IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 28 February 2023, accepted 7 March 2023, date of publication 10 March 2023, date of current version 16 March 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3255885

RESEARCH ARTICLE

Achieving Decentralized and Dynamic SSO-Identity Access Management System for Multi-Application Outsourced in Cloud

SOMCHART FUGKEAW, (Member, IEEE)
Sirindhorn International Institute of Technology, Thammasat University, Khlong Nueng, Pathum Thani 12121, Thailand
e-mail: somchart@siit.tu.ac.th

This work was supported by the Sirindhorn International Institute of Technology (SIIT) Young Researcher Grant under Contract SIIT2019-YRG-SF02.

ABSTRACT Existing Single Sign-On (SSO) access control systems typically rely on the traditional protocols requiring additional authentication mechanism and/or identity providers. As the growing demand in outsourcing system resources such as data and applications to the cloud platform, implementing traditional SSO models to support efficient and fine-grained access control for multi-user and multi-application environment is not practical. In this paper, we propose a blockchain-based identification and access management (IAM) scheme called D^2-IAM to provide strong security measures for controlling SSO access to resources in the cloud. At a core of D^2-IAM, core access control processes are done by the smart contracts and blockchain where the access transactions are well retained for the accountability. In our system, the SSO authentication is based on the highest authentication level and the hashed-based token management. Owing to the autonomous authentication management, the communication overhead regarding the interaction with identity providers and third-party verification mechanism for multi-system authentication is minimized. For the authorization system, D^2-IAM enables fine-grained access through the access policy modeled in the document database written and enforced to each customer. Finally, we conducted the experiments on Google cloud to show that our D^2-IAM system is efficient for the implementation. The performance test showed that our proposed system was approximately 4 times efficient than the average processing time of three existing works.

INDEX TERMS SSO, authentication, access control, blockchain, access policy, document database.

I. INTRODUCTION

Access Control system is generally regarded as the most fundamental and crucial mechanism of any information systems. An effective access control typically refers to the secure provision of 3As including authentication, authorization, and auditing. In cloud computing, most service providers provide basic authentication such as user/password, one-time password (OTP) to their users. Such authentication mechanisms may not be sufficient for ensuring the secure access to critical or sensitive resources such as applications or database located in the cloud.

Most organizations deploying their applications or any services on the cloud need to implement their access control mechanism such as multi-factor authentication, PKI authentication [1] together with their authorization model on top of the services deployed. For those organizations having multiple application services on cloud may have different access control mechanisms to different groups of users. The cost for handling authentication options, access control policies are very expensive.

It is possible that some cloud providers may provide Single sign-on authentication system to support enterprises having multiple systems. SSO enables a user to use a single set of login credentials such as identity attributes, user/password, two-factor authentication (2FA) or

The associate editor coordinating the review of this manuscript and approving it for publication was Aneel Rahim.

multi-factor authentication (MFA) to gain access to multiple systems. It can be implemented by using security assertion markup language (SAML) standard [11], token-based [12], authentication standards [6], [30]. Recently, SSO applications leverage OpenID Connect [2] and FIDO [31] to enable more flexible and secure feature of the authentication process. OpenID Connect has been developed as the identity layer over OAuth 2.0 protocol which allows various types of clients such as web-based applications, mobile applications, and JavaScript clients, to verify the identity and acquire the profile of the end-user through the authorization server. Furthermore, The FIDO Alliance launched FIDO2 Authentication standard based on public key cryptography for authentication. It is realized as a new passwordless authentication standard. For the authentication, users can register and then select a FIDO2 security key upon the sign-in interface as their authentication method. This technique is a more secure authentication that provides secure and fast login experiences across websites and applications.

However, the SSO authentication service comes with additional service fee charged the resource owners can only use the default authentication method provided by the cloud provider. Also, using different authentication standards for SSO service might deal with the clients' system compatibility such as browsers, run-time services.

Regarding the access policy management, they need to migrate and implement separate authorization policies for limiting the access privilege to their users in accessing different resources. Here, the auditing is achieved through the application log files and access logs supported by the cloud provider.

Nevertheless, using the authentication service provided by the providers could render security and privacy issue. This is because the identity information may be leaked or compromised. In addition, dealing with multiple access policies configuration for multiple resources in cloud is not efficient in practice. The management cost is high in such environment. Furthermore, SSO service that relies on centralized authentication in the local host server or in the cloud [14] typically encounters the single point of failure and it induces substantial impact to the users and business.

Recently, blockchain technology has been adopted by several research works focusing on IAM [15], [16], [17], [18], [19] and data sharing [24], [25], [26], [27], [28], [29], [34], [35], [36]. Various industries tend to adopt blockchain for supporting their business transaction processing and sharing as it supports decentralized, traceable, and tamper-resistant data access control architecture. These properties are desired for serving scalability and dynamic control of multiple resources sharing with minimization of single point of failure of system entities deployed in cloud environment. In addition, the functions of IAM can be efficiently enforced by the execution of smart contracts.

Even though blockchain technology empowers the access control management with robust decentralized authentication, high availability of service and immutable access transaction traceability, there is no specific solution to integrate the decentralization model of blockchain technology and a cloud-based access control system having the properties of lightweight SSO authentication, multiple access authorization models with the privacy-preserving policy, and the enforcement of preventive-based access control. Furthermore, existing non-blockchain and blockchain-based access control systems for cloud computing share common gaps as follows.

(1) Existing SSO authentication schemes generally rely on the generic authentication methods such as SAML, OAuth which do not take the authentication options of multiple resources into their model. If the authentication service or any trusted entity who issues the identity or access ticket is compromised, the whole access control is collapse.
(2) Access policy is managed and enforced to users based on the centralized authorization service located on the cloud server. This renders the risk of single point of failure in centralized cloud-based architecture. In addition, the access policies stored in the authorization service are not encrypted.

Therefore, the need for full-fledged access control that calibrates the advantages of blockchain and the stated strong access control properties is a real challenge for the cloud-based access control research and applications.

In this paper, we proposed a secure and dynamic access control system called D^2-IAM which is primarily used in the cloud environment where multiple resources such as applications and outsourced data service are available. Our proposed system provides strong access control mechanisms for multiple resources. We used blockchain and smart contracts to provide autonomy and decentralization. All core access control functions are run in the decentralized setting. This work furthermore proposes a preventive access control mechanism that uses logs and defines a set of actions and rules to keep the system safe.

The technical ingredients of our proposed scheme are based on our proposed a lightweight SSO token, cryptographic protocols, and the blockchain technology.

The contributions of our proposed approach are described as follows.

1.We proposed a blockchain-based IAM system to enable secure and efficient SSO-authentication, authorization, and preventive-based access control. All of the core access control functions are handled by a set of smart contracts. This enables more autonomous and traceable access control activities.

2. We devised a lightweight SSO-authentication token based on highest authentication level of the resources requested to access. The authentication is thus bound to the privilege of resource access instead of relying on additional authentication mechanism. Our proposed SSO-authentication token significantly reduces the communication overhead for multi-system authentication.

3. For the authorization system module, the access control policies are modeled in the JSON file or the document database and they are all encrypted and stored in the cloud storage. Compared to traditional access policy models such as role-based access control (RBAC), attribute-based access control (ABAC), using JSON file supports fine-grained with more expressive authorization as each document policy is assigned to each user. In addition, instead of specifying access rules in a shared repository, the policy specified in the JSON file gives faster policy validation and enforcement for individual user because the system does not to deal with multiple rules stored in the global policy. The responsible smart contract verifies the encrypted policies and enforces the authorization control in an efficient and secure manner.

4. Our system supports the preventive access control that is considered essential for a large-scale access control environment. Our system provides a set of rules and actions that are able to prevent unwanted security incidents.

5. We developed a prototype system and conducted the experiments to validate the efficiency of our D^2-IAM.

The rest of the paper is structured as follows. Section II discusses related works. Section III presents our D^2-IAM system model. Section IV presents the security analysis. Section V describes the implementation and evaluation. Finally, the conclusion and future work are given in section VI.

II. RELATED WORK

In this section, we reviewed the cloud-based access control solutions in both non-blockchain access control and blockchain-based access control.

A. NON-BLOCKCHAIN ACCESS CONTROL SCHEMES

Most research works related to a cloud-based or distributed IAM focuses on the specific model such as role based [9], policy-based access control, IAM standards [3], [5], [10], and SSO Service.

The authentication service such as OpenID connect [2] and FIDO2 [31] have been employed as the SSO feature in many applications. OpenID connect is built on OAuth 2.0 protocol [30] that allows users to use SSO to access across applications using OpenID Providers (OPs) to authenticate their identities. However, it lacks user authorization data such as the privilege or permission data. FIDO removes replace the password authentication with the PKI authentication method based on the key generated upon sign-in or the key stored in the device. This enables both secure and fast login to several applications. To deal with this authentication method, users need to undergo an additional security step and need to aware of the security of the key stored in their devices.

In [3], Moghaddam et al. proposed a policy-based authentication model to control user authentication through the ontology schema. The authorization is done via access policies stored in the policy database. However, the paper does not deal with multiple resources deployed in the cloud system.

In [7], Tang et al. introduced the general framework of policy-based access and behavior control management as well as the policy-driven network access and behavior control management model. In this model, the network access and behavior management control process are implemented through abstract policy configuration, network device and application server.

In [8], Zhou and Zhu focused on authentication and SSO service design based on centralized authentication service (CAS) protocol to realize SSO function which includes a small part of a custom encryption method. Their model enables multiple systems to share a secure single-login.

In [39], Erdem and Sandikkaya proposed a cloud-based OTP authentication architecture that is resistant to replay attack between an OTP user, a service provider, and a cloud OTP provider. The focus of the paper leverages the multiple pre-shared keys in addition to the OTP and it allows the user to have multiple profiles for different purposes in the cloud OTP provider.

In [40], Yang et al. proposed a cloud-based cryptographic-based authentication scheme based on elliptic curve cryptography (ECC) without pairing operations. Hence, the secret key is a primary credential for the authentication and gaining access to the data outsourced in the cloud. However, the proposed scheme only deals with the authentication to access outsourced data.

In [41], Vinoth et al. proposed a pre-authentication and pre-authorization models for cloud systems. In this scheme, the cloud server generates the pseudo-identity used to anonymously authenticate the legitimate user requesting to access the cloud. If the authentication is successful, the cloud server and the user share the session key for the entire access session. However, the proposed scheme did not support access control to multiple resources.

B. BLOCKCHAIN-BASED ACCESS CONTROL SCHEMES

Recently, many research works [5], [14], [15], [16], [17], [18], [19] have employed blockchain technology as a platform to support decentralized identity management as well as the access control for data sharing in cloud computing. For example, Wang et al. [5] proposed a cloud user identity management protocol based on Ethereum blockchain. The identity management is controlled by using smart contracts. This enables decentralized access control and avoid the intervention by the cloud provider. It also enables users and cloud providers to jointly manage the system through smart contracts. However, the proposed system does not deal with the authorization management in the cloud setting.

In [14], Chung et al. proposed a trust-based access control model for cloud computing. The model focuses on adding trust value in RBAC model. It is capable to perform dynamic update of authorization status in the model. After the user accesses the system and completes the operation, cloud service provider will give trust feedback value for each operation upon the access activity. Hence, the authorization

is dynamic. However, this approach only considers the authorization part. The core idea of the proposed scheme is to design a cryptographic protocol for encrypting the data stored in cloud while major access control protocols are handled by the smart contracts of the Ethereum blockchain.

For example, Wang et al. [18] proposed a blockchain-based access control for cloud storage system by using Ethereum blockchain and ciphertext policy attribute-based encryption (CP-ABE).

In [19], Yang et al. proposed a blockchain-based access control framework called AuthPrivacyChain by focusing on the privacy of shared resource while user authentication and authorization function are done by the smart contracts. In this scheme, the authentication source and authorization policy are encrypted and stored in the blockchain. Even though the proposed scheme fully supports the privacy of both data and access policy, the cost for validating encrypted credentials and the policies in the block bring the performance issue.

Recently, Ping and Sato et al. [20] proposed a decentralized framework called Sunspot for privacy-preserving data sharing access control using blockchain. Sunspot accommodates two access control mechanisms including a fine-grained mechanism and a payment-based mechanism. The access control is based on the combination of CP-ABE and their proposed multi-blockchain protocols. The proposed scheme enables flexible data sharing in various scenarios.

In [32], Al-Zubaidie et al. proposed an authorization scheme based on the pseudonymization and the anonymization with the XACML model and ECDSA signatures to support the security and privacy of access policy that contain the personal data of healthcare users. However, the proposed scheme did not provide the SSO authentication.

In [33], Park et al. proposed an access control model by focusing on the activity control (ACON) to support the user activity-based access control enforcement in the collaborative computing systems. The proposed model is capable to identify and understand access control models through the analysis of a set of design principles for activity control including abstraction, controllability, containment, automation, accountability and searchability followed by their applicability in a smart health use case.

Nevertheless, all of the above research works have not entailed the dynamic SSO authentication and authorization of IAM model in multi-cloud resources environment. Existing SSO authentication schemes for both non-blockchain and blockchain-based access control solutions encounter the common problems related to the limitation on high communication cost and high dependency on the identity or authentication service provider. Furthermore, there are no existing SSO schemes that support the preventive-based access control enforcement.

III. OUR PROPOSED D^{2}IAM SYSTEM

This section describes the system model and the system process of our proposed scheme. We also present our proposed preventive-based access control technique.

A. SYSTEM MODEL

This section presents the system model of D^2-IAM and provides the details of its system components. Figure 1 represents the D^2-IAM system model that we have designed.

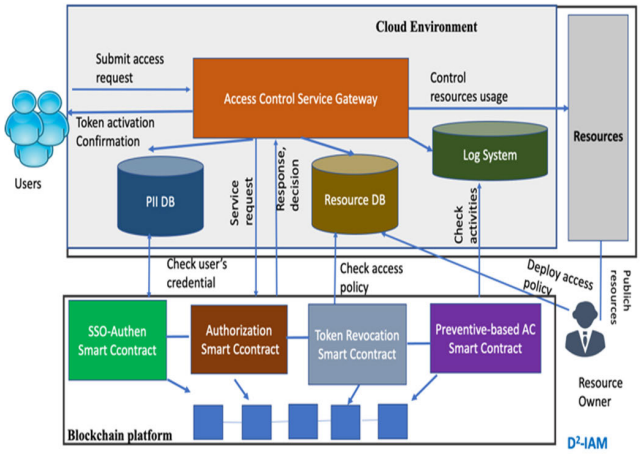

FIGURE 1. D^2-IAM system model.

D^2-IAM System Model consists of the following entities and functional systems.

- **Resource owner** publishes or provides resources such as application services on a public cloud and allows authorized users to access.
- **Users** are allowed to access multiple application services upon the access policies regulated by the resource owner.
- **Access Control Service Gateway (ACSG)** is the core system service located in the cloud. It is responsible for accepting access requests from the users and liaising with the smart contracts for performing SSO-authentication, token revocation, enforcing authorization, and invoking preventive access control. The gateway also controls the users' access session while they use the applications. All related-access decisions are enforced through the ACSG.
- **Smart contracts** (SCs) are a set of executable programs running on the blockchain to execute the tasks based on logical function. Blockchain assures that all smart contracts are immutable and transactions executed are autonomously and truthfully executed with full traceability. All initiated transactions are timestamped and distributed across many nodes in the network. This prevents spoofing or modification attack of the content or the rule of the contract. Furthermore, smart contracts do not require centralized authority to verify their authenticity. Rather, they allow participants to do particular transactions in a faster manner. To cope with other security attacks, there are works that proposed the techniques to provide the defensive solutions such as static analysis [37], runtime detection [38], and language-based security [42]. In our system, we control the access to smart contracts based on the hash value

of system identity code and address such as IP or MAC address of the system modules stored in the smart contract. If the caller's information fails to match the ones stored in the contract storage, the request for executing the contract is dropped.

There are several blockchain platforms that support smart contracts such as Ethereum, Hyperledger Fabric, Corda, Stellar and Rootstock.

In our system, there are four types of smart contracts designed to serve the core access control of D^2-IAM.

- Single Sign-On Authentication smart contract (SSO-Auth SC). This SC authenticates the users who request to access resources such as application services in the cloud. Then, it generates SSO token to enable the user can access multiple resources without several logons. This system module connects to the PII database and resource database.
- Authorization smart contract (AZSC). This SC is responsible for validating and enforcing user privilege based on the access policy expressed in the document database.
- Token revocation smart contract terminates the SSO token based on the expiry of the confirmation of the activation of a newly generated SSO token or the request from system administrator. In both cases, the AZSC checks the conditions and activates the smart contract to revoke the token.
- Preventive-based access control smart contract (PACSC) contains a set of preventive access rules used to support a more advanced access control feature. It monitors user access activities and provides some actions to control the access session. For example, if there are some events that violates the rule, the system will disable the access session and send the notification to the system administrator. This module also connects and makes use the log data in the log files.

- **Blockchain** stores transaction records using blocks of special data structures and prevents tampering with historical data. In our system, it is also used to support data auditing and data integrity verification.
- **Resource database** stores a collection of access policies. All policies are expressed in document database format and they are all encrypted with the authorization smart contract's public key before they are sent to the cloud. It serves for user's credentials and resource mapping during the SSO service confirmation and authorization validation done by the authentication smart contract and authorization smart contract. The example of data stored in the resource database is kept as a tuple of < Hash value of Uid, Hash value of identity1, Hash value of identity2, doc no.>. The attribute doc no refers to the access policy that belongs to each user id.
- **Personally identifiable identity (PII) database** keeps a set of hash value of users 'credentials. Hence, the privacy of PII instances of user is preserved. The user generally submits the ID via SSL channel and then the system hashes the received IDs and then it will be sent as a part of request to the PII database for checking the identity of user. The PII agent verifies the request and compares the received hash value with the value stored in PII record of the requested user. Then, only successful or failed verification status is returned to the ACSG.
- **Log system** stores all system and user activities that occur in the system. It is also used to support a preventive-based access control for checking the log data against the rules.

B. SYSTEM PROCESS

Our model consists of four major phases including System Initialization, SSO Authentication, Authorization, and Token Revocation.

Table 1 lists the notations used in our scheme.

TABLE 1. Notations used in our scheme.

Notation	Meaning
$PubK_{AZSC}$	A public key issued to authorization smart contract
$PrivK_{AZSC}$	A private key issued to authorization smart contract
SSO_tk	An sso token issued upon the successful authentication
md	Message digest computed from taking hash function over the data.
$ACP_{doc\_no.}$	An access control policy for document database doc no.
$Enc\_PrivK_{AZSC.}$	An encrypted AZSC's private key
$SymKey$	A symmetric key created from the AES algorithm.
$ACP^{ENC\_doc\_no}$	An encrypted access control policy of document no.
R	Random number generated by Pseudo random number generator
Sig_{SCid}	Digital signature value of the smart contract $SCid$

Phase 1: System Initialization

This phase consists of the following four algorithms run by the resource owner.

1. **EncACP**$(PubK_{AZSC}, ACP_{doc\_no}) \rightarrow ACP^{ENC\_doc\_no}$. This algorithm is used to encrypt the access control policies for the document database doc no. It takes as inputs authorization smart contract's public key $PubK_{AZSC}$, and an access control policy ACP_{doc_no}. Then it returns an encrypted $ACP^{ENC\_doc\_no}$.
2. **KeyPairGen**$(RSAKeyGen, param) \rightarrow (PubK_{SCid}, PrivK_{SCid})$. The wallet in the blockchain generates RSA KeyPair for the smart contract. It takes two input parameters, RSAkey_algorithm and system parameters $param$, to generate a 2048-bit public key $PubK_{AZSC}$ and private key $PrivK_{AZSC}$. The private key will be then encrypted and then it is stored with its corresponding

public key in the blockchain for further use by the smart contact.

3. **EncKey**(R, $PrivK_{AZSC}$) $\rightarrow$ Enc_PrivK_{AZSC}.This algorithm generates pseudo random number generator [21] to encrypt the private key of the AZSC that will be stored in the blockchain. Hence, the confidentiality of AZSC's private key is preserved.
4. **SigGen_RSA**($PrivK_{SCid}$,md) $\rightarrow$ Sig_{SCid}: the function computes a digital signature value Sig_{SCid}, of message digest md using the private key $PrivK_{SCid}$ which belongs to the smart contract $SCid$.
5. **Verify_RSA**($PubK_{SCid}$, Sig_{SCid}) $\rightarrow$ md: the function verifies whether the value Sig_{SCid} is the correct signature value of md by using the public key $PubK_{SCid}$to decrypt the signature.

Phase 2: SSO Authentication

In our system, there are three steps for completing the SSO-authentication. In this process, SSO-authentication smart contract is responsible to execute three functions including user ID verification, authentication, and SSO token generation. Technical contributions offer lightweight ID verification and authentication based on selective verification of user credentials and highest authentication. Then, the SSO-authentication is generated based on the user ID, secure authentication option, and user access session. Figure 2 illustrates the SSO-authentication procedure performed by the SSO-Auth smart contract.

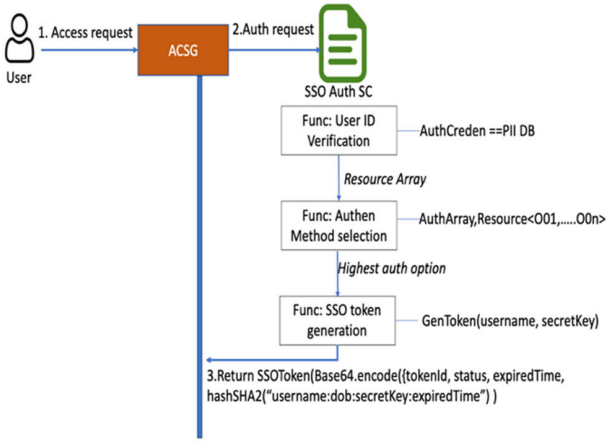

FIGURE 2. SSO-authentication procedure.

As shown in Fig.2, the system verifies the identity of the user. Here, the user needs to present a few identities or credential information to the ACSG where all transactions are secure through the SSL communication. Then, the submitted identities are combined and hashed. The derived message digest will be compared with the corresponding message digest stored in the PII database. If they are matched, the smart contract will check the resources of which the user can access and prompt to the authentication step. The function of ID verification process performed by the SSO-authentication smart contract is shown below.

User ID Verification Function

```
Function verifyUser(authCreden) {
  matchOrNot = find if authCreden matches in the PII database
 if (matchOrNot == true){
 Fetch all resources for user to choose
 Logging.info("identified")
 } else {
 Alert(errorMessage)
 Logging.error("error")
 }
}
```

For the step 2, the smart contract checks the authentication method based on the highest authentication level of all resources the user requests to access. Here, each resource profile detailing the list of resources and corresponding authentication option is checked. With this method, the authentication module checks all authentication method for all resources the requestor has. For example, User A makes a request for using SSO to access the web application X which requires user password for accessing the system. After checking User A's capability, there are additional web applications namely Y, and Z, that the user has the access right. The authentication methods for Y and Z are user/password and OTP respectively. The system will enforce the OTP option for the SSO-authentication. Then, the user is prompted with the authentication option provided by the system. In the final step, if the user is successfully authenticated, the smart contract will generate SSO token to allow the user to access all system resources without multiple logins. The token has the expiration time. If the token expires, the user has to recall the SSO Service Authentication again.

To generate a token, the *SSO-authen SC* fetches the hash value of the username from the database. Then, it assigns the token to be get along with the hash value of the user id. The token will then be stored inside the database. Whenever the user wants to use the SSO service, the integrity of the token will be verified by using the user id. The token value will be compared to the username value of the user. The function used to check the authentication option is shown below.

Algorithm 2 Authentication method selection

```
Function getAuthenMethod(setOf ChosenResources, SSO){
 Array AuthArray
 If (not SSO){
        For(each r in setOf ChosenResources {
           Auth = authentication method binds to O
           AuthArray.push(Auth)
        }
 }
 return highest auth level in AuthArray to the user
 } else{
           Call SSOService(SSO.authCreden)
 }
}
```

In essence, we apply highest authentication method to the user who requests for accessing multiple resources and the SSO token will be then issued based on the successful authentication step. This enables secure and minimized communication cost for one-time granting of SSO token. Since the token is constructed by the identity of the user with the randomized secret key, it serves for both the uniqueness and security of the user access and system integrity. For its usability, the user requests for an SSO Service, the smart contract will verify if there is any SSO token bound to the requested user's access, and it is still valid. If there is no token granted, the system will prompt the user to perform the highest authentication method. After the user is successfully authenticated, the smart contract will generate the token based on the user's identity. Then the hash value is generated and the token is produced. Each token is assigned with the unique ID, session ID, and expired time. Essentially, ACSG retains these parameters for runtime verification of the token requested by the applications. Therefore, our system assures that only valid token used by the legitimate user is allowed to enable access to resources. The communication between system modules and clients is secured by SSL protocol. The function of SSO authentication and token generation is presented as follows.

Algorithm 3 SSO Authentication Service

```
Function authenticateUser(authCreden){
 Activate = false
 if (verifyUser(authCreden) == valid){
 if (SSOService == true){
 SSOToken = generateToken(authCreden.username,
authCreden.secretkey)
 Call ACSG module for sending activation request to user
 tokenId = Base64.decode(SSOToken).tokenId
Loop (30 secs for token activating confirm from user) {
 if user.request.activate == true {
 Call activateToken(tokenId)
       Activate = true
 }
}
 if (not activate) {
  Call revokeToken(Base64.decode(SSOToken).tokenId)
 }
} else {
 alert(errorMessage)
 Logging.error(''error'')
 }
}
Function generateToken(username, secretKey){
 expiredTime = dateTimeNow + 30 minutes
 status = Deactive
 // Hash value for token validation with sessionId checking
validationHash = hashSHA2(''username:dob:secretKey:
expiredTime:Request.sessionId'')
 SSOToken = Base64.encode({tokenId, status, expiredTime,
validationHash})
 Save SSOToken to DB
 return SSOToken
}
Function activateToken(tokenId) {
 SSOToken = Get user token from DB where tokenId =
tokenId
 SSOTokenDecoded = Base64.decode(SSOToken)
 SSOTokenDecoded.status = Active
 SSOToken = Base64.encode(SSOTokenDecoded)
 Save SSOToken to DB
}
```

Phase 3: Authorization

In the authorization module, we define the enforcement process based on the authorization policy. In our system, all resources are bound with the corresponding access control policy database. All policies are expressed in the document database and all documents are encrypted by the public key of authorization smart contract. The hash values of ''user id'' and ''document number'' are stored in the resource database on cloud.

Definition 1: Access Control Policy is a collection of documents where each document contains access rule for a particular user (*Uid*), resources or objects (*O*), and privilege (*P*) associated to the object.

The example of access policy for each user expressed in the document database is shown in Fig.3.

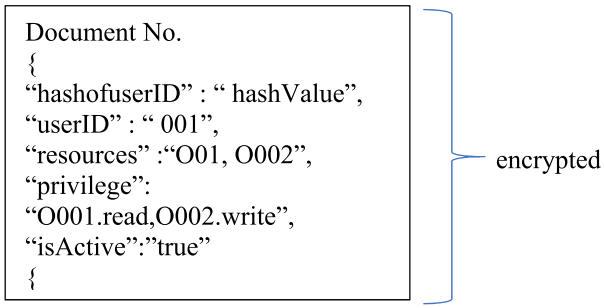

FIGURE 3. Access policy database.

The information about the Document No. representing the access policy for each user is stored in the encrypted JSON file. Since the access policy is stored in the resource database located in the cloud and it contains the details of user capability that depicts the association of the user, resource, and privilege, the privacy-preserving of the policy content is thus crucial.

In our system, the authorization enforcement is done through two three-protocols: realization, verification, and preventive-based access control.

Realization. In this protocol, SSO-authen SC sends the confirmation of token *tk* to ACSG. ACSG then runs the following steps.

1. ACSG acknowledges the issuance of SSO token SSO_tk for Uid with valid time *time start ts*, and *time end* t_e [t_s, t_e]. If the validation result is $\perp$, it terminates the protocol. Otherwise, it proceeds the protocol;

2. ACSG emits an authorization request to AZSC with *value.hashofUserID*.

Verification. In this protocol, AZSC executes the below procedures to verify the access rule of the user *Uid*.

1. Gets Uid from valid SSO_tk.
2. Checks resource db directing to the Uid associated to document policy.
3. Gets a doc no. from a tuple < Hash value of Uid, Hash value of identity1, Hash value of identity2, doc no.>
4. Decrypts the encrypted policy by using its private key. The decryption function is done as follows.

$$\mathrm{DEC}_{\mathrm{RSA}}(R, Enc\_PrivK_{AZSC}., ACP^{ENC\_doc\_no.}) \equiv ACP_{doc\_no.}$$

5. Emit privilege *p* of *Uid* and fetch the authorization state to the ACSG.

In essence, the technical contributions executed by the AZSC are the invocation of document based on the retrieval of the hash value of the User ID and set of identities associated to the JSON document and the verification of access policy through the policy decryption. The decryption function performed by the *AZSC* is technically built in the configuration function; no parts of the crypto elements are revealed. Then, the privilege of the application requested by the user is enforced during the access session.

In our authorization enforcement, resource owners can import their local policy to the access policy database located in the cloud. Also, the resource owners can also dynamically update their policies via cloud anytime and anywhere. To support the integrity of all policy databases, we exploited the integrity agent to calculate hash value of each database. If there are any changes on any policies, the system administrator will be notified. The authorization request and response transactions sent and received between entities in cloud and blockchain are done via secure web service protocol.

Phase 4: Token Revocation

In our system, the token generated can be revoked based on either the expiry of the confirmation period of the token activation by the user or the request initiated by the system administrator. The revocation is done through the activation of the token revocation smart contract. The code below shows how the smart contract revokes the token and how the service verifies the token.

Algorithm 4 SSO Token Revocation

```
Function revokeToken(tokenId) {
 SSOToken = Get user token from DB where tokenId =
tokenId
 SSOTokenDecoded = Base64.decode(SSOToken)
 SSOTokenDecoded.status = revoked
 SSOToken = Base64.encode(SSOTokenDecoded)
 Save SSOToken to DB
}
Function verifyToken(inputSSOToken){
 inputSSOTokenDecoded =
Base64.decode(inputSSOToken)
 if (SSOToken.status == Active and dateTimeNow <
SSOToken.expiredDate) {
 SSOToken = Get user token from DB where tokenId =
inputSSOTokenDecoded.tokenId
 SSOTokenDecoded = Base64.decode(SSOToken)
 MD = hashSHA2(''username:dob:secretKey:
inputSSOTokenDecoded.expiredTime'')
 if (MD == SSOTokenDecoded.hash){
      return true
 } else {
      return false
 }
} else {
      return false
 }
}
```

For the revocation process, the smart contract checks the user whom the token has been granted. Then, the smart contract verifies the active SSO token and its validity period before it immediately revokes the token. Therefore, token invalidation takes place immediately and the token cannot be used again after the revocation.

C. PREVENTIVE-BASED ACCESS CONTROL

In this system function, we formally define a set of rules to prevent unwanted access activities from the users. The *PACSC* collaboratively works with the log file and it will trigger when the any pre-defined activities are detected.

Definition 2: Preventive Access Control Policy (PACP) is a set of pre-defined rules consisting of events or activities that are considered illegal or harmful to the system resources and corresponding preventive actions to stop those activities.

The policy is expressed as

if<$condition_1$,$condition_2$,.....,$condition_n$>,

then <$action_1$,$action_2$, $action_n$>

In our system, we classify the transactional cases as our preventive controls that are entailed by the PACP. Here are examples of our PACP used to enforce the activities occurring in the system.

Case 1: Login transaction

Rule1: If the user provides false credentials >3 times, then this user and login ip address are blocked.

Rule 2: If the authentication is unsuccessful > 3 times, then this user and login ip address are blocked.

Rule 3: If the user colludes attributes with other users to gain access to Resource $O1$, then the user account is suspended and the system sends an alert to the admin.

Rule 4: If the same user logs in using different IPs as a concurrent session, then all access sessions are disabled.

Case 2: In-Service transaction

Rule 5: If the user tries to elevate the privilege of certain application services > 2 times, then disable access grant to the service.

Rule 6: If the user uses the services > 8 hrs, then the system sends an alert to the admin.

Case 3: Temporal-aware transaction

Rule 7: If the user makes a request to access Resource *O2* and *O3*, then the services are only available during day time (8.00 am -5.00 pm)

Case 4: Location-based transaction

Rule 8: If the user makes a request to access Resource R4 and the login IP is 192.168.xx.xx, then the service is enabled.

The major contributions of the PACSC are the enforcement of dynamic access decision triggered by the access events. All policy rules are signed by SigGen function using the private key of PACSC. It can be updated flexibly and dynamically by the resource owners and these policy sets are executed by the system agent of the PACSC that monitors access transactions.

IV. SECURITY ANALYSIS

This section describes the security threat and security property of our proposed scheme.

A. SECURITY THREAT OF SSO TOKEN-BASED APPROACH

In SSO token-based model, the attacker may try to either compromise the elements of SSO infrastructure or compromise the global administrator account. For the first method, once the system is compromised, the attacker steals the credentials of any valid users and requests for the token to gain access on the behalf of the user. For the latter method, once the account is compromised, the attacker can create fake credentials which will be used to register in the system.

In our system, the scope of possible attack thus mainly deals with the exploitation of fake or compromised user credentials to gain access to the system. In our system, obtaining creating valid credentials alone or exploiting fake credentials in the system cannot reach out the generation of the token. There is the authentication process requiring the correct authentication factor(s) that must be supplied to the system through verification of the authentication smart contract. In addition, compromising system elements in blockchain such as smart contracts or authentication transactions is computationally infeasible. Furthermore, we assume that the activation of the SSO token is done by the token confirmation at the user's mobile phone. Therefore, our proposed model is resistant to the mentioned SSO token attacks.

B. SECURITY PROPERTY

1) CONFIDENTIALITY OF USER CREDENTIALS

In our scheme, all identity records of the users are stored in the cloud and blockchain in the privacy-preserving manner. Only selected identity information is hashed for the authentication purpose. Hence, the content of user credentials is stored in unreadable format.

2) CONFIDENTIALITY OF ACCESS POLICIES STORED IN THE CLOUD

In our scheme, access policy is written in the JSON file and each document represents the access policy for each user. All documents are encrypted by the public key encryption. The encryption key is also encrypted by the random encryption and securely stored in the configuration file for fast execution. Only the legitimate AZSC smart contract can perform the decryption and verify the policy.

3) RESISTANCE TO REPLAY ATTACK AND MAN-IN-THE-MIDDLE ATTACK

To prevent the token replay attack, the access control system gateway (ACSG) keeps a list of user ID and session ID of generated token, for the lifetime of the token. Hence, the application can check whether the token is legitimate or not based on the verification confirmation from the ACSG. We also use SSL secured connection between system modules and clients communication. Hence, all mentioned system parameters are encrypted. This helps to prevent Man-in-the-Middle attacks. Also, the user interacts with the system by presenting identity information to the ACSG where all transactions are secure through the SSL communication. The system will get back to the user with access decision. Therefore, the request and response message are secure through the encrypted channel.

4) SECURE TRANSACTIONS BETWEEN CLOUD AND BLOCKCHAIN COMMUNICATION

In our scheme, all communication processes that occur between the cloud server, smart contracts and the blockchain are done via secure web service. This prevents the eavesdropping and session hijacking.

5) ACCOUNTABILITY AND AVAILABILITY OF ALL SERVICE REQUEST TRANSACTIONS

Our system retains activity records of all entities in blockchain. The SSO authentication status and authorization enforcement, and possible preventive-based access control are recorded in the tamper-proof manner in the blockchains. This supports data and service integrity for the authentication function. In addition, blockchain platform offers high degree of service and transaction availability as it is implemented in the decentralized setting and the committed transactional data are available in multiple nodes of the blockchain. Crucially, the above security features play a key role enabling D^2IAM to be practical in the real implementation in the public cloud environment. This is because the strong access control is guaranteed based on the lightweight, secure, and privacy-preserving authentication, flexible authorization, and immutable access transaction logs.

V. EVALUATION AND EXPERIMENT

This section provides the evaluation of our scheme by giving the details of functional analysis, computation cost analysis, and implementation of our D^2-IAM.

A. FUNCTIONALITY ANALYSIS

We compare the functionality of our scheme with three works including [18], [19], and [20] which are all blockchain-based access control schemes.

As presented in Table 2, only our scheme provides SSO authentication to use multiple resources in cloud while other works only focus on applying a blockchain technology for data access control in cloud computing. As for the

TABLE 2. Functionality comparison.

Scheme	SSO	Block chain and Cloud-based	Permission checking	Policy Enc.	Preventive policy
[18]	No	Yes	CP-ABE policy	No	No
[19]	No	Yes	Policy in blockchain	Yes	No
[20]	No	Yes	CP-ABE	No	No
Our scheme	Yes	Yes	Document db policy	Yes	Yes

TABLE 3. Comparison of computation cost of ID or authentication verification cost.

Scheme	ID/Authentication Verification Cost	Authorization Cost
[18]	C_{sym}	$\|N\|C_e + (1+2\|S\|)C_p + \|N\|C_p$
[19]	C_{pubk}	$C_{pubk} + C_{Hash}$
[20]	$R + C_{Hash}$	$R + \|N\|C_e + (1+2\|S\|)C_p + \|N\|C_p$
Ours	C_{Hash}	C_{pubk}

SSO function, the system checks the authentication option for each resource the user requests to access. The highest authentication method is then applied. Regarding the privacy access policy used to enforced the permission, scheme [18] and [20] uses CP-ABE access policy access constructing from the set of user attributes and logical operators. In [19], the access policy is stored in the blockchain in the encrypted format. In our scheme, the access policy is expressed in the document database for each user and it is encrypted using public key encryption. Regarding the privacy-preserving access policy, only scheme [19] and ours provide the policy encryption to support the confidentiality of the policy stored in the blockchain and cloud. Finally, only our scheme has a feature of preventive access control in the cloud environment.

B. COMPUTATION COST ANALYSIS

This section presents the computation cost of our proposed scheme with three blockchain-based authentication schemes including [18], [19], [20]. Table 3 presents the computation cost of user identify verification, encryption, and decryption of all schemes. For ease of analysis, we use the following notations to measure the computation cost.

- C_{Hash} is the cost of 256-bit hash computation
- C_{sym} is the cost of encryption or decryption with a symmetric key.
- C_{pubk} is the cost of encryption or decryption with a public key encryption.
- R is 256-bit random number
- C_e: Exponentiation and XOR operation cost
- C_p: Pairing operation cost
- $|S|$: The size of user attribute set
- $|N|$: The number of nodes in access control policy.

As shown in Table 3, the identity verification cost of scheme [18] is based on the Diffie-Hellman key exchange method using the AES symmetric encryption. In [19], the user identification is done through the process of public key decryption method. In [20], the identity verification cost is subject to the computation of 256-bit random number and the create of wallet address using hash value. Our scheme only requires the SHA-256 hashing cost of the ID data and it is compared with the existing one in the database. If it is matched, the user is prompted with the highest authentication method of the resource requested to access. Therefore, our scheme yields constant costs of user id verification and require no key for the authentication stage. Regarding the authorization cost, Wang et al.'scheme [18] applies CP-ABE that deals with pairing and exponentiation operation and the cost is subject to the number of attributes set contained in the key and access policy.

In [19], the authorization is done through the public key decryption of the authorization request initiated from the user to cloud system and the cloud needs to calculate hash value CHash for confirming the authorization from the corresponding blockchain. In [20], the major authorization costs are done through the verification of secret key using the random number for the verification and CP-ABE decryption. Our scheme only requires checking the authorization policy in the encrypted document policy by using the public key decryption. Obviously, the cost of authorization of our scheme is also less than other works. Therefore, our scheme offers efficient authentication and authorization with the support of SSO capability.

C. PERFORMANCE EVALUATION

In this paper, we aim at evaluating the practicality and efficiency of the entire process of the authenticationand the authorization of our proposed system. This signifies the capability of the authentication and authorization protocols of our system. For the token revocation and preventive-based access control enforcement, their cost is much smaller than the SSO authentication process as it only deals with the user and token verification and small number of predefined rules.

To evaluate the efficiency of our scheme, we conducted the experiments to measure the total access control processing time including identity verification and authorization. Furthermore, we measure the system scalability by assessing the throughput of our D^2-IAM in accommodating the concurrent requests. For assessing the processing time and system throughput, we did the simulation to compare our scheme and three schemes including [18], [19], and [20]. We conducted the test on the container run on the Google cloud CPU1, RAM 128MB. The programming language is java and solidity. The standard AES and RSA encryption and signing provided by Ethereum blockchain [23] are used for [18] and [19], and our scheme. The CP-ABE toolkit and Java Pairing-Based Cryptography [22] are used for [18] and [20]. In the

experiment, the sample document size is 20 KB, the access policy used in all schemes contain 5 attributes. We conducted the experiment 20 times and used their average score to plot the graph as shown in Figure 3.

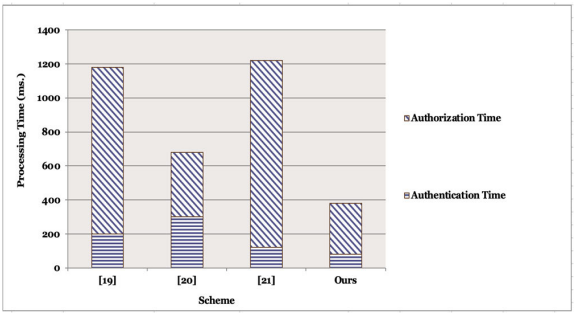

FIGURE 4. ID verification/authorization and authorization runtime.

According to the result shown in Fig.4, our scheme provides least processing time for both authentication and authorization. In scheme [19], the processing time for access control is slightly greater than ours because it uses public key decryption in both the authentication (ID verification and SSO authentication) and authorization. Scheme [18] and [20] took more time for authorization since they rely on CP-ABE decryption. Here, if there are more attributes involved, they will need more decryption time for privilege checking.

Additionally, we performed the experiment to assess the scalability of the proposed system. Here, we measured the access control throughput by using JMeter. In the test, the communication cost for contacting the blockchain was excluded. Figure 5 shows the throughput of all schemes in supporting a number of access control requests.

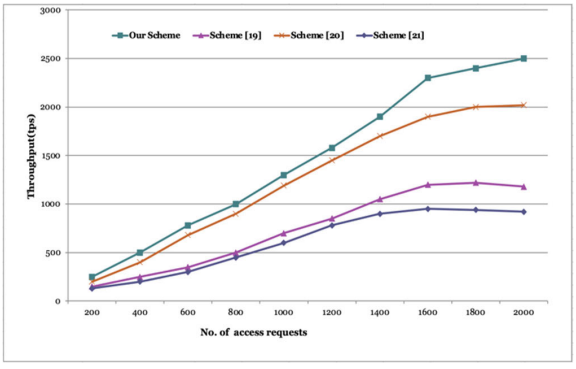

FIGURE 5. Access throughput.

As shown in Fig.5, our scheme yields highest throughput as it can accommodate high volume of access requests based on smaller cost of cryptographic operations. The graph shows that our scheme continuously produces more throughput when the workload increases and it slightly inclines when it the requests are large than 1,600. At 1,600 requests, our scheme attains the throughput approximately 1,300 tps. The throughput of scheme [19] is comparable to ours and the slope of the graph is about to mature when the requests are larger than 1,400. In scheme [18] and [20], the throughput is two times less than ours and [19] and both schemes started to saturate when the requests are greater than 1,400.

In conclusion, the overall performance of the access control is subject to the core processing cost of the cryptographic operation of identity verification and authorization. Our scheme achieves higher efficiency and practicality as it enjoys lightweight ID verification and fast SSO authentication by using hashing and SSO token generation based on highest authentication option.

VI. CONCLUSION

We have proposed a blockchain-based access control system called D^2-IAM system to support strong SSO-authentication, dynamic authorization, and preventive-based access control with accountability in cloud computing. Our system optimized the cost of SSO-authentication and authorization process through the design and implementation of smart contracts and blockchains. In addition to achieving high efficiency of authentication and authorization, the access policy is modeled in the document database which is ease of management. Also, the confidentiality of its content is guaranteed based on the public key encryption. We provide the efficiency analysis and experiment to show that D^2-IAM is efficient in practice and its performance outperforms existing works.

For future works,it is a need to devise the auditing protocol to validate the integrity of access policies located on cloud. Even though the policies are encrypted, the assurance of their integrity is still crucial. The public cloud auditing techniques [43], [44] are worth to explore. In addition, the adoption of decentralized storage platforms, such as Inter Planetary File System (IPFS) for storing the policies, PII database can be used to replace the general cloud storage since IPFS provides more efficient file handling with data indexing. Finally, it is worth to develop the anomaly detection method based on machine learning to detect the authentication protocol attack or the misuse of the SSO authentication ticket.

REFERENCES

[1] S. Fugkeaw, P. Manpanpanich, and S. Juntapremjitt, "Exploiting X.509 certificate and multi-agent system architecture for role-based access control and authentication management," in *Proc. 7th IEEE Int. Conf. Comput. Inf. Technol. (CIT)*, Oct. 2007, pp. 733–738.
[2] *The Open ID Connect*. Accessed: Jan. 14, 2023. [Online]. Available: https://openid.net/connect/
[3] F. F. Moghaddam, P. Wieder, and R. Yahyapour, "A policy-based identity management schema for managing accesses in clouds," in *Proc. 8th Int. Conf. Netw. Future (NOF)*, Nov. 2017, pp. 91–98.
[4] N. Naik and P. Jenkins, "A secure mobile cloud identity: Criteria for effective identity and access management standards," in *Proc. 4th IEEE Int. Conf. Mobile Cloud Comput., Services, Eng. (MobileCloud)*, Mar. 2016, pp. 89–90.
[5] S. Wang, R. Pei, and Y. Zhang, "EIDM: A Ethereum-based cloud user identity management protocol," *IEEE Access*, vol. 7, pp. 115281–115291, 2019.
[6] N. Hossain, M. A. Hossain, M. Z. Hossain, M. H. I. Sohag, and S. Rahman, "OAuth-SSO: A framework to secure the OAuth-based SSO service for packaged web applications," in *Proc. 17th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun./12th IEEE Int. Conf. Big Data Sci. Eng.*, New York, NY, USA, Aug. 2018, pp. 1575–1578, doi: 10.1109/TRUSTCOM/BIGDATASE.2018.00227.

[7] C. Tang, X. Fu, and P. Tang, "Policy-based network access and behavior control management," in *Proc. IEEE 20th Int. Conf. Commun. Technol. (ICCT)*, Oct. 2020, pp. 1102–1106.
[8] H. Zhou and L. Zhu, "Research and design of CAS protocol identity authentication," in *Proc. Int. Conf. Comput. Vis., Image Deep Learn. (CVIDL)*, Jul. 2020, pp. 384–387.
[9] M. A. Thakur and R. Gaikwad, "User identity and access management trends in IT infrastructure—An overview," in *Proc. Int. Conf. Pervasive Comput. (ICPC)*, Jan. 2015, pp. 1–4.
[10] M. Uddin, S. Islam, and A. Al-Nemrat, "A dynamic access control model using authorising workflow and task-role-based access control," *IEEE Access*, vol. 7, pp. 166676–166689, 2019.
[11] S. Fugkeaw, P. Manpanpanich, and S. Juntapremjitt, "A development of multi-SSO authentication and RBAC model in the distributed systems," in *Proc. 2nd Int. Conf. Digit. Inf. Manage.*, 2007, pp. 297–302.
[12] L. Hui, *Computational Intelligence and Security* (Lecture Notes in Computer Science) vol. 3802. Berlin, Germany: Springer, 2005.
[13] L. Chunge, M. Mingji, L. Bingxu, and C. Shuxin, "Design and implementation of trust-based access control model for cloud computing," in *Proc. IEEE 5th Adv. Inf. Technol., Electron. Autom. Control Conf. (IAEAC)*, Mar. 2021, pp. 1934–1938.
[14] B. Cusack and E. Ghazizadeh, "Evaluating single sign-on security failure in cloud services," *Bus. Horizons*, vol. 59, no. 6, pp. 605–614, Nov. 2016.
[15] B. Qin, J. Huang, Q. Wang, X. Luo, B. Liang, and W. Shi, "Cecoin: A decentralized PKI mitigating MitM attacks," *Future Gener. Comput. Syst.*, vol. 107, pp. 805–815, Jun. 2020.
[16] W. Jiang, H. Li, G. Xu, M. Wen, G. Dong, and X. Lin, "PTAS: Privacy-preserving thin-client authentication scheme in blockchain-based PKI," *Future Gener. Comput. Syst.*, vol. 96, pp. 185–195, Jul. 2019.
[17] 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, doi: 10.1109/ACCESS.2019.2939368.
[18] S. Wang, X. Wang, and Y. Zhang, "A secure cloud storage framework with access control based on blockchain," *IEEE Access*, vol. 7, pp. 112713–112725, 2019, doi: 10.1109/ACCESS.2019.2929205.
[19] C. Yang, L. Tan, N. Shi, B. Xu, Y. Cao, and K. Yu, "AuthPrivacyChain: A blockchain-based access control framework with privacy protection in cloud," *IEEE Access*, vol. 8, pp. 70604–70615, 2020, doi: 10.1109/ACCESS.2020.2985762.
[20] Y. Ping and H. Sato, "A decentralized framework enabling privacy for authorizable data sharing on transparent public blockchains," in *Proc. Int. Conf. Algorithms Archit. Parallel Process.*, Dec. 2021, pp. 693–709.
[21] J. Viega, "Practical random number generation in software," in *Proc. 19th Annu. Comput. Secur. Appl. Conf.*, Las Vegas, NV, USA, 2003, pp. 129–140, doi: 10.1109/CSAC.2003.1254318.
[22] *PBC (Pairing-Based Cryptography) Library*. Accessed: Oct. 20, 2022. [Online]. Available: https://crypto.stanford.edu/pbc/
[23] G. Wood, et al., "Ethereum: A secure decentralized generalised transaction ledger," Ethereum Project Yellow Paper, Tech. Rep. 151, 2014, pp. 1–32.
[24] G. Zhang, T. Li, Y. Li, P. Hui, and D. Jin, "Blockchain-based data sharing system for AI-powered network operations," *J. Commun. Inf. Netw.*, vol. 3, no. 3, pp. 1–8, 2018.
[25] Y. Ding and H. Sato, "Derepo: A distributed privacy-preserving data repository with decentralized access control for smart health," in *Proc. 7th IEEE Int. Conf. Cyber Secur. Cloud Comput. (CSCloud)/6th IEEE Int. Conf. Edge Comput. Scalable Cloud (EdgeCom)*, Aug. 2020, pp. 29–35.
[26] S. Gao, G. Piao, J. Zhu, X. Ma, and J. Ma, "TrustAccess: A trustworthy secure ciphertext-policy and attribute hiding access control scheme based on blockchain," *IEEE Trans. Veh. Technol.*, vol. 69, no. 6, pp. 5784–5798, Jun. 2020.
[27] F. Ghaffari, E. Bertin, N. Crespi, S. Behrad, and J. Hatin, "A novel access control method via smart contracts for internet-based service provisioning," *IEEE Access*, vol. 9, pp. 81253–81273, 2021.
[28] S. Fugkeaw, "Enabling trust and privacy-preserving e-KYC system using blockchain," *IEEE Access*, vol. 10, pp. 49028–49039, 2022.
[29] Y. Fan, X. Lin, W. Liang, J. Wang, G. Tan, X. Lei, and L. Jing, "TraceChain: A blockchain-based scheme to protect data confidentiality and traceability," *Software: Pract. Exper.*, vol. 52, no. 1, pp. 115–129, Jan. 2022, doi: 10.1002/spe.2753.
[30] *OAuth 2.0*. Accessed: Jan. 14, 2023. [Online]. Available: https://oauth.net/2/
[31] *FIDO Alliance*. Accessed: Jan. 14, 2023. [Online]. Available: https://fidoalliance.org
[32] M. Al-Zubaidie, Z. Zhang, and J. Zhang, "PAX: Using pseudonymization and anonymization to protect patients' identities and data in the healthcare system," *Int. J. Environ. Res. Public Health*, vol. 16, no. 9, p. 1490, Apr. 2019, doi: 10.3390/ijerph16091490.
[33] J. Park, R. Sandhu, M. Gupta, and S. Bhatt, "Activity control design principles: Next generation access control for smart and collaborative systems," *IEEE Access*, vol. 9, pp. 151004–151022, 2021, doi: 10.1109/ACCESS.2021.3126201.
[34] M. J. H. Faruk, H. Shahriar, M. Valero, S. Sneha, S. I. Ahamed, and M. Rahman, "Towards blockchain-based secure data management for remote patient monitoring," in *Proc. IEEE Int. Conf. Digit. Health (ICDH)*, Sep. 2021, pp. 299–308.
[35] Z. Ullah, B. Raza, H. Shah, S. Khan, and A. Waheed, "Towards blockchain-based secure storage and trusted data sharing scheme for IoT environment," *IEEE Access*, vol. 10, pp. 36978–36994, 2022, doi: 10.1109/ACCESS.2022.3164081.
[36] J. Wang, J. Chen, N. Xiong, O. Alfarraj, A. Tolba, and Y. Ren, "S-BDS: An effective blockchain-based data storage scheme in zero-trust IoT," *ACM Trans. Internet Technol.*, Feb. 2022, doi: 10.1145/3511902.
[37] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Buenzli, and M. Vechev, "Securify: Practical security analysis of smart contracts," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 67–82.
[38] M. Rodler, W. Li, G. O. Karame, and L. Davi, "Sereum: Protecting existing smart contracts against re-entrancy attacks," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2019.
[39] E. Erdem and M. T. Sandikkaya, "OTPaaS—One time password as a service," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 3, pp. 743–756, Mar. 2019, doi: 10.1109/TIFS.2018.2866025.
[40] X. Yang, X. Yi, S. Nepal, I. Khalil, X. Huang, and J. Shen, "Efficient and anonymous authentication for healthcare service with cloud based WBANs," *IEEE Trans. Services Comput.*, vol. 15, no. 5, pp. 2728–2741, Sep. 2022, doi: 10.1109/TSC.2021.3059856.
[41] R. Vinoth, L. J. Deborah, P. Vijayakumar, and B. B. Gupta, "An anonymous pre-authentication and post-authentication scheme assisted by cloud for medical IoT environments," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 5, pp. 3633–3642, Sep. 2022, doi: 10.1109/TNSE.2022.3176407.
[42] Ethereum Foundation. (2020). *Vyper Documentation*. Accessed: Jan. 14, 2022. [Online]. Available: https://vyper.readthedocs.io/en/latest/?badge=latest
[43] F. Wang, L. Xu, J. Li, and K.-K.-R. Choo, "Lightweight public/private auditing scheme for resource-constrained end devices in cloud storage," *IEEE Trans. Cloud Comput.*, vol. 10, no. 4, pp. 2704–2716, Oct. 2022, doi: 10.1109/TCC.2020.3045806.
[44] X. Li, S. Liu, R. Lu, M. K. Khan, K. Gu, and X. Zhang, "An efficient privacy-preserving public auditing protocol for cloud-based medical storage system," *IEEE J. Biomed. Health Informat.*, vol. 26, no. 5, pp. 2020–2031, May 2022, doi: 10.1109/JBHI.2022.3140831.

SOMCHART FUGKEAW (Member, IEEE) received the bachelor's degree in management information systems from Thammasat University, Bangkok, Thailand, the master's degree in computer science from Mahidol University, Thailand, and the Ph.D. degree in electrical engineering and information systems from The University of Tokyo, Japan, in 2017. He is currently an Assistant Professor with Sirindhorn International Institute of Technology, Thammasat University. His research interests include information security, access control, cloud computing security, big data analysis, and high-performance computing. He has served as a Reviewer for several international journals, such as IEEE Access, IEEE Transactions on Information Forensics and Security, IEEE Transactions on Dependable and Secure Computing, IEEE Transactions on Cloud Computing, IEEE Transactions on Big Data, IEEE Transactions on Network Science and Engineering, IEEE Transactions on Network and Service Management, IEEE Transactions on Parallel and Distributed Systems, Computer and Security, IEEE System Journal, and *ACM Transactions on Multimedia Computing Communications and Applications*.