Using Blockchain for IOT Access Control and Authentication Management

Abdallah Zoubir Ourad[1(✉)], Boutheyna Belgacem[1(✉)], and Khaled Salah[2(✉)]

[1] Universität Passau, Innstraße 43, 94032 Passau, Germany
abdallah.ourad@uni-passau.de, be@sec.uni-passau.de
[2] Khalifa University, Al Zafranah, Abu Dhabi, United Arab Emirates
khaled.salah@kustar.ac.ae
https://web.sec.uni-passau.de
https://www.kustar.ac.ae

Abstract. Securing Access to IOT devices is a challenging task as IoT devices are resource-constrained devices in terms of processing, storage, and networking capacity. Because of their fast spreading and deployment, significant disadvantages are seen in today's authentication and access control schemes. This paper proposes a blockchain-based solution which allows for authentication and secure communication to IOT devices. Our solution benefits greatly from the intrinsic features of blockchain and also builds on existing authentication schemes. Specifically, our proposed blockchain-based solution, architecture, and design allow for accountability, integrity, and traceability with tamper-proof logs. The paper provides overall system design and architecture, and details on testing and implementation of a realistic scenario as a proof of concept.

Keywords: Internet-Of-Things · Blockchain · Smart contract
Authentication · Access control

1 Introduction

Internet-Of-Things concept can be defined as a system of connected computing devices in various fields and forms that can be deployed everywhere. These devices can communicate with other devices, gather, share, and process information to deliver a certain service [12]. According to experts from CISCO, Ericsson and other companies, by the year 2020, we will have over 50 billion devices connected to the INTERNET [3,18]. IOT devices are functioning in all fields nowadays: house appliances, medical area, personal accessories, etc. To allow such functionality, these devices must have certain characteristics. They should posses the abilities of operating using low energy and communicating with other heterogeneous devices. Also, They should maintain stable connection with the back-end if existed and be able to receive patches when necessary.

Authentication and access control are key concepts to securely manage computer resources and networks. Based on the previously mentioned characteristics, these concepts should be redefined in the context of IOT. Authentication

D. Georgakopoulos and L.-J. Zhang (Eds.): ICIOT 2018, LNCS 10972, pp. 150–164, 2018.
https://doi.org/10.1007/978-3-319-94370-1_11

techniques and access control policies need to take the limited resources drawback into consideration. Likewise, when it comes to IOT conditions, classical approaches for access control like **ABAC** (Attribute-based access control) and **RBAC** (Role-based access control) are proven to be inflexible, unscalable and difficult to upgrade [6,13].

In addition, the centralized perception of authentication where all devices have to contact a certain entity creates a major drawback. Building a system that depends on a trusted third party implies the assumption of a TTP that is always available and authentic. This creates a bottleneck around the trusted party which affects availability. Also, the model fails when the centralized entity is compromised. Additionally, the TTP can tamper records without accountability. Such disadvantages in the IOT design can be overcome using the blockchain technology.

Blockchain is known as the underlying technology for **Bitcoin** [9]. It can be defined as a growing chain of records. By design, the blockchain inherits effective characteristics in which the blocks of records are decentralized, tamper-proof and can be accessed by all nodes equally. This concept can be extended to all types of applications that require a trusted third party to validate records or transactions. Blockchain made it possible to replace the trusted third party with a transparent untampered block of records that is available via a distributed form, hence, the trust is moved from a single entity to a decentralized community of blockchain nodes.

An effective method that utilizes blockchain concept is the smart-contract. Smart-Contract was first defined in 1996 as a self-operating or self-executing program [16]. This method was reintroduced in Ethereum blockchain to facilitate the development of blockchain automated applications.

Events and logs are Ethereum blockchain features. An event is a reply (returned value) from the smart contract to the user interface interacting with it. The main goal of using events and logs is to ease the communication between smart contracts and programs communicating with them.

Figure 1 demonstrates a sample scenario for an Ethereum Smart Contract Application. First, the client requests access for a certain resource/asset from the smart contract. Second, the smart contract checks if the asset is free then books the fee from the client. For this example, the client is paying via Ethereum crypto-currency. Third, the smart contract reserves the resource for the current client. Fourth, The client uses the resource as approved. Finally, if everything went according to the contract rules, the smart contract charges the client as agreed. It is important to note that the smart contract acts fully autonomously and the owner is not involved in any steps.

The rest of the paper is organized as follows. Section 2 will review various implementations of IOT authentication solutions. Then, Sect. 3 will explain the detailed design of the proposed solution. Section 4 will clarify the implementation phases, elements, technologies and techniques. After that, Sect. 5 will illustrate the tests performed on this paper's approach, and the outcome results and

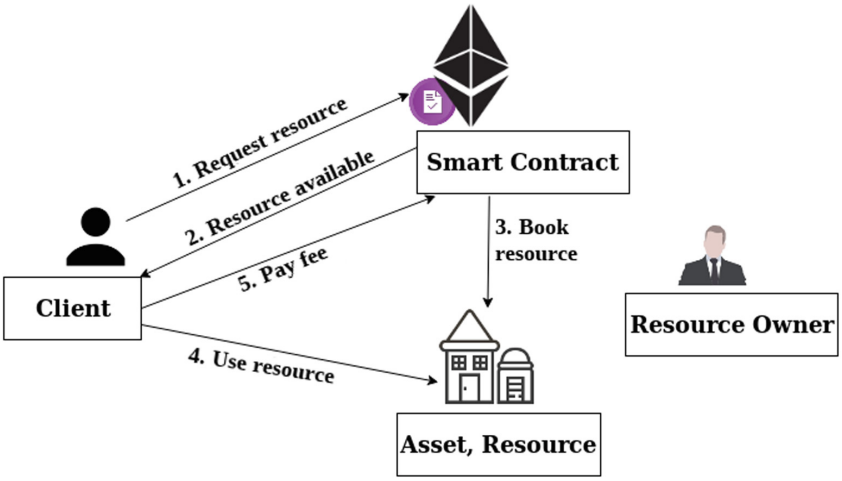

Fig. 1. A sample Ethereum smart contract scenario.

evaluation. Finally, the paper is concluded with the lessons learned and future plans that will help improving this solution in Sect. 6.

2 Related Work

This section presents different existing approaches to solve the problem of authentication management and access control in IOT devices. Section 2.1 will discuss traditional approaches used for authentication and access control in addition to their advantages and disadvantages. Solutions presented in Sect. 2.2 are distinguished by the fact that they all use blockchain as a backbone for their ideas.

2.1 IOT Authentication Traditional Models

A basic approach is to authenticate to each device directly using a combination of (username, password). This method provides decent access control because each authenticated user has his roles and permissions specified and stored on the device by the admin (owner). However, since the user must authenticate to each machine independently, this method produces an overhead and doesn't scale. This technique can be seen in classic IOT devices like IP cameras and Internet accessed home utilities.

Authenticating using single-sign-on protocols is a more advanced solution. For instance, when **oauth2** is deployed as an authentication method, users try to access a device by authenticating to a trusted oauth2 provider. This trusted third party can be GOOGLE, FACEBOOK, etc. If they successfully authenticate and have the required permissions, the trusted entity grants access. All devices

managed by the same individual can be accessed by authenticating to the trusted entity [14]. Figure 2 illustrates an abstract of the Oauth2 protocol flow in the context of IOT. First, when the user tries to access the IOT device resources, the device act as the oauth2 client and sends an authorization request to the user. Second, the user grants the client the authority to communicate with the authorization server i.e. the oauth2 provider. Then, the IOT device contacts the oauth2 provider to access the user's information in order to check if such user is allowed to use its resources [14].

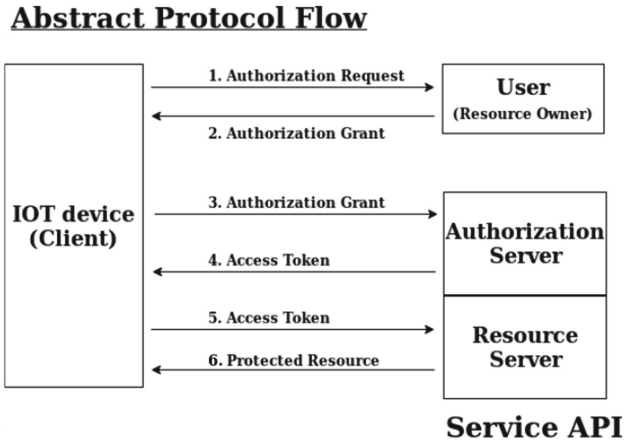

Fig. 2. An abstract for Oauth2 flow.

Using such approach saves time and effort since the user accesses multiple entities by authenticating to a single entity. Also, the oauth2 provider is usually a trusted third party with a high reputation which eases the integration of such solution [14].

On the other hand, trusting a centralized entity increases the threat of having a single point of failure which threatens the availability of the presented approach. Moreover, if the user's account or the centralized entity were compromised, the whole system is affected. An essential attack vector that may lead to this model's failure is phishing which has a high successful rate. In addition, spear phishing campaigns are getting more intensive, precise and sophisticated nowadays which may even trick the educated users [2]. According to Symantec Latest Intelligence Report for June 2017, 76% of organizations came forward of being phishing victims in 2016 [11].

The approaches discussed so far present a valid implementation for IOT authentication. However, they suffer from some drawbacks that may affect scalability, performance and availability. The following approaches have the potential for an IOT authentication management and access control solution that uses blockchain technology.

2.2 Blockchain Based Authentication Models

Auth0 introduced a method to authenticate to a server via Ethereum blockchain using a challenge-response method. The drawback discovered in auth0 approach is the need of a 3rd party authentication server. It is a hybrid solution that combines the decentralization of blockchain with the centralization of a trusted third party. The problem of "Single Point of Failure" or "Single Point of Trust" will reappear with this approach. According to **Auth0**, the centralized server holds an essential role since it is involved in 62.5% of the whole operation. This increases the dependency on a centralized entity which contradicts the benefit of using blockchain technology [10].

Blockstack is presenting the concept of a new decentralized INTERNET. This network contains applications for authentication and storage. The system utilizes a user's keypair to authenticate in a similar manner to PKI. When a users is authenticated correctly, a unique JSON Web Token is created. The JWT permits access to all pre-authorized resources via the one authentication process validated previously.

On the downside, blockstack has many requirements to operate. The goal of this system is to shift to a newly decentralized network therefore, the first requirement is to use a blockstack browser. If not, then blockstack app should be installed on the user's machine. In addition, the system built an additional two layers on top of blockchain **- peer network layer and storage layer -** which increases the operational overhead. Finally, the entities interaction is based on a new domain name protocol called the **Blockchain Name System** as a replacement for traditional DNS [7]. BNS is one of many novel implementations that are aiming to replace classic domain name system. More examples include NameCoin and Ethereum Name Server [1].

3 Proposed Systems Design and Architecture

The paper offers a blockchain based solution with distinct system architecture. It addresses the drawbacks of current solutions. Also, it should be portable and run on any network with minimum dependencies unlike **blockstack**. It is targeting IOT devices that lack strong processing power. It is also presenting the idea of Oauth implementation via smart contract to login once and control all the authorized devices without the need to login separately for each IOT device. In addition, the IOT devices can run smart contracts to become self profiting devices.

Section 5.1 will discuss the costs of this solution. Section 5.2 will go over the tests operated on a running prototype in addition to the test outcome. Tests will include performance experiments and crafted attacks against them. Then, Sect. 5.3 will evaluate in terms of availability, scalability, decentralization, tamper proof and performance advantages and drawback.

There are many advantages for using Ethereum blockchain as a platform for this solution. Ethereum has a stable development framework with existing

incentive for miners to solve challenge hashes. Also, Ethereum light client protocol can run on IOT devices with low processing power and memory which is essential for the proposed solution [8].

One Time Authentication: Authenticate once Directly to the Blockchain then Access the Resource Using Smart Contract Tokens: In this scenario, the user authenticates to the smart contract which verifies his/her identity. The smart contract then determine if the user is allowed to access the resources. The authentication is performed in an isolated phase. When succeeded, the user can interact with the device in any preferred method e.g., ssh, http, https, etc. This accomplishes decentralized authentication in an efficient way. The evaluation of this solution will be discussed in Sect. 5.

3.1 Assumptions

To implement such solution, the following assumptions must be taken into consideration:

- The user owns one or more IOT devices.
- The user's Ethereum keystore is not compromised i.e. the private key is protected.
- The user has an Ethereum account.
- The user and the IOT device are connected to the Ethereum blockchain
- The user will deploy his smart contract.

The system can budge the last assumption with full functionality. A centralized smart contract that authenticate users to their respective IOT devices can be created. However, since one of the goals of this paper is to avoid depending on a central entity. It is more suitable to ask users to deploy their own smart contracts so they can achieve full control of their own systems. This will shape a system of decentralized smart contracts running on a decentralized blockchain where each user owns his/her smart contract.

3.2 System Architecture

The message sequence diagram shown in Fig. 3 illustrates the steps of the authentication process as follows:

1. The user Authenticates to the smart contract using his Ethereum wallet address.
2. If the user is valid, the smart contract broadcasts an Access token and the sender's ethereum address. The user and the IOT device receive the broadcasted information from the smart contract.
3. The user crafts a package that contains (Access token, User IP, Access duration and the ethereum public key). This package is signed using the ethereum private key then sent with the corresponding public key. The package can be encrypted if wanted. However, It is not required for the protocol to operate. Integrity is what matters in this scenario hence the signing of the message.

4. When the IOT device receives the package, it verifies its content. If succeeded, the device grants access to the user from the sender's IP for the duration specified. Otherwise, if any of those checks fails, the request is dropped. The verification phase is described in Sect. 4.2.

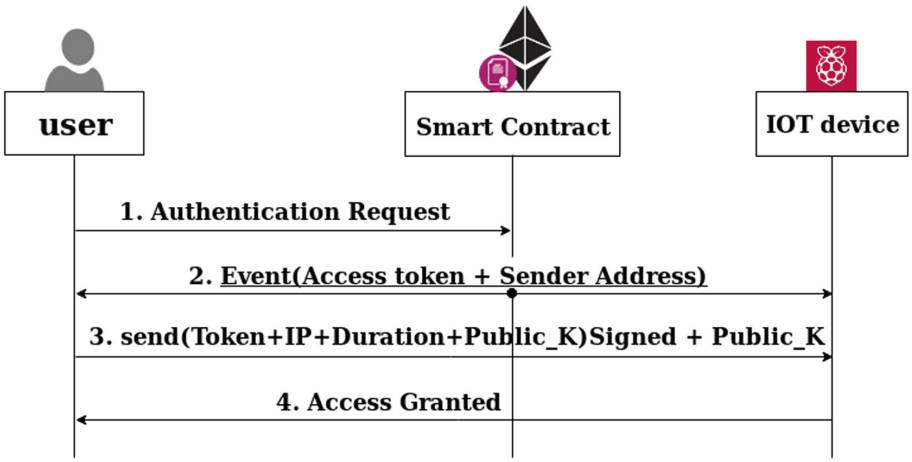

Fig. 3. The second solution authentication scenario.

The smart contract completes its authentication task in the first step. It is noted that the IOT device needs to perform many verification steps. However, as discussed in Sect. 5, this solution runs successfully on a standard raspberry Pie 3 Model B. More details on the technical specifications of the smart contract and the IOT server are explained in Sect. 4.

4 Implementation and Deployment

Implementing the proposed solution can be compartmentalized to the separate phases that can be built individually then integrated accordingly. This will facilitate the developing process. Also, debugging will be easier once challenges are faced. The presented solution will be explained in two subsections. The first subsection will explain the Smart contract functionality and how it performs authentication as the first phase. The second subsection discusses the authentication interaction between the user and the IOT device after the first phase is successfully executed.

4.1 Phase 1: Smart Contract

As mentioned in Sect. 3, the first phase occurs when the user authenticates to the smart contract to prove he/she is a legitimate user. Pseudo-code below describes

the source code of the smart contract. This version of the smart contract only considers the admin user as a legitimate user. In other words, the user who deployed this smart contract on the blockchain is the admin user. Adding more users can be achieved by adding an *addUser()* method to the smart contract.

The Smart Contract of the Proposed Solution:

```
contract Login2
Declare Private owner, hash, token_raw, random_number
//begin constructor:
owner = msg.sender;
End Constructor

Private Function login_admin()
IF msg.sender == owner
   set random_number = random(1,100);
   set hash = keccak256(msg.sender,now,random_number);
   trigger even LoginAttempt(msg.sender, hash);
Endif
End Function
End Class
```

(Source code is available on: https://github.com/sasukeourad/OTA)

When users want to authenticate, they use their Ethereum client to call method *login_admin*. The *login_admin* function requires no parameters and it is protected from public usage i.e. only authorized users can call it because a modifier performs the verification of the sender's Ethereum address. If a verified user calls this function, *login_admin* creates a random hash using function *rand*. Then, *login_admin* creates a token by hashing the user's ethereum address, block time and the random hash created in the latest step. After that, an event is initiated to send the token and the authenticated user's address back to the IOT device and user to proceed with phase 2.

4.2 Phase 2: User-IOT Interaction

When phase 1 is completed successfully, the user and the IOT device receive an authentication token and the Ethereum address of the authorized user. Phase 2 connects the two entities together. Note that this solution assumes that the user knows the address - ip or domain name - of the IOT device. In case this wasn't true, the device address can be sent by event LoginAttempt.

User Side Implementation Flow: First, by using **nodejs** and **web3 Ethereum client**, the user's script connects to the Ethereum blockchain listening for events coming from the deployed smart contract in phase 1. When the event arrives, the user's script accesses the keystore directory that contains the user's secret keys and extracts the private key. Note that the script needs the

key pass in order to perform such action. Then, the script extracts the public key from the private key. The public key is hashed using **keccak256** algorithm. The last 40 bytes of the resulting hash is the Ethereum address of the user. This Ethereum address is compared with the one received from the smart contract event. This is the formal method to obtain an Ethereum address from a keystore directory. Note that the public key usage is minimized and replaced by the Ethereum address since the address is shorter and easier to use. This is achieved using **keythereum** to access the private key and **elliptic** to derive the public key. Those nodejs libraries can be found online [4,17]. Figure 4 shows a message sequence diagram for the process of extracting Ethereum addresses from keystores [4,17].

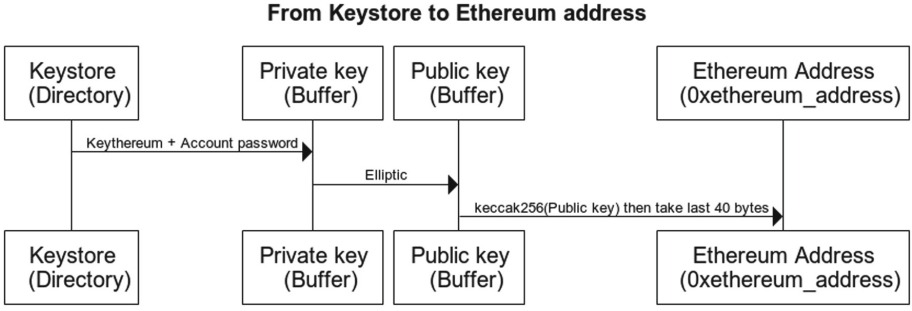

Fig. 4. The process of deriving the Ethereum address from the user's ethereum keystore

After assuring that the Ethereum address received from smart contract is the user's address. The script starts crafting the authentication message. The message can be described as following:

$$message = [token + src\_ip + Auth\_dur + Pub_K] \tag{1}$$

Where:

- **token:** is the token received from the smart contract.
- **src_ip:** is the ip address the user will connect from.
- **Auth_dur:** is the duration for the authentication validity before it is revoked and another authentication is required.
- **Pubk:** is the public key of the user's ethereum account.

Finally, the message is signed using the private key of the user's ethereum account. The following authentication package is sent to the IOT device:

$$message + Signature + Pub_K \tag{2}$$

IOT Side Implementation Flow: The IOT device script starts in a similar manner to the user script. It connects to the smart contract deployed in phase 1 and listens for the desired event. When the event occurs, the script gets the authentication token and the Ethereum address of the authenticated user. The script waits for the user's authentication package to arrive. If the package arrived, the verification phase starts. If any of the verification steps fail to succeed, the authentication package is dropped to minimize the workload on the IOT device processing power. The process moves to the next step only if the current step is verified.

Verification Phase Steps:

1. Is the authentication package and message in the correct format as shown in Eqs. 1 and 2?
2. Is the message signature valid? This is checked using the public key in Eq. 2
3. Is the public key in the authentication package in Eq. 2 similar to the one in the message in Eq. 1?
4. Is the token in the message similar to the token from the smart contract?
5. Is the source ip in the message similar to the source ip of the authentication package sender?
6. By hashing the public key in the message and taking the last 40 bytes ... Is the result similar to the Ethereum address from the smart contract?

If all those phases were verified, the user is authenticated. Otherwise, the authentication package is dropped. According to BigO standards, the IF-statement adds a linear execution complexity to the program depending on the input. It is denoted as:

$$O(kn) = O(n) \tag{3}$$

Where k is a constant.

5 Testing and Evaluation

After implementing the prototype of the proposed solution, this section discusses the tests performed to assure their security and functionality. Also, the system is evaluated against the other solutions discussed in Sect. 2. In addition, the cost required to establish the communication between the user and the smart contract is covered. Note that generally, only "setter" functions will cost the user since they require miners to modify the blockchain where the "getter" functions don't cost any time or money.

To modify smart contract attributes, a transaction should be made in the Ethereum blockchain. The fee is calculated in GAS and paid in Ether. When the smart contract is deployed, the owner has the choice to set the value of the amount of gas required. A higher gas price means this transaction is mined first. It is a trade-off between priority and cost. In December 2017 the price of "21K" gas used is $0.01. This gas amount translates to **2 gwei** which is the standard

speed to perform a transaction. This cost is arguably acceptable since it provides a decentralized authentication platform in addition to a tamper-proof block of records. Otherwise, the alternative solution would be to trust a centralized entity with your data where the threats of single point of failure and losing sensitive data increase. A third option is to manage a self-owned decentralized database which costs more for maintenance and management.

The fluctuating value of Ethereum price can raise a challenge for smart contract users. Since the deployment test of this paper's smart contract and according to Fig. 5, Ethereum value has peaked for certain interval then returned to the normal price. The fluctuating price of cryptocurrencies is a drawback that affects smart contract usage stability. However, this is planned to be solved in the future consensus algorithms proposed by Ethereum.

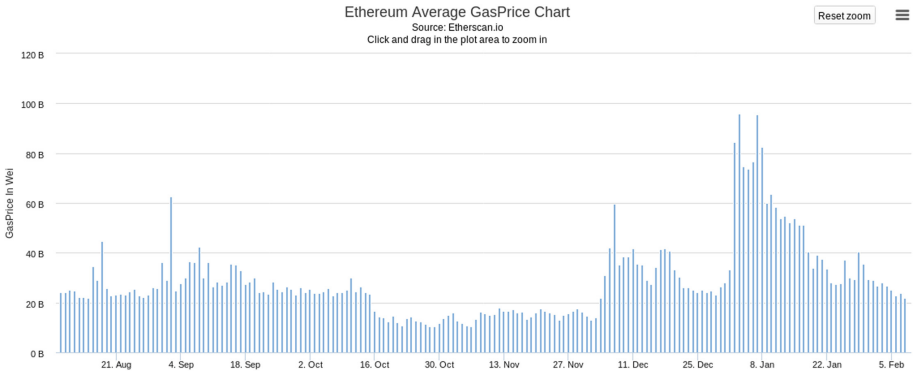

Fig. 5. Ethereum average gas price

5.1 Costs

Table 1 illustrate the transaction cost, execution cost and the equivalent price in US dollars for deploying and using the smart contract.

Table 1. Gas cost for running functions

Function	Transaction cost	Execution cost	Total cost	Price in USD
Deployment	275153	170457	445610	$0.212
login_admin	64089	42817	106906	$0.051

First rows show the price of deploying the smart contract which is only done once. It is clear that deploying the smart contract is more expensive since it writes it on the blockchain. On the other side, the login functionality is cheaper.

The login function can be optimized to cost less. The reason for current price is that it generates the authentication token by hashing, generating a random number and retrieving the block hash. These prices represent the cost of using the traditional proof-of-work consensus protocol. With Ethereum moving to standardizing the proof-of-authority protocol, these costs will decrease noticeably since the miners' work load will decrease too.

5.2 Testing

The testing phase was divided into different subsections. First, manual tests are performed against the proposed solution to assure its robustness, security and performance. Manual tests include malicious scenarios in additional to the ideal test cases. Second, static analysis tools are used to perform automated security assessment for the smart contracts.

Manual Testing: After running the solution prototype, the ideal scenario was tested first. A legitimate user calls the smart contract function: *login_admin* using his/her MIST Ethereum client. The smart contract sends the authentication token and the user's Ethereum address to the user and IOT device simultaneously. According to the test, the first phase was completed in less than 4 s on a private blockchain. Then, the user connects to the IOT device by sending the authentication package explained in Eq. 2.

Bypassing the verification steps in the IOT authentication script were tested by performing few malicious attacks as explained below:

- A replay attack failed since the attacker's source IP needs to be identical to the source IP in the signed authentication message.
- Modifying the signed authentication message failed since the script verifies the message signature.
- Injecting the attacker's own authentication package failed since the public key should lead to the Ethereum address of the legitimate user.
- A man-in-middle may be able to sniff outgoing authentication packages. However, integrity is protected since he/she cannot modify the signed authentication message.

The outcome of the test phase proves that this solution is secure as long as the user's keypairs are not compromised. The authentication tokens should be invalidated and replaced once the authentication is successfully completed. The next steps are outside the scope of this paper.

When testing the current solution, the test environment varies. First, most tests are conducted in a private ethereum blockchain. This eases the process of mining and validating transactions. After that, when the smart contract is tested in the public Ethereum blockchain test-net, it is recommended to use **Rinkeby** instead of **Ropsten** since it uses Proof-of-Authority instead of Proof-Of-Work which is used in **Ropsten**. This will also ease the public testing process.

The proof of work concept is the current method used in Ethereum main network and Ropsten network to confirm transactions. A miner solves a mathematical puzzle to validate the transaction for an incentive. This approach requires a lot of processing power to execute. On the other hand, the proof of authority implemented in the Rinkeby test-net depends on a set of explicitly authorized nodes instead of miners solving mathematical problems therefore, it is considered the future of mining techniques where mining doesn't require as much processing power [19].

Automated Testing: Static Security Analysis: Security assessment via static analysis was performed using **mythril** by ConsenSys. It uses concolic analysis to detect security issues in smart contracts. It can operate in both whitebox and blackbox testing scenarios. Since the smart contract source code is available, whitebox testing was performed on it. Mythril returned no issues.

In addition, a control flow graph is created for the smart contract source code to assure all possible paths are checked. The graph is created using Ethereum Laser Symbolic Virtual Machine. Figure 6 shows the starting point of the smart contract control flow diagram.

Fig. 6. Starting point of Login2 CFG

Finally, another metric that can be proposed for future testing is performing formal method tests on the smart contract to verify that all possible execution paths are covered and anticipated. Current efforts are documented in [5, 15].

5.3 Evaluation

To assure the quality of the proposed solution, the next step is to compare it to previous solutions presented in Sect. 2. The evaluation metric is based on

Table 2. Comparing and evaluating authentication solutions

	Auth/device	Oauth2	Auth0	Blockstack	Paper sol.
Availability	✓	**X**	**X**	**X**	✓
Scalability	**X**	**X**	✓	✓	✓
Decentralization	✓	**X**	**X**	✓	✓
Tamper proof	**X**	**X**	**X**	✓	✓

whether the offered authentication scheme solved problems occurring in the other authentication mechanisms proposed for the IOT devices.

Table 2 shows a comparison between the proposed solution based on Availability, Scalability, decentralization and tamper proof. For this comparison, the evaluation metrics is defined more specifically. Availability is described as removing the bottleneck and functioning without a single point of failure. Scalability is used here to explain the added overhead to the usage of the application when more devices are added. Decentralization is the ability for the authentication application to run without depending on a central entity that may break the system if taken down. Tamper proof is the assurance that saved data and transactions cannot be tampered once registered in the logs of the system.

6 Conclusion

In this paper, we have proposed a blockchain-based solution to proved authenticate users to access securely IoT devices. We demonstrated how our approach overcomes the shortcomings of existing authentication schemes. We showed that our blockchain based solutions, using Ethereum smart contracts, can provide tamper proof records and decentralization to improve current approaches. We designed and implemented our solution considering real life scenarios using available IoT devices and technologies. Specifically, we showed how to successfully authenticate legitimate users to access their IOT devices. Also, we showed that our approach withstood crafted attacks that were attempting to hijack legitimate sessions and brute force credentials. As a future work, we plan to extend our the proposed approach with a massive scale access and authentication to include huge number of IoT devices and end users. We also plan to test the approach on real Ethereum blockchain network and measure performance in terms of cost (or gas) consumption and scalability. We also considering monetization aspects related to IoT devices and their data, whereby usage is paid through crypto-token of ether.

References

1. Al-Bassam, M.: SCPKI: a smart contract-based PKI and Identity System (2017)
2. Amadeo, R.: Don't trust OAuth: why the "Google Docs" worm was so convincing. https://arstechnica.com/security/2017/05/dont-trust-oauth-why-the-google-docs-worm-was-so-convincing/
3. Evans, D.: The Internet of Things. How the Next Evolution of the Internet Is Changing Everything (2011)
4. George V.: A Next-Generation Smart Contract and Decentralized Application Platform (2018)
5. Hirai, Y.: Formal Verification of Ethereum Contracts (2018)
6. Liu, J., Xiao, Y., Chen, C.: Authentication and access control in the Internet of Things. In: 2012 32nd International Conference on Distributed Computing Systems Workshops, pp. 588–592 (2012)
7. Ali, M., Shea, R., Nelson, J.: Blockstack: A New Internet for Decentralized Applications (2017)
8. McKinney, J.: Light client protocol (2017)
9. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008)
10. Peyrott, S.: An Introduction to Ethereum and Smart Contracts: an Authentication Solution. https://auth0.com/blog/an-introduction-to-ethereum-and-smart-contracts-part-3/
11. Symantec: Latest Intelligence for June 2017. https://www.symantec.com/connect/blogs/latest-intelligence-june-2017
12. Minerva, R., Biru, A., Rotondi, D.: Towards a definition of the Internet of Things (IoT) (2015)
13. Gusmeroli, S., Piccione, S., Rotondi, D.: IoT access control issues: a capability based approach. In: Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (2012)
14. Sandoval, K.: Why OAuth 2.0 Is Vital to IoT Security. https://nordicapis.com/why-oauth-2-0-is-vital-to-iot-security/
15. Swamy, N., Hriţcu, C., Keller, C., Rastogi, A., Delignat-Lavaud, A., Forest, S., Bhargavan, K., Fournet, C., Strub, P., Kohlweiss, M., Zinzindohoue, J., Zanella-Béguelin, S.: Dependent types and multi-monadic effects in F*. SIGPLAN Not. **51**, 256–270 (2016)
16. Szabo, N.: Smart Contracts: Building Blocks for Digital Markets (1996)
17. theethereum: Accounts, Addresses, Public and Private Keys, and Tokens. https://theethereum.wiki/w/index.php/Accounts,_Addresses,_Public_And_Private_Keys,_And_Tokens
18. Thomson, D.: IoT and the problem of identity. https://www.symantec.com/connect/blogs/iot-and-problem-identity
19. Tosh, D., Shetty, S., Liang, X., Kamhoua, C., Njilla, L.: Consensus protocols for blockchain-based data provenance: Challenges and opportunities. In: 2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON), pp. 469–474. IEEE (2017)