Blockchain-based identity and authentication scheme for MQTT protocol

Mwrwan Abdelrazig Abubakar
School of Computing, Edinburgh Napier University, Edinburgh, UK
M.Abubakar@napier.ac.uk

Zakwan Jaroucheh
School of Computing, Edinburgh Napier University, Edinburgh, UK
Z.Jaroucheh@napier.ac.uk

Ahmed Al-Dubai
School of Computing, Edinburgh Napier University, Edinburgh, UK
A.Al-Dubai@napier.ac.uk

Xiaodong Liu
School of Computing, Edinburgh Napier University, Edinburgh, UK
X.Liu@napier.ac.uk

ABSTRACT

The publish and subscribe messaging model has proven itself as a dominant messaging paradigm for IoT systems. An example of such is the commonly used Message Queuing Telemetry Transport (MQTT) protocol. However, the security concerns with this protocol have presented vital security challenges in most IoT applications. For example, the MQTT protocol does not have secure authentication mechanisms implemented and leaves that task to the developer as all the included native security services are fragile. This paper will present a well-thought approach involving a lightweight authentication and authorization scheme together with a decentralized identity system to manage the users' identities. This mechanism helps in facilitating the authentication for both subscribers and publishers by utilizing a smart contract in Ethereum blockchain to guarantee trust, accountability and preserve user privacy. We provided a proof-of-concept implementation to prove our work, which involves a decentralized MQTT platform and dashboard using our approach. The usability of this approach was further analyzed, particularly concerning CPU and memory utilization. Our analysis proved that our approach satisfies IoT applications' requirements since it reduces the consumption of resources and that smart contracts help in the automation of data management processes.

CCS CONCEPTS

• **Security and privacy** → Network security; Security protocols.

ACM Reference Format:
Mwrwan Abdelrazig Abubakar, Zakwan Jaroucheh, Ahmed Al-Dubai, and Xiaodong Liu. 2021. Blockchain-based identity and authentication scheme for MQTT protocol. In *2021 The 3rd International Conference on Blockchain Technology (ICBCT '21), March 26–28, 2021, Shanghai, China.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3460537.3460549

ICBCT '21, March 26–28, 2021, Shanghai, China

ACM ISBN 978-1-4503-8962-4/21/03…$15.00
https://doi.org/10.1145/3460537.3460549

1 INTRODUCTION

IoT and its application were first introduced as a concept of cooperation and interaction of several components, such as wireless sensor network and radio frequency identification (RFID) [1]. The interconnected devices must be reachable either physically or virtually and must produce content that can access regardless of location. However, it is plagued by the lack of a link between the physical and information world, which prevents the effective processing of data obtained from the interaction between users and the electronic equipment [2]. Sensors are used to collect data, such as humidity and temperature, which will be stored in a database located in the server or at times displayed in a comprehendible application interface. The HTTP is currently used in a request-reply messaging model to facilitate web data exchange and handle data acquisition from the hardware. However, it has proved unsuitable for use in resource-constrained IoT systems [3]. Since IoT is mainly used in resource-constrained devices, its protocols mandate that it uses low energy, gives back a real-time response and less bandwidth. The use of CoAP [4] provides a temporary fix as a lightweight request-reply protocol that is ideal for resource-constrained IoT systems due to its low level of consuming resources. However, it lacks the portability and scalability required by most functions. Over the past several years, various messaging models have emerged known for their resource efficiency and low communication overhead [5]. This includes NesC [6], LooCI [7] and MQTT [8]. The MQTT messaging protocol is a lightweight publish-subscribe messaging protocol popular for key advantages over others, such as low energy consumption. It can also be used from smartphones, a functionality that other models such as the COAP fail at. Compared to the HTTP protocol, MQTT has superior features when used on android devices by slowing down energy consumption. Currently, the large-scale deployment and adoption of IoT have increased privacy and security challenges. IoT remains vulnerable to attacks because: a) being that communication is wireless, the system faces increased risks of attacks such as message tampering, message eavesdropping and identity spoofing. b) most devices will often have access to limited resources that will prevent them from effecting advanced security solutions [9]. These resources include processing capacity, memory and energy. Another key challenge faced by IoT is the centralization of efficient security solutions such as Public Key Infrastructure

(PKI), which leads to scalability issue owing that there are thousands of nodes connected in such an environment. Finally, IoT is also plagued by integrating new scenarios or services since each component relies on a different type of architecture, deployment and security approach. This necessitates proposals for new security solutions that will work on the whole system. It is therefore, the system can a) allow the integration of other devices, b) does not rely on the case design or architecture of component and c) fully adapted to the need and requirements of IoT.

1.1 Problem definition

IoT related issues such as privacy and security remain the most critical issues that plagued the use of IoT systems. For instance, there are a number of issues and vulnerabilities that arise from different scenarios in both the devices and protocols. An example of a potential source of vulnerability to IoT systems is the use of the Message Queueing Telemetry Transport (MQTT), which only comes with a weak security mechanism. The authentication system uses a connect message to transmit the username and password in plain text. The choice of the security approach is left to the application designer. The Transport Layer Security (TLS) [10] is not mandatory while most real-life applications will rely on it, which increases the likelihood of bad or incomplete implementations of security approaches and protocols. This creates the need for new approaches to enhance the security of MQTT.

1.2 Our contribution

Our main contributions in the paper can be summarised as follows.

•We introduced a conceptual model of a lightweight weight approach, which embodies a methodology to achieve authentication of remote IoT devices without relying on any third party.

•We proposed a blockchain-based authentication and authorisation mechanism for the MQTT protocol.

•We developed a decentralised identity system to manage the users' identity.

•We developed a smart contract in the Ethereum blockchain to facilitate the publisher and subscriber access to the MQTT broker without the need for a centralised trust.

•We provided a proof-of-concept implementation to verify the feasibility of our solution.

The remainder of this paper is organised as follows. The related work is discussed in section 2. In section 3, we provided an overview background on the blockchain technology and the MQTT messaging protocol. The proposed approach is presented in section 4. The whole authentication process is described in section 5. In section 6, we described the implementation of our mechanism. To evaluate our work, we provided an intensive performance and security analysis in section 7. Finally, we concluded the paper and provided a suggestion for future work.

2 RELATED WORK

The security of the Internet of Things has received significant interest from the scientific society. A significant of current researches are focused on the authentication and authorisation in the IoT messaging and communication protocols. An example of such researches, the works presented in [11] [12] [13] [14] [15] which provided authentication and authorisation solutions for the MQTT messaging protocol. However, the main issue with these approaches is centralisation. The problem with the centralised authentication approaches is that it requires to store authentication data on a centralised local server, which is prone to a single point of failure. Similar to our approach, the work presented in [16] has a similar focus as it is presenting a blockchain-based OTP authentication approach for resource-constrained IoT devices. The implementation is based on MQTT protocol. The Ethereum blockchain is being used to provide an independent channel to manage the second-factor authentication through the use of a smart contract. Similarly, the authors in [17] proposed a blockchain-based OTP authentication scheme for the MQTT messaging protocol. The proposed approach utilises Ethereum blockchain to provide an out of band channel for implementing the second-factor authentication. Our approach advantage in comparison with other approaches is that it provides an integrated solution. In addition to authentication, our smart contract is also responsible for managing the user's policies and the authorisation. Therefore, our mechanism works as access control to prevent unauthorised access to the IoT systems. Besides, storing and sharing a secret between entities remains a challenging task as it requires a secure channel before it can be proven via blockchain. Therefore, the random challenge generation process in our approach is also based on the smart contract. The smart contract uses the blockchain-based random values to act as the source of entropy, facilitating the random challenge generation. This will help in reducing the risk associated with storing, protecting and sharing a secret between a verifier and a prover.

3 OVERVIEW OF THE BLOCKCHAIN AND MQTT

3.1 The MQTT Protocol

Most IoT devices will often depend on the MQTT protocol to exchange data. MQTT, which is now registered under the standard ISO/IEC 20922:2016 is specifically created for IoT and requires low energy, offers real-time response and less bandwidth [8]. MQTT relies on TCP/IP, Bluetooth or UDP, thereby a lightweight protocol that requires constant connection and minimizes messages. The use of MQTT offers various benefits such as low energy use and can be used across all smartphones. The MQTT session is divided into the following four stages:

- a. Connection
- b. Authentication
- c. Communication
- d. Termination

3.2 The Publish-Subscribe Messaging Model

A publish-subscribe communication model is made up of three main components. These are the subscriber, the publisher and the broker, as shown in Figure 1 below. The broker receives messages from the publishers, which are in the form of topics. The topic in this case is the metadata that has the information about the data in string format. Under the MQTT protocol, the process starts by the creation of the TCP/IP connection by the client to the broker. The client then moves to the authentication phase.

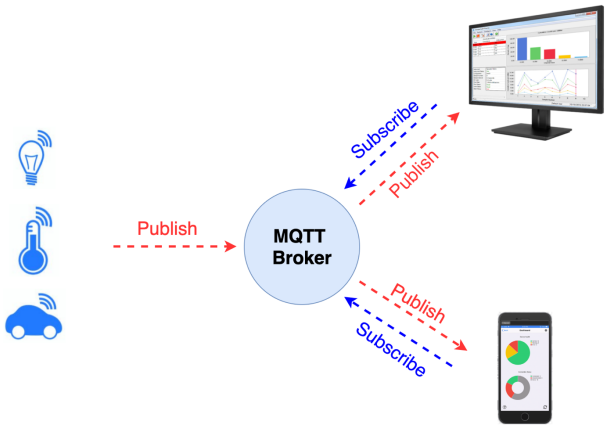

Figure 1: MQTT messaging protocol

3.3 BLOCKCHAIN

Blockchain was first introduced as an open-source project aimed at providing secure financial transactions developed by Satoshi Nakamoto in the year 2008 [18]. This technology is considered as the main drive behind the innovation of the Bitcoin cryptocurrency. Blockchain made it possible to store linked lists of blocks, which allows for the appending of new blocks while preventing their deletion. Additionally, blockchain uses a consensus protocol to validate the number of transactions in each block and facilitate the peer to peer moving and sharing of the cryptocurrencies. The list cannot be changed since each block's completion is done in a well-formed manner and that no possible deletion is possible. The consensus protocol facilitates the agreement between the nodes. The nodes represent the miners that work to expand the block and calculates and verifies the proof of work. The blockchain framework relies on public-key cryptography to help with the validation process and the creation of new keys. All members in the blockchain network will receive a public and private key for them to be allowed to participate in the consensus and the block creation process [18].

3.4 Smart contract

Nick Szabo first introduced this term in 1994 [19] before the use of the decentralised ledger to host smart contracts. This was introduced as a theoretical concept of computer software to enforce and verify the negotiation of a contract digitally. In the case of Bitcoin, smart contracts have been used to transfer values from one user to another when certain conditions are met, but this limited to digital currency only. However, in Ethereum, the smart contract is considered as an object that has its own account and address. This account can communicate with other contracts, send digital currency to another account or even store data. Ethereum replaced the Bitcoin's complicated language with a Turing complete language, which called solidity to allow developers to program their own smart contract.

3.5 Decentralised identity

User identities are determined using the identity management methods. Most current systems have their user's identity controlled and maintained by third-party applications or protocols such as single-point services or identity providers. However, the main issue with the secure management of data lies in trusting people or institutions contracted to ensure protection. In this case, the identity is owned by the mentioned providers rather than the rightful owners. As blockchain technology grows the use of self-sovereign identities (SSI) [20] has increased, which utilised the benefits associated with decentralisation. The use of SSI allows users to control and own their personal identities and other benefits such as decentralised control and privacy. For SSI to be realized, there is a need for the implementation of two key standards, such as Verifiable Credentials (VCs) [21] and Decentralized Identifiers (DIDs) [22]. VCs are instrumental in facilitating the authenticated attribute disclosure and the privacy-aware while DIDs focus on cryptographic identification.

4 THE PROPOSED SOLUTION

To resolve significant privacy concerns and remove the need for remembering passwords and prevent weak passwords from being used for authentication, we proposed using blockchain technology to facilitate decentralised authentication and authorisation in the MQTT messaging protocol. We will rely on Ethereum blockchain, which allows us to use smart contracts. We rely on smart contracts to implement on-chain access control decisions and other policies. The smart contract issues the tokens used by the subscribers and publishers to connect to the MQTT broker. The users can interact with the smart contract by issuing transactions signed by their private key. The hash of the used key is taken to be the user's address and will be used to associate the user to their access token. Our proposed approach will reduce the entropy required and storage requirements. As the smart contract will generate access tokens and manage the authentication process; therefore, there is no need for users to store their tokens locally. The access token will serve as a one-time password (OTP). A simple lookup in the distributed ledger will help verify the authenticity and integrity of all the tokens. Our solution further looks into the provision of decentralised identity management services such as secure and fair exchange, revocation and verifiable credentials by using the power of smart contracts. During the programs preliminary phase, users are prompted to register their remote devices, assign users to specific topics, set policy for the users and get back a verifiable claim. The smart contract helps in assigning the users' policy, after which we implement a decentralised identity model, which will be tasked with managing the users' identities. The users will have the ability to control their identities. This will be made possible through the use of self-sovereign identity (SSI). For this, we will utilise Uport decentralised identity functionality, which will help us to securely share and communicate information between the user's mobile application to our application. With our approach, we no longer need a central trusted authority. The application allows user to store their information straight on their mobile devices, which can be accessed during the validation phase, which makes use of our decentralised web application to retrieves users' credentials and request signing from the user.

5 THE SYSTEM MODEL

Before we can proceed to the authentication procedure and formalise the required concepts, it is important that we extract our solution's model. Our system established a link between the blockchain system and the MQTT components (broker, publisher and subscriber). Our system has three main entities: the Blockchain, MQTT system, and the decentralised identity system.

5.1 The blockchain

Blockchain in our model is used to store information in a distributed manner while maintaining consistency. For this, we used Ethereum blockchain, which allows smart contracts to be deployed in their blockchain system. The smart contract is used in our approach to interfacing with data stored on the blockchain. We developed our smart contract to set the users' policies and register the client's remote devices. We will also use smart contracts in our approach to storing a trusted mapping between the authorised access's public key and its access token. The smart contract is responsible for any operations involved with the authorisation and authentication process such as the whitelisting of all addresses that authorised the access token, generating the users' tokens, and retrieving access tokens issued for the authorised users. The smart contract will also be responsible for authenticating users by receiving signed challenges from users. It works similar to the PKI, as the client identity denotes the public part of the asymmetric key pair. This will guarantee that anyone has access to the blockchain can verify the authenticity of the message signed by the owner of the keypair.

5.2 Subscriber/Publisher

Our system's subscribers and publishers will be the Ethereum clients, all who have a public and private key that facilitates transactions with the Ethereum blockchain. When looking to distinguish subscribers and publisher authentication in MQTT, our system's subscribers represent clients who need to connect to the broker and subscribe to a specific topic to receive data published from the publishers on that topic. In contrast, Publishers are the resource-constrained IoT devices, mainly the sensors which need to authenticate to connect to the MQTT broker to publish their sensor reading in a specific topic. Anytime a publisher or subscriber initiates a connection with the broker, they will receive a challenge back from the smart contract. This will work as the first step towards security as the only authorised addresses can receive a challenge to sign it. The subscribers and publishers rely on the private key to sign a transaction and send the challenge back to the smart contract, enabling them to interact with the smart contract and cryptographically prove their identity. The challenge will serve as a one-time password to access to the broker.

5.3 The broker

The MQTT broker is similar to publishers and subscribers as all are on the Ethereum blockchain and having a public and private key. When the MQTT broker receives a connect request from a client, it will first extract the client ID, which is the public part of the client key pair. Then the broker will send the client's public key to the smart contract. The smart contract will verify the user's permissions in the blockchain. If it is allowed access, the smart contract will generate a challenge and assigned it to the client. Then the client signs the challenge using the private key and authenticates to the smart contract. This challenge will serve as a one-time password to access to the broker. The broker will authenticate the client through the smart contract and verify the procedure's correctness. The broker is no longer needed to reside in a specific physical location when using our MQTT-based architecture. That is, the broker can reside on the cloud or run on a specific host. A general overview of our proposed approach is illustrated in Figure 2 below.

1) The user first registers remote devices and set users' policies. 2) Publisher or Subscriber send a connect request to our MQTT broker using their public keys as a client ID. 3) The MQTT broker signs a transaction and sends the client ID to the smart contract to be authenticated. 4) The smart contract verifies the user's permissions and generates a challenge to be mapped to the client public key. 5) Publisher/Subscriber signs the challenge using their private key. 6) The broker verifies the procedure's correctness and connect the client.

5.4 The decentralised identity model

This section presents our decentralised identity model. For this, we proposed a self-sovereign identity model to preserve the privacy of our users. Therefore, giving them more control over their personal data rather than have it managed and stored by a third party. We utilised Uport identity [23] to help with storing the identity data on the user's mobile wallet, which allows for the enhancing of MQTTs security. Our system recognises users as real people with the flexibility to express themselves fully when interacting with the smart contract compared to having them as abstract hex-encoded addresses that interact with each other.

5.4.1 Registration. A new user needs to download the Uport identity mobile app. The users will then have public/private keys, which will be stored on their mobile devices. The user's private key will be stored securely in the client's mobile device and used to sign transactions from the client's account. First, the owner is supposed to associate his/her user's identity with the Ethereum public key. For this to happen, the resource owner has to access our decentralised web application. The users need to register to our decentralised web application by clicking "Register Using Uport" on our website. The application will then redirect the user to a new QR code page. The user needs to scan the QR code using his/har Uport mobile app in order to allow the web application to receive the user's credentials from the user's mobile wallet. Then the user will be redirected to a form that needs to be completed by the resource owner. The user requires an IoT device that can communicate with the broker and the blockchain to help authenticate the owned remote IoT device. The user needs to submit the required information, such as the remote device ID (public address), the user's roles and topic name. The web application will proceed and calls functions on the smart contract that set the users' policies and add the remote IoT devices to the user's web of trust. After that, the user will receive a verifiable claim in their Uport mobile app. Each claim has to have a corresponding security token, which helps with requesting the proof of the claim when a client needs to login to the web application. The registration process is shown in Figure 3

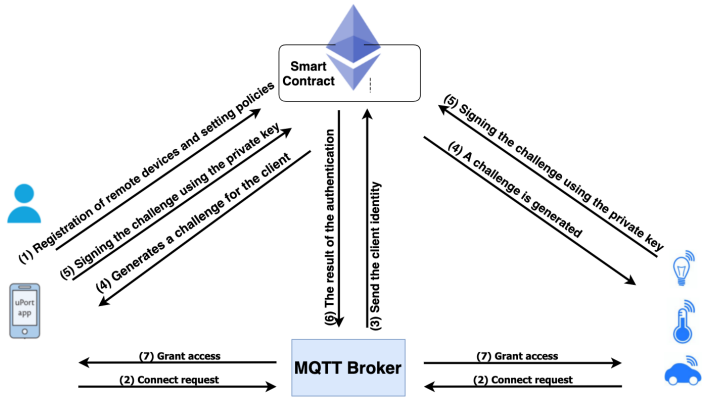

Figure 2: The proposed solution for authentication

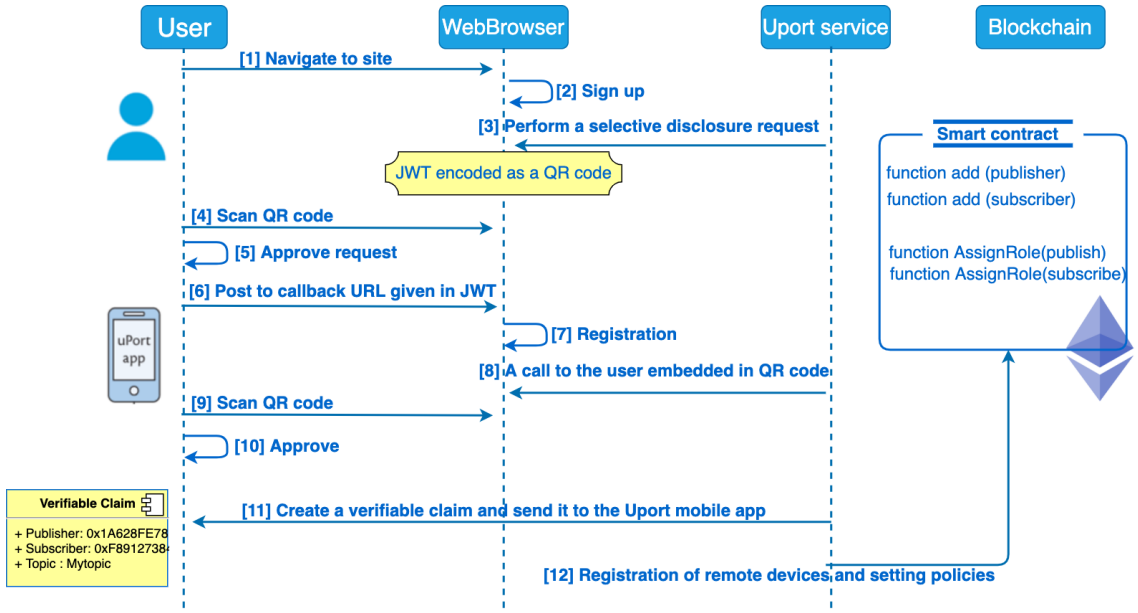

Figure 3: The registration model

5.4.2 Login to the user's dashboard. The subscriber needs to login to the user's dashboard and gets authenticated through the smart contract to access the broker. When a subscriber requests access to the broker from the web interface, it first needs to click "Login with Uport" button on our website. The web application will begin by requesting a verifiable claim from the client before it requests access to the MQTT broker. The verifiable claim will be requested via a QR code, which needs to be scanned by the user's Uport mobile app. The web application will extract the client ID, topic name and the action requested (publish or subscribe) from the verifiable claim. The application will then send a connect request to the broker to request access and pass this information to the broker. The broker will then start the first step toward authentication by querying the smart contract to verify the user's permissions. This done by sending the client ID, a topic and the requested action (subscribe/publish) to the smart contract. The smart contract will verify the user's policies. If it is permitted, the smart contract will generate a challenge. The users can then use the Uport app on their mobile to sign the challenge using their private key. The user will get notified of the request on their Uport app and will be asked to either deny or approve. The approval needs to be confirmed using the user's fingerprint or pin code of their mobile, allowing users to present themself as real people. The final step in the authentication involves retrieving the verification result by the broker from the smart contract to establish an authorised connection with the client. The user's login and verification process is presented in Figure 4 below.

6 IMPLEMENTATION

This section describes the implementation of our system and explaining the technologies that were used. For the purposes of the implementation of our framework, we will rely on MQTT with

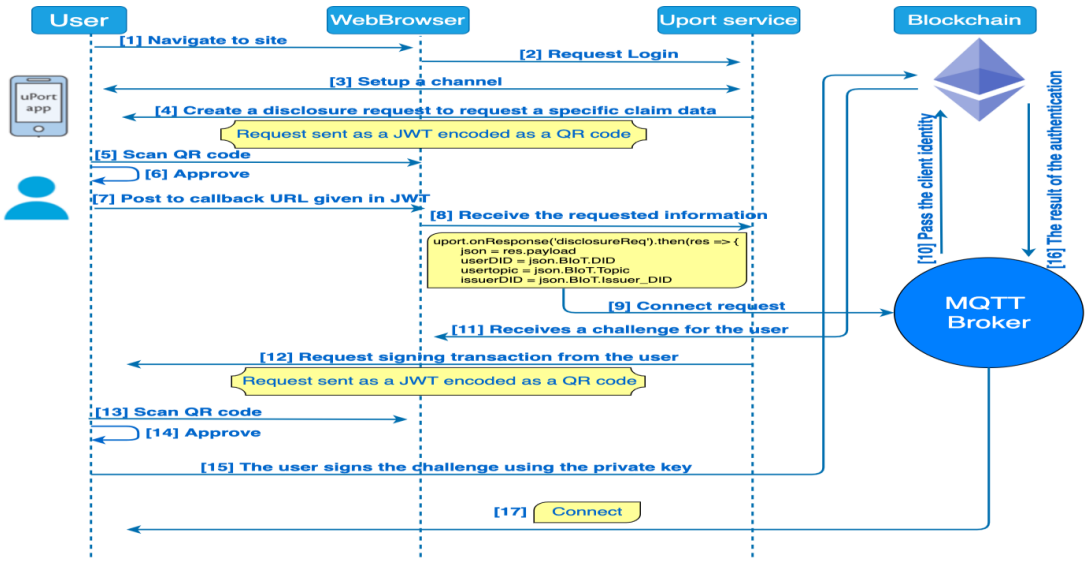

Figure 4: The user's login and verification

the Ethereum blockchain framework. Our prototype contains three main components, the decentralised web application, a smart contract for interfacing with data stored on the blockchain, and the MQTT application, which contains the broker, subscriber and publisher. We relied on Ethereum blockchain, the most prominent cryptocurrency measured by the market capitalisation. The primary impulse behind our choice of Ethereum blockchain is the amount of support that is provided due to its popularity and the ability to deploy smart contracts to their network. We implemented our smart contract in Solidity, the Turing complete language that designed to develop smart contracts in the Ethereum blockchain system. To write, evaluate and deploy the smart contract in Ethereum network, we utilised the Ethereum web browser-based IDE Remix [24]. The role of the contracts in our implementation is similar to the ones of self-signed certificates. It implements any function that carries out operations in the authorisation and authentication process and generates access tokens for authorised users.

To implement the MQTT broker, we adapted the open-source Mosca MQTT broker [25]. The MQTT broker helps with data exchanging between the publishers and subscribers, or between devices and the web and mobile applications. We also built a publish/subscribe client application in JavaScript, which uses MQTT library called PAHO [26]. We relied on the JavaScript API to bridge the block chain's framework to the MQTT application. We then considered the need for a user dashboard to allow the subscribers to connect to the broker from the web interface. It additionally helps subscribers with reading the data detailed regarding specific topics. Our web application consists of various types of resources. These include the HTML templates, JavaScript files, CSS files and server-side implementation code. CSS and images files are used as static resources and particularly influence the display of our application. We implemented our web-server to host the website on ubuntu using apache server. The web-server is built to demonstrate how our decentralised web-based solution can allow users to communicate with blockchain to set users' policies and managing their identity to subscribe to the broker securely. The user interface communicated to the MQTT broker through a Web Socket. Therefore, the exchange of data will occur in real-time. Running the MQTT over a WebSocket to allow implementing MQTT in the user interface. To allow the communication between our application and the Ethereum blockchain, we have relied on the web3.js Ethereum JavaScript API, to interact with an Ethereum node run on Infura. We will then use Ethereum-based Uport identity mobile application as an Ethereum user's wallet. We have provided an open-source code implementation of our project in GitHub [27]. The project's root directory contains three main components. The first folder is the web app folder, which stores the source code for the front end and the back end implementation of the web application, including the registration web page and the login web page. Secondly, the smart contract folder, which hosts the solidity programming code for our implemented smart contract. Finally, the MQTT folder, which contains the Broker and publishers' JavaScript implementation, which are based on the open-source Mosca MQTT broker.

7 EVALUATION

7.1 Performance analysis

To evaluate our system's performance, we have implemented the MQTT broker and clients based on our proposed scheme. We compare our approach with the build-in authentication of the MQTT and the authentication over a secure TLS channel. Compared with TLS, our approach's memory utilisation is around 200 MB less than the TLS, as shown in Figure 5. This is a significant factor for efficient IoT applications. From our evaluation, we observed that the current TLS, which is being used widely to secure authentication on the MQTT, is consumed a higher RAM than our approach because it requires to allocate additional buffers. TLS will also cause more overhead for each MQTT message sent. However, the overhead is

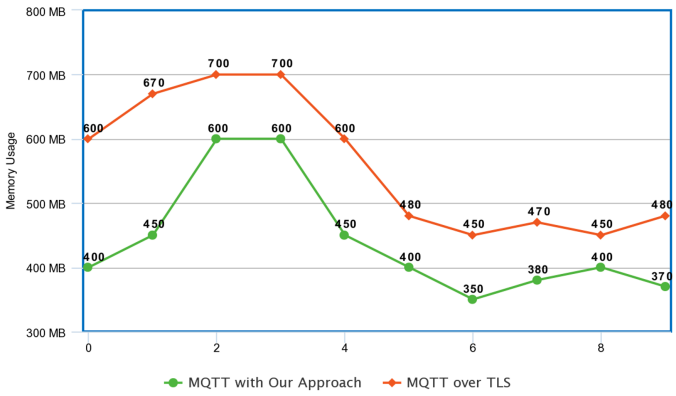

Figure 5: Memory Utilisation

varied at runtime depending on the cypher suite used for the TLS connection.

Moreover, we observed that the computational overhead of the TLS scales up to 81% of the CPU, while the CPU overhead of our approach is around 24%. Our approach is around 57% less in CPU overhead compared with the TLS. Our analyses unveiled that the initialisation of the authentication is the heaviest operation, as shown in Figure 6 below. TLS handshake has shown significant CPU overhead when the clients connecting to the MQTT broker, particularly with a certificate that uses a large key length. It has also shown a high drop in processor usage after the publishers and subscribers have connected.

7.2 Transactions cost

In this section, we are going to evaluate the gas cost of each event that happens in the system. We used the Rinkeby Ethereum test network for testing our proposed system. For reliable results, we will rely on using a public test network compared to a private one. Our result showed that the highest cost is the cost associated with the deployment of the smart contract to Ethereum network. However, this action will only be performed once when setting up the system for the first time. The cost of deploying our smart contract was around 0.000101ETH Ethers, which corresponding to $0.03732657 using the average quotation of $369.57 per Ether on October 17, 2020 [28]. In contrast, the cost of calling a function in the smart contract was around 0.000041ETH, which equates to $0.01515237 per transaction. Nonetheless, this cost is variant, as it depends on the time expected for sending the transaction and the storage and computational resources required. However, it is necessary to stress that our implementation is not based on the main Public Ethereum network because of the financial costs included. Instead, we built our implementation based on Rinkeby test net, which provides a blockchain testing environment with similar characteristics to the Ethereum's main public network, and without financial cost as Rinkeby provides a faucet to request free Ethers to this testing network.

7.3 End-to-end delay

We observe that in order for a user to complete the authentication and get authorised access to the broker, two main transactions need to be performed, which call functions in the smart contract to facilitate the authentication process. The first function is the acessToken (), which is called from the broker to authenticate the user through the smart contract. The problem with this function is that the values that need it to perform as a source of entropy to facilitate the generation of the random challenge are depending on the block mining process because it needs values such as block's timestamp and the block difficulty, which will not be available until the block is mined. Therefore, for a user to get authenticated via a first factor and receive a challenge to be signed by the user, it needs around 13-15 second. Furthermore, it will also need another 13-15 seconds to sign a challenge to be sent back to the smart contract. Besides, other functions are defined as view functions, in that they incur no CPU overhead, delay or cost as they only read the state of the blockchain without doing any forms of modification, such as the getClientToken(), which is used by the broker to retrieve the token that submitted by the user. However, the issue of blocks being mined too slowly will not be a big problem for two-factors authentications and similar approaches. In comparison with real-world applications that depend on a third-party for implementing two-factor authentication, it has also shown a considerable delay.

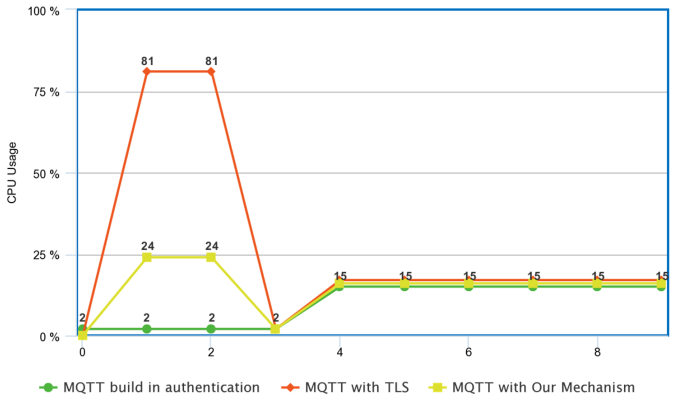

Figure 6: CPU Utilisation

For instance, emails and SMS experiencing a considerable delay at the busy networks. Therefore, this would only be a problem for applications that need argent access to the public ledger. Moreover, our approach can be significantly improved by implementing it in a private permissioned blockchain. In addition to the private chain, replacing the PoW consensus mechanism with another less computational mechanism such as PoS or PoA will significantly reduce the time needed to mine the block and would also improve the transactions speed.

7.4 Security analysis

7.4.1 Compromised Website. A common threat that will affect any website is the attackers who have gained access to a website where a user has an account. Therefore, attackers might gain access to the user's password and other accounts that use the same password. In our system, we eliminated using a user password to gain access to the user's dashboard and authenticate to the MQTT broker. Instead, we rely on verifiable credentials, which refer to the assigned DID. Our approach assures a high level of confidentiality and integrity. It is backed with the issuer's DID and its cryptographic proof to secure data transfer between users and our web application.

7.4.2 Cryptographic attack on the user's keypair. The cryptographic schema used by Ethereum is being widely accepted within the cryptocurrency systems, which based on the KECCAK-256 hash function. The probabilities of guessing a randomly produced Ethereum private key are 1/(2^256), equal to 1 in 115 quattuorvigintillions [29]. This is approximately around the number of atoms in the universe. We sustain the same brute-force resistance similar to other Ethereum keypair. Therefore, if this a potential attack in our users' keypairs, it would can then be possible in any other Ethereum wallet.

7.4.3 Physical access to a user's device. Our solution will guarantee strong security against an attacker who gained physical access to a user's device, as Uport client application on the user's smartphone stores the user's private key securely in an encrypted file. The application decrypts the user's private key once the user has provided their fingerprint or device pin code. However, in a situation where the attacker possesses the user's mobile device and their pin or fingerprint, the attacker will have total control over the identity linked to it. The attacker can then add new subscribers and publishers to the user net of trust. The attacker will also be able to sign a transaction from the victim account to the smart contract and get authorised access to the MQTT broker. To overcome the risks of such an attack, we proposed the use of the recovery mechanism provided by Uport identity. Uport addresses problems such as this with a delegation mechanism, which built into special contracts called the controller contract and the proxy contract. This allows Uport users to recover their identities in the event of such an attack.

7.4.4 Attacker on the network. Our system relies on Ethereum public test network, it therefore, transactions will be available for all nodes involved in the blockchain system and other users who perform a simple lookup in the blockchain. Therefore, more attention needs to be considered to ensure that attackers do not compromise the system by authenticating themself to the MQTT broker using an existing token. Our system remains resistant against such attacks as our smart contract ensures that a unique challenge is produced and mapped to a single user. We further require that a user sign this challenge using his/her private key in order to complete the process and grant access to the system.

8 CONCLUSIONS AND FUTURE WORK

This paper has presented a proof-of-concept design and implementation of a blockchain-based authentication and authorisation schema. The provided solution relied on Ethereum blockchain. Our solution further provided a decentralised identity model that allows users to have full control over their identity and data without the need for centralised authority to manage identity. The paper described the implementation of our approach and the specific technologies that were needed for the implementation. We have also provided analysis on the performance together with the associated costs that our system uses. We observe that our approach will provide a lightweight approach to facilitate authentication of MQTT protocol in a distributed and secure way. However, our approach has shown a high delay since the transactions require to be added to the blockchain but still within an acceptable range compared with similar two-factor authentication approaches that use a third party to maintain authentication like SMS and Emails. On the contrary, our approach performs much better in the computation and storage overhead as it has shown a negligible CPU and memory usage compared with the TLS, which allows resource-constrained IoT devices to work in parallel with other applications without suffering from the overloading problem. It is feasible that our approach satisfies the security requirements for IoT applications and meet future demands. Overall, we hope our design provides advantages in the area of user authentication compared to current alternatives. Our future work intends to expand the smart contract's authentication functionality to handle an extensive range of policies and conditions. Besides, we will implement our solution using different blockchain systems such as Hyperledger and IOTA, and we will evaluate various consensus mechanism to improve the performance and end-to-end delay.

REFERENCES

[1] Peña-López, I. (2005). ITU Internet report 2005: the internet of things. https://www.itu.int/osg/spu/publications/internetofthings/.

[2] Tawalbeh, L. A., Muheidat, F., Tawalbeh, M., & Quwaider, M. (2020). IoT Privacy and security: Challenges and solutions. Applied Sciences, 10(12), 4102.

[3] Yokotani, T., & Sasaki, Y. (2016, September). Comparison with HTTP and MQTT on required network resources for IoT. In 2016 international conference on control, electronics, renewable energy and communications (ICCEREC) (pp. 1-6). IEEE.

[4] Shelby, Z., Hartke, K., & Bormann, C. (2014). The constrained application protocol (CoAP). https://tools.ietf.org/html/rfc7252.

[5] Jaikar, S. P., & Iyer, K. R. (2018). A survey of messaging protocols for IOT systems. International Journal of Advanced in Management, Technology and Engineering Sciences, 8(II), 510-514.

[6] Gay, D., Levis, P., Von Behren, R., Welsh, M., Brewer, E., & Culler, D. (2003). The nesC language: A holistic approach to networked embedded systems. Acm Sigplan Notices, 38(5), 1-11.

[7] Hughes, D., Thoelen, K., Horré, W., Matthys, N., Cid, J. D., Michiels, S., ... & Joosen, W. (2009, December). LooCI: a loosely-coupled component infrastructure for networked embedded systems. In Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia (pp. 195-203).

[8] Standard, O. A. S. I. S. (2014). MQTT version 3.1. 1. *URL* http://docs.oasis-open.org/mqtt/mqtt/v3, *1*.

[9] Samaila, M. G., Neto, M., Fernandes, D. A., Freire, M. M., & Inácio, P. R. (2018). Challenges of securing Internet of Things devices: A survey. *Security and Privacy*, *1*(2), e20.

[10] Independent, E. R. T. D. (2008). *The transport layer security (tls) protocol version 1.2.* Technical report, RTFM Inc.," https://tools. ietf. org/html/rfc5246.
[11] Patel, C., & Doshi, N. (2020). A Novel MQTT Security framework In Generic IoT Model. Procedia Computer Science, 171, 1399-1408.
[12] Pahlevi, R. R., Sukarno, P., & Erfianto, B. (2019). Implementation of Event-Based Dynamic Authentication on MQTT Protocol. Jurnal Rekayasa Elektrika, 15(2).
[13] Lohachab, A. (2019). ECC based inter-device authentication and authorization scheme using MQTT for IoT networks. Journal of Information Security and Applications, 46, 1-12.
[14] Bali, R. S., Jaafar, F., & Zavarasky, P. (2019, January). Lightweight authentication for MQTT to improve the security of IoT communication. In Proceedings of the 3rd International Conference on Cryptography, Security and Privacy (pp. 6-12).
[15] Cruz-Piris, L., Rivera, D., Marsa-Maestre, I., De La Hoz, E., & Velasco, J. R. (2018). Access control mechanism for IoT environments based on modelling communication procedures as resources. Sensors, 18(3), 917.
[16] Buccafurri, F., & Romolo, C. (2019, September). A Blockchain-Based OTP-Authentication Scheme for Constrainded IoT Devices Using MQTT. In Proceedings of the 2019 3rd International Symposium on Computer Science and Intelligent Control (pp. 1-5).
[17] Buccafurri, F., De Angelis, V., & Nardone, R. (2020). Securing MQTT by Blockchain-Based OTP Authentication. Sensors, 20(7), 2002.
[18] Nakamoto, S., & Bitcoin, A. (2008). A peer-to-peer electronic cash system. Bitcoin.–URL: https://bitcoin. org/bitcoin. pdf, 4.
[19] Szabo, N. (1994). Smart contracts. Unpublished manuscript. https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html
[20] Tobin, A., & Reed, D. (2016). The inevitable rise of self-sovereign identity. The Sovrin Foundation, 29(2016).
[21] Sporny, M., Longley, D., Burnett, D., & Kellogg, G. (2018). Verifiable Credentials Data Model. Retrieved July, 26, 2018.
[22] Reed, D., Sporny, M., Longley, D., Allen, C., Grant, R., Sabadello, M., & Holt, J. (2020). Decentralized identifiers (dids) v1. 0: Core architecture, data model, and representations. World Wide Web Consortium, Cambridge, MA.
[23] Lundkvist, C., Heck, R., Torstensson, J., Mitton, Z., & Sena, M. (2017). Uport: A platform for self-sovereign identity. URL: https://whitepaper. uport. me/uPort_whitepaper_DRAFT20170221. pdf.
[24] Remix-Solidity, I. D. E. (2018). Remix. ethereum. Org
[25] Collina, M. (2013). Mosca : the MQTT server for node.js that can be backed up by AMQP, Redis, ZeroMQ or just MQTT.
[26] Paho, E. Eclipse Paho-MQTT and MQTT-SN software. Saatavissa (viitattu 26.10. 2018): http://www. eclipse. org/paho.
[27] BC-of-Every-Thing. 2021. A Blockchain-based Authentication and authorisation mechanism for MQTT. https://github.com/BC-of-Every-Thing/DecentralisedMQTT.
[28] Cap, C. M. (2020). All cryptocurrencies. Retrieved from Coin Market Cap: https://coinmarketcap. com/all/views/all.
[29] Help Net Security. April 24, 2019. Research on private key generation reveals theft of ETH funds from accounts with discoverable keys. URL:https://www.helpnetsecurity.com/2019/04/24/ethercombing/