problem is described as follows.

We define the event E_1 as $\mathcal{B}$ does not abort in any queries. We define the event E_2 as $\mathcal{A}$ successfully forges an authentication message sent by the legitimate user U_c to the legitimate server S_d. Let l denote the number of bits of biometric information, q_s denote the number of send-queries, and ε demote the probability of $\mathcal{A}$ successfully forges an authentication message sent by the legitimate user U_c to the legitimate server S_d. We use Zipf's law [33] to enhance the security of the security model. C' and s' are Zipf's parameters. We can get $\Pr(E_1) \geq (1 - (1/q_s + 1))^{q_s}$ and $\Pr(E_2|E_1) \geq \varepsilon$, and then we get

$$\begin{aligned} \Pr_{\mathcal{B}\text{win}}(\text{ECCDH}) &= \frac{1}{q_h} \cdot \Pr(E_2 \cap E_1) \\ &= \frac{1}{q_h} \cdot \Pr(E_2|E_1) \cdot \Pr(E_1) \\ &\geq \frac{1}{q_h} \cdot \left(1 - \frac{1}{q_s + 1}\right)^{q_s} \cdot \varepsilon \\ &\quad + \max\left\{C' \cdot q_s^{s'}, \frac{q_s}{2^l}\right\}. \end{aligned} \tag{5}$$

It implies that $\mathscr{B}$ can solve an instance of the ECCDH problem with nonnegligible probability. This is contradicting with the hardness of the ECCDH problem. Therefore, we conclude that no polynomial-time adversary can forge an authentication message sent by the legitimate user to the legitimate server with a nonnegligible probability if the ECCDH problem is hard.

Lemma 2. *No polynomial-time adversary $\mathscr{A}$ can forge a response message sent by the legitimate server to the legitimate user with a nonnegligible probability if the ECCDH problem is hard.*

Proof

Suppose that a polynomial-time adversary $\mathscr{A}$ can forge a response message sent by the legitimate server to the legitimate user with a nonnegligible probability, then the challenger $\mathscr{B}$ can solve an instance of the ECCDH problem by a nonnegligible advantage. Given an instance of the ECCDH problem, we assume that $\mathscr{A}$ eventually chooses to forge the response message sent by the legitimate server S_d to the legitimate user U_c as a challenge, and the objective of challenger $\mathscr{B}$ is to compute $SK_{S_d} \cdot x \cdot P$ as SK_{S_d}, x are unknown elements in $\mathbb{Z}_q^*$, where $PK_{S_d} = SK_{S_d} \cdot P$, $X = x \cdot P$, and $\mathrm{PK}_{S_d}, X, P \in G$. In the following, we will demonstrate how $\mathscr{B}$ can solve the ECCDH problem by exploiting the adversary $\mathscr{A}$.

$\mathscr{B}$ runs the system building algorithm: $\mathscr{B}$ initializes the public parameters and hash function $\mathrm{params} = \{\mathbb{F}_p, E, G, P, p, q, h_1, h_2\}$ and sends them to $\mathscr{A}$. Meanwhile, $\mathscr{B}$ maintains several lists L_H, L_U, L_S, which are empty at the beginning. Then, $\mathscr{A}$ makes the following queries:

(1) $\mathrm{Hash}_n(\mathrm{str})$: $\mathscr{B}$ maintains a list $L_{Hn} = \{\mathrm{str}, h_n(\mathrm{str})\}$, which is empty at the beginning. After receiving the query $\mathrm{Hash}_n(\mathrm{str})$ from $\mathscr{A}$, $\mathscr{B}$ checks whether $\{\mathrm{str}, h_n(\mathrm{str})\}$ exists in the list L_{Hn}; if it exists, $\mathscr{B}$ returns the corresponding value $h_n(\mathrm{str})$; otherwise, $\mathscr{B}$ takes a random number $h_n(\mathrm{str})$, stores $\{\mathrm{str}, h_n(\mathrm{str})\}$ in L_{Hn}, and finally returns $h_n(\mathrm{str})$ to $\mathscr{A}$.

(2) ExtractU(ID_{U_i}): $\mathscr{B}$ maintains a list $L_U = \{ID_{U_i}, SK_{U_i}, PK_{U_i}\}$. After receiving the query ExtractU (ID_{U_i}) from $\mathscr{A}$, $\mathscr{B}$ checks whether $\{ID_{U_i}, SK_{U_i}, PK_{U_i}\}$ exists in L_U; if it exists, $\mathscr{B}$ returns PK_{U_i} to $\mathscr{A}$. Otherwise, $\mathscr{B}$ takes $SK_{U_i} \xleftarrow{\$} \mathbb{Z}_q^*$ as the private key of U_i, then calculates the public key $PK_{U_i} = SK_{U_i} \cdot P$, then stores $\{ID_{U_i}, SK_{U_i}, PK_{U_i}\}$ into the list L_U, and returns PK_{U_i} to $\mathscr{A}$.

(3) ExtractS (ID_{S_j}): $\mathscr{B}$ maintains a list $L_S = \{ID_{S_j}, SK_{S_j}, PK_{S_j}\}$. After receiving the query ExtractS (ID_{S_j}) from $\mathscr{A}$, $\mathscr{B}$ checks whether $\{ID_{S_j}, SK_{S_j}, PK_{S_j}\}$ exists in L_S or not. If it exists, $\mathscr{B}$ returns PK_{S_j} to $\mathscr{A}$. Otherwise, $\mathscr{B}$ executes the operations as follows:

(i) If $j \neq d$, $\mathscr{B}$ takes $SK_{S_j} \xleftarrow{\$} \mathbb{Z}_q^*$ as the private key of S_j and calculates the public key $PK_{S_j} = SK_{S_j} \cdot P$, then stores $\{ID_{S_j}, SK_{S_j}, PK_{S_j}\}$ into the list L_S, finally returns PK_{S_j} to $\mathscr{A}$.

(ii) If $j = d$, $\mathscr{B}$ takes $PK_{S_d} \xleftarrow{\$} G$ as the public key of S_d and sets the private key SK_{S_d} to $\perp$ (which means an unknown value), then stores $\{ID_{S_d}, SK_{S_d}, PK_{S_d}\}$ into the list L_S, finally returns PK_{S_d} to $\mathscr{A}$.

(4) Send($\prod_{i,j}^t, m$): $\mathscr{A}$ sends a message m belonging to the session $\prod_{i,j}^t$ to $\mathscr{B}$. $\mathscr{B}$ processes the message m according to the following rules:

(i) If $\prod_{i,j}^t \neq \prod_{c,d}^t$, $\mathscr{B}$ processes the received message according to our scheme and sends the result to $\mathscr{A}$.

(ii) If $\prod_{i,j}^t = \prod_{c,d}^t$ and m = 'begin', $\mathscr{B}$ picks a random point $X \xleftarrow{\$} G$, sets x to $\perp$ (which means an unknown value), picks a random number $\mathrm{SID} \xleftarrow{\$} \mathscr{H}_2$, and sends $\{X, \mathrm{SID}\}$ to $\mathscr{A}$.

(iii) If $\prod_{i,j}^t = \prod_{c,d}^t$ and $m = \{X, \mathrm{SID}\}$, $\mathscr{B}$ picks a random number $y \xleftarrow{\$} \mathbb{Z}_q^*$ and calculates $Y = y \cdot P$, and sends $\{Y, y\}$ to $\mathscr{A}$.

(iv) If $\prod_{i,j}^t = \prod_{c,d}^t$ and $m = \{Y, \mathrm{AUTH}_{S_d}\}$, $\mathscr{B}$ aborts the game.

(v) If $\prod_{i,j}^t = \prod_{c,d}^t$ and $m = \{M_{U_c}\}$, $\mathscr{B}$ aborts the game.

(5) CorruptU (ID_{U_i}): if $\mathscr{A}$ has already sent ExtractU (ID_{U_i}) query to $\mathscr{B}$, $\mathscr{A}$ can query the user U_i's private key. $\mathscr{B}$ searches for U_i's private key SK_{U_i} in the list L_U and returns SK_{U_i} to $\mathscr{A}$.

(6) CorruptS (ID_{S_j}): if $\mathscr{A}$ has already sent ExtractS (ID_{S_j}) query to $\mathscr{B}$, $\mathscr{A}$ can query the application server S_j's private key. If $j = d$, $\mathscr{B}$ aborts the game; otherwise, $\mathscr{B}$ searches for S_j's private key SK_{S_j} in the list L_S and returns SK_{S_j} to $\mathscr{A}$.

(7) Reveal ($\prod_{i,j}^t$): $\mathscr{A}$ issues query for the session key of the session $\prod_{i,j}^t$. If $\prod_{i,j}^t = \prod_{c,d}^t$, $\mathscr{B}$ aborts the game; otherwise, $\mathscr{B}$ returns the session key generated by the session $\prod_{i,j}^t$.

According to the above queries, if $\mathscr{A}$ can successfully forge a response message sent by the legitimate server S_d to the legitimate user U_c, it means that $\mathscr{A}$ can output AUTH_{S_d} belonging to the session $\prod_{c,d}^t$ where $\mathrm{AUTH}_{S_d} = h_1(TK_{S_d} XY \| ID_{U_c} \| ID_{S_d})$. According to the given instance of the ECCDH problem, $\mathscr{B}$ can transform the equation

$$TK_{S_d} = \left(SK_{S_d} + y\right) \cdot \left(PK_{U_c} + X\right), \tag{6}$$

to get

$$SK_{S_d} \cdot x \cdot P = TK_{S_d} - SK_{U_c} \cdot PK_{S_d} - SK_{U_c} \cdot Y - y \cdot X, \tag{7}$$

and $\mathscr{B}$ gets $SK_{U_c}, PK_{S_d}, X, y, Y$ by searching the query records and gets TK_{S_d} by search the hash-query records with

the probability of $(1/q_h)$ (q_h denotes the number of hash-queries). Then, $\mathscr{B}$ outputs $TK_{S_d} - SK_{U_c} \cdot PK_{S_d} - SK_{U_c} \cdot Y - y \cdot X$ as the solution to the given instance of the ECCDH problem. The probability that $\mathscr{B}$ can solve the ECCDH problem is described as follows.

Defining the event E_1 as $\mathscr{B}$ does not abort in any queries. Defining the event E_2 as $\mathscr{A}$ successfully forges an authentication message sent by the legitimate user U_c to the legitimate server S_d. Let l denote the number of bits of biometric information, q_s denote the number of send-queries, and ε demote the probability of $\mathscr{A}$ successfully forges an authentication message sent by the legitimate user U_c to the legitimate server S_d. We can get $\Pr(E_1) \geq (1 - (1/q_s + 1))^{q_s}$, $\Pr(E_2|E_1) \geq \varepsilon$, and then we get

$$\begin{aligned} \Pr_{\mathscr{B}\text{win}}(\text{ECCDH}) &= \frac{1}{q_h} \cdot \Pr(E_2 \cap E_1) \\ &= \frac{1}{q_h} \cdot \Pr(E_2|E_1) \cdot \Pr(E_1) \\ &\geq \frac{1}{q_h} \cdot \left(1 - \frac{1}{q_s + 1}\right)^{q_s} \cdot \varepsilon. \end{aligned} \tag{8}$$

It implies that $\mathscr{B}$ can solve an instance of the ECCDH problem with nonnegligible probability. This is contradicting with the hardness of the ECCDH problem. Therefore, we conclude that no polynomial-time adversary can forge an authentication message sent by the legitimate user to the legitimate server with a nonnegligible probability if the ECCDH problem is hard.

Theorem 1. *Our scheme is MA-secure if the ECCDH problem is hard.*

Proof

According to Lemma 1 and Lemma 2, we learn that if the ECCDH problem is hard, no polynomial-time adversary can forge an authentication message sent by the legitimate user to the legitimate server with a nonnegligible probability and no polynomial-time adversary can forge a response message sent by the legitimate server to the legitimate user with a nonnegligible probability. Therefore, our scheme is MA-secure.

Theorem 2. *Our scheme is AKA-secure if the ECCDH problem is hard.*

Proof

In the proof of Lemma 1, we learn if $\mathscr{A}$ can successfully forge an authentication message sent by the legitimate user U_c to the legitimate server S_d, it means that $\mathscr{A}$ sends M_{U_c} belonging to the session $\prod_{c,d}^{t}$ to $\mathscr{B}$, where $M_{U_c} = h_1(\text{KEY}\|X\|Y\|ID_{U_c}\|ID_{S_d})$. Then, $\mathscr{B}$ can get KEY as the session key of the session $\prod_{c,d}^{t}$ from the query records with probability of $\Pr(\mathscr{A}_{U_i \longrightarrow S_j})$. However, we learn that no polynomial-time adversary can forge an authentication message sent by the legitimate user to the legitimate server with a nonnegligible probability if the ECCDH problem is hard in Lemma 1. Therefore, our scheme is AKA-secure if the ECCDH problem is hard.

6.3. Security Requirement Analysis. The research of Wang et al. [34] revealed a variety of security defects that are common in the multiserver architecture, such as no truly multifactor security and temporary information attack. Our scheme takes these security defects into consideration and meets the following security requirements.

6.3.1. User Anonymity. In our scheme, user's identity ID_{U_i} exists in the message SID, M_{U_i}, and AUTH_{S_j}, where $\text{SID} = (X\|ID_{U_i}\|e_i\|L_i) \oplus h_2(x \cdot PK_{S_j})$, $M_{U_i} = h_1(\text{KEY}\|X\|Y\|ID_{U_i}\|ID_{s_j})$, and $\text{AUTH}_{S_j} = h_1(\text{TK}_{S_j}\|X\|Y\|ID_{U_i}\|ID_{s_j})$. It is impossible for the adversary to get ID_{U_i} by inferring SID because $(X\|ID_{U_i}\|e_i\|L_i)$ is encrypted by $h_2(x \cdot PK_{S_j})$, and at the same time, because x is an unknown number, based on the difficulty of the ECDL problem, the adversary cannot get the value of $x \cdot PK_{S_j}$. And it is impossible for the adversary to know ID_{U_i} by inferring M_{U_i} or AUTH_{S_j} because of the one-way property of the hash function. Therefore, our scheme can provide user anonymity.

6.3.2. Untraceability. In our scheme, each session generates new random secret values x and y. Due to the randomness of x and y, the values of message SID, M_{U_i}, and AUTH_{S_j} which contain ID_{U_i} or ID_{S_j} are different in each session and the polynomial-time adversary cannot analyze their interconnection even for the same user and server's session. Therefore, our scheme can provide untraceability.

6.3.3. Perfect Forward Secrecy. We assume that the adversary obtains the private keys of the user U_i and the application server S_j and intercepts the messages $\{X, \text{SID}\}$, $\{Y, \text{AUTH}_{S_j}\}$, $\{M_{U_i}\}$ of a past session between the user U_i and the server S_j, but the adversary does not obtain the temporary session secrets x, y of that session, because the temporary session secrets are deleted immediately after used. If the adversary wants to get the session key of that session, it needs to calculate $\text{KEY} = h_1(TK\|X\|Y\|\text{AUTH})$, where $TK = (SK_{U_i} + x) \cdot (SK_{S_j} + y) \cdot P$, and x, y is unknown elements. It means that the adversary must solve an instance of the ECCDH problem. Therefore, if the ECCDH problem is hard, our scheme can provide the perfect forward secrecy.

6.3.4. No Smart Card Lose Attack. According to the definition of the adversary, we assume that the adversary obtains the data A_i and B_i in the user's smart card by using the side-channel attack. Because we use the fuzzy-verifier techniques [30,32] in the process of recovering the user's private key through smart card, different input $\{\text{PW}_i, R_i\}$ may output the same B_i', where $B_i' = h_1(ID_{U_i} + h(PW_i\|R_i) \bmod n_0)$.

Thus, we suppose that the adversary gets $\{PW_i', R_i'\}$ which makes $B_i' = B_i$, and then it calculates $SK_{U_i}' = A_i \oplus h_1(PW_i' \| R_i')$, and it cannot verify the correctness of SK_{U_i}'. Therefore, our scheme can resist the smart card lose attack.

6.3.5. No Stolen Verifier Table Attack. In the stolen verifier table attack, the adversary misuses valued data which are stored at servers' side such as passwords or other parameters and masquerade as legal users [35]. In our scheme, the blockchain maintains the repository that stores public keys, rather than servers or RC maintain the verifier table. Therefore, our scheme can resist the stolen verifier attack.

6.3.6. Three-Factor Security. In our scheme, user's password PW_i, personal biometrics ω_i, and the secret data A_i, B_i make up the three-factor security. Based on the constructed security model, the adversary cannot obtain all three of these security factors at the same time. Therefore, we discuss the case where the adversary obtains two security factors:

(i) The adversary obtains both the user's password PW_i and the secret data A_i, B_i in the smart card or both the user's personal biometrics ω_i and the secret data A_i, B_i in the smart card: according to the security analysis in Section 6.3.4, we can get the adversary cannot verify the correctness of the user's private key SK_{U_i}'.

(ii) The adversary obtains the user's personal biometrics ω_i and the password PW_i: the adversary cannot calculate $SK_{U_i} = A_i \oplus h_1(PW_i \| R_i)$ without the secret data A_i in the smart card. Therefore, the adversary cannot obtain the user's private key SK_{U_i}.

Therefore, our scheme can provide three-factor security.

6.3.7. Decentralized Registration. In our scheme, users register their accounts on the blockchain network. The blockchain network is composed of multiple nodes, allowing up to one-third of the nodes to be down without affecting the function of the blockchain network.

6.3.8. Decentralized Authentication. In our scheme, participants only need to obtain each other's public key from any blockchain node to complete mutual authentication and key agreement according to our scheme. According to the distributed feature of blockchain nodes, our scheme will not suffer from the security risks caused by centralization such as the leak of master key or the downtime of the registration center server.

6.3.9. Hierarchical Access Control. In our scheme, the application server needs to verify the access permission of the user before responding to the user login. The n-level application server has the permission key array $\{a_n, a_{n+1}, \ldots, a_{\max-2}, a_{\max-1}, a_{\max}\}$, so it cannot verify the user's permission whose access permission level is lower than it. If the user wants to impersonate a higher level of access permission, the application server calculates $s_i' = h_1(d \| h_1(e_i \| a_{L_i}))$, verifies $s_i' \stackrel{?}{=} s_i$, and if it does not hold, the application server rejects the user.

6.3.10. Resistance to User Impersonation Attack. According to the proof of Lemma 1, no polynomial-time adversary can forge an authentication message sent by the legitimate user U_i to the legitimate server S_j without the private key SK_{U_i} of U_i and the server's temporary session secret y. The server can find out about the attack by verifying the authentication message sent by the user. Therefore, our scheme can resist the user impersonation attack.

6.3.11. Resistance to Server Spoofing Attack. According to the proof of Lemma 2, no polynomial-time adversary can forge a response message sent by the legitimate server S_j to the legitimate user U_i without the private key SK_{S_j} of S_j and the user's temporary session secret x. The user can find out about the attack by verifying the response message sent by the server. Therefore, our scheme can resist the server spoofing attack.

6.3.12. Resistance to Replay Attack. In our scheme, each session generates new random temporary session secrets x and y. We discuss the replay attack in the following two scenarios:

(i) Replay the user's message: after receiving the message sent by the user, the server will calculate $M_{S_j} = h_1(\mathrm{KEY} \| X \| Y \| ID_{U_i} \| ID_{s_j})$, where Y is generated by the server based on temporary session secret y, and its value in each session is different. If the adversary replays the user's message in the previous session $\{X', \mathrm{SID}'\}$ and $\{M_{U_i}'\}$, where $M_{U_i}' = h_1(\mathrm{KEY}' \| X' \| Y' \| ID_{U_i} \| ID_{s_j})$. The application server can determine whether the authentication message belongs to the current session of this server S_j with user U_i by verifying whether M_{U_i} is equal to M_{S_j}.

(ii) Replay the application server's message: after receiving the message sent by the server, the user will calculate $\mathrm{AUTH}_{U_i} = h_1(TK_{U_i} \| X \| Y \| ID_{U_i} \| ID_{S_j})$, where x is the temporary session secret, and its value in each session is different. If the adversary replays the server's message in the previous session $\{Y', \mathrm{AUTH}_{S_j}'\}$, where $\mathrm{AUTH}_{S_j}' = h_1(TK_{S_j} \| X' \| Y' \| ID_{U_i} \| ID_{s_j})$. The client can determine whether the response messages belong to the current session by verifying whether AUTH_{S_j} is equal to AUTH_{U_i}.

Therefore, our scheme can resist the replay attack.

6.3.13. Resistance to Modification Attack. According to the proof of Lemma 1 and Lemma 2, no polynomial-time adversary can modify the values of the authentication messages

Table 7: Security comparison.

Security requirements	Our scheme	Kou et al.'s scheme [2]	Ying and Nayak's scheme [36]	Xiong et al.'s scheme [11]	He and Wang's scheme [4]	Odelu et al.'s scheme [5]
User anonymity	✓	✓	✓	✓	✓	✓
Untraceability	✓	✓	✓	✓	✓	✓
Decentralized registration	✓	✗	✗	✓	✗	✗
No smart card lose attack	✓	✓	✓	✓	✗	✗
Impersonation attack	✓	✓	✓	✓	✓	✓
Replay attack	✓	✓	✓	✓	✓	✓
Man-in-the-middle attack	✓	✓	✓	✓	✗	✓
Hierarchical access control	✓	✓	✗	✗	✗	✗

or the response message so as to make the message pass the verification. Therefore, our scheme can resist the modification attack.

6.3.14. Resistance to Temporary Information Attack. We assume that the adversary obtains the temporary session secrets x, y and intercepts the massages $\{X, \mathrm{SID}\}$, $\left\{Y, \mathrm{AUTH}_{S_j}\right\}$, and $\left\{M_{U_i}\right\}$ of a past session between the user U_i and the server S_j, but the adversary does not obtain the private keys of the user U_i and the application server S_j. If the adversary wants to get the session key of that session, it needs to calculate $\mathrm{KEY} = h_1(TK \| X \| Y \| \mathrm{AUTH})$, where $\mathrm{TK} = (SK_{U_i} + x) \cdot (SK_{S_j} + y) \cdot P$, and SK_{U_i}, SK_{S_j} are unknown elements. It means that the adversary must solve an instance of the ECCDH problem. Therefore, if the ECCDH problem is hard, our scheme can resist the temporary information attack.

6.3.15. Resistance to Insider Attack. In our scheme, the participant's private key is generated by itself and does not rely on the registration center, so there is no insider attack.

6.3.16. Resistance to Man-in-the-Middle Attack. As an attack that aims at circumventing mutual authentication, the man-in-the-middle attack can succeed only when the adversary can impersonate each participant to their satisfaction as expected from the legitimate participant. Accounting to the previous security analysis, no adversary can impersonate a legitimate participant in our scheme. Therefore, our scheme can resist the man-in-the-middle attack.

7. Performance Analysis

In this section, we compare the performance of the proposed scheme with other multiserver architecture schemes which include Kou et al.'s scheme [2], Ying and Nayak's scheme [36], Xiong et al.'s scheme [11], He and Wang's scheme [4], and Odelu et al.'s scheme [5]. Among them, Kou et al.'s scheme [2], Ying and Nayak's scheme [36], and Xiong et al.'s scheme [11] are the latest schemes since 2019.

7.1. Security Comparison. Table 7 compares the security performance of our scheme with Kou et al.'s scheme [2], Ying and Nayak's scheme [36], Xiong et al.'s scheme [11], He and Wang's scheme [4], and Odelu et al.'s scheme [5]. According to Table 7, only our scheme can meet all 9 security requirements.

7.2. Computation/Communication Overhead Comparison. To demonstrate the advantages of our scheme in terms of computation and communication overheads, our scheme is compared with Kou et al.'s scheme [2], Ying and Nayak's scheme [36], Xiong et al.'s scheme [11], He and Wang's scheme [4], and Odelu et al.'s scheme [5]. In all of these schemes, user and server registration is only required once, and the authentication and key agreement processes are the main processes in the practical application of the schemes. Thus, we only consider the overheads in the authentication and key agreement processes of the user and application server in our comparison.

We take an elliptic curve E over the finite field $\mathbb{F}_p$: $y^2 = x^3 + ax + b$, where $p(> 2^{160})$ is a large prime number, as the elliptic curve used in our scheme, Ying and Nayak's scheme [36], Xiong et al.'s scheme [11], He and Wang's scheme [4], and Odelu et al.'s scheme [5]. The point P is an element with large prime order $q(> 2^{160})$ on the additive group G on E. Thus, we get the size of the point in G to be 320 bits and the size of the number in $\mathbb{Z}_q^*$ to be 160 bits. The output length of the hash functions h_1, h_2 is 160 bits and 512 bits. We take $\hat{e}$: $G_1 \times G_1 \longrightarrow G_2$ as the bilinear map used in Kou's scheme [2].

7.2.1. Computation Overhead. In order to compare the computation overhead, we calculate the sum of the computation times of the cryptographic operations used in the authentication phase for each scheme. The execution time of cryptographic operations comes from He et al.'s compute [37]. The computing was run on a cryptographic library MIRACL, and it was based on a hardware platform consists of an Intel I7-4770 processor with 3.40 GHz clock frequency and 4 gigabytes memory and runs Windows 7 operating system. The notations

Table 8: Computation overhead comparison.

	Our scheme	Kou et al.'s scheme [2]	Ying and Nayak's scheme [36]	Xiong et al.'s scheme [11]	He and Wang's scheme [4]	Odelu et al.'s scheme [5]
Client	$3T_{sm-ecc} + T_{pa-ecc} + 4T_h \approx 1.3282$	$3T_{sm-bp} + 2T_{exp} + T_{pa-bp} + 6T_h \approx 5.2327$	$4T_{sm-ecc} + 2T_{pa-ecc} + 7T_h \approx 1.7723$	$3T_{sm-ecc} + T_{mul} + 5T_h \approx 1.3265$	$3T_{sm-ecc} + 7T_h \approx 1.3267$	$3T_{sm-ecc} + T_{(e/d)} + 7T_h \approx 1.3269$
Server	$3T_{sm-ecc} + T_{pa-ecc} + 6T_h \approx 1.3284$	$2T_{bq} + T_{sm-bp} + 4T_{exp} + T_{pa-bp} + 7T_h + T_{mul} \approx 10.3348$	$4T_{sm-ecc} + 2T_{pa-ecc} + 3T_h \approx 1.7719$	$4T_{sm-ecc} + T_{mul} + 5T_h \approx 1.7685$	$3T_{sm-ecc} + 5T_h \approx 1.3265$	$2T_{sm-ecc} + 2T_{(e/d)} + 6T_h \approx 0.885$
Total time (milliseconds)	≈2.6566	≈15.5675	≈3.5442	≈3.095	≈2.6532	≈2.2119

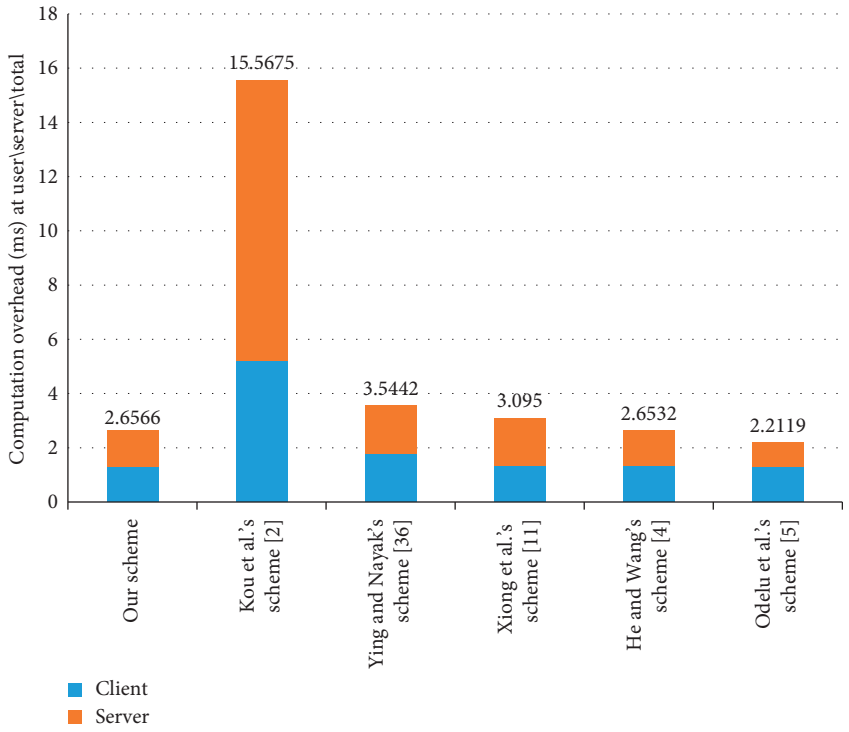

Figure 4: Computation overhead comparison.

about the execution time of the cryptographic operations are defined as follows:

(i) T_{bq} (=4.211 ms): the execution time of a bilinear pairing operation.

(ii) T_{sm-bp} (=1.709 ms): the execution time of a scale multiplication operation related to the bilinear pairing.

(iii) T_{pa-bp} (=0.0071 ms): the execution time of a point addition operation related to the bilinear pairing.

(iv) T_{mtp} (=4.406 ms): the execution time of a hash-to-point operation related to the bilinear pairing, where the hash function maps a string to a point of G_1.

(v) T_{exp} (=0.049 ms): the execution time of an exponentiation operation in G_2.

(vi) T_{sm-ecc} (=0.442 ms): the execution time of a scale multiplication operation related to the ECC.

(vii) T_{pa-ecc} (=0.0018 ms): the execution time of a point addition operation related to the ECC.

(viii) T_h (=0.0001 ms): the execution time of a general hash function.

(ix) T_{mul} (=0.00001 ms): the execution time of a multiplication operation in $\mathbb{Z}_q^*$.

(x) $T_{(e/d)}$ (=0.0002 ms): the execution time of a symmetric key encryption/decryption.

The authentication phase of our scheme has three scale multiplication operations, one point addition operation, and four general hash functions. And it has three scale multiplication operations, one point addition operation, and six general hash functions at the application server side. Table 8 and Figure 4 show the comparison of computation overhead between our scheme and other schemes. Compared with Xiong et al.'s scheme [11], our scheme reduces one scale multiplication operation at the application server side, and the computation overhead is reduced by 14.16%. And compared with Kou et al.'s scheme [2], our scheme does not use the bilinear pairing operation which has a large computation overhead, and the computation overhead is reduced by 82.93%.

7.2.2. Communication Overhead. In the mutual authentication phase of our scheme, the client and the application server communicate three times, sending $\{X, \text{SID}\}$, $\{Y, \text{AUTH}_{S_j}\}$, and $\{M_{U_i}\}$, and the application server also queries from the nearest blockchain node to get $\{ID_{U_i}, PK_{U_i}, s_i\}$. Messages $XYPK_{U_i} \in G$ and their length are both 320 bits. Messages M_{U_i}, AUTH_{S_j}, $s_i \in \mathscr{H}_1$, and their length are both 160 bits. Message $\text{SID} \in \mathscr{H}_2$, its length is 512 bits, and ID_{U_i} is a 32-bit data; thus, the communication overhead is $(320 + 512) + (320 + 160) + 160 + (32 + 320 + 160) = 1984$ bits. Table 9 and Figure 5 show the comparison of the communication overhead of our scheme with other schemes. In comparison, our scheme's communication overhead is close to the nearest scheme, Ying and Nayak's scheme [36] and Xiong's scheme [11].

Table 9: Communication overhead comparison.

	Our scheme	Kou et al.'s scheme [2]	Ying and Nayak's scheme [36]	Xiong et al.'s scheme [11]	He and Wang's scheme [4]	Odelu et al.'s scheme [5]
Rounds of message exchange	3	2	2	2	5	5
Number of bits required	1984	1568	1632	1568	3520	2944

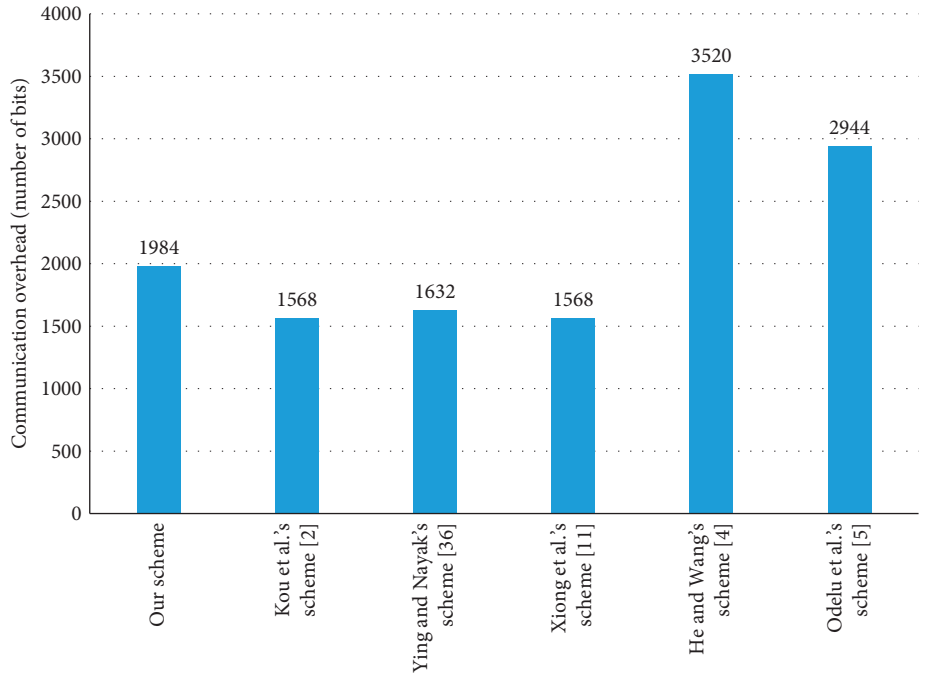

Figure 5: Communication overhead comparison.

8. Conclusions

With the wide application of blockchain technology, some blockchain-based multiserver authentication schemes have been proposed to deal with the centralization problem under the original multiserver architectures. However, most of them do not have effective user permission control and do not have satisfactory performance in terms of computation and communication overheads. In this paper, a blockchain-based hierarchical authentication scheme for multiserver architecture has been proposed. It is decentralized, and it has hierarchical access control functions. Our security proof and performance analysis show that our scheme is securer and more efficient than some existed multiserver authentication schemes such as Xiong et al.'s scheme [11] and Kou et al.'s scheme [2]. Our scheme is well suited for blockchain-based applications, such as a cross-bank blockchain digital currency system or a blockchain-based medical case system of multiple hospitals.

Data Availability

The data used to support this novel scheme are included within the article.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

Acknowledgments

This research was partially supported by the Key Program of the National Natural Science Foundation of China (no. 61772166) and the Natural Science Foundation of Zhejiang Province of China (no. LZ17F020002).

References

[1] L. Lamport, "Password authentication with insecure communication," *Communications of the ACM*, vol. 24, no. 11, pp. 770–772, 1981.

[2] J. Kou, M. He, L. Xiong, and Z. Lv, "Efficient hierarchical authentication protocol for multiserver architecture," *Security and Communication Networks*, vol. 2020, Article ID 2523834, 14 pages, 2020.

[3] T. Wan, Z. Liu, and J. Ma, "Authentication and key agreement protocol for multi-server architecture," *Journal of Computer Research and Development*, vol. 53, no. 11, pp. 2446–2453, 2016.

[4] D. He and D. Wang, "Robust biometrics-based authentication scheme for multiserver environment," *IEEE Systems Journal*, vol. 9, no. 3, pp. 816–823, 2014.

[5] V. Odelu, A. K. Das, and A. Goswami, "A secure biometrics-based multi-server authentication protocol using smart cards," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 9, pp. 1953–1966, 2015.

[6] V. Odelu, A. K. Das, S. Kumari, X. Huang, and M. Wazid, "Provably secure authenticated key agreement scheme for distributed mobile cloud computing services," *Future Generation Computer Systems*, vol. 68, pp. 74–88, 2017.

[7] D. He, S. Zeadally, B. Xu, and X. Huang, "An efficient identity-based conditional privacy-preserving authentication scheme for vehicular ad hoc networks," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 12, pp. 2681–2691, 2015.

[8] B. Liu, Y. Zhou, f. Hu, F. Li et al., "User authentication and key agreement protocol for mobile client-multi-server environment," *Journal of Cryptologic Research*, vol. 5, no. 2, pp. 111–125, 2018.

[9] Q. Shao, C. Jin, Z. Zhang et al., "Blockchain: architecture and research progress," *Chinese Journal of Computers*, vol. 41, no. 5, pp. 969–988, 2018.

[10] B. Bhushan, C. Sahoo, P. Sinha, and A. Khamparia, "Unification of blockchain and internet of things (BIoT): requirements, working model, challenges and future directions," *Wireless Networks*, vol. 27, no. 1, pp. 50–90, 2021.

[11] L. Xiong, F. Li, S. Zeng, T. Peng, and Z. Liu, "A blockchain-based privacy-awareness authentication scheme with efficient revocation for multi-server architectures," *IEEE Access*, vol. 7, pp. 125840–125853, 2019.

[12] Y. Ren, F. Zhu, K. Zhu, P. K. Sharma, and J. Wang, "Blockchain-based trust establishment mechanism in the internet of multimedia things," *Multimedia Tools and Applications*, pp. 1–24, 2020.

[13] L. H. Li, L. C. Lin, and M. S. Hwang, "A remote password authentication scheme for multiserver architecture using neural networks," *IEEE Transactions on Neural Networks*, vol. 12, no. 6, pp. 1498–1504, 2001.

[14] M.-C. Chuang and M. C. Chen, "An anonymous multi-server authenticated key agreement scheme based on trust computing using smart cards and biometrics," *Expert Systems with Applications*, vol. 41, no. 4, pp. 1411–1418, 2014.

[15] V. Odelu, A. K. Das, and A. Goswami, "Cryptanalysis on "robust biometrics-based authentication scheme for multi-server environment"" *IACR Cryptology*, vol. 2014, 2014.

[16] J.-L. Tsai and N.-W. Lo, "A privacy-aware authentication scheme for distributed mobile cloud computing services," *IEEE Systems Journal*, vol. 9, no. 3, pp. 805–815, 2015.

[17] Y. Hei, J. Liu, and G. Yewei, "A blockchain-based authentication scheme for identity information sharing," *Journal of Cryptologic Research*, vol. 7, no. 5, pp. 605–615, 2019.

[18] Z. Zhou, L. Li, S. Guo, and Z. Li, "Biometric and password two-factor cross domain authentication scheme based on blockchain technology," *Journal of Computer Applications*, vol. 38, no. 6, pp. 1620–1627, 2018.

[19] A. Mohsin, A. Zaidan, B. Zaidan et al., "Based blockchain-pso-aes techniques in finger vein biometrics: a novel verification secure framework for patient authentication," *Computer Standards & Interfaces*, vol. 66, Article ID 103343, 2019.

[20] S. Nakamoto, *A Peer-To-Peer Electronic Cash System*, Manubot, Canada, 2019.

[21] B. Bhushan, A. Khamparia, K. M. Sagayam, S. K. Sharma, M. A. Ahad, and N. C. Debnath, "Blockchain for smart cities: a review of architectures, integration trends and future research directions," *Sustainable Cities and Society*, vol. 61, Article ID 102360, 2020.

[22] S. Goyal, N. Sharma, I. Kaushik, B. Bhushan, and A. Kumar, "Blockchain as Lifesaver of IoT," *Security and Trust Issues in Internet of Things: Blockchain to the Rescue,*, pp. 209–238, 2020.

[23] S. Goyal, N. Sharma, B. Bhushan, A. Shankar, and M. Sagayam, "Iot enabled technology in secured healthcare: applications, challenges and future directions,," in *Cognitive Internet of Medical Things for Smart Healthcare*, pp. 25–48, Springer, Berlin, Germany, 2020.

[24] G. Madaan, B. Bhushan, and R. Kumar, "Blockchain-based cyberthreat mitigation systems for smart vehicles and industrial automation," in *Multimedia Technologies in the Internet of Things Environment*, pp. 13–32, Springer, Berlin, Germany, 2020.

[25] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy extractors: how to generate strong keys from biometrics and other noisy data,," in *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 523–540, Springer, Berlin, Germany, 2004.

[26] R. P. Durbin, P. Sinha, K. M. Sagayam, and J. Andrew, "Letter: acid secretion by gastric mucous membrane," *The American Journal of Physiology*, vol. 229, no. 6, p. 1726, Article ID 106897, 1975.

[27] E. Androulaki, A. Barger, V. Bortnikov et al., "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, pp. 1–15, Porto, Portugal, April 2018.

[28] The-Linux-Foundation, *Install, Build and Deploy Solutions Using Hyperledger Technologies*, The-Linux-Foundation, San Francisco, CA, USA, 2021, https://www.hyperledger.org/use.

[29] The-Linux-Foundation, *Hyperledger-fabric: A Blockchain Platform for the Enterprise*, The-Linux-Foundation, San Francisco, CA, USA, 2018.

[30] D. Wang and P. Wang, "Two birds with one stone: two-factor authentication with security beyond conventional bound," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 4, pp. 708–722, 2016.

[31] W. Wang, H. Huang, L. Xue, Q. Li, R. Malekian, and Y. Zhang, "Blockchain-assisted handover authentication for intelligent telehealth in multi-server edge computing environment," *Journal of Systems Architecture*, vol. 2021, Article ID 106897, 2021.

[32] D. Wang, D. He, P. Wang, and C.-H. Chu, "Anonymous two-factor authentication in distributed systems: certain goals are beyond attainment," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 4, pp. 428–442, 2014.

[33] D. Wang, H. Cheng, P. Wang, X. Huang, and G. Jian, "Zipf's law in passwords," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 11, pp. 2776–2791, 2017.

[34] D. Wang, X. Zhang, Z. Zhang, and P. Wang, "Understanding security failures of multi-factor authentication schemes for multi-server environments," *Computers & Security*, vol. 88, Article ID 101619, 2020.

[35] H. U. Rehman, A. Ghani, S. A. Chaudhry, M. H. Alsharif, and N. Nabipour, "A secure and improved multi server authentication protocol using fuzzy commitment," *Multimedia Tools and Applications*, pp. 1–25, 2020.

[36] B. Ying and A. Nayak, "Lightweight remote user authentication protocol for multi-server 5g networks using self-certified public key cryptography," *Journal of Network and Computer Applications*, vol. 131, pp. 66–74, 2019.

[37] D. He, S. Zeadally, N. Kumar, and W. Wu, "Efficient and anonymous mobile user authentication protocol using self-certified public key cryptography for multi-server architectures," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 9, pp. 2052–2064, 2016.