An Ethereum Based Attribute-Based Access Control for IoT

M.A. Maalla[I], S.V. Bezzateev[II]

[I] ITMO University, Saint Petersburg, Russia
[II] Saint Petersburg State University of Aerospace Instrumentation, Saint Petersburg, Russia

Аннотация. An Ethereum-based integrated access control is proposed in this paper to provide a scalable access control since the policies are not related to users, instead they are related to the attributes which provide more generic and scalable way to handle the increase of users in the future. Proposed Ethereum-based ABAC framework includes several principal elements: users, Ethereum, a ChainLink gateway, IPSF, and devices. This Ethereum-based system utilizes smart contracts (SCs) to facilitate access control like AccessRequestContract, AdminPolicyManager, IoTDataManager, PolicyEvaluator, IPFSDataHandler, and ChainLinkOracleAdapter. Storing data on distributed system like IPFS and using Chainlink to handle communications between on/off-chain brings more efficient to the proposed model. Together, these components enable our proposed model to deliver distributed, efficient, and immutable access control management for the IoT devices.

Ключевые слова: *Ethereum, Blockchain, ABAC, Chainlink, IPFS.*
DOI: 10.14357/20790279240104 **EDN: IBRCUX**

Introduction

The expansion of IoT sphere makes it so difficult to keep up with all the new security risks and attack trends. Moreover, it has become essential to come up with a more efficient access control mechanism that is generic and dynamic to not be applicable only to certain IoT devices and also to be scalable to service wide range of users. However, with the limited computing capability of most of the IoT devices, it's essential to shift the processing from IoT devices to other component in the system, Attributes-based access control is a great, simple and scalable type of access control that fits the need of the system, and it's based on attributes instead of users, which can be different types of attributes that makes the system more scalable and dynamic [1]. However, this access control alone is not immune and it's hard to keep the integrity of the access management, Integrating this technique with blockchain technology like Ethereum, deliver the immutability required for such system and also the decentralization and transparent due to the characteristics of the blockchain technology[2] [3]. The issue of this system alone is where should we store the data related to the access management, security policies, and user management, this issue is resolved by using a decentralized system IPFS which is designed for distributed peer-to-peer sharing which solve the centralization problem. IPFS offers a decentralized way of storing and sharing data, enhancing efficiency and speed by retrieving files from the nearest node. It resists censorship, reduces redundancy, and provides a more robust, version-controlled system for a persistent and resilient internet. This technology is particularly beneficial for decentralized applications, content distribution, and digital archiving [4].

We can communicate with this IPFS network by using ChainLink which connects existing systems to any public or private blockchain and enables secure cross-chain communication, in this way we shift storing data on-chain to off-chain IPFS in secure way using ChainLink and provide better performance on Ethereum network[5].

1. Proposed Model

Our proposed model is an innovative and solid access control that combines between the scalability of ABAC and immutability and transparency of Blockchain designed for IoT. This model is integrated with IPFS (InterPlanetary File System) through ChainLink, is a novel approach that provide efficient, transparent, decentralized, immutable access control. In this model, we use the traditional ABAC to provide a scalable access control, the whole management and mechanism process is done by the blockchain network through

SCs, which brings immutability to the system, all required data for the model is stored on IPFS decentralized system, and the whole communication between on/off-chain is controlled by SCs.

Blockchain is the core element in the model since it controls the whole process, verifies the attributes provided by the users, grants/denies the access, and also user management and all other data related to the objects on IPFS [6].

One of the innovative aspects of this model is the integration with Chainlink, a decentralized oracle network. As shown in Figure 1, This integration allows the blockchain network to communicate with off-chain services. Chainlink oracles provide a bridge between these data stored on IPFS and the blockchain network, since we can't store all data on blockchain and we need a third party to access the storing systems. This ensures that the AC system can access up-to-date and accurate information about users and objects, which is essential for grant/deny accessing the objects [7][8].

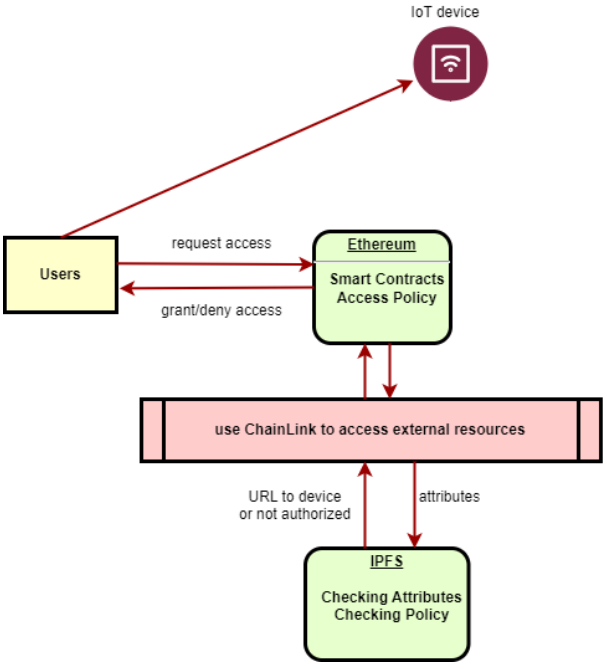

Fig. 1. The system flow

1.1. ABAC Model and Requirements

Attributes can be grouped , symbolized as A ∈ {S, O, P, E} [7][9], where A = {name: value}:

- S signifies the subject's attribute, which can be presented as an ID, profession… etc.
- O pertains to the object's attribute. Which can be IP address, category, device…etc.
- P relates to the permission attribute, like read, write, delete, or executing.
- E is the environment's attribute, like time, date, physical location… etc.

1.2. Policy Definition

We represent the policy as a generic equation that accommodates various combinations of attributes from the sets S, O, P, and E, we use logical operators to encapsulate the possible conditions. Each policy P_i can be thought of as a logical statement that combines conditions on subject attributes S, object attributes O, permission attributes P, and environmental attributes E.

Let's denote each set of attributes as follows: $S = \{s_1, s_2, \ldots, s_m\}$, $O = \{o_1, o_2, \ldots, o_n\}$, $P = \{p_1, p_2, \ldots, p_k\}$, and $E = \{e_1, e_2, \ldots, e_l\}$, A generic policy P_i can be represented as a logical statement combining elements from these sets:

$$\mathrm{P_i} = \left(C_s(S) \wedge \mathrm{C_o(O)} \wedge \mathrm{C_p(P)} \wedge \mathrm{C_e(E)}\right) \Rightarrow AccessGranted$$

Where: $C_s(S)$, $\mathrm{C_o(O)}$, $\mathrm{C_p(P)}$ and $\mathrm{C_e(E)}$ are conditions or combinations of conditions applied to the respective attribute sets.

1.3. Policy Evaluating

We represent this step by **EvaluatePolicy** procedure which is done by SCs to assure the validity of the attributes, as an equation for evaluating whether all attributes are correct, we can define it in a logical format. Let's denote the policy as a set of conditions $C = \{c_1, c, \ldots, c_n\}$ and the attributes provided in the access request as $A = \{a_1, a_2, \ldots, a_n\}$. Each c_i corresponds to a condition in the policy, and each a_i corresponds to an attribute in the request.

$$EvaluatePolicy(C, A) = \bigwedge_{i=1}^{n} (c_i \leftrightarrow a_i)$$

This equation states that *EvaluatePolicy* will return true if and only if each condition ci in the policy matches its corresponding attribute ai in the access request. The operator ⋀ denotes a logical AND across all conditions, ensuring that the function returns true only when every single condition c_i exactly matches the corresponding attribute a_i.

1.4. Smart Contracts

The SC is the core of our system since it controls the whole process. There are six SCs employed to achieve this model:

1. AccessRequestContract: Handles user access requests and communicates with policy evaluation.
   - submitAccessRequest(attributes): Users send the attributes to the blockchain network.
   - retrievePolicy(): Retrieves the relevant policy matching the submitAccessRequest for the uers.
   - requestEvaluation(): Initiates the policy evaluation process.
2. AdminPolicyManager: Allows admins to manage the whole mechanism.

- addPolicy(policyData): Add a new policies .
- updatePolicy(policyId, newPolicyData): Update an existing policy.
- deletePolicy(policyId): Remove an existing policy.
- viewPolicy(policyId): View details of a specific policy.

3. IoTDataManager: Manages data related to objects.
   - addIoTDevice(deviceData): Register a new object.
   - updateIoTDevice(deviceId, newDeviceData): Update details of an existing object.
   - deleteIoTDevice(deviceId): Remove an object from the system.
   - getIoTDeviceInfo(deviceId): Retrieve information about a specific object.
4. PolicyEvaluator: Evaluates access requests against policies.
   - evaluatePolicy(userAttributes, policy): Check the attributes and the policies and make a decision to grant/deny access.
5. IPFSDataHandler: Interfaces with IPFS for data retrieval and storage.
   - storeData(data): Store data on IPFS.
   - retrieveData(hash): Retrieve data from IPFS using its hash.
6. ChainLinkOracleAdapter: Facilitates communication with ChainLink oracles for off-chain data.
   - requestDataFromOracle(dataRequest): Request data from an off-chain source via ChainLink.
   - receiveDataFromOracle(): Receive data from ChainLink oracle.

1.5. Integration with IPFS

To detail the integration of our model with IPFS, let's construct an algorithm that illustrates the process. This algorithm will include the functions and steps necessary to interact with IPFS through the hashes only which is stored on the blockchain network, and getting object's data from IPFS using the same hash stored on blockchain as in fig. 2.

1.6. ChainLink Oracle Integration

Integrating Chainlink oracles is a necessity for our system, which will interact with the Chainlink network to retrieve or send data to external sources. Here is a LaTeX representation of the algorithm for integrating Chainlink oracles with our SC as shown in fig. 3:

In this way we connect the Ethereum platform with IPFS network and we provide on/off-chain communication in efficient way.

2. Security Considerations

Security is a critical aspect of our system involving Ethereum blockchain, IPFS, and Chainlink oracles

Algorithm 1 IPFS Integration Algorithm

```
procedure PrepareData(data)
    serializedData ← serialize(data)
    return serializedData
procedure UploadToIPFS(serializedData)
    ipfsHash ← IPFS.upload(serializedData)
    return ipfsHash
procedure StoreIPFSHashInContract(recordID, ipfsHash)
    txID ← EthereumSmartContract.storeHash(recordID, ipfsHash)
    return txID
procedure GetIPFSHashFromContract(recordID)
    ipfsHash ← EthereumSmartContract.getHash(recordID)
    return ipfsHash
procedure RetrieveDataFromIPFS(ipfsHash)
    data ← IPFS.fetch(ipfsHash)
    return data
data ← "Policy or IoT Device Information"
recordID ← "UniqueIdentifierForData"
serializedData ← PrepareData(data)
ipfsHash ← UploadToIPFS(serializedData)
StoreIPFSHashInContract(recordID, ipfsHash)
retrievedHash ← GetIPFSHashFromContract(recordID)
retrievedData ← RetrieveDataFromIPFS(retrievedHash)
```

Fig. 2. IPFS Integration Algorithm

for access control in IoT environments. We consider

Algorithm 2 Chainlink Oracle Integration for Data Retrieval

```
procedure RequestDataFromOracle(requestData)
    jobId ← specify Chainlink job ID for the task
    payment ← specify LINK token payment amount
    oracleAddress ← specify the Chainlink oracle contract address
    chainlinkRequest ← create a Chainlink request structure
    chainlinkRequest.add("get", requestData.apiUrl)
    chainlinkRequest.add("path", requestData.jsonPath)
    requestId ← sendChainlinkRequestTo(oracleAddress, chainlinkRequest, payment)
    return requestId
procedure ReceiveData(requestId, callbackFunction)
    fulfill Oracle response
    data ← read the data sent by the Chainlink oracle
    callbackFunction(data)
requestData ← structure with API URL and JSON path
requestId ← RequestDataFromOracle(requestData)        ▷ This will initiate the request to the Chainlink oracle
RecieveData(requestId, processRetrievedData) ▷ Callback function to process data
```

Fig. 3. Chainlink Integration

the following when we want to address and maintain the security of our model:

2.1. Data Confidentiality

We can achieve that by storing encrypted version of the data by AES algorithm and keep the corresponding hash on the blockchain network. And for key management, we can use blockchain network itself for secure distribution of the encrypted keys.

According to that we can update Algorithm 1 mentioned earlier in fig. 2 to include AES-encryption to the data stored on IPFS, and the updated version of the algorithm is shown in fig. 4:

The updated algorithm for integrating IPFS with Ethereum now incorporates additional steps to enhance data security significantly. Firstly, data is encrypted using AES before being uploaded to IPFS ensuring the data remains secure and confidential. Upon retrieval from IPFS, this data is then decrypted, allowing for secure access to the original information. To facilitate this, the algorithm also includes a robust procedure for the distribution and retrieval of AES keys through Ethereum SCs. This method

Algorithm 3 Integration of IPFS with Ethereum using AES Encryption

```
procedure EncryptData(data, aesKey)
    encryptedData ← AES.encrypt(data, aesKey)
    return encryptedData
procedure DecryptData(encryptedData, aesKey)
    data ← AES.decrypt(encryptedData, aesKey)
    return data
procedure DistributeAESKey(aesKey, authorizedEntities)
    keyHash ← EthereumSmartContract.storeKey(aesKey, authorizedEntities)
    return keyHash
procedure RetrieveAESKey(keyHash, entityID)
    aesKey ← EthereumSmartContract.getKey(keyHash, entityID)
    return aesKey
procedure PrepareData(data, aesKey)
    encryptedData ← EncryptData(data, aesKey)
    serializedData ← serialize(encryptedData)
    return serializedData
procedure UploadToIPFS(serializedData)
    ipfsHash ← IPFS.upload(serializedData)
    return ipfsHash
procedure StoreIPFSHashInContract(recordID, ipfsHash)
    txID ← EthereumSmartContract.storeHash(recordID, ipfsHash)
    return txID
procedure GetIPFSHashFromContract(recordID)
    ipfsHash ← EthereumSmartContract.getHash(recordID)
    return ipfsHash
procedure RetrieveDataFromIPFS(ipfsHash, aesKey)
    encryptedData ← IPFS.fetch(ipfsHash)
    data ← DecryptData(encryptedData, aesKey)
    return data
data ← "Policy or IoT Device Information"
recordID ← "UniqueIdentifierForData"
aesKey ← "AESKeyForEncryption"
authorizedEntities ← "List of Authorized Entity IDs"
keyHash ← DistributeAESKey(aesKey, authorizedEntities)
serializedData ← PrepareData(data, aesKey)
ipfsHash ← UploadToIPFS(serializedData)
StoreIPFSHashInContract(recordID, ipfsHash)
retrievedHash ← GetIPFSHashFromContract(recordID)
retrievedKey ← RetrieveAESKey(keyHash, "YourEntityID")
retrievedData ← RetrieveDataFromIPFS(retrievedHash, retrievedKey)
```

Fig. 4. Integration of IPFS with Ethereum using AES

guarantees that only authorized entities have access to these keys, thereby maintaining the integrity and confidentiality of the data throughout the process.

2.2. Hashing and Data Integrity

The concept of using IPFS hashes could indirectly ensure data integrity since IPFS generates a unique hash for each piece of data based on its content. However, additional cryptographic hash functions (like SHA-256) for pre-encryption integrity checks could be used so we can add another layer of integrity to the data delivered by the IPFS, and we only care about that part of the system, since the integrity of data managed by Ethereum, and SCs are already reserved due to the nature of blockchain itself.

3. Performance Analysis

To assess the performance and viability of our model, we utilized a PC equipped with an Intel i7 processor (2.60GHz) and 16 GB of RAM for the prototype implementation. For the development of smart contracts, the Solidity language was used. These smart contracts were created using Solidity and deployed on the Goerli testnet, which serves as a testing platform for Ethereum smart contract development.

In our experiment, we quantified the cost time expense associated with smart contracts' procedures deployed on the Goerli testnet. The specific contracts examined included *AccessRequestContract*, *AdminPolicyManager*, *IoTDataManager*, *PolicyEvaluator*, *IPFSDataHandler*, and *ChainLinkOracleAdapter*.

The performance tests have been conducted by different number of concurrent accesses to the six

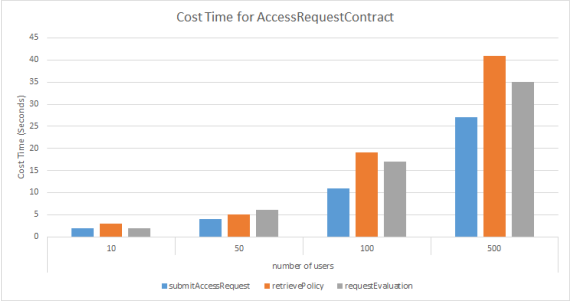

Fig. 5. Cost Time for AccessRequestContract

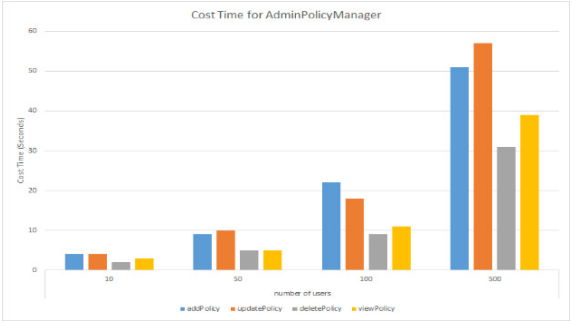

Fig. 6. Cost Time for AdminPolicyManager

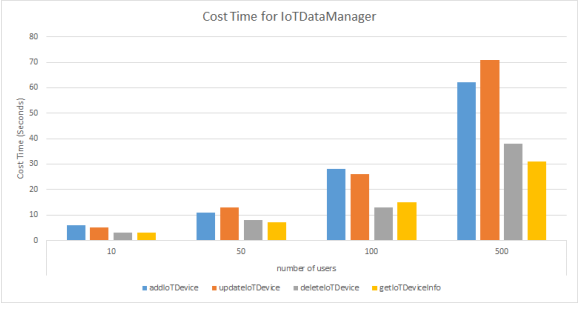

Fig. 7. Cost Time for IoTDataManager

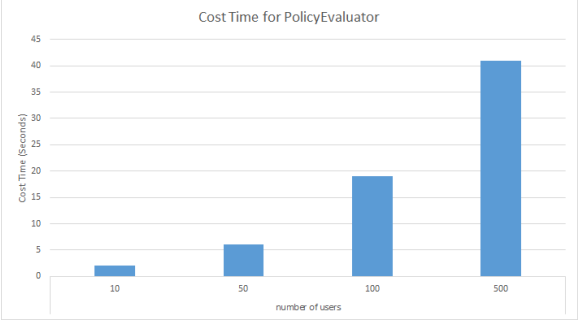

Fig. 8. Cost Time for PolicyEvaluator

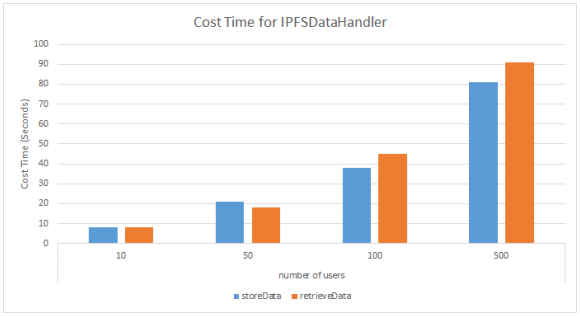

Fig. 9. Cost Time for IPFSDataHandler

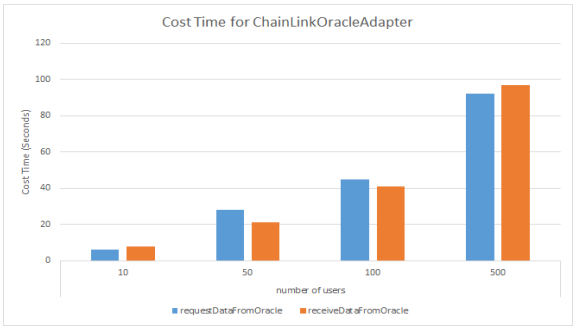

Fig. 10. Cost Time for ChainLinkOracleAdapter

smart contracts with some test data. Those numbers are 10, 50, 100, 500 requests and the results of cost time (in seconds) were measured for each smart contract as shown in fig. 5-10.

The results show the cost time for calling each procedure in the six smart contracts, which is considered acceptable considering the complexity of the deployed system. We can notice that the cost time of each procedure differs due to the differences in complexity of the contract, the more complex the smart contract, the more computational resources are needed to execute it. This includes the number of functions, the complexity of the logic within those functions, and the amount of data being processed.

Conclusion

The paper presents an innovative and solid access control that combines the scalability of ABAC and immutability and transparency of Blockchain designed for IoT. This approach effectively leverages Ethereum's decentralization, tamper-proofing, and immutability capabilities, moreover, it provides more distributed method to store data, also enhancing data availability and system resilience by using IPFS instead of traditional databases, with applying AES encryption to assure the security of data. Integration with Chainlink provides a secure, reliable and efficient channel to communicate between on/off-chain systems. Our experiments were conducted on all smart contracts: AccessRequestContract, AdminPolicyManager, IoTDataManager, PolicyEvaluator, IPFSDataHandler, and ChainLinkOracleAdapter and the results shows the cost time of each one. Moreover, storing data on distributed system like IPFS and using Chainlink to handle communications between on/off-chain brings more efficient to the proposed model.

References

1. *Shahid H. et al.* "Machine learning-based mist computing enabled Internet of Battlefield Things," ACM Trans. Internet Technol., vol. 21, no. 4, pp. 1–26, 2021.
2. *Andaloussi Y., El Ouadghiri M.D., Maurel Y., Bonnin J.-M. and Chaoui H.* "Access control in IoT environments: Feasible scenarios," Procedia Comput. Sci., vol. 130, pp. 1031–1036, 2018.
3. *Deebak B.D. and Fadi A.-T.* "Privacy-preserving in smart contracts using blockchain and artificial intelligence for cyber risk measurements," J. Inf. Secur. Appl., vol. 58, p. 102749, 2021.
4. *Norvill R., Pontiveros B.B.F., State R., and Cullen A.* "IPFS for reduction of chain size in Ethereum," in 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 2018, pp. 1121–1128.
5. *Breidenbach L. et al.* "Chainlink 2.0: Next steps in the evolution of decentralized oracle networks," Chain. Labs, vol. 1, pp. 1–136, 2021.
6. *Ouaddah A.* "A blockchain based access control framework for the security and privacy of IoT with strong anonymity unlinkability and intractability guarantees," in Advances in Computers, vol. 115, Elsevier, 2019, pp. 211–258.
7. *Figueroa S., Añorga J., Arrizabalaga S., Irigoyen I. and Monterde M.* "An attribute-based access control using chaincode in RFID systems," in 2019 10th IFIP international conference on new technologies, mobility and security (NTMS), 2019, pp. 1–5.
8. *Cruz J.P., Kaji Y. and Yanai N.* "RBAC-SC: Role-based access control using smart contract," Ieee Access, vol. 6, pp. 12240–12251, 2018.
9. *Liu H., Han D. and Li D.* "Fabric-IoT: A blockchain-based access control system in IoT," IEEE Access, vol. 8, pp. 18207–18218, 2020.

Maalla Maher A. PhD student, ITMO University, Kronverksky Pr. 49, bldg. A, St. Petersburg, 197101, Russia, E-mail:maher.malla7@gmail.com

Bezzzateev Sergey V. Head of department, Doctor of technical science, State University of Aerospace Instrumentation, Bolshaya Morskaya St. 67, bldg. A, St. Petersburg, 190000, Russia, E-mail: bsv@aanet.ru

Атрибутивная система контроля доступа на основе Ethereum для IoT

М.А. Маалла[I], С.В. Беззатеев[II]

[I] ITMO Университет, г. Санкт-Петербург, Россия

[II] Государственный университет аэрокосмического приборостроения, г. Санкт-Петербург, Россия

Аннотация. Предлагается использовать интегрированный контроль доступа на основе Ethereum для обеспечения масштабируемости, что обеспечивает более общий способ управления увеличением количества пользователей. Предлагаемая система ABAC на основе Ethereum включает несколько основных элементов: пользователь, Ethereum, шлюз ChainLink, IPSF и устройства. Эта система на основе Ethereum использует смарт-контракты AccessRequestContract, AdminPolicyManager, IoTDataManager, PolicyEvaluator, IPFSDataHandler и ChainLinkOracleAdapter для упрощения контроля доступом. Хранение данных в распределенной системе, такой как IPFS, и использование Chainlink для обработки коммуникаций между внешними и внутренними сетями делают предложенную модель более эффективной. Вместе эти компоненты позволяют предложенной модели обеспечивать распределенное, эффективное и неизменяемое управление доступом для устройств IoT.

Ключевые слова: *Ethereum, Блокчейн, ABAC, Chainlink, IPFS*
DOI: 10.14357/20790279240104 **EDN: IBRCUX**

Литература

1. *Shahid H. et al.* “Machine learning-based mist computing enabled Internet of Battlefield Things,” ACM Trans. Internet Technol., vol. 21, no. 4, pp. 1–26, 2021.
2. *Andaloussi Y., El Ouadghiri M.D., Maurel Y., Bonnin J.-M. and Chaoui H.* “Access control in IoT environments: Feasible scenarios,” Procedia Comput. Sci., vol. 130, pp. 1031–1036, 2018.
3. *Deebak B.D. and Fadi A.-T.* “Privacy-preserving in smart contracts using blockchain and artificial intelligence for cyber risk measurements,” J. Inf. Secur. Appl., vol. 58, p. 102749, 2021.
4. *Norvill R., Pontiveros B.B.F., State R., and Cullen A.* “IPFS for reduction of chain size in Ethereum,” in 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 2018, pp. 1121–1128.
5. *Breidenbach L. et al.* “Chainlink 2.0: Next steps in the evolution of decentralized oracle networks,” Chain. Labs, vol. 1, pp. 1–136, 2021.
6. *Ouaddah A.* “A blockchain based access control framework for the security and privacy of IoT with strong anonymity unlinkability and intractability guarantees,” in Advances in Computers, vol. 115, Elsevier, 2019, pp. 211–258.
7. *Figueroa S., Añorga J., Arrizabalaga S., Irigoyen I. and Monterde M.* “An attribute-based access control using chaincode in RFID systems,” in 2019 10th IFIP international conference on new technologies, mobility and security (NTMS), 2019, pp. 1–5.
8. *Cruz J.P., Kaji Y. and Yanai N.* “RBAC-SC: Role-based access control using smart contract,” Ieee Access, vol. 6, pp. 12240–12251, 2018.
9. *Liu H., Han D. and Li D.* “Fabric-IoT: A blockchain-based access control system in IoT,” IEEE Access, vol. 8, pp. 18207–18218, 2020.

Маалла Махер Аднан. Университет ИТМО, г. Санкт Петербург, Россия. Аспирант. Область научных интересов: информационная безопасность, блокчейн технологии. E-mail: maher.malla7@gmail.com

Беззатеев Сергей Валентинович. Государственный университет аэрокосмического приборостроения, г. Санкт-Петербург, Россия. Заведующий кафедрой. Доктор технических наук, доцент. Область научных интересов: теория кодирования, криптография, информационная безопасность. E-mail: bsv@aanet.ru (Ответственный за переписку)