MQTT PROTOCOL DATA SECURITY WITH OTP BLOCKCHAIN-BASED IDENTITY AND DATA VERIFICATION

BATUHAN PARLAKAY

MEF UNIVERSITY
AUGUST 2023

MEF UNIVERSITY

GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

MASTER'S IN MECHATRONICS AND ROBOTICS ENGINEERING

M.Sc THESIS

MQTT PROTOCOL DATA SECURITY WITH OTP BLOCKCHAIN-BASED IDENTITY AND DATA VERIFICATION

Batuhan PARLAKAY

ORCID NO: 0009-0004-6186-3587

Thesis Advisor: Asst. Prof. Dr. Tuba AYHAN

AUGUST 2023

ACADEMIC HONESTY PLEDGE

This is to certify that I have read the graduation project and it has been judged to be successful, in scope and in quality and is acceptable as a graduation project Master's Degree in Mechatronics and Robotics Engineering.

Name Surname: Batuhan PARLAKAY

Signature:

ABSTRACT

MQTT PROTOCOL DATA SECURITY WITH OTP BLOCKCHAIN-BASED IDENTITY AND DATA VERIFICATION

Batuhan PARLAKAY

M.Sc. in Mechatronics and Robotics Engineering

Thesis Advisor: Asst. Prof. Dr. Tuba AYHAN

August 2023, 133 Pages

The widespread Internet of Thing presence in almost every aspect of our lives has been made possible by the fast development of technology these past few years. The internet of things is in a wide area. For example cell phones, tablets, computers and all other devices with sensors. Among the technologies used to facilitate efficient communication, between these IoT devices the MQTT protocol stands out. Exposure of security vulnerabilities existing in MQTT, and the development of effective countermeasures is a key objective of this thesis.

The MQTT server's client architecture was built between the Raspberry Pi and the computer. To be used by MQTT Broker and publisher subscribers, Python programs have been developed. The use of the wireshark API has been recommended to check for system security vulnerabilities. During that process, the safety issues at packet and module level have been examined in an experimental manner. The MQTT protocol has been found to be vulnerable to attacks.

Although encryption can be performed on port 8388 with Secure Sockets (SSL) and Transport Layer Security (TLS) protocols to address the security vulnerabilities found in the standard MQTT configuration, this is not preferred and is not scalable. Instead, a structure has been developed on port 1883 again, using smart contracts, digital signatures to only allow authorized users to connect to the MQTT broker, providing authentication and encryption for the publisher and subscriber. Extra security measures are offered with authentication, message denial, data integrity, and selective privacy.

In the area of Smart Contracts major progress has been made. A smart contract, ensuring transparency and traceability in every transaction offering benefits, plays an important role. Smart contracts consist of an automated set of instructions which, when certain conditions have been fulfilled, shall be executed automaticly. Users' permissions as well as Digital Signatures could be included in these conditions. Better scalability is also provided by this system.

Using A system has been developed using Smart Contract technology to perform user authentication and permission management for users connected to an MQTT broker. Authorization processes such as adding, removing, granting, or denying user permissions can be executed through a smart contract. Similarly, a user seeking to access and publish or read data on the MQTT broker must not only possess the necessary authorization but also approve a digitally signed message based on their unique OTP (One-Time Password) information. The Elliptic Curve digital signature algorithm is used for this signature.

Users with access permissions can verify the digital signature using their public key and OTP information. Once the smart contract confirms the user's permission to publish data, they can proceed with their publication or perform encrypted data readings This design is intended to stop entry and manipulation of data, within the system.

In contrast with the data security offered by SSL or TLS, this new and effective method provides additional protection against attacks on Data Centres such as potential Distributed Denial of Service attack from Sybil.

Upon completion of this thesis, it was determined that the Broker gained immunity against attacks due to the implemented security measures.Consequently the thesis offers an examination of MQTT in relation, to attacks and suggests an enhanced security mechanism to counteract these attacks.

Keywords : IoT, MQTT, OTP, Blockchain, Smart Contract, HbMQTT, ECDSA, Digital Signature

Numeric Code of the Field : 92905

ÖZET

MQTT PROTOKOLÜ VERİ GÜVENLİĞİNİN OTP BLOKZİNCİR TABANLI KİMLİK VE VERİ DOĞRULAMA İLE SAĞLANMASI

Batuhan PARLAKAY

Mekatronik ve Robotik Mühendisliği Tezli Yüksek Lisans Programı

Tez Danışmanı: Dr. Öğr. Üyesi Tuba AYHAN

Ağustos 2023, 133 Sayfa

Son yıllarda teknolojinin hızla gelişmesiyle birlikte, Nesnelerin İnterneti (IoT) adını verdiğimiz akıllı cihazlar ve sistemler hayatımızın her alanında kendine yer bulmaktadır. IoT, akıllı telefonlar, tabletler, PC'ler ve üzerinde sensör bulunan neredeyse her şeyi kapsayan geniş bir kavramdır. Bu kapsamda, IoT cihazları arasında verimli ve güvenli iletişim sağlamak amacıyla MQTT protokolü önemli bir role sahiptir. Bu tezde, MQTT protokolünün güvenlik zafiyetlerini tespit etmek ve bu zafiyetlere karşı etkili önlemler geliştirmek hedeflenmektedir.

Tez kapsamında, Raspberry Pi ve bilgisayar arasında MQTT server client mimarisi oluşturulmuştur. MQTT Broker ve yayıncı/abone istemcileri için Python programları geliştirilmiş, sistem üzerinde çeşitli güvenlik açıklarını tespiti sağlanırken Shodan API'sinden ve Wireshark'dan yararlanılmıştır. Bu süreçte, paket ve konu düzeyindeki güvenlik sorunları deneysel olarak incelenmiştir.

Akıllı sözleşmeler, dijital imzalar, OTP kullanarak 1883 numaralı portunda bir yapı geliştirilmiştir. Bu, yayıncı ve abone için kimlik doğrulama ve şifreleme sağlayarak yalnızca yetkili kullanıcıların MQTT brokerine bağlanmasına olanak tanır. Kimlik doğrulama, mesaj reddetme, veri bütünlüğü ve seçici gizlilik gibi ek güvenlik önlemleri sunulmaktadır.

MQTT brokerına erişim, veri yayınlama veya okuma isteyen bir kullanıcının sadece gerekli yetkiye sahip olması yetmez, aynı zamanda benzersiz OTP (Tek Seferlik Şifre) bilgilerine dayalı olarak dijital imzalı bir mesajı onaylaması da gereklidir. Bu imza Elips Kavisli Dijital İmza Algoritması'na dayanır. Erişim izinlerine sahip kullanıcılar, dijital imzayı genel anahtar ve OTP bilgilerini kullanarak doğrulayabilirler.

Dijital imzalar, asimetrik şifreleme tekniklerini kullanarak, iletilen her bir MQTT mesajının bütünlüğünü ve kökenini doğrular. Bu, iletilen verinin değiştirilmediğini ve belirli bir cihaz veya kullanıcıdan geldiğini garantiler.

SSL/TLS, bağlantı bazında çalışır ve tüm bağlantıyı şifreler. Geniş ölçekteki sistemlerde, SSL/TLS sertifikalarının yönetimi ve sürekli şifreleme/şifre çözme işlemleri, özellikle düşük kapasiteli IoT cihazlarında ek işlem yükü oluşturabilir. Akıllı sözleşmeler ve dijital imzaların kullanımı, geniş ölçekteki çok sayıda yayınlayıcı ve abone içeren sistemlerde, SSL/TLS'ye göre daha ölçeklenebilir bir çözüm sunabilir.

Tez çalışması sonucunda, geliştirilen güvenlik önlemleri sayesinde Broker'ın saldırılara karşı bağışık olduğu tespit edilmiştir. Bu tez, MQTT protokolündeki güvenlik tutarsızlıklarının ve alınabilecek önlemlerin özlü bir incelemesini sunarak, alanındaki çalışmalara katkı sağlamayı hedeflemektedir. Bu sayede, IoT sistemlerinde veri iletişiminin daha güvenli ve etkin bir şekilde gerçekleştirilmesine yardımcı olunacaktır.

Anahtar Kelimeler : IoT, MQTT, OTP, Blockchain, Akıllı Sözleşme, HbMQTT, ECDSA, Dijital İmza

Bilim Dalı Sayısal Kodu: 92905

ACKNOWLEDGEMENT

I am incredibly grateful, to my advisor, Associate Professor Tuba AYHAN for the help and support she provided throughout my masters thesis journey. Her guidance, at every step of this research process has been absolutely essential. Greatly appreciated.

I am truly thankful, for the support I have received from my father, Muharrem PARLAKAY, my mother, Nermin PARLAKAY and my brother, Tunahan PARLAKAY. Thank you all much. I am grateful for everything that you've been giving to me in my career as a teacher.

I would also like to thank my colleagues at DACEL, the company where I had my first professional experience.

I sincerely thank everyone mentioned above for your contributions in bringing this master's thesis to completion.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

IOT	**:** Internet Of Things
SCADA	: Supervisory Control and Data Acquisition
OTP	**:** One Time Password
MQTT	**:** Message Queuing Telemetry Transport
PoW	**:** Proof of Work
PoS	**:** Proof of Stake
TLS	: Transport Layer Security
SSL	: Secure Sockets Layer
ECDSA	: Elliptic Curve Digital Signature Algorithm
RSA	: Rivest–Shamir–Adleman
DSA	: Digital Signature Algorithm

INTRODUCTION

The popularity of Internet of Things (IoT) applications has seen a rise, in the year leading to a corresponding increase, in concerns regarding their security. The MQTT protocol, known as Message Queuing Telemetry Transport is a communication protocol utilized extensively in Internet of Things (IoT) systems due, to its nature and energy efficiency. MQTT, however, has a number of serious security vulnerabilities. User authentication is one of the most important processes [1].

The The objective The objective herein is to devise a fortified security and authorization mechanism for the MQTT user authentication process by integrating one- time password (OTP) systems with blockchain technology, smart contracts, digital signature, asymmetric encryption mechanisms.

Particular, it should be considered that username and password information in the MQTT protocol can be easily decrypted using network monitoring tools, such as Wireshark, and manipulated by malicious individuals [2]. To enhance the MQTT user authentication process, the use of smart contracts is suggested. In this method individuals are incorporated into the system by validating their permissions through contracts. If the user has access permission, they can create a username and password as desired. These login details will be utilized to establish a connection via MQTT.

One-time password (OTP) systems can be implemented to further increase the security of MQTT user authentication. OTP systems generate different and temporary passwords for each use, reducing the risk of unauthorized access for users [3]. Furthermore if we combine OTP with signatures it can enhance the security mechanism to a level. In the scope of this thesis, a demo is conducted using a Raspberry Pi, utilizing MQTT's broker, publisher, and subscriber features to read temperature and humidity values connected to the Raspberry Pi on another computer within the same network, thereby establishing a security mechanism.

The developed security mechanism utilizes smart contracts, OTP, and digital signatures to enable the observation of MQTT traffic. Asymmetric encryption has been preferred for data encryption. The use of the Elliptic Curve Digital Signature Algorithm has become prevalent, in signatures. Additional details of the reasons for making such a choice shall be given under the following sections. Access can be controlled, monitored, and limited through smart contracts, OTPs, and digital signatures. Additionally, coupled with data encryption, unauthorized access becomes nearly impossible. Four fundamental strategies have been identified to enhance IoT security: employing blockchain for communication protection, authenticating users, recognizing legitimate IoT devices, and configuring IoT settings.

The fundamentals of the MQTT protocol and existing literature on security vulnerabilities in user authentication is examined first. Subsequently, the integration of smart contracts is discussed in detail, along with how they can be used to add users and verify their permissions.

In the following sections, the inclusion of digital signature and OTP systems in the user authentication process and the role they play in this process is discussed. Various approaches, algorithms, and technologies related to OTP will be examined, and the most suitable method will be selected for use in the application section of the thesis.

The details of the demo conducted with Raspberry Pi and the steps of the integrations carried out is described in the see, providing a guiding resource for readers to implement this security mechanism in their projects.

Solutions to the security problems in the authentication process for MQTT will be presented. In order to increase security, it is aimed to develop a security mechanism based on smart contracts, digital signatures and OTP. The level of security will thus be increased and protection of user data privacy and safety improved.

1. IOT PROTOCOLS

1.1. IoT Session Layer Protocols: A Comparative Analysis

Different communication protocols are used for the internet of things system, and these have been optimized according to individual parameters, in particular taking into account those criteria which include data transmission, energy efficiency or security. A brief introduction of main Internet protocols, including MQTT,AMQP, DDS, CoAP and XMPP, as well as the reasons why it's a good protocol compared to others will be provided here [4].

MQTT (Message Queuing Telemetry Transport): This protocol stands out for its lightweight, energy-efficient, and low bandwidth requirements, making it particularly suitable for low-resource devices. MQTT uses the publisher subscriber model for data transmission to provide low latency and excellent reliability.

Communication may be facilitated by the Advanced Message Queuing Protocol, AMQP. The AMQP is more sophisticated and fluid in its communication capabilities, but requires higher resource consumption as well as the requirement to carry larger numbers of channels.

DDS (Data Distribution Service): Designed for real-time systems, DDS delivers high performance and low latency. Though this protocol is widely used in large scale and distributed systems, due to resource scarcity and bandwidth constraints it may not be appropriate for lowresource Internet of Things systems.

CoAP (Constrained Application Protocol): CoAP is designed for devices with low power and resource constraints. This protocol, like HTTP, is designed to respond in a fast and simplified way. However, due to CoAP operating over UDP, it may not be as successful as MQTT in terms of reliability.

XMPP (Extensible Messaging and Presence Protocol): his protocol provides features such as instant messaging and presence information. XMPP is designed especially for real-time and interactive applications but comes with higher resource consumption and complexity.

MQTT is known for its design minimal bandwidth needs, energy efficiency and quick response time. Therefore, it is advantageous, especially for low- resource IoT systems. While some applications may find the flexibility, security, and complexity features offered by other protocols important, MQTT has become a more common choice in IoT systems thanks to its simplicity and performance. The publisher- subscriber model of MQTT facilitates the management of data exchange and communication. This particular model guarantees that as the number of devices grows the system will also be able to expand and remain easily manageable. Compared to the protocols, this allows communication in a manner that is energy efficient and minimises transmission of data. However, other protocols are not scalable and manageable, providing no flexibility for the development of a security mechanism. And undoubtedly, MQTT is the simplest and most stable among them [5].

2. BLOCKCHAIN & MQTT: CORE CONCEPTS

2.1. The MQTT Protocol

The system shall be set up as a MQTT client server structure, which allows clients to interact with the MQTT broker for sending and receiving messages. The main components of MQTT are the publisher, subscriber, and broker [6]. The MQTT protocol has been standardized by OASIS (Organization for the Advancement of Structured Information Standards), a consortium that develops and publishes a range of data protocols and standards. According to the work of the OASIS MQTT Technical Committee, the publisher is responsible for generating messages and sending them to the broker. The subscriber, on the other hand, listens to specific topics to receive messages from the broker. The broker shall act as a intermediary, manage the communication routes and ensure that messages reach their intended recipients.

The general architecture of the MQTT protocol [7], one of the most important aspects is the topic-based filtering mechanism. Topics are hierarchical and can be organized in a way that reflects the structure of the data being transmitted. For example, a topic hierarchy for a smart home system might be organized as "home/room/device/sensor." When a subscriber wants to receive messages related to a specific topic, it subscribes to that topic by sending a subscription request to the broker. The broker then forwards any relevant messages to the subscriber as they are published.

The quality of service level, which determines the guarantee of the delivery of messages, is another important aspect of MQTT. QoS 0 provides no guarantee of message delivery, while QoS 1 ensures that the message will be delivered at least once, possibly with duplicates. QoS 2 guarantees that the message will be delivered exactly once, with no duplicates. The choice of QoS level depends on the application requirements and the trade-offs between message delivery guarantees, latency, and network overhead.

MQTT also provides a mechanism for maintaining persistent sessions between clients and the broker. When a client connects to the broker, it can specify whether it wants to establish a clean session or a persistent session. In a clean session, all subscriptions and messages are discarded when the client disconnects, while in a persistent session, the broker retains the client's subscriptions and any undelivered messages until the client reconnects. This guarantees that clients can receive any messages they missed even if there were disconnections or interruptions, in the network.

In terms of security, MQTT is supporting the Transport Security LayerTLS protocol for communicating with clients in encrypted mode to a broker. In the studies that were carried out, some deficiencies have been noted. These protocols only provide encryption and have weak scalability. With smart contracts and digital signatures, the security mechanism is rebuilt from the ground up. In addition, the user authentication system of MQTT is now supplemented with an OTPOneTimePassword mechanism which previously relied only on a username and password. Security is greatly enhanced by personalised passwords with a six digit number.

MQTT was not designed for security reasons, although it has been widely used. According to a survey conducted by the IoT developer community, MQTT is the second most popular IoT messaging protocol [8]. The rapid growth of MQTT necessitates that a sequential security version and mitigation process should be designed considering the resource constraint structure of the device. Therefore, for MQTT's security implementations, it has been determined that TLS/SSL restricts resources [9].

As far as safety is concerned, the decentralized structure of blockchain technology has its advantages. It is believed that through the use of smart contracts and digital signatures, the security mechanisms on MQTT can be replaced. Furthermore by implementing these proposed methods we aim to not authenticate user identities. Also ensure the utmost data integrity and accuracy. Smart contracts, records become immutable and actions can be traced. As a result, safety measures and assurance on the reliability of data will be introduced.

SSL/TLS, though a robust security protocol, still has some weaknesses. For instance, it is vulnerable to attacks through certificate authorities, which can lead to the breaching of security in the system. Also, for SSL/TLS to operate successfully, it requires a complex certificate infrastructure that needs to be properly configured and managed.

On By virtue of its decentralized architecture, it remains resilient against vulnerabilities associated with certificate authorities. A higher degree of simplicity and less resource requirements, as well as a reduction in complexity compared to the complexities encountered by SSL/TLS protocols, characterises Smart Contracts' configuration and management.

In summation, the security fortification of MQTT can be effectively facilitated through the integration of smart contracts, digital signatures, and OTP mechanisms. However, in order for this concept to be fully achieved it is crucial that we effectively tackle obstacles. These concerns revolve around making sure that the technologies utilized in relation, to the Internet of Things are suitable and capable of adapting to scales.

A substantial body of literature [10] [11] [12] [13], are adopting blockchain-based solutions to ensure the security of complex IoT systems. Hence, by incorporating MQTT's secure communication into a blockchain-based IoT system, the system can leverage the benefits of both technologies. As MQTT provides a simple and effective communication model that ensures the real time and reliable exchange of data between IoT devices, blockchain can provide safe, transparent or tamper proof records for transactions. A solution to create Internet of Things systems meeting the needs of security and performance, while also offering flexibility and higher levels of protection is provided by combining MQTT with blockchain.

2.1.1. The Benefits and Use Cases of Publish-Subscribe Messaging

The publish-subscribe model is a communication structure consisting of three main elements: publisher, subscriber, and broker. Within this structure, publishers send messages according to specific topics, and these messages carry information containing metadata related to the topics. In the MQTT protocol, clients initiate the communication process by establishing a TCP/IP connection with the broker. After completing the authentication stage, the client becomes part of the system by subscribing to relevant topics or publishing messages to those topics, as shown in Figure 1.

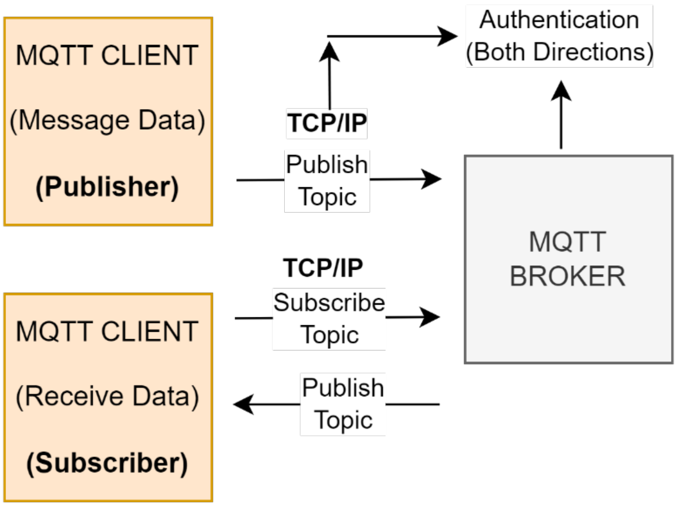


Figure 1:The primary components of the MQTT protocol

Prada et al. [14] have deliberated on the efficiency of the MQTT protocol in facilitating communication with devices limited by resources. There are benefits for a range of Internet of ThingsIoT applications that this model offers, e.g. adaptability and capability to adapt as the needs evolve. There is no direct connection between publishers and subscribers, allowing the system to be expandable and accommodating for the addition of new devices. Moreover the use of the publish subscribe model enhances the efficiency of communication while also minimizing power consumption.

The broker is responsible, for overseeing and improving the communication, between publishers and subscribers. The system therefore has a way of dealing with data transmission and security controls. The MQTT protocol effectively harnesses the benefits offered by this model in the realm of IoT guaranteeing energy efficiency and minimal delays. Therefore, the publish-subscribe model and MQTT protocol have become a popular choice in IoT applications [15]. Figure 2 shows the Publish-Subscribe methodology as presented.

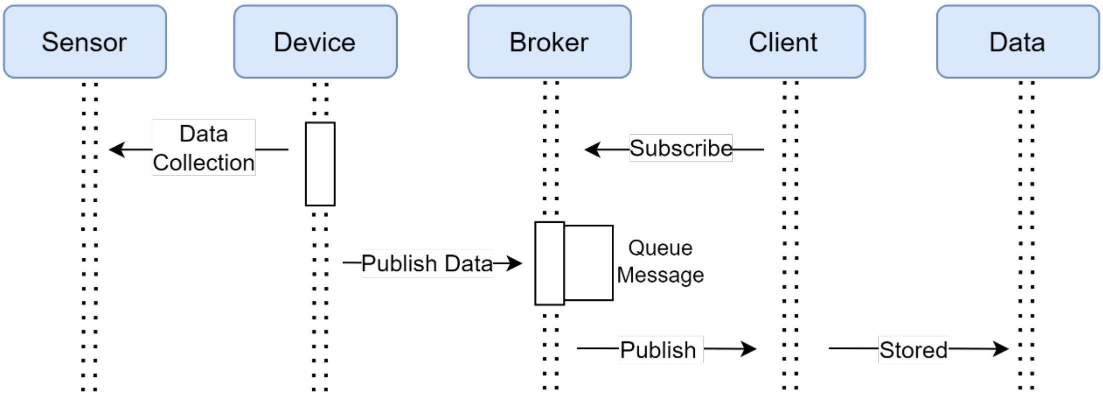


Figure 2: The publish-subscribe messaging sequence diagram

2.1.2. HBMQTT

HBMQTT is an open-source MQTT broker written in the Python programming language. Utilizing the publisher-subscriber model of the MQTT protocol, it enables data transmission between IoT devices. HBMQTT allows users to develop and customize their own private MQTT brokers [16].

One of advantages of HBMQTT is the flexible configuration options it offers. Users can write configuration files in Python, allowing them to tailor the broker's features and behavior. In particular in areas such as security, encryption and authorisations this is of great importance.

There have been no issues with integrating smart contracts directly into the broker configuration. The security and reliability of this system is further enhanced by the use of asymmetric cryptographic mechanisms for subscribers, while at the same time publishers benefit from Digital Signatures. Security and flexibility of authentication and authorization procedures will be significantly improved through the integration of systems.

HBMQTT, a customizable and secure MQTT broker developed with Python, provides a strong and reliable communication infrastructure for IoT systems. The HBMQTT assures the security and reliability of data transfer in Internet of Things environments, by offering flexible configuration options, features to protect users from unauthorised access as well as integrated blockchains.

2.2. Blockchain

Blockchain, a decentralized and distributed ledger technology, emerged in 2008 alongside Bitcoin, creating a significant revolution [17]. This new technology allows for safe, reliable and secure data storage with due regard to the management of supply chains, identity verification as well as other types of information in a manner that is also compatible with transactions.

As illustrated in Figure 3, the Blockchain consists of an database and a network of nodes. A blockchain database maintains records in the form of blocks, which are shared, distributed, tamper-resistant, and append-only. Although blocks are accessible to all blockchain users, they cannot be deleted or modified by them. Blocks are chained together because each block has the hash value of the previous one. There are several verified transactions in every block. Additionally, each block includes a timestamp indicating the time of its creation and a random number (nonce) for cryptographic operations. The blockchain network consists of nodes that safeguard the network as a distributed peer, to peer system. All the nodes are able to access the blocks. They don't have authority, over them [18] [19].

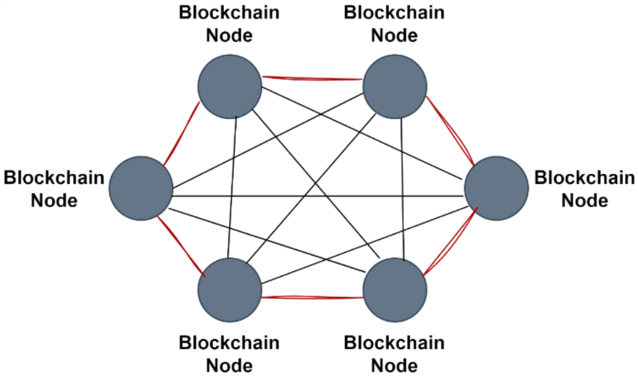


Figure 3: Blockchain network

The blockchain architecture is generally divided into four layers. The bottommost layer, the data layer, ensures the immutability of the overall data of the blockchain by utilizing asymmetric encryption, digital signatures, timestamps, cryptographic hash functions, and Merkle trees composed of blocks. The network layer verifies the accuracy of transmitted data by validating them through a dissemination mechanism. Using algorithms, the consensus layer ensures that node confidence and data consistency are maintained.

The application layer encompasses real-world scenarios in which the blockchain is implemented [20]. According to research Ethereum serves as a Blockchain platform utilized for creating security solutions based on smart contracts. Furthermore the initial studies have shown that Solidity is widely preferred as the programming language, for developing Ethereum contracts. The common reason that leads researchers to choose Ethereum as a Blockchain platform and prefer Solidity as the smart contract language is Ethereum's support for customizable smart contract programming and Solidity's similarity to JavaScript in terms of syntax [21] [22] [23]. To tackle cyber security issues and vulnerabilities in the Internet of Things, some initial studies have also explored IBM Hyperledger Fabric as a Blockchain platform.

Additionally, other primary research [24] [25] employed custom-built Blockchain platforms. Moreover when it comes to tackling security concerns, in the Internet of Things (IoT) primary studies have utilized characteristics found in Blockchain platforms. It is worth noting that the primary studies suggested smart contract-based security solutions that can be implemented on any Blockchain platform capable of executing smart contracts [26] [27].

As a result of all these findings, it has been determined that there is potential and a need to utilize blockchain for identity authentication in the MQTT and IoT mechanisms. Blockchain can be employed to securely and immutably store and share individuals' identity information. Furthermore it is crucial to develop methods and strategies that can enhance the utilization of technology for this specific objective.

2.2.1. Smart Contract

Smart contract, introduced with the advent of blockchain technology, are self-executing contracts with the terms of the agreement directly written into code. These contracts, providing a safe, transparent and decentralised way of conducting various operations and processes, shall be automaticly applied and enforced by the conditions set out in this Agreement. Smart contracts to deal with user authentication issues within the MQTT protocol have been developed for this particular case.

Ethereum is the preferred smart contract platform. The technology, behind the Ethereum platform was created by Vitalik Buterin. A user transaction or a message from another smart contract can activate a smart contract.Because a smart contract is stored on the blockchain it cannot be. Will only come into effect when one of these two events takes place.The code of the contract is executed by the Ethereum Virtual Machine (EVM) and provides a set of instructions that enable programmers to build arbitrary contracts applicable to any situation [28].

There has been research conducted to showcase how Blockchain can be applied in domains, including Identity Verification and Access Control [29] [30] [31] [32] [33].

Current security solutions in existing internet and IoT systems generally rely on a single trusted central authority, which makes them vulnerable to various attacks starting from a single point of failure, such as Denial of Service. On the other hand, using Smart Contracts based security solutions does not require significant changes in existing network infrastructure but rather takes advantage of key features of both cryptocurrencies and smart contracts. Most primary studies have relied on the decentralized, immutable, and auditable properties of blockchain technology. Some primary studies have utilized the customizable nature of smart contracts and the intrinsic properties of blockchain to develop identity verification, authorization, and access control solutions on top of blockchain for IoT [34].

The smart contract named AccessControl is designed to manage user access and authentication in a secure and decentralized manner. The contract stores allowed users, their usernames, and passwords in a mapping structure. The contract owner has the exclusive authority to grant and revoke access, as well as add, delete, or retrieve user information. Since a smart contract is stored on the blockchain it cannot be. Only if certain conditions are met will it enter into force. By utilizing the features of technology the smart contract adds an extra layer of protection, to the MQTT protocol.

This method is seen as an advancement, in enhancing the security , dependability of systems. In the future utilizing contracts and blockchain technology has the potential to enhance trustworthiness and security not for MQTT but, for other protocols.

The potential of combining the blockchain technology with current communication protocols has been highlighted when a contract for user authentication is integrated into MQTT.

2.2.2. Public and Private Key Cryptography in MQTT

Consider the blockchain as a daily ledger. Records are grouped into timestamped blocks. Each one is defined by an encryption hash. Each block refers to the one preceding it. This establishes a connection between these blocks, thereby creating a chain, or blockchain - see Figure 4. Any node with access can read this sequential, backward-linked list of blocks [35], understand what the state of ongoing data [36] is in the network as it changes in real time.

In order to ensure safe communication and identity verification, a component of cryptography, namely public and private key cryptography, has a critical role to play. In this encryption system each user possesses a pair of keys; a key that can be freely shared with others and a private key that remains confidential. The public key would be used to encrypt messages when the private key was used for decryption. This way of using keys allows for communications to be made in the absence of compromising any information.

Users interact with the blockchain through a pair of private/public keys [37]. They use their private keys to sign their transactions, and they can be addressed in the network through their public keys. Authentication, integrity and unrepudiation in the network are brought about by asymmetric cryptography. Each signed transaction is broadcast by a user's node to its peer-to-peer hops.

Neighbor peers ensure the incoming transaction is valid before further forwarding it; invalid transactions are discarded. The transaction is then transmitted all the way across the network.

The transactions collected and validated by the network are arranged and packaged in a time-stamped candidate block over a predetermined time interval. It's called the mining process. The mining node broadcasts this block back to the network. The selection of the mining node and the content of the block depends on the consensus mechanism used by the network.

Nodes validate the proposed block, ensuring through transaction and hash that it references the correct previous block in their chain. If this is the case, they add the block to their chain and apply the transactions it contains, updating accordingly. The proposed block shall be discarded if this does not happen, indicating the end of each round.

In this high-level security mechanism, it is evident that public and private key cryptography will play a crucial role in enhancing security within the context of integrating blockchain technology for MQTT user authentication. Granting access permission to a user through a smart contract is only possible with administrative authority. To possess this authority, the user must be able to validate the digitally signed message created with a private key unique to their public key. Upon obtaining authorization, the user can proceed to authorize specific publishers and subscribers in MQTT broadcasts, a process that is also integrated with OTP for an added layer of security.

The system that incorporates public and private cryptography into the MQTT user authentication process leverages the strengths of both blockchain technology and cryptographic methods. Security measures to limit access and allow authorised users to use the system shall be implemented in the process of integration. Furthermore in the realm of devices that utilize both private keys there is a promising opportunity to merge communication protocols with cryptographic techniques to effectively tackle the increasing security challenges.

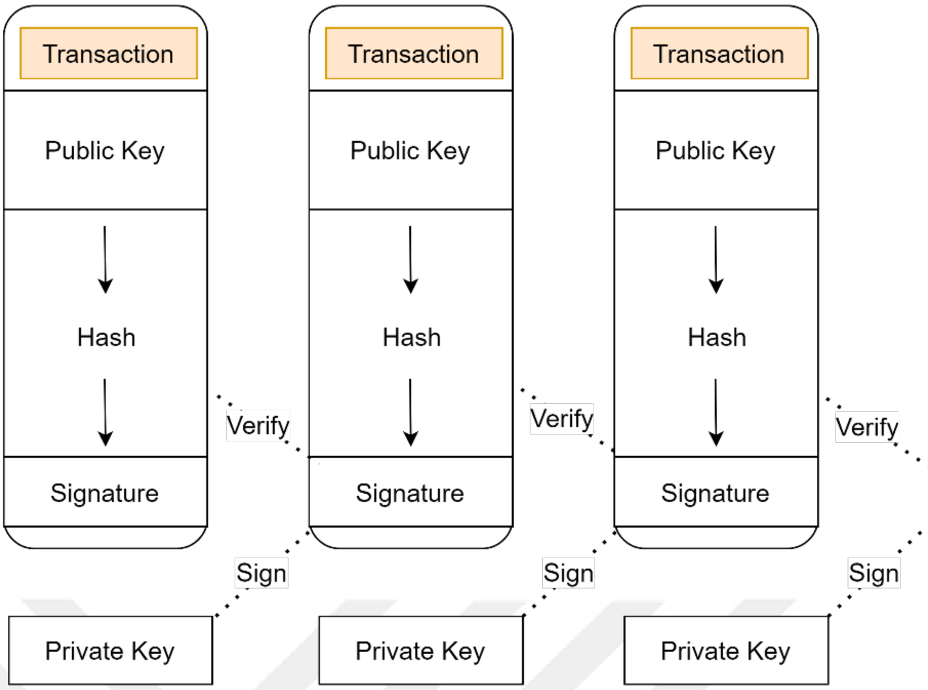


Figure 4: An overview of blockchain transaction processes

2.2.3. Consensus Algorithm

The smart contract developed for user security operates on the Ethereum network. Smart contracts running on the Ethereum network indirectly benefit from Ethereum's consensus algorithm. The Proof of Work consensus algorithm is currently used for Ethereum. It is currently undergoing a transition, towards the Proof of Stake (PoS) algorithm. The consensus algorithm enables the nodes, within the network to reach an agreement regarding the legitimacy of transactions and blocks. This ensures the accuracy and validity of transactions performed on the blockchain [38].

Proof of Work algorithm carries out transaction approvals through a process called mining. During this process miners strive to solve puzzles. The miner who successfully discovers the solution is then able to add the block and receive transaction fees and rewards as a result. This method leads to energy inefficiency and centralization due to the high energy consumption of nodes in the network [39].

Proof of Stake algorithm, on the other hand, uses nodes called "validators" instead of miners. These nodes participate in the validation process by locking ("staking") their cryptocurrencies in the network. Block validators are randomly selected based on the amount of cryptocurrency they hold and the duration of the lock.

It made the network safer and more transparent, thanks to this method which substantially reduces energy consumption [40].

The constraints of computational resources for any single entity make the imitation of multiple entities on a network futile. Specifically, any node can find the precise nonce, a random number that results in the SHA-256^2 hash of a block header, including the anticipated number of leading zeros. This will configure its block to be the next mining block in the network. Any node that can solve this puzzle establishes the so-called Proof-of-Work (PoW) and earns the privilege to shape the next block in the chain. Because this includes a one-way cryptographic hash function, any other node can easily verify that the given answer meets the requirement [41] [42].

Apart from SHA-256, other hashing algorithms such as Blake-256 [43] and scrypt [44], which is used in Litecoin, can be employed for PoW. There are mechanisms that incorporate multiple algorithms, such as Myriad [45]. An alternative to PoW, Proof-of-Stake (PoS), requires much less CPU computation for mining. In PoS, a node's chance of mining the next block is proportional to the balance of that node [46]. PoS schemes have their own strengths and weaknesses, and their real-world applications have proven to be quite intricate [47].

Smart contracts operating on the Ethereum network benefit from the security and verification advantages provided by the consensus algorithm. For example, in the specified smart contract, the accuracy and security of transactions managing user access control are ensured through the Ethereum network's consensus algorithm. In this case the consensus algorithm allows users to have a taste for the benefits of decentralization technology, by ensuring that digital contracts are reliability and transparency.

In the defence of contracts against cyber attacks and malicious acts, the consensus algorithm also plays a role. private networks where participants are whitelisted, there is no need for costly consensus mechanisms like PoW; there is no risk of a Sybil attack [48]. This removes the requirement for a motivation, for mining. Gives us the opportunity to select from a broader selection of consensus protocols. The Ethereum networks smart contracts provide a framework for transactions and storing data thanks, to the level of security and verification offered by this robust consensus algorithm.

Furthermore, the network of smart contracts is guaranteed to be trusted and transparent by a consensus algorithm. Every Ethereum node is storing an upto date copy of the blockchain, and continuous synchronization between nodes remains in place. As a result, transactions and changes can be seen and verified by everyone. The general processing flow can be observed in Figure 5. As far as the data are concerned, that feature provides advantages such as identity verification and access control [49].

Smart contract on the Ethereum network benefit from the security, verification, and transparency advantages provided by the consensus algorithm. Applications or services running on a secure, transparent structure can be made available to and used by users and developers. In the wide acceptance and efficient application of intelligent contracts and decentralised applications, consensus algorithm is an integral part of technology which serves as a major factor.

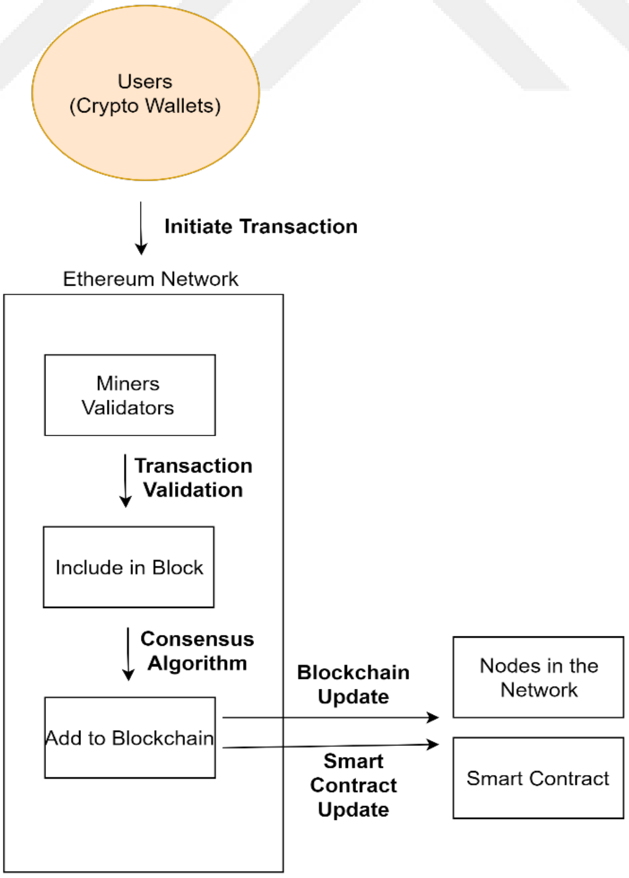


Figure 5: Smart contract execution in ethereum network

2.3. Digital Signature and OTP

To verify the authenticity and accuracy of documents or data, an electronic signature is a verification mechanism. It uses a pair of keys (private and public). The public key is used to generate a digital signature of the message, whereas the private key will be used for validation in Figure 6. This technique is carried out using asymmetric encryption algorithms like RSA (Rivest–Shamir–Adleman) or DSA (Digital Signature Algorithm).

A hash function generates a unique value from the data and this value is encrypted with the private key. The digital signature formed in this way is attached to the message. The receiver side decrypts this signature with the public key and calculates the hash value by applying the same hash function to the rest of the message. If these two hash values match, the integrity of the message is verified [50].

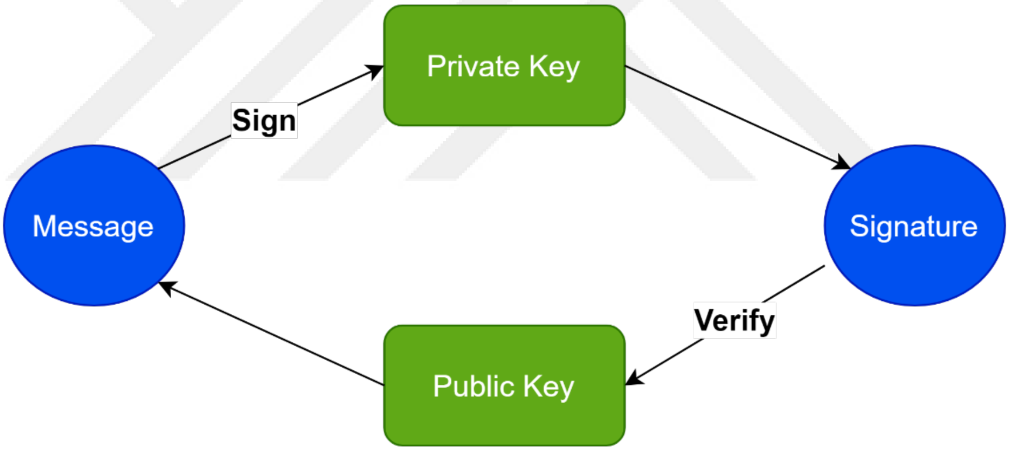


Figure 6: Digital signature diagram

Digital signatures are indeed crucial for verifying and preserving the integrity of data coming from IoT (Internet of Things) devices, especially when working with lightweight protocols such as MQTT (Message Queuing Telemetry Transport).

On On the other hand, SSL/TLS protocols bring a heavier computational load. This is due to the need to compute a MAC (Message Authentication Code) to verify the integrity of every message and the requirement to decrypt each message. Moreover, the "handshake" implementation of SSL/TLS may have a significant effect on performance for networks with insufficient capacity as well as high latency.

Therefore, Digital Signatures can be more advantageous in IoT devices that are typically resource-limited and in applications where energy efficiency is critical. Moreover they plan to combine these elements with contracts in order to simplify the management and execution of keys and transactions. This simplifies the implementation of signature systems leading to a decrease, in the requirement, for securely storing and managing account keys.

The elliptic curve digital signature algorithm (ECDSA) has been chosen as the digital signature method. ECDSA operates on the basis of group theory defined over an elliptic curve. Each user selects a random point (private key) on the elliptic curve and obtains another point (public key) by multiplying this point a certain number of times. While this multiplication process can be performed easily, the reverse operation (i.e., the division operation) to return to the starting point is computationally impractical, forming the foundation of ECDSA's security.

Despite having smaller key sizes, ECDSA provides a similar level of security compared to RSA and DSA. For instance the security level of a 256 bit ECDSA key is equivalent, to that of a 3072 bit RSA key. This leads to a decrease, in the amount of memory. Bandwidth used by ECDSA, which ultimately leads to faster processing times and lower overall energy consumption. These attributes play a role especially when it comes to devices that operate with constrained energy and processing capabilities [51].

The ECDSA is a method to guarantee safety through the use of curve cryptography and also known as Elliptic Curve Digital Signature Algorithm. The same level of safety is ensured. But it requires shorter key lengths. Such specific functionality may be appreciated by devices that are restricted in access to resources such as those available through the Internet of Things. There are advantages, to utilizing a time span, which include decreased utilization of processing power and memory reduced energy consumption and enhanced overall performance.

ECDSA provides an advantage over more resource efficient algorithms such as RSA that are widespread in the application of standard SSL andTLS protocols. Given ECDSA's ability to provide an optimal level of security by using SSL/TLS protocols with equivalent or lower resource usage, it has been found that its use is a very effective tool in securing the integrity and verification of Internet of Things devices.

When combined with smart contracts, digital signature becomes even stronger. Smart contracts allow for the secure management and verification of digital signatures. With this it makes digital signatures widely available and easier to use.

In digital signature applications, the OTP (One-Time Password) mechanism is often used as an additional layer of security. Users have the freedom to generate and modify their usernames and passwords frequently as they desire. These passwords are included in the content of the digital signature and are known only to a specific user. In this way, validating the integrity and authenticity of a message requires not only the use of the public key but also the input of this OTP password. In order to prevent unauthorised access and data manipulation, this process shall ensure that the confidentiality of signatures is strengthened. In particular, if it helps to enhance overall security, e.g. as regards devices and large data flows, it will be of great benefit.

Lastly, MQTT security can be greatly improved if the correct application of ECDSA is made in Figure 7. This offers a faster and more efficient solution, and ensures the integrity, verification, and encryption of data coming from IoT devices [52].

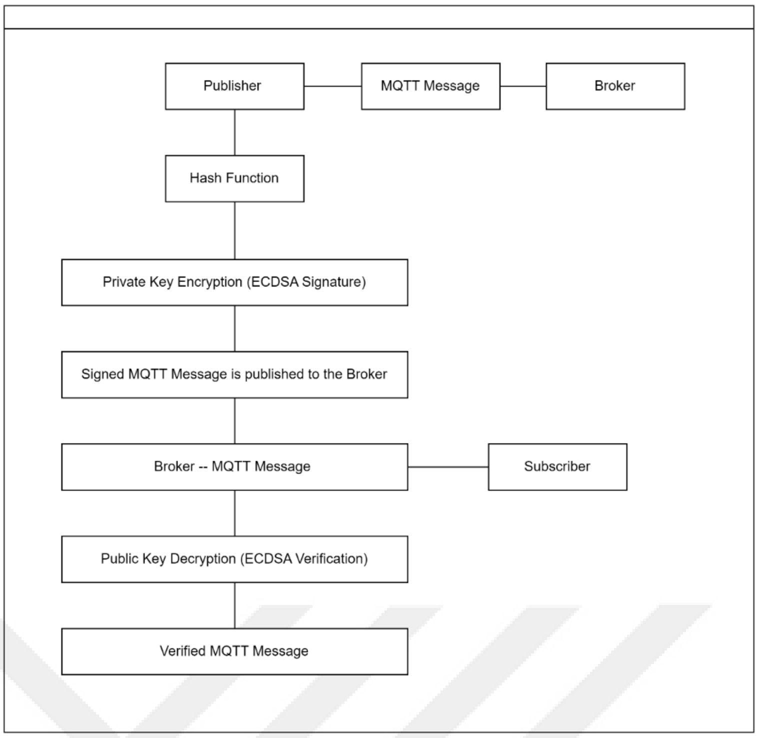
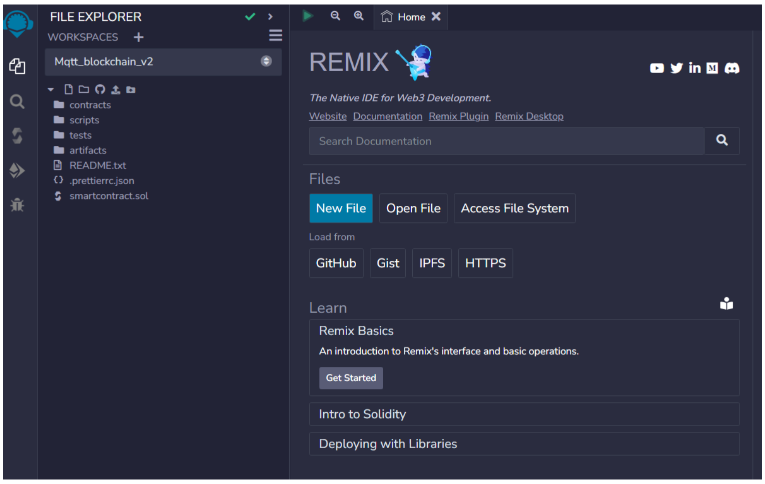

Publisher
MQTT Message
Broker
Hash Function
Private Key Encryption (ECDSA Signature)
Signed MQTT Message is published to the Broker
Broker -- MQTT Message
Subscriber
Public Key Decryption (ECDSA Verification)
Verified MQTT Message

Figure 7:Elliptic curve digital signature algorithm

2.4. Remix IDE and Ganache

Figure8, Remix IDE is a popular and user-friendly web tool used for writing, compiling, and testing Ethereum-based smart contracts. It is utilized to ensure that smart contracts written in Solidity language function correctly and produce the expected results [53].

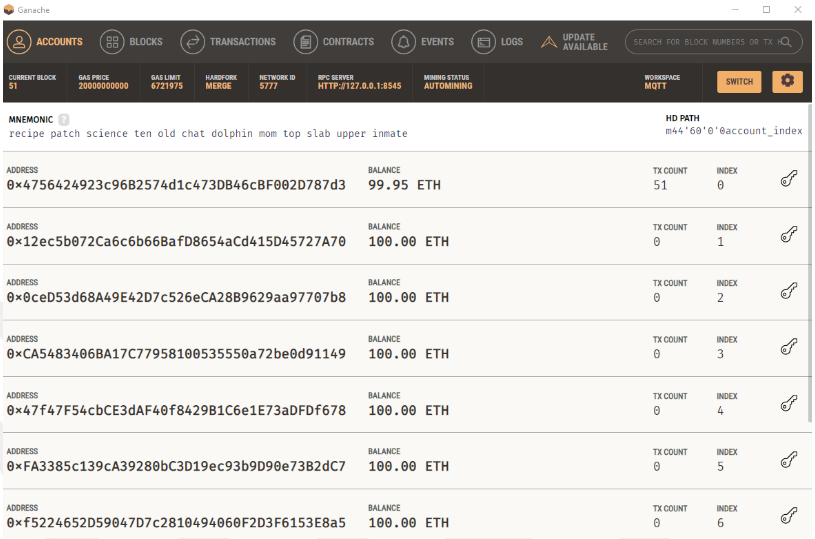


The Native IDE for Web3 Development.

An introduction to Remix's interface and basic operations.

Figure 8: Remix Ide

Figure 9, Ganache is a personal blockchain for Ethereum development, allowing developers to deploy and test smart contracts in a safe and controlled environment. It provides a built-in network of test Ethereum accounts with pre-allocated test Ether, making it easy to simulate transactions and interactions with smart contracts without the risk of spending real Ether [54].



CURRENT BLOCK	GAS PRICE	GAS LIMIT	HARDFORK	NETWORK ID	RPC SERVER	MINING STATUS	WORKSPACE
51	20000000000	6721975	MERGE	5777	HTTP://127.0.0.1:8545	AUTOMINING	MQTT

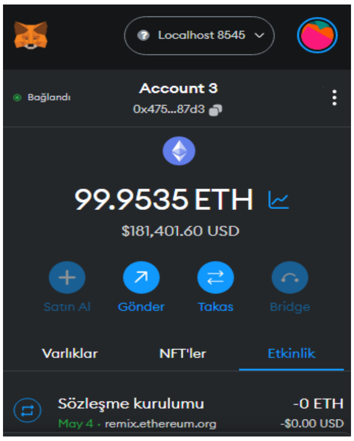
MNEMONIC
recipe patch science ten old chat dolphin mom top slab upper inmate

HD PATH
m44'60'0'0account_index

ADDRESS	BALANCE	TX COUNT	INDEX
0×4756424923c96B2574d1c473DB46cBF002D787d3	99.95 ETH	51	0
0×12ec5b072Ca6c6b66BafD8654aCd415D45727A70	100.00 ETH	0	1
0×0ceD53d68A49E42D7c526eCA28B9629aa97707b8	100.00 ETH	0	2
0×CA5483406BA17C77958100535550a72be0d91149	100.00 ETH	0	3
0×47f47F54cbCE3dAF40f8429B1C6e1E73aDFDf678	100.00 ETH	0	4
0×FA3385c139cA39280bC3D19ec93b9D90e73B2dC7	100.00 ETH	0	5
0×f5224652D59047D7c2810494060F2D3F6153E8a5	100.00 ETH	0	6

Figure 9: Ganache truffle suite

Connecting Ganache to a MetaMask wallet Figure 10 enables seamless testing of smart contract functionality with test Ether transactions, further enhancing the development process [55]. In order to ensure the security and reliability of Smart Contracts prior to their deployment on the Ethereum network, a complete testing environment is created in combination with IDE Remix and Ganache.



Bağlandı

Account 3
0x475...87d3

99.9535 ETH
$181,401.60 USD

Sözleşme kurulumu -0 ETH
May 4 · remix.ethereum.org -$0.00 USD

Figure 10: Metamask wallet

2.5. Decentralized Identity Management

Decentralizing identity management is an important part of the solution with a view to improving data security and privacy in MQTT protocols. The secure storage of data related to identity, authorisation credentials and Digital Rights can already be used in these systems. By ensuring the immutability of data, blockchain technology can be leveraged [56] .

The ability of individuals to have control over identity and the protection of personal data is one aspect which makes a difference. In order to eliminate the requirement of authority, centralised identity management plays an important role.

In October 2016 Dyn, a provider of DNS services experienced a DDoS attack. Caused major websites, like Twitter, Amazon, Tumblr, Reddit, Spotify and others to go offline for several hours [57]. The fact that IP and user identity spoofing attacks can be very effective is demonstrated by the "Dyn attack" incident.

Better defense against IP and user identity spoofing can be provided by the use of cryptography based identities and access management systems. When IP spoofing attacks are launched, such as in subsequent versions of the Mirai botnet, the immutability of validated blockchains can prevent devices from connecting to a network by disguising themselves with fake signatures [58]. In this context, the example of Filament's Taps explains this point effectively [59].

Embracing decentralized identity management for user authentication in the MQTT protocol not only strengthens overall system security but also fosters trust among users. By incorporating technology into identity and data verification processes MQTT has the capability to provide an trustworthy authentication system. As a result this establishes the groundwork, for communication systems that're more secure and dependable guaranteeing the privacy, integrity and accessibility of data throughout the network.

3. SYSTEM

The experimental setup and configuration, as shown in Figure 11, are explained. In this experimental setup, an MQTT client running on a Raspberry Pi 2 [60], an MQTT broker executed with a Python script, and an integrated blockchain security mechanism at the broker's location are utilized. An integrated sensor for measuring temperature and humidity is included in the Raspberry Pi. DHT11 sensor was employed for measuring temperature and humidity levels [61]. Secondly, by setting up the MQTT broker on another computer, a connection is established with the Raspberry Pi. During this connection and data exchange, MQTT packets are analyzed using the Wireshark network analysis tool. Through these analyses, the contribution of OTP blockchain-based identity and digital signature data verification methods to data security over the MQTT protocol will be demonstrated. In this chapter, the experimental setup will be explained step by step in detail through a figure, and the analyses and results obtained during this process will be evaluated.

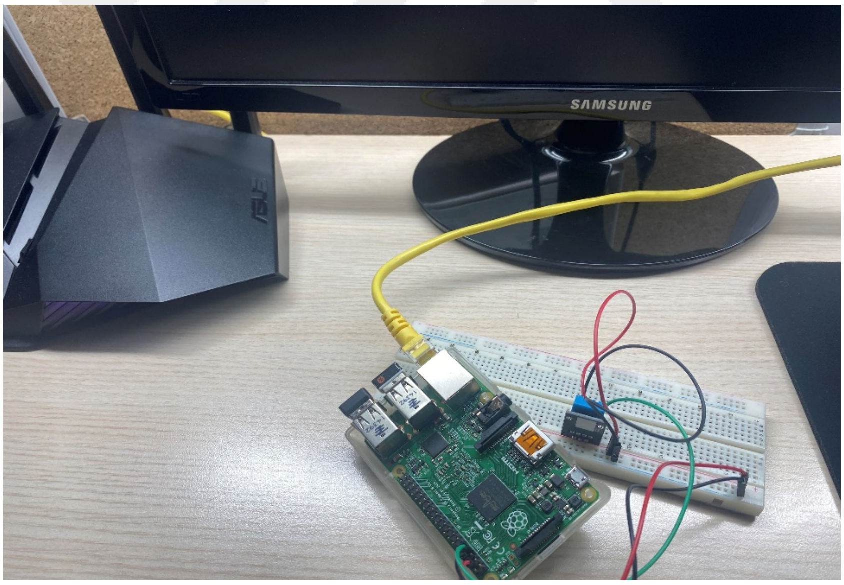


Figure 11:The introduction and configuration of devices

In our blockchain model, using the Ethereum blockchain due to its distributed information transfer capabilities, smart contracts are deployed in order to enhance user security. Allowing easy access to everyone is not desired because malicious manipulations can lead to problems. Therefore, a mechanism has been created where only users added and granted access permission through a smart contract can access and publish data. The system will terminate a connection if an unauthorised user tries to access it. Hence, the authorization mechanism is of utmost importance.

As indicated in Figure 12, users can be added or removed with the help of a smart contract. If a user is added to the system, access permission to the desired data is given to the desired user again with the help of a smart contract. If desired, access can also be blocked.

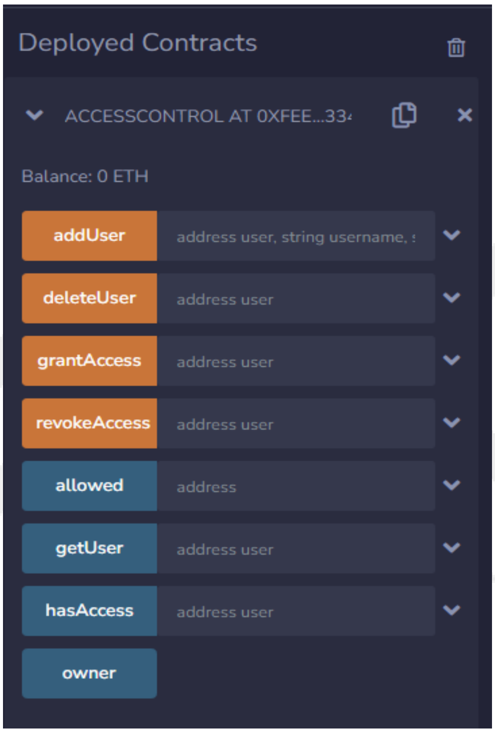


Figure 12: Smart contract function blocks

Only the users you specify and the data you permit can be monitored in the system. As previously mentioned, it offers a simple mechanism with its unchangeable structure and personalized address structure for everyone. Thus, blockchain technology and smart contracts together have developed a user security mechanism.

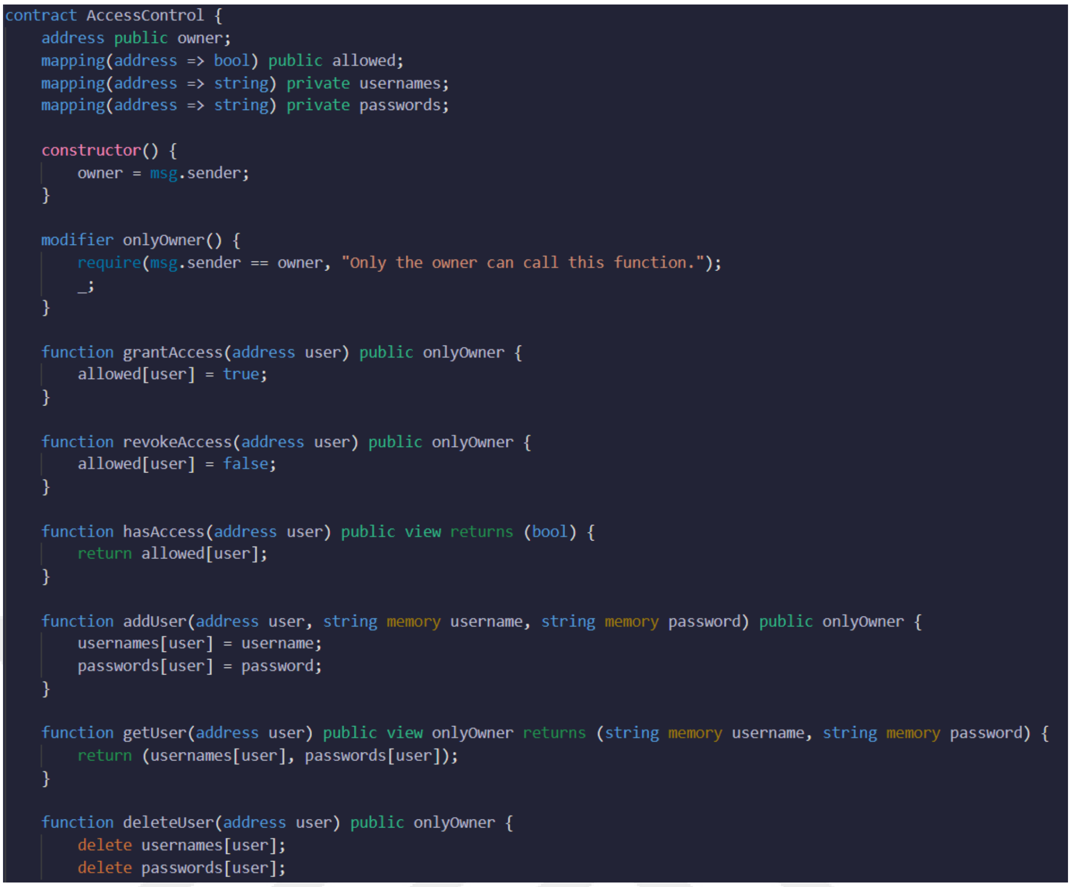
The standard version of MQTT does not incorporate a robust mechanism to authenticate users and devices. Consequently, unauthorized users or devices could potentially conduct publish (pub) or subscribe (sub) operations.

MQTT lacks a mechanism for proper permission control. There's a deficiency in constraining which topics users or devices can access or broadcast. The security problems associated with MQTT are commonly addressed by using the SSL Sockets Layer and TLS Transport Layer Security. Encryption and verification techniques, which are designed to strengthen the security measures for transmitting data over networks, shall be covered by these protocols.

However, the solution of SSL/TLS has certain limitations. It does not offer a solution regarding permission control. It does not provide a mechanism to determine who can broadcast to which topics or subscribe to them. It may be difficult and time consuming to distribute and manage certificates, in particular on large Internet of Things networks. Smart contracts offer a better proposal for solutions at this stage.

In the Ethereum platform Solidity is a programming language that was specially designed to build contracts. The increasing popularity of it is starting to be embraced by developers. Is now considered the standard language for writing smart contracts due to its seamless integration with the Ethereum Virtual Machine (EVM) and its capability to handle intricate blockchain logic effectively. The contract's coded on the Ethereum platform using Solidity, a programming language that can be used to create smart contracts.

In this smart contract, the aim is to implement a simple access control system for managing user permissions on the blockchain. The smart contract named "**AccessControl**" consists of several variables, functions, and a modifier. Please refer to Figure 13 for an explanation of AccessControls contract structure and components. To gain an understanding of how the contract functions this diagram provides a depiction of the variables, functions and modifier involved.



```
contract AccessControl {
    address public owner;
    mapping(address => bool) public allowed;
    mapping(address => string) private usernames;
    mapping(address => string) private passwords;

    constructor() {
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Only the owner can call this function.");
        _;
    }

    function grantAccess(address user) public onlyOwner {
        allowed[user] = true;
    }

    function revokeAccess(address user) public onlyOwner {
        allowed[user] = false;
    }

    function hasAccess(address user) public view returns (bool) {
        return allowed[user];
    }

    function addUser(address user, string memory username, string memory password) public onlyOwner {
        usernames[user] = username;
        passwords[user] = password;
    }

    function getUser(address user) public view onlyOwner returns (string memory username, string memory password) {
        return (usernames[user], passwords[user]);
    }

    function deleteUser(address user) public onlyOwner {
        delete usernames[user];
        delete passwords[user];
```

Figure 13: A general template for an accesscontrol smart contract:

The first step in the proceess is for the user to deploy the AccessControl smart contract on the Ethereum blockchain. This is done by compiling the Solidity code and submitting a contract creation transaction to the Ethereum node. A unique contract address that can serve as a reference point for future interactions with smart contracts will be generated on the Ethereum node once this transaction has been confirmed.

After receiving the contract address from the Ethereum node, the user stores it for future reference. This address will be used whenever the user wants to interact with the AccessControl smart contract, such as when adding users, granting or revoking permissions, and checking access statuses.

A user will call the **addUser(...)** function on a smart contract, which returns an Ethereum address, username and password argument when adding a new user to access control system.This function links the provided Ethereum address to the specified username and password and creates a new entry in the contract's username and password mapping.

After the new user has been added to the system, the contract owner (the user who deployed the contract – admin) can grant them access permission by calling the **grantAccess(...)** function, passing the new user's Ethereum address as an argument. The allowed mapping is updated with this function and the value of a given address is set to true.

To check the access permission status of a user, the contract owner can call the **hasAccess(...)** function, passing the user's Ethereum address as an argument. To determine whether a specified user has been granted access rights, this function provides either true or false value.

Figure 14, the sequence diagram provides a visual representation of the steps involved in deploying and interacting with the AccessControl smart contract on the Ethereum blockchain. It's a method to comprehend the flow of events and communication, among the elements, within the system. The user can effectively administer access permissions for different users in the EthereumBased Access Control System by performing these steps.

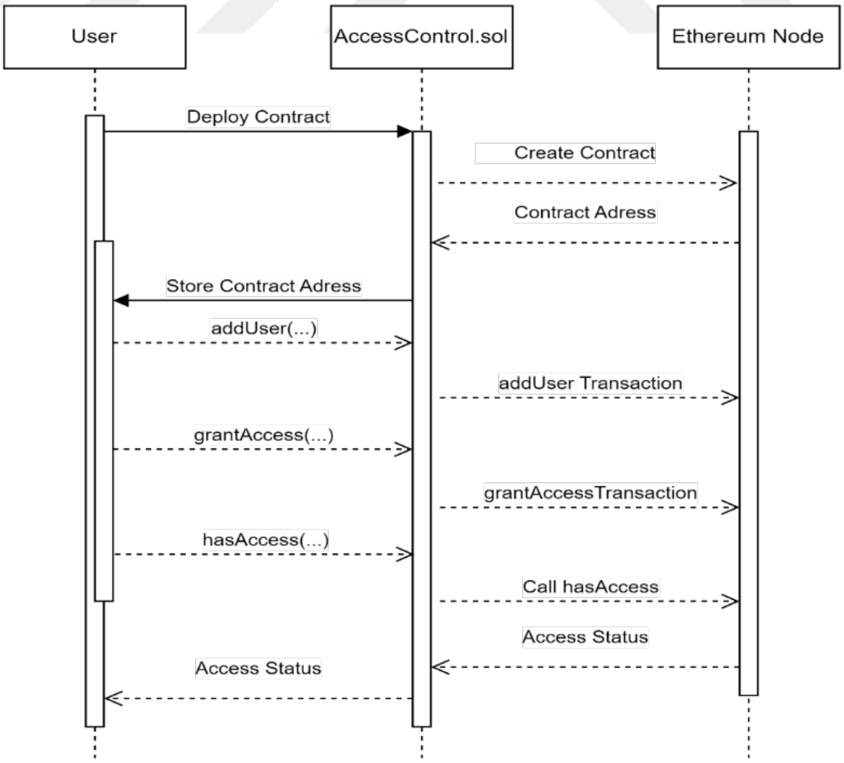
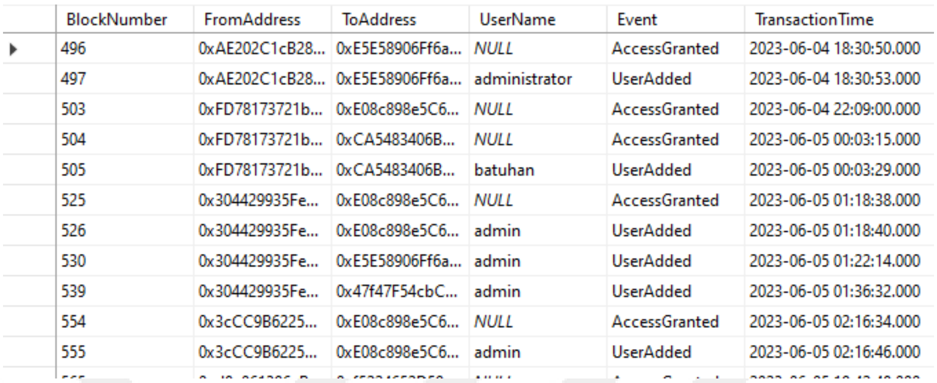


Figure 14:MQTT and OTP blockchain-based system configuration

Additionally, since every operation provides a decentralized and immutable transaction structure, monitoring of the data is also ensured Figure 15. An infrastructure that automatically records all permissions granted, denied, user additions and removals has been established.



	BlockNumber	FromAddress	ToAddress	UserName	Event	TransactionTime
▶	496	0xAE202C1cB28...	0xE5E58906Ff6a...	*NULL*	AccessGranted	2023-06-04 18:30:50.000
	497	0xAE202C1cB28...	0xE5E58906Ff6a...	administrator	UserAdded	2023-06-04 18:30:53.000
	503	0xFD78173721b...	0xE08c898e5C6...	*NULL*	AccessGranted	2023-06-04 22:09:00.000
	504	0xFD78173721b...	0xCA5483406B...	*NULL*	AccessGranted	2023-06-05 00:03:15.000
	505	0xFD78173721b...	0xCA5483406B...	batuhan	UserAdded	2023-06-05 00:03:29.000
	525	0x304429935Fe...	0xE08c898e5C6...	*NULL*	AccessGranted	2023-06-05 01:18:38.000
	526	0x304429935Fe...	0xE08c898e5C6...	admin	UserAdded	2023-06-05 01:18:40.000
	530	0x304429935Fe...	0xE5E58906Ff6a...	admin	UserAdded	2023-06-05 01:22:14.000
	539	0x304429935Fe...	0x47f47F54cbC...	admin	UserAdded	2023-06-05 01:36:32.000
	554	0x3cCC9B6225...	0xE08c898e5C6...	*NULL*	AccessGranted	2023-06-05 02:16:34.000
	555	0x3cCC9B6225...	0xE08c898e5C6...	admin	UserAdded	2023-06-05 02:16:46.000

Figure 15: Storage in smart contract transaction

Enhancing the security of the MQTT protocol through the implementation of an MQTT Broker. Developed using the Python language and the hbmqtt library, the MQTT Broker replaces traditionally resource-intensive SSL and TLS protocols with the lighter and more energy-efficient OTP and digital signatures to enhance security. The Elliptic Curve Digital Signature Algorithm is preferred for digital signatures.

Whether a client is a publisher or a subscriber, they must have received authorization in a smart contract on the Ethereum blockchain. As a publisher, a user can create an MQTT message. However, the broker, which is awaiting the message, expects the content of the signed message to include the authorized address and the OTP information specifically generated for that address. Therefore, besides having access permission, the publisher's message must contain accurate information to pass the verification of the digitally signed content by the broker. Otherwise, the broker will not validate the message.

The broker is committing a transaction to the blockchain when he has verified his client's identity. This ensures that the data integrity and its origin are checked, as well as enabling a broker to operate independently of where it is located.

The overall overview of the suggested method is depicted in Figure 16. This is a very important step in guaranteeing the security of MQTT protocol, as Smart Contracts, Digital Signers and OTP techniques will be integrated into this approach.

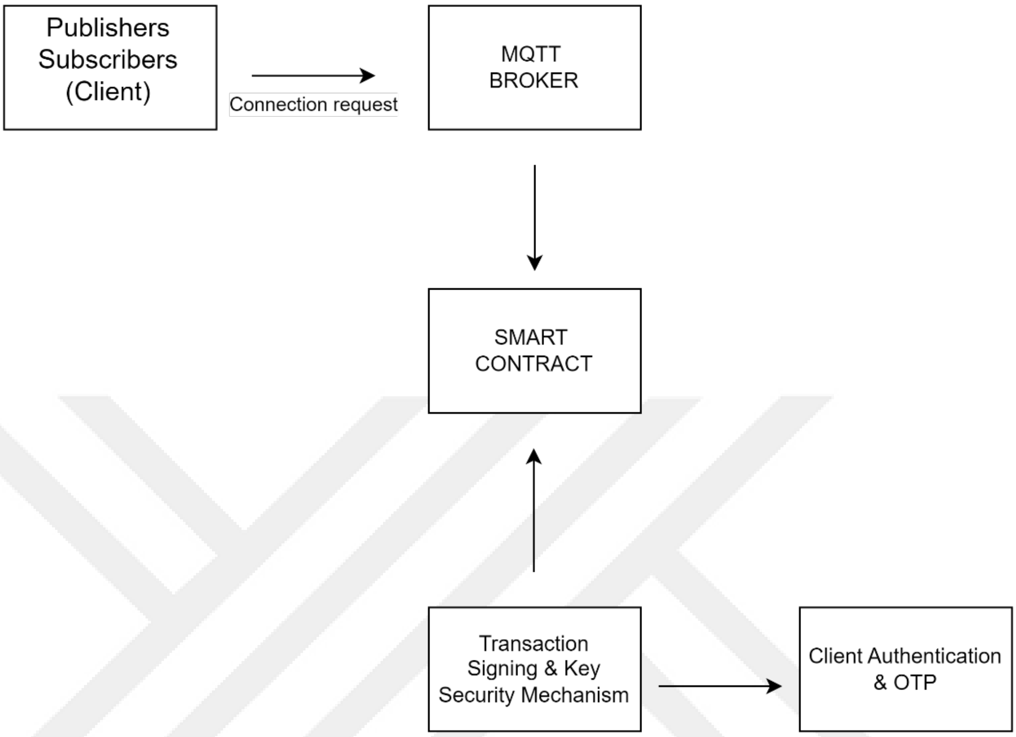


Figure 16:Recommendation-based overview

Several crucial steps are involved in the authentication and access control process for internet of Things devices, with MQTT brokering, smart contracts or digitally signature playing an important role. The application that registers and configures users and devices by means of Smart Contracts shall manage the first step. Then the application will request a connection to MQTT broker, which in turn shall facilitate transfer of clients' identity to Smart Contracts.

Smart contracts inform the application of client information and access control data in order to keep the process going. An OTP (One-Time Password) specific to the user is obtained. This one-time password (OTP) is assigned to each user and is constantly renewed. When initializing the broker, the user's access-permitted address and their corresponding OTP information are awaited.

Publishers create MQTT messages and include the expected information in their digital signatures. The security and integrity of published information shall be ensured by this Digital Signatories. Subscribers similarly decrypt the asymmetrically encrypted data, provided they are authorized users.

Every authorized operation is recorded and viewable on the blockchain. In order to efficiently and safely manage the process of authentication and access control for connected devices, this approach effectively complements Smart Contracts, OTP or digital signatures set out in Figure 17.

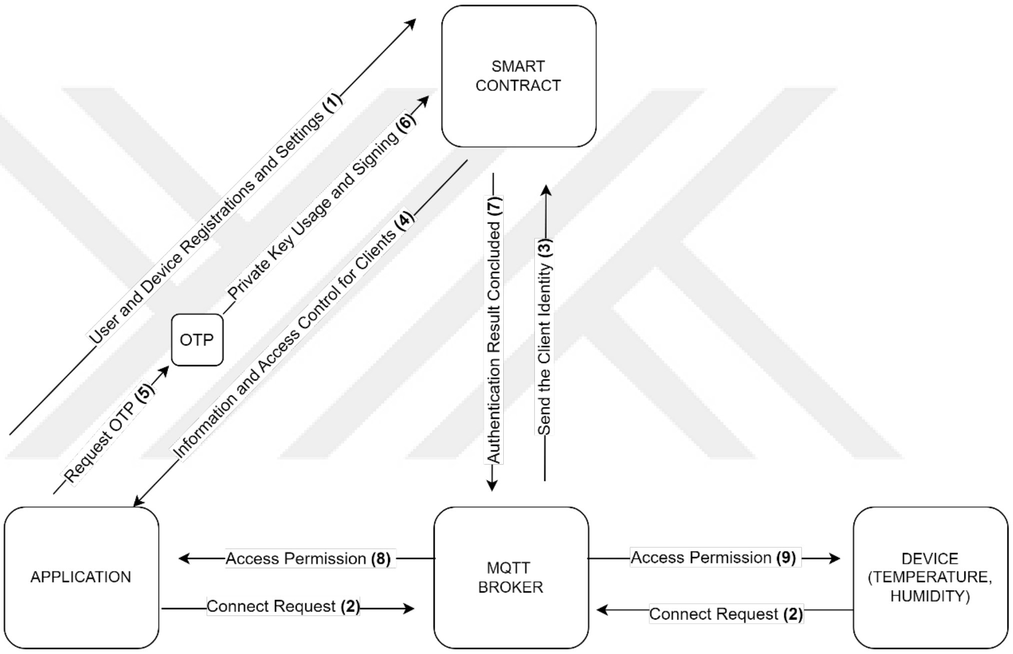

SMART CONTRACT
User and Device Registrations and Settings (1)
Private Key Usage and Signing (6)
Information and Access Control for Clients (4)
OTP
Request OTP (5)
Authentication Result Concluded (7)
Send the Client Identity (3)
APPLICATION
Access Permission (8)
Connect Request (2)
MQTT BROKER
Access Permission (9)
Connect Request (2)
DEVICE (TEMPERATURE, HUMIDITY)

Figure 17:MQTT and OTP blockchain-based system configuration

4.EVALUATION

4.1. Comparative Analysis

SSL/TLS technology has been utilized on the internet for a period making it extensively adopted. The issue of SSL certificates by certification authorities is one aspect of this system. Numerous companies have come to recognize the significance of keeping their SSL certificates up, to date after the OpenSSL Heartbleed vulnerability was uncovered. This incident triggered a search for more advanced technologies [62] .

The Heartbleed hack is the kind of security vulnerability that was discovered in 2014 due to a problem with an OpenSSL library. This vulnerability surfaces in the "heartbeat" extension used in TLS/SSL communication. An attacker could exploit this vulnerability to steal critical information from your server by using a data exchange error with the client. Such attacks allow attackers to access information that is not publicly available in server memory, e.g. user logon credentials. A major concern for the safety of the Internet was raised by the Heartbleed attack that has caused a large number of websites to review and reinforce their security measures.

A comprehensive comparison of integrated authentication systems using MQTT is presented. Figure 18 includes a sequence diagram depicting the functioning of TLS/SSL, while Figure 19 contains the sequence diagram of the proposed solution. Upon comparing the two processes, it has been determined that the existing method, integrated with TLS (Transport Layer Security) and SSL (Secure Socket Layer), exhibits certain adverse effects.

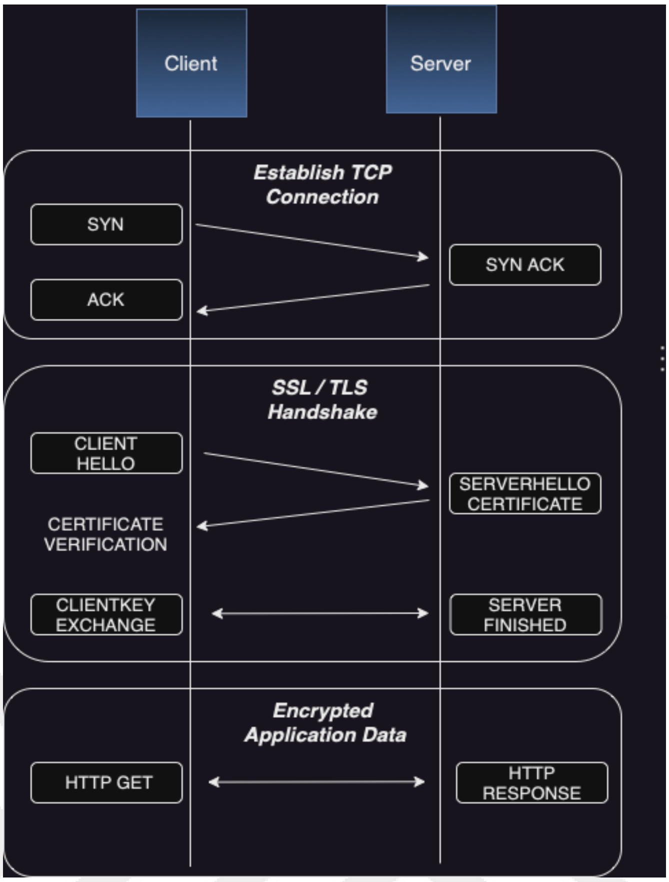


Figure 18: TLS/SSL Sequence Diagram

In research and the measured results [63], TLS requires additional processes and security handshakes during the connection establishment process. Increased connection times can result from these other processes, especially in scenarios with an increasing loss rate for packets or a deterioration of network conditions. This is because network issues such as packet loss hinder the proper transmission of data and impact the connection establishment process.

As a result of this, the TLS/SSL protocol increases RAM consumption in data processing by creating additional buffers for authentication and data encryption processes. In the context of certificates based authentication processes, which use large key lengths, this increase is especially pronounced. As the key length increases, the size of the generated buffers also grows, resulting in a significant rise in RAM usage.

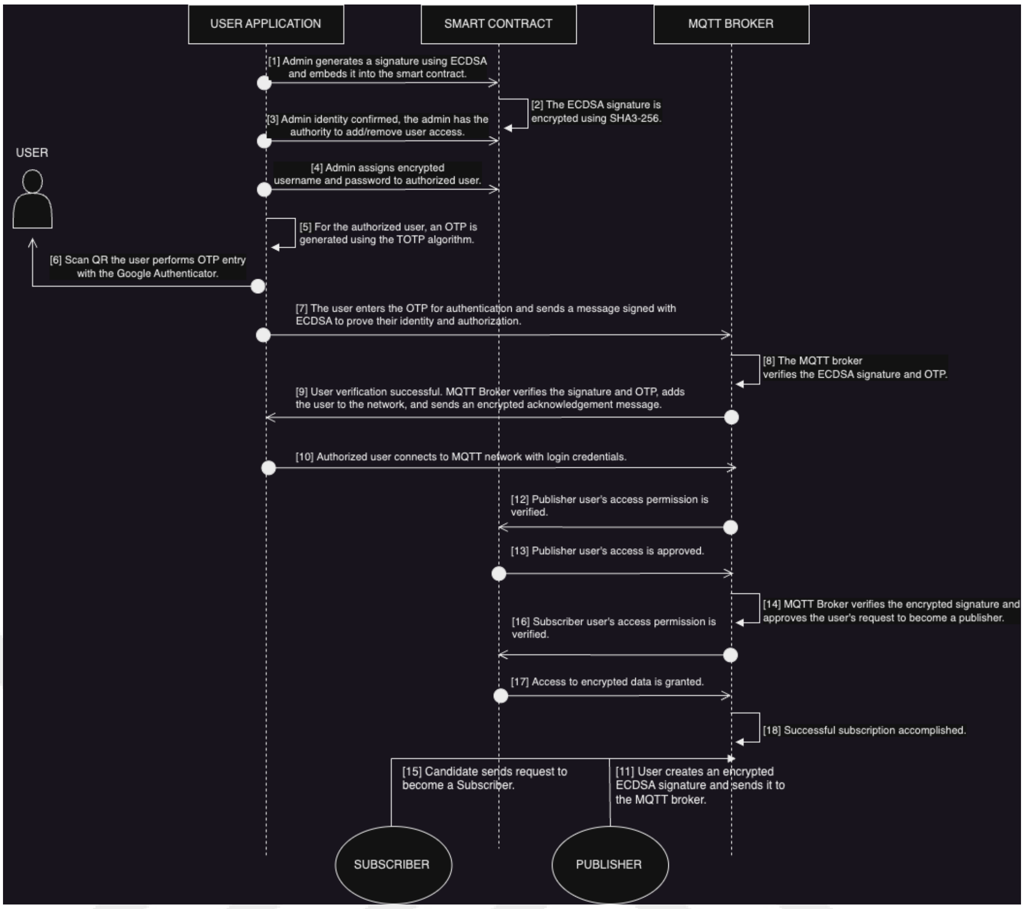

USER APPLICATION
SMART CONTRACT
MQTT BROKER
[1] Admin generates a signature using ECDSA and embeds it into the smart contract.
[2] The ECDSA signature is encrypted using SHA3-256.
[3] Admin identity confirmed, the admin has the authority to add/remove user access.
USER
[4] Admin assigns encrypted username and password to authorized user.
[5] For the authorized user, an OTP is generated using the TOTP algorithm.
[6] Scan QR the user performs OTP entry with the Google Authenticator.
[7] The user enters the OTP for authentication and sends a message signed with ECDSA to prove their identity and authorization.
[8] The MQTT broker verifies the ECDSA signature and OTP.
[9] User verification successful. MQTT Broker verifies the signature and OTP, adds the user to the network, and sends an encrypted acknowledgement message.
[10] Authorized user connects to MQTT network with login credentials.
[12] Publisher user's access permission is verified.
[13] Publisher user's access is approved.
[14] MQTT Broker verifies the encrypted signature and approves the user's request to become a publisher.
[16] Subscriber user's access permission is verified.
[17] Access to encrypted data is granted.
[18] Successful subscription accomplished.
[15] Candidate sends request to become a Subscriber.
[11] User creates an encrypted ECDSA signature and sends it to the MQTT broker.
SUBSCRIBER
PUBLISHER

Figure 19:Proposed Solution Sequence Diagram

However, in the context of identity authentication based on smart contracts, the mechanism of creating extra buffers is mitigated. Smart contracts transparently store digital certificates and other authentication-related information in a blockchain, ensuring data immutability and traceability. This means that changes such as the setting up of additional buffers are not being implemented, thus making data processing more efficient.

Therefore, cryptographic algorithms like ECDSA, used in conjunction with smart contracts, can have shorter key lengths, resulting in lower RAM consumption and reduced creation of extra buffers. This enables resource constrained devices to have a performance advantage and increases the efficiency of security authentication processes.

Especially when performing a TLS handshake with a certificate that has a large key length between MQTT clients connecting to the broker, a significant CPU workload has been observed.

Furthermore, during user authentication using smart contracts, smart contracts have been employed to add, remove, and grant communication access permissions to authorized users. At this point, it has been observed that the key length of smart contracts has a significant impact. Particularly, using a shorter key length in user authentication with ECDSA (Elliptic Curve Digital Signature Algorithm) results in a considerable reduction in CPU usage. The lower CPU workload requirement of ECDSA positions smart contracts as a faster and more efficient authentication method compared to TLS/SSL. This advantage can make the ECDSA and smart contracts a fast, more effective authentication method than TLS SSL offering significant benefits both in terms of efficiency and performance.

Smart contracts also enable all authentication processes to be recorded transparently and traceably on the blockchain. A system of more transparently and reliable certificates may also be established and managed. Moreover, the security of certificates can be enhanced against fraudulent activities such as counterfeiting and tampering due to the immutability of data on the blockchain.

Finally, an independent design and management of the identity authentication process can be achieved as a result of the decentralised nature of Smart Contracts. In order to ensure security and reliability, this prevents a single central authority from controlling all processes.

4.2 Performance Benchmarking

When examining the methods employed and the results obtained in the comparative analysis described in Section 4.1, it is observed that the conducted tests yield similar outcomes in terms of performance evaluation. Furthermore, in [64] , [65] and [66] research articles, comparisons between smart contracts and SSL/TLS have been carried out, and similar results have been reported in those studies. As a result, it has been found that low energy consumption is the major factor in Internet of things applications and their results are confirmed through performance tests.

In the pursuit of performance analysis for the system under consideration, an MQTT broker and its associated clients have been developed in alignment with the proposed plan. The authentication and authorization system of the evolved MQTT, in addition to the authentication conducted via the TLS SSL channel, have been juxtaposed against a proprietary method. After the authorization process, data transmission to the broker was progressively carried out with 50 publisher users.

Upon examining the data in Figure 20-21, it is observed that the implemented approach leads to a reduction in memory usage by approximately 34.54% compared to TLS. For internet of things applications such a result can be regarded as favourable.

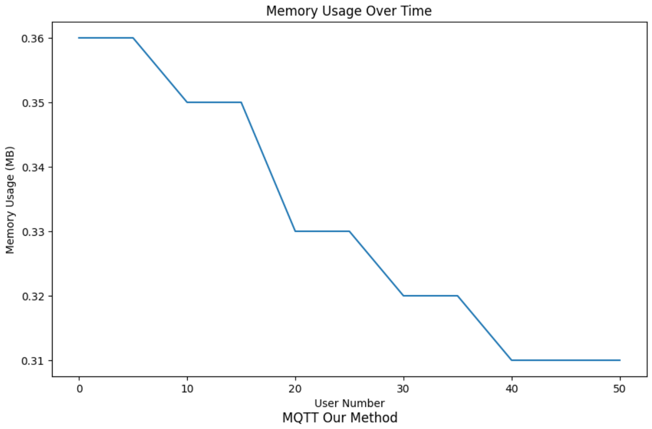


Figure 20: Our Method Memory Usage

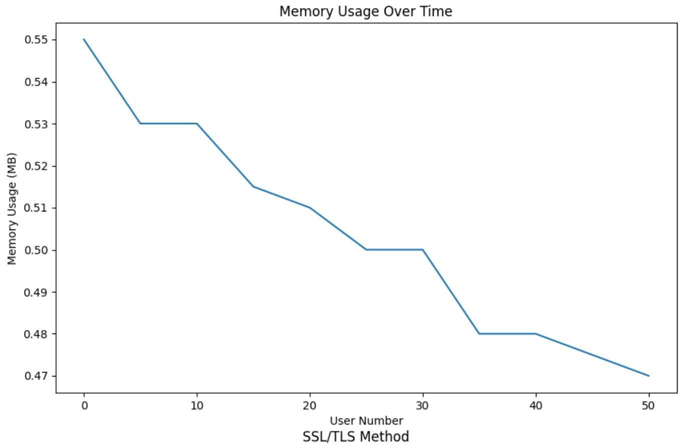


Figure 21:SSL/TLS Memory Usage

Additionally, in Figure 22-23, it can be seen that the computational overhead of TLS reaches approximately 3.4% of the CPU, while our approach's CPU overhead is around 1.190%. As a result, in terms of CPU overhead, our approach shows approximately 65 % less CPU workload compared to TLS.

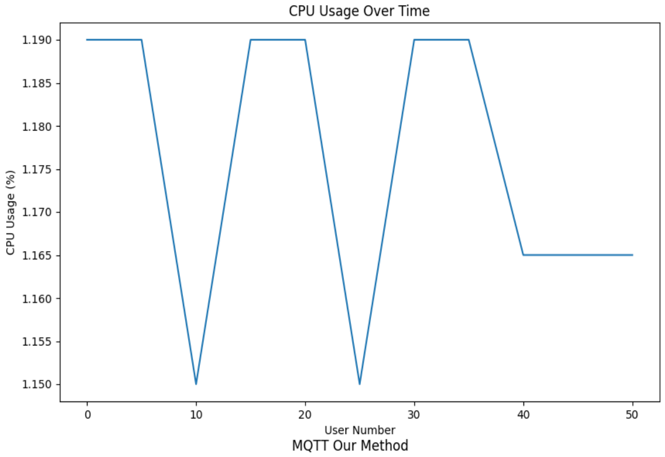


Figure 22:Our Method CPU Usage

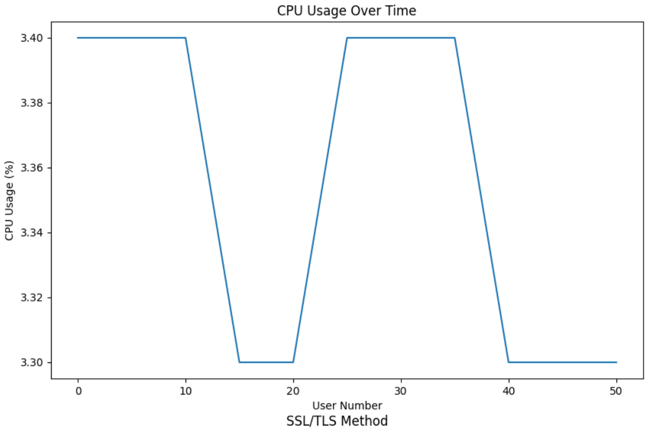
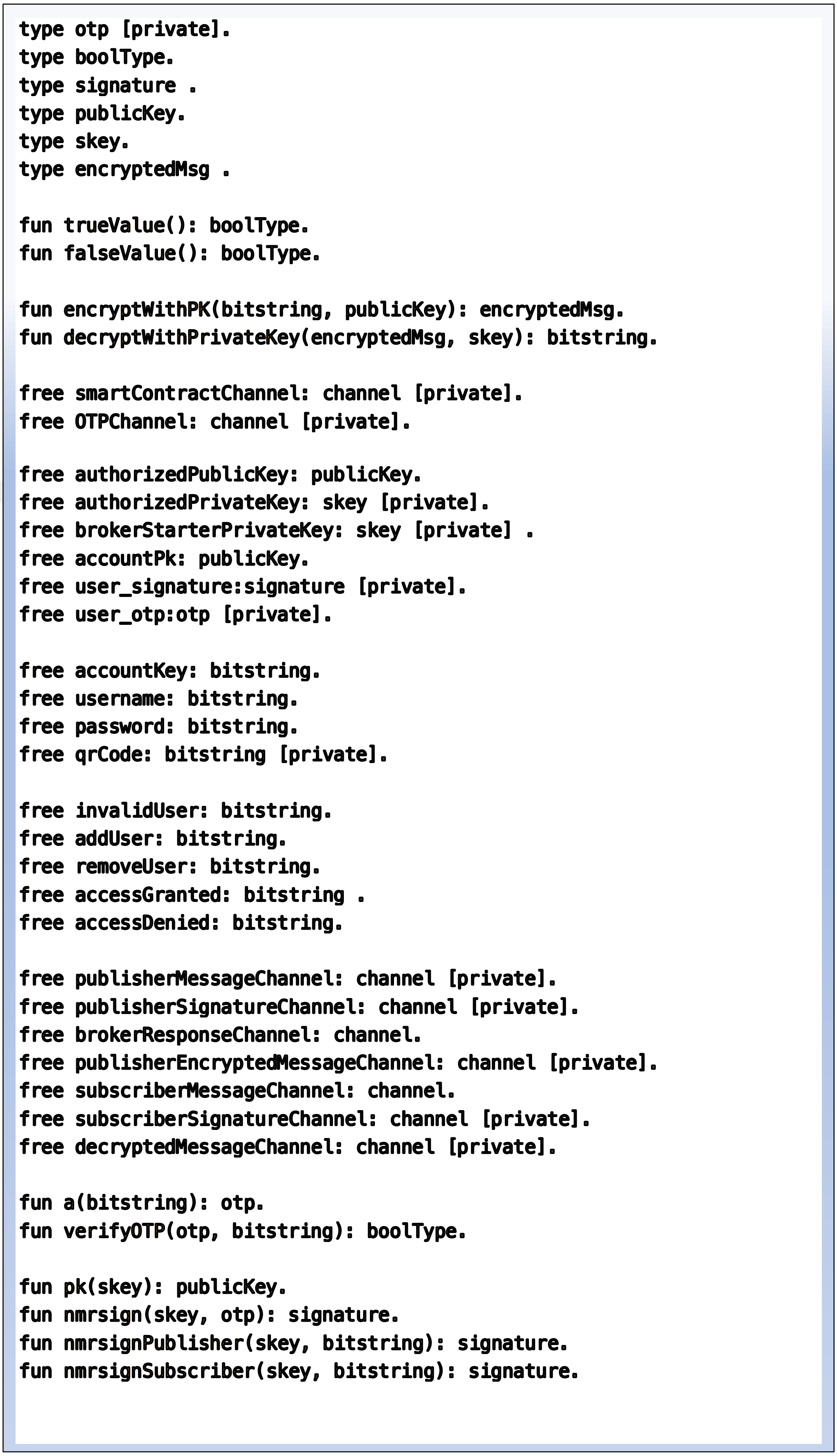


Figure 23:SSL/TLS CPU Usage

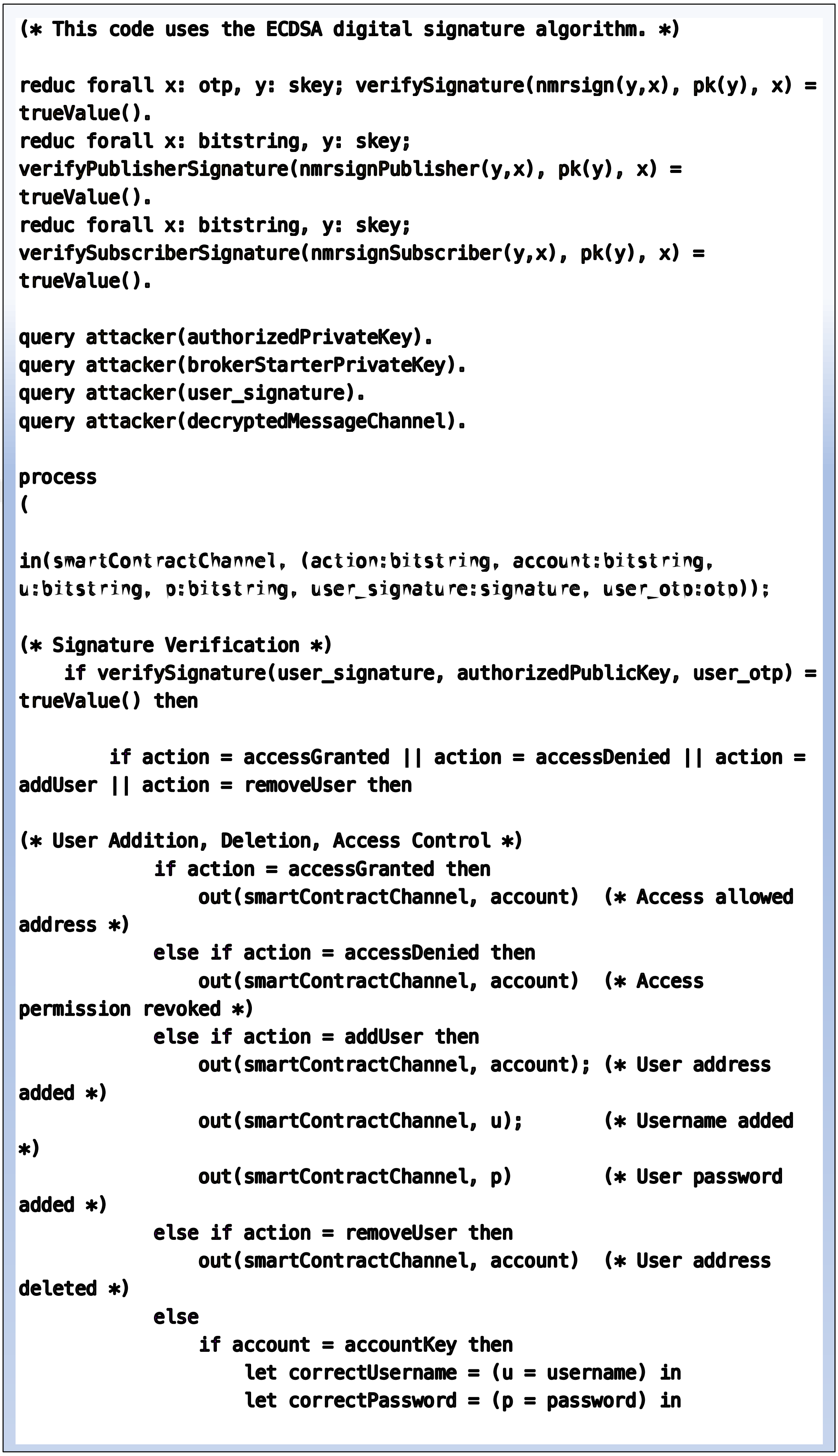
4.3 Security Analysis

In the digital era, information security and cryptographic measures are vital for all technological solutions. This section discusses the thorough cryptographic assessment of 'MQTT Otp Blockchain-based Identity and Data Verification' using renowned tools such as ProVerif and Scyther. [67] [68]

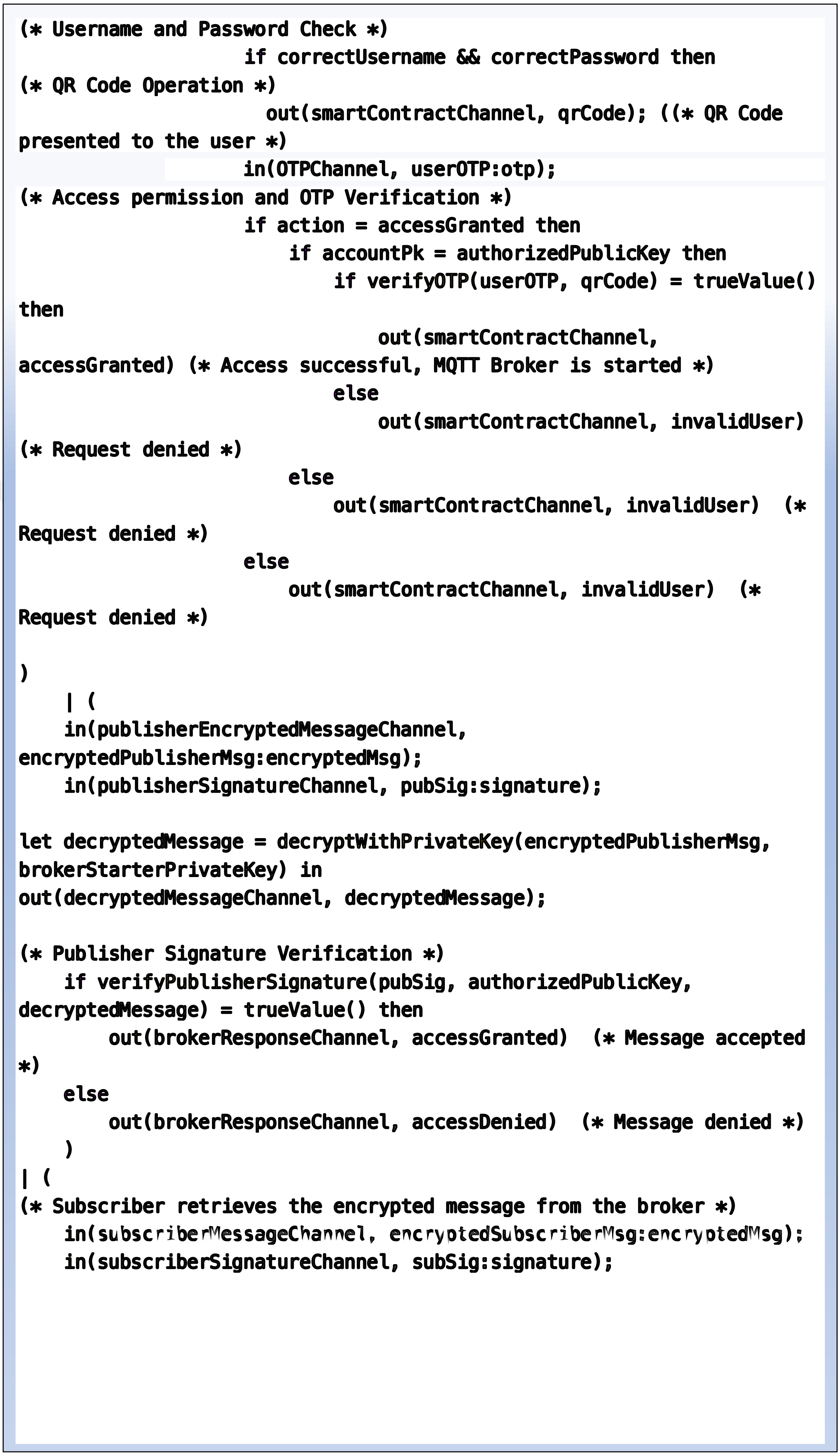
Firsly, ProVerif is recognized as an instrumental tool tailored for the automatic verification of protocols. It is intended to verify the integrity of security features which are inherent in a specific protocol. Through an exhaustive exploration of all conceivable execution pathways of such protocols, ProVerif discerns whether the stipulated security attributes are consistently upheld. With this context, an in-depth scrutiny of the underlying code and the subsequent results derived from this analysis will be presented.



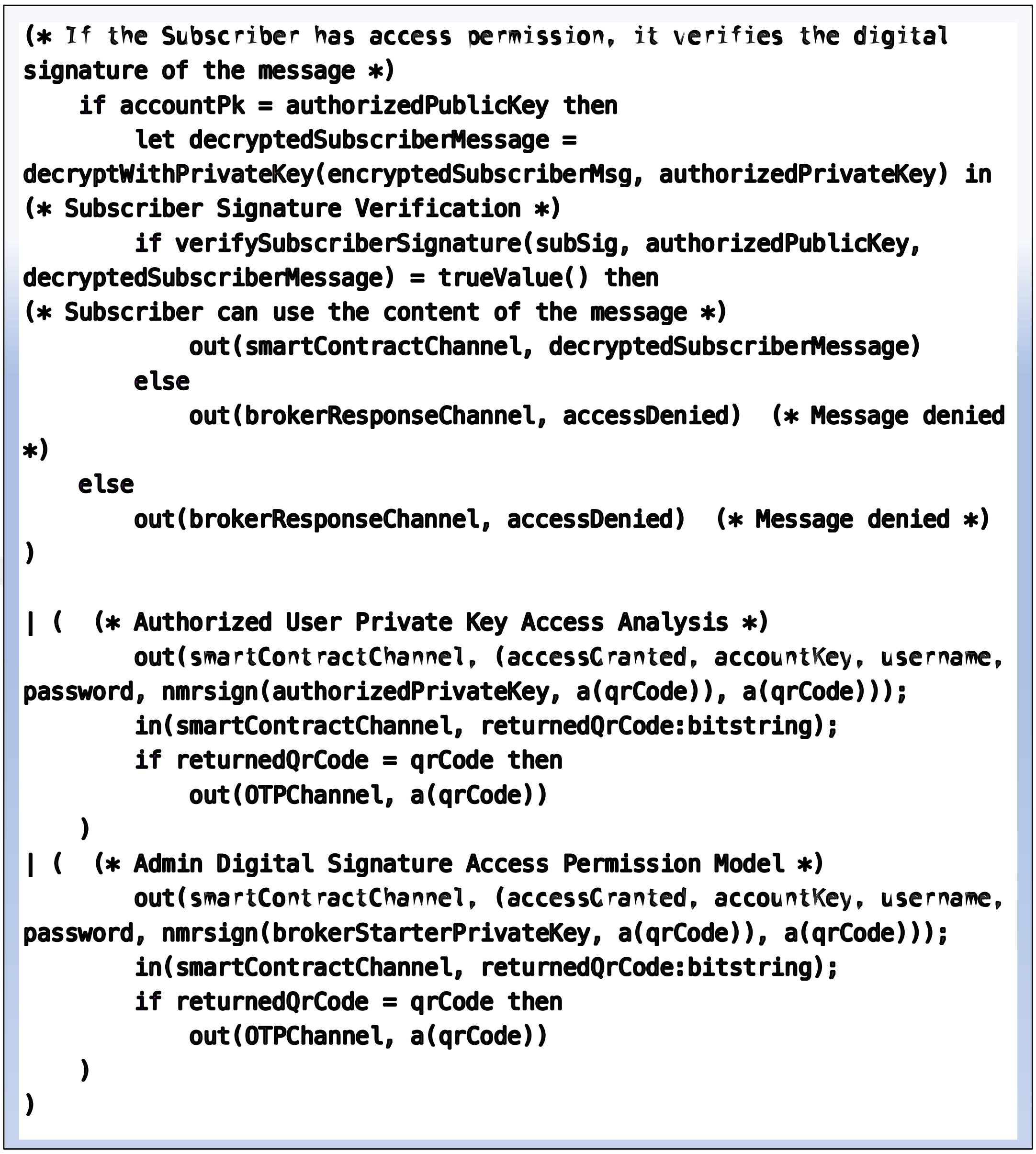
```
type otp [private].
type boolType.
type signature .
type publicKey.
type skey.
type encryptedMsg .

fun trueValue(): boolType.
fun falseValue(): boolType.

fun encryptWithPK(bitstring, publicKey): encryptedMsg.
fun decryptWithPrivateKey(encryptedMsg, skey): bitstring.

free smartContractChannel: channel [private].
free OTPChannel: channel [private].

free authorizedPublicKey: publicKey.
free authorizedPrivateKey: skey [private].
free brokerStarterPrivateKey: skey [private] .
free accountPk: publicKey.
free user_signature:signature [private].
free user_otp:otp [private].

free accountKey: bitstring.
free username: bitstring.
free password: bitstring.
free qrCode: bitstring [private].

free invalidUser: bitstring.
free addUser: bitstring.
free removeUser: bitstring.
free accessGranted: bitstring .
free accessDenied: bitstring.

free publisherMessageChannel: channel [private].
free publisherSignatureChannel: channel [private].
free brokerResponseChannel: channel.
free publisherEncryptedMessageChannel: channel [private].
free subscriberMessageChannel: channel.
free subscriberSignatureChannel: channel [private].
free decryptedMessageChannel: channel [private].

fun a(bitstring): otp.
fun verifyOTP(otp, bitstring): boolType.

fun pk(skey): publicKey.
fun nmrsign(skey, otp): signature.
fun nmrsignPublisher(skey, bitstring): signature.
fun nmrsignSubscriber(skey, bitstring): signature.
```



```
(* This code uses the ECDSA digital signature algorithm. *)

reduc forall x: otp, y: skey; verifySignature(nmrsign(y,x), pk(y), x) =
trueValue().
reduc forall x: bitstring, y: skey;
verifyPublisherSignature(nmrsignPublisher(y,x), pk(y), x) =
trueValue().
reduc forall x: bitstring, y: skey;
verifySubscriberSignature(nmrsignSubscriber(y,x), pk(y), x) =
trueValue().

query attacker(authorizedPrivateKey).
query attacker(brokerStarterPrivateKey).
query attacker(user_signature).
query attacker(decryptedMessageChannel).

process
(

in(smartContractChannel, (action:bitstring, account:bitstring,
u:bitstring, p:bitstring, user_signature:signature, user_otp:otp));

(* Signature Verification *)
    if verifySignature(user_signature, authorizedPublicKey, user_otp) =
trueValue() then

        if action = accessGranted || action = accessDenied || action =
addUser || action = removeUser then

(* User Addition, Deletion, Access Control *)
            if action = accessGranted then
                out(smartContractChannel, account)  (* Access allowed
address *)
            else if action = accessDenied then
                out(smartContractChannel, account)  (* Access
permission revoked *)
            else if action = addUser then
                out(smartContractChannel, account); (* User address
added *)
                out(smartContractChannel, u);       (* Username added
*)
                out(smartContractChannel, p)        (* User password
added *)
            else if action = removeUser then
                out(smartContractChannel, account)  (* User address
deleted *)
            else
                if account = accountKey then
                    let correctUsername = (u = username) in
                    let correctPassword = (p = password) in
```



```
(* Username and Password Check *)
                    if correctUsername && correctPassword then
(* QR Code Operation *)
                      out(smartContractChannel, qrCode); ((* QR Code
presented to the user *)
                    in(OTPChannel, userOTP:otp);
(* Access permission and OTP Verification *)
                    if action = accessGranted then
                        if accountPk = authorizedPublicKey then
                            if verifyOTP(userOTP, qrCode) = trueValue()
then
                                out(smartContractChannel,
accessGranted) (* Access successful, MQTT Broker is started *)
                            else
                                out(smartContractChannel, invalidUser)
(* Request denied *)
                        else
                            out(smartContractChannel, invalidUser)  (*
Request denied *)
                    else
                        out(smartContractChannel, invalidUser)  (*
Request denied *)

)
    | (
    in(publisherEncryptedMessageChannel,
encryptedPublisherMsg:encryptedMsg);
    in(publisherSignatureChannel, pubSig:signature);

let decryptedMessage = decryptWithPrivateKey(encryptedPublisherMsg,
brokerStarterPrivateKey) in
out(decryptedMessageChannel, decryptedMessage);

(* Publisher Signature Verification *)
    if verifyPublisherSignature(pubSig, authorizedPublicKey,
decryptedMessage) = trueValue() then
        out(brokerResponseChannel, accessGranted)  (* Message accepted
*)
    else
        out(brokerResponseChannel, accessDenied)  (* Message denied *)
    )
| (
(* Subscriber retrieves the encrypted message from the broker *)
    in(subscriberMessageChannel, encryptedSubscriberMsg:encryptedMsg);
    in(subscriberSignatureChannel, subSig:signature);
```

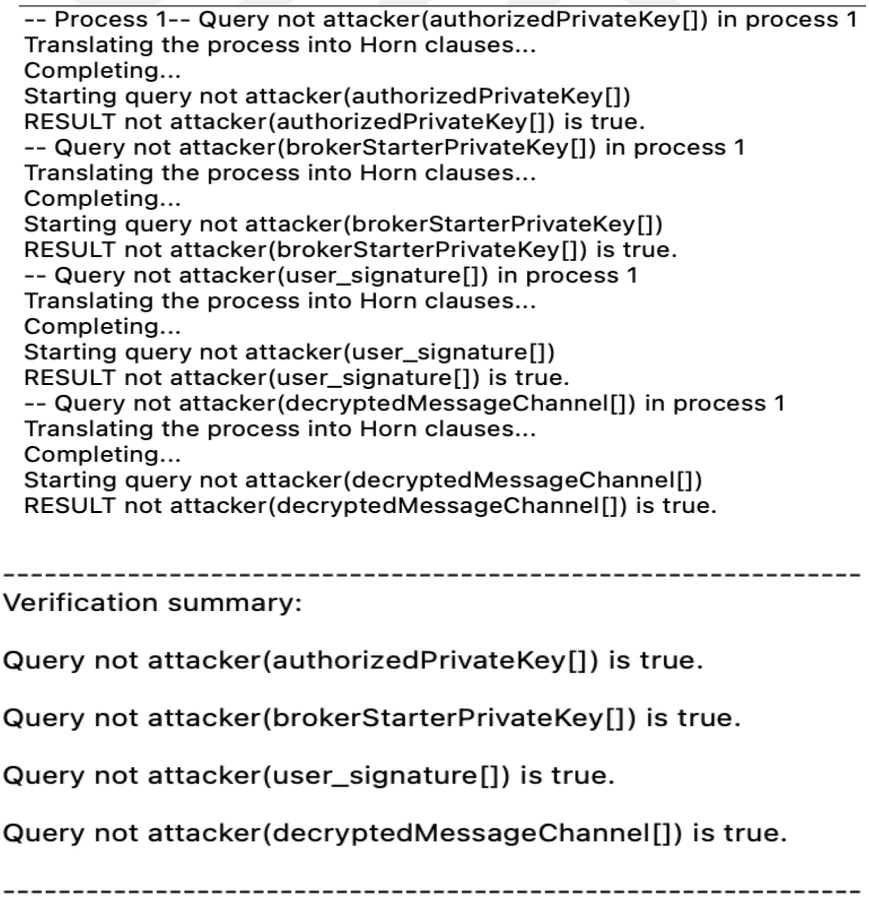


```
(* If the Subscriber has access permission, it verifies the digital
signature of the message *)
    if accountPk = authorizedPublicKey then
        let decryptedSubscriberMessage =
decryptWithPrivateKey(encryptedSubscriberMsg, authorizedPrivateKey) in
(* Subscriber Signature Verification *)
        if verifySubscriberSignature(subSig, authorizedPublicKey,
decryptedSubscriberMessage) = trueValue() then
(* Subscriber can use the content of the message *)
            out(smartContractChannel, decryptedSubscriberMessage)
        else
            out(brokerResponseChannel, accessDenied)  (* Message denied
*)
    else
        out(brokerResponseChannel, accessDenied)  (* Message denied *)
)

| (  (* Authorized User Private Key Access Analysis *)
        out(smartContractChannel, (accessGranted, accountKey, username,
password, nmrsign(authorizedPrivateKey, a(qrCode)), a(qrCode)));
        in(smartContractChannel, returnedQrCode:bitstring);
        if returnedQrCode = qrCode then
            out(OTPChannel, a(qrCode))
    )
| (  (* Admin Digital Signature Access Permission Model *)
        out(smartContractChannel, (accessGranted, accountKey, username,
password, nmrsign(brokerStarterPrivateKey, a(qrCode)), a(qrCode)));
        in(smartContractChannel, returnedQrCode:bitstring);
        if returnedQrCode = qrCode then
            out(OTPChannel, a(qrCode))
    )
)
```

Critical information about our protocol's reliability and robustness was gained from an extensive security analysis carried out via ProVerif. It was our primary objective to ensure the security and integrity of private keys, digital signatures or messages in this protocol.

It was determined that the authorization processes are under admin control, and these controls are safeguarded in a way that attackers cannot breach. Unauthorized attempts to start the broker were detected and blocked, proving these attack attempts were unsuccessful. The integrity of the digital signatures in the system was fully preserved, and it was ascertained that these signatures are secure against unauthorized modifications. It was proven that decrypted messages were not displayed to unauthorized individuals and these messages can only be accessed by authorized users.

The results obtained clearly show that a security protocol we have implemented has been effectively and safely applicable in all key areas as described in Figure 24.

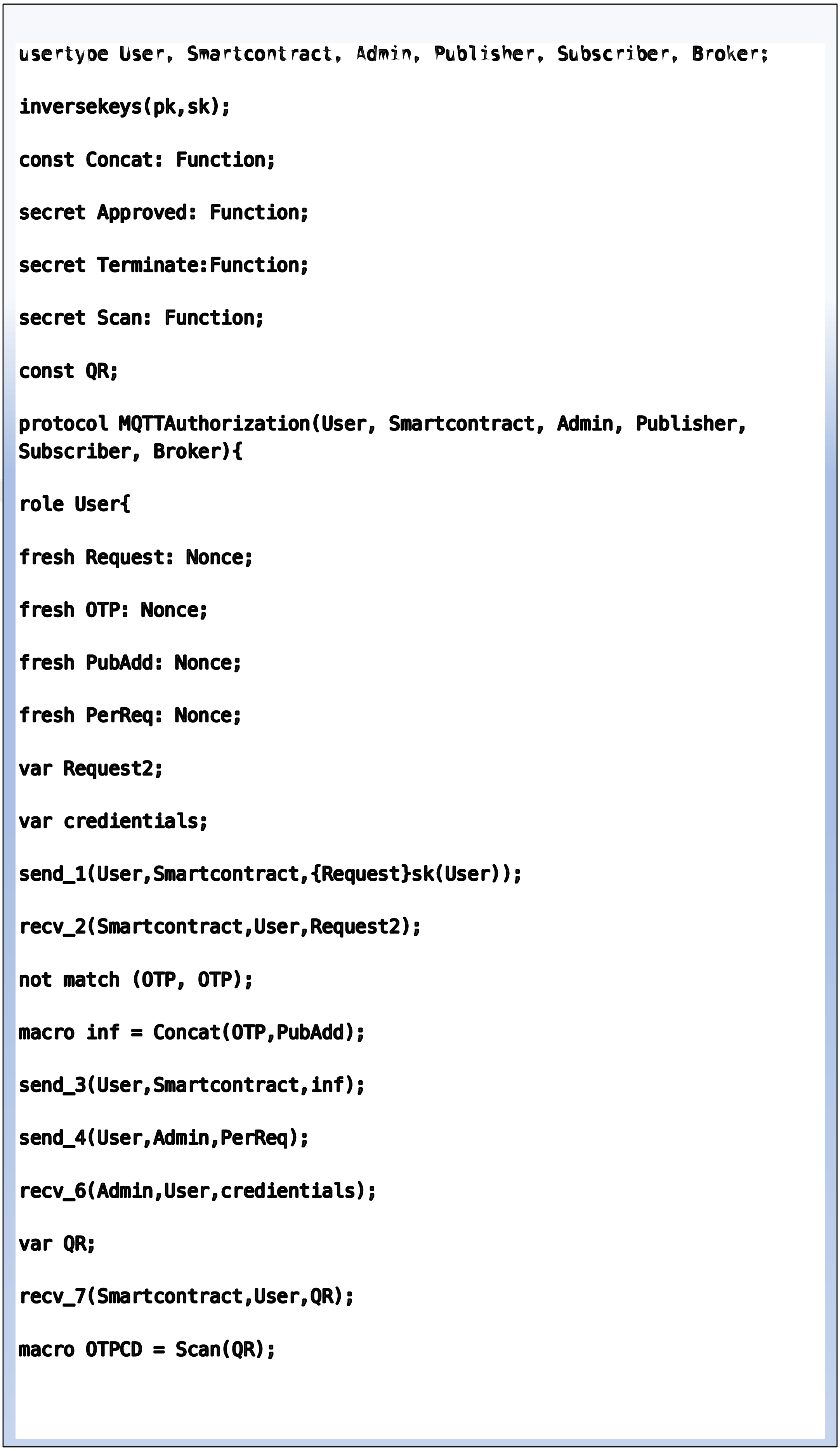


```
-- Process 1-- Query not attacker(authorizedPrivateKey[]) in process 1
Translating the process into Horn clauses...
Completing...
Starting query not attacker(authorizedPrivateKey[])
RESULT not attacker(authorizedPrivateKey[]) is true.
-- Query not attacker(brokerStarterPrivateKey[]) in process 1
Translating the process into Horn clauses...
Completing...
Starting query not attacker(brokerStarterPrivateKey[])
RESULT not attacker(brokerStarterPrivateKey[]) is true.
-- Query not attacker(user_signature[]) in process 1
Translating the process into Horn clauses...
Completing...
Starting query not attacker(user_signature[])
RESULT not attacker(user_signature[]) is true.
-- Query not attacker(decryptedMessageChannel[]) in process 1
Translating the process into Horn clauses...
Completing...
Starting query not attacker(decryptedMessageChannel[])
RESULT not attacker(decryptedMessageChannel[]) is true.

--------------------------------------------------------------
Verification summary:

Query not attacker(authorizedPrivateKey[]) is true.

Query not attacker(brokerStarterPrivateKey[]) is true.

Query not attacker(user_signature[]) is true.

Query not attacker(decryptedMessageChannel[]) is true.

--------------------------------------------------------------
```

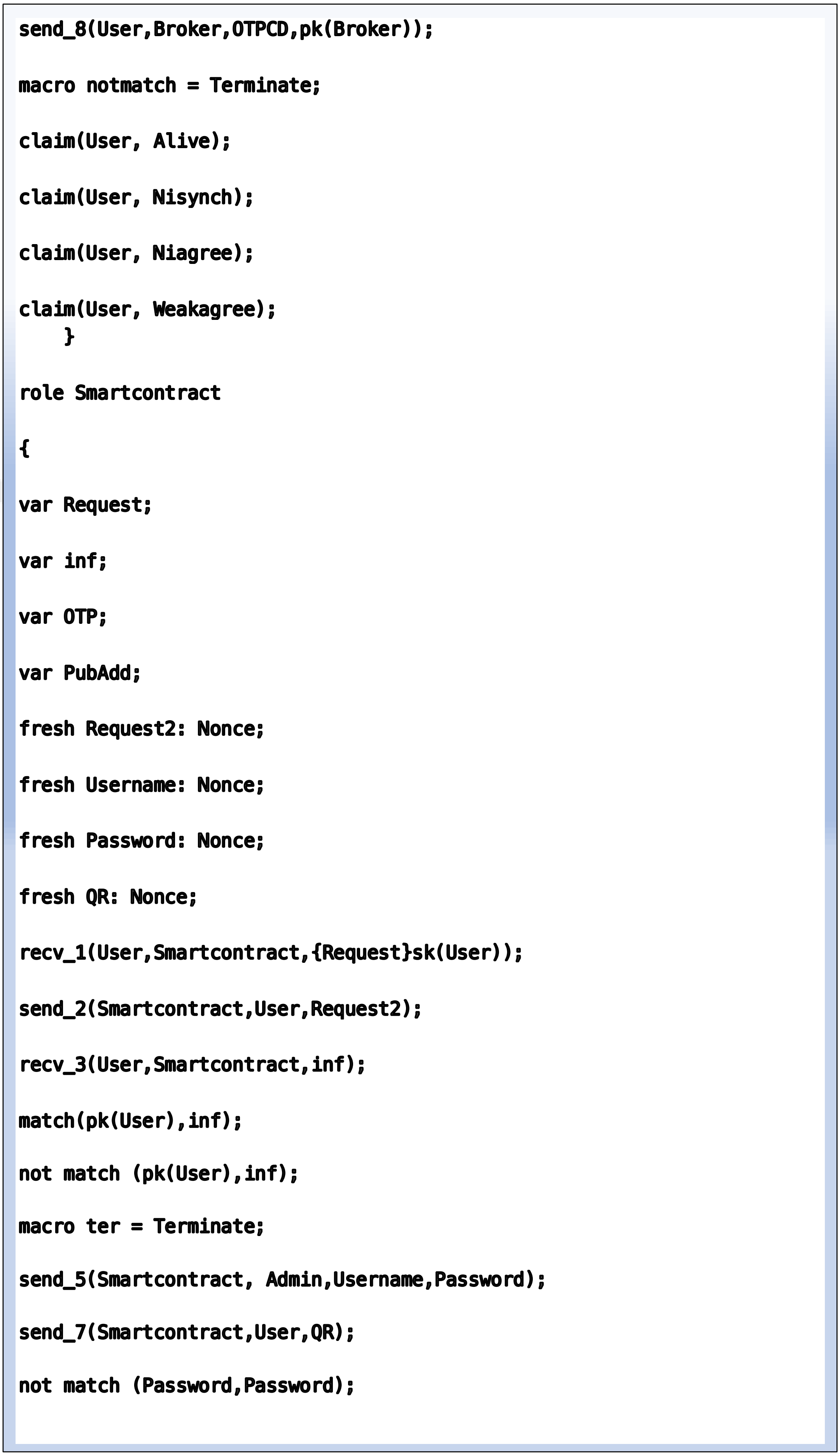
Figure 24:Results of the security analysis conducted with ProVerif.

Following ProVerif, Scyther, a tool that automates the security analysis of protocols, was utilized to evaluate the security of the MQTTAuthorization protocol. This code delineates how the MQTT Authorization protocol interacts with various roles and operations, such as User, Smartcontract, Admin, Publisher, Subscriber, and Broker. The protocol covers a number of processes such as digital signature validation, data transmission using encryption keys and authorizations from different roles.

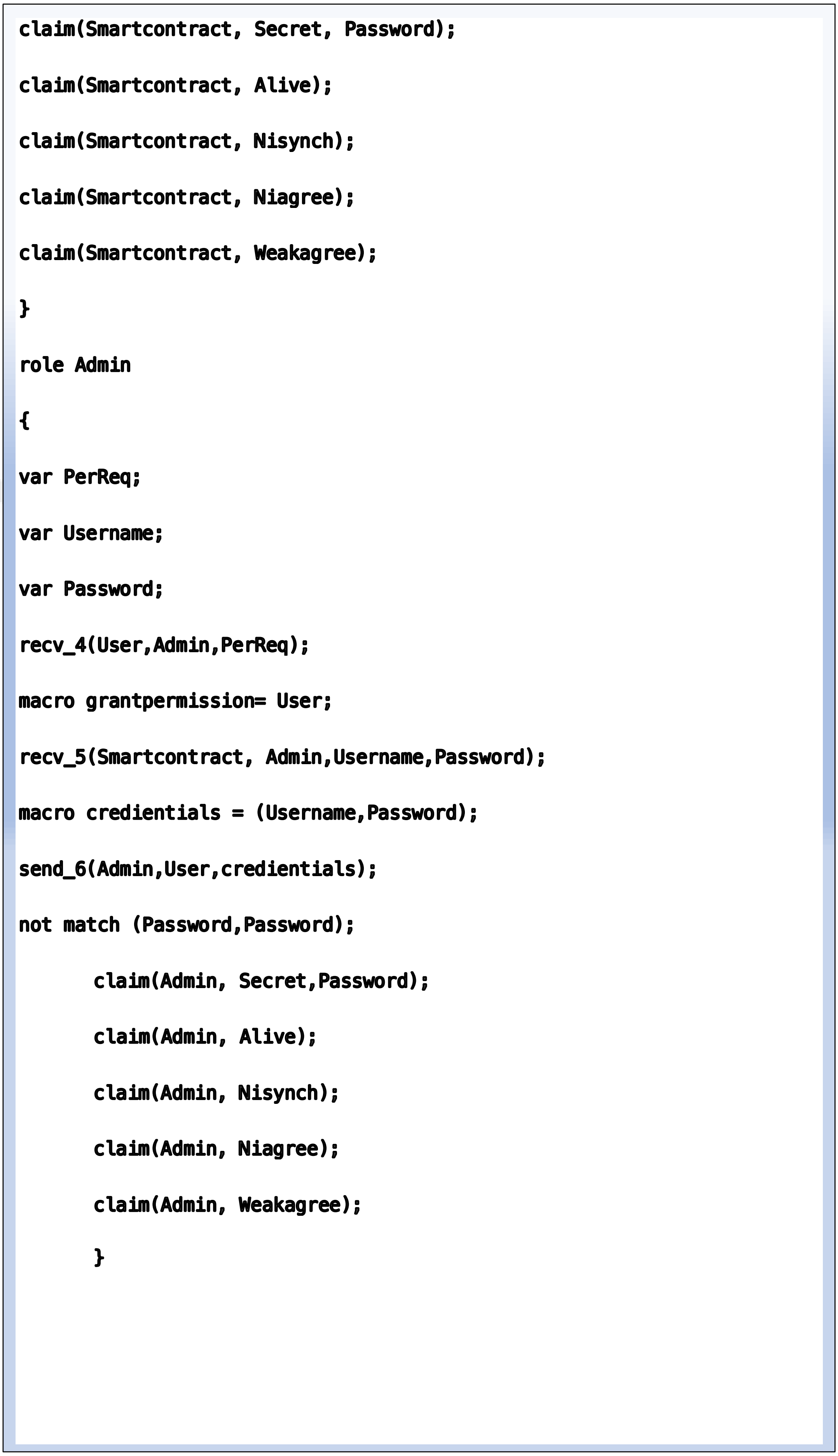
Interactions between commands are represented by 'send' and 'recv', which indicate data transfer between two parties. The 'match' and 'not match' statements check the correctness of digital signatures and other data. 'Macro' statements display the outcomes of specific operations and definitions; whereas 'claim' statements test specific security attributes and states of the protocol. In Scyther, 'claims' are used to test security attributes and potential attacks. For instance, 'Nisynch' checks against replay attacks, while 'Niagree' guarantees messages are securely and correctly transmitted. 'Alive' confirms that the protocol steps are approved by both parties, and 'weakagree' ensures an attacker cannot impersonate another entity. The secrecy or 'secret' claim confirms the safety of information exchanged during communication.



```
usertype User, Smartcontract, Admin, Publisher, Subscriber, Broker;

inversekeys(pk,sk);

const Concat: Function;

secret Approved: Function;

secret Terminate:Function;

secret Scan: Function;

const QR;

protocol MQTTAuthorization(User, Smartcontract, Admin, Publisher,
Subscriber, Broker){

role User{

fresh Request: Nonce;

fresh OTP: Nonce;

fresh PubAdd: Nonce;

fresh PerReq: Nonce;

var Request2;

var credientials;

send_1(User,Smartcontract,{Request}sk(User));

recv_2(Smartcontract,User,Request2);

not match (OTP, OTP);

macro inf = Concat(OTP,PubAdd);

send_3(User,Smartcontract,inf);

send_4(User,Admin,PerReq);

recv_6(Admin,User,credientials);

var QR;

recv_7(Smartcontract,User,QR);

macro OTPCD = Scan(QR);
```

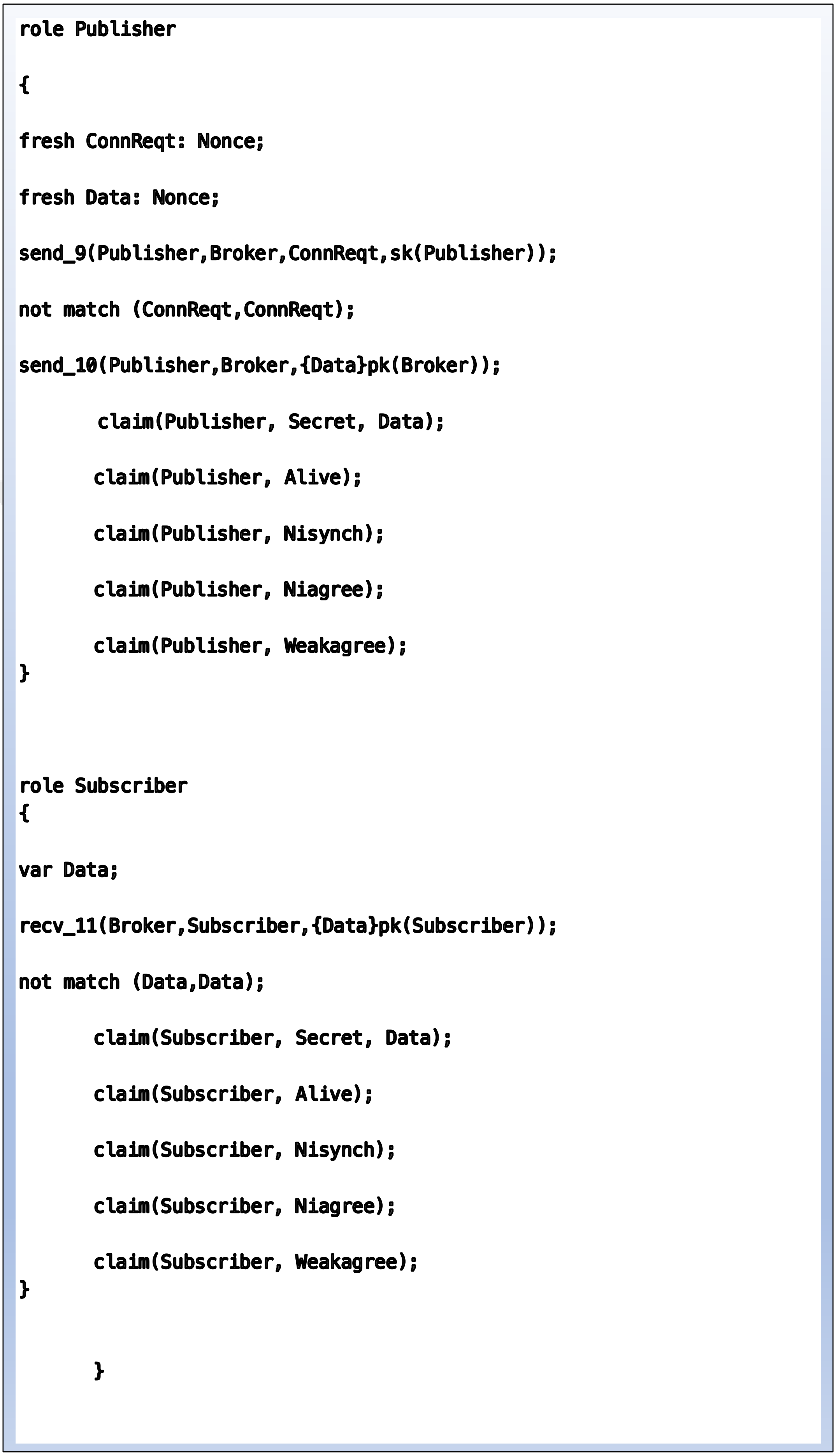


```
send_8(User,Broker,OTPCD,pk(Broker));

macro notmatch = Terminate;

claim(User, Alive);

claim(User, Nisynch);

claim(User, Niagree);

claim(User, Weakagree);
    }

role Smartcontract

{

var Request;

var inf;

var OTP;

var PubAdd;

fresh Request2: Nonce;

fresh Username: Nonce;

fresh Password: Nonce;

fresh QR: Nonce;

recv_1(User,Smartcontract,{Request}sk(User));

send_2(Smartcontract,User,Request2);

recv_3(User,Smartcontract,inf);

match(pk(User),inf);

not match (pk(User),inf);

macro ter = Terminate;

send_5(Smartcontract, Admin,Username,Password);

send_7(Smartcontract,User,QR);

not match (Password,Password);
```



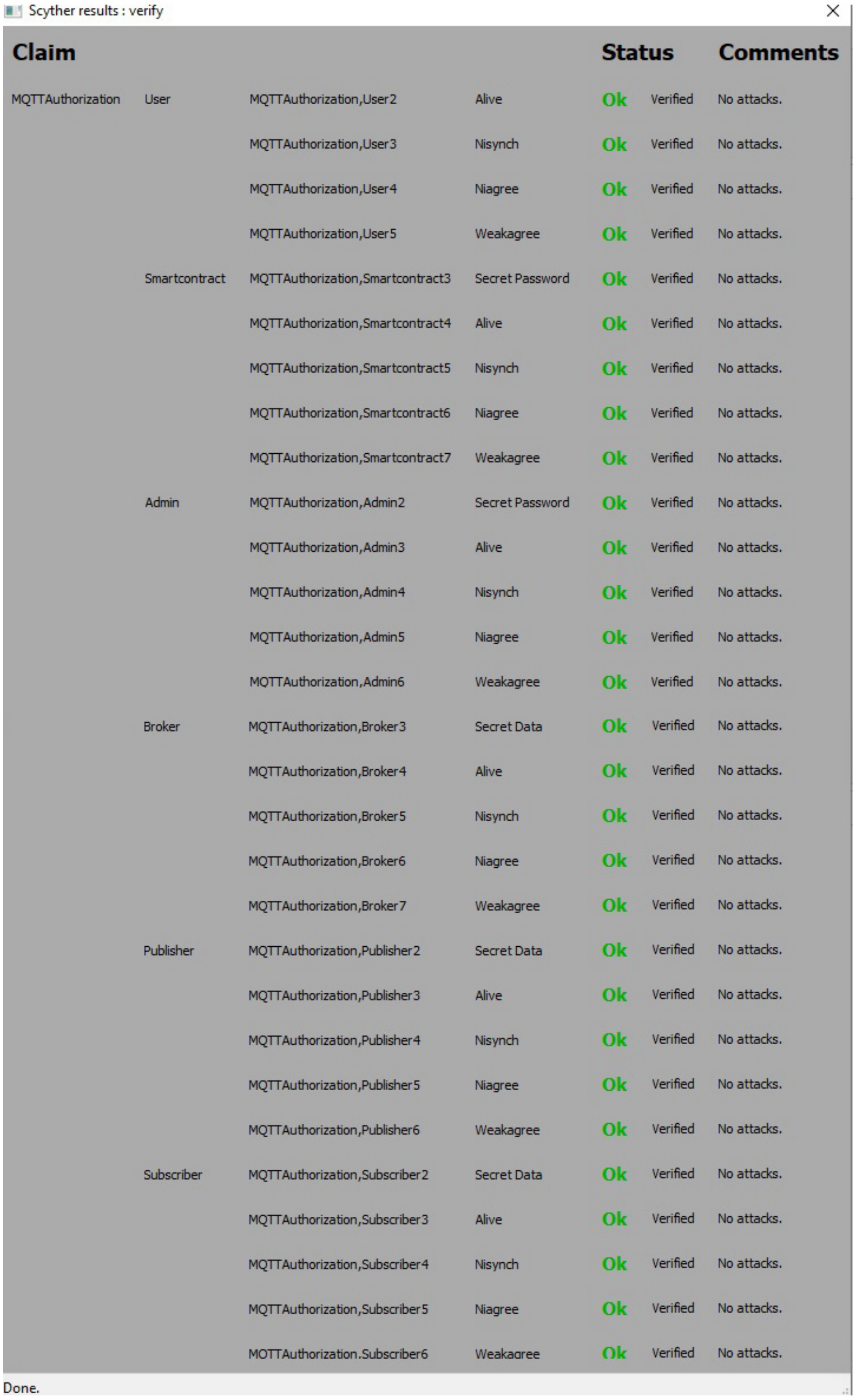
```
claim(Smartcontract, Secret, Password);

claim(Smartcontract, Alive);

claim(Smartcontract, Nisynch);

claim(Smartcontract, Niagree);

claim(Smartcontract, Weakagree);

}

role Admin

{

var PerReq;

var Username;

var Password;

recv_4(User,Admin,PerReq);

macro grantpermission= User;

recv_5(Smartcontract, Admin,Username,Password);

macro credientials = (Username,Password);

send_6(Admin,User,credientials);

not match (Password,Password);

      claim(Admin, Secret,Password);

      claim(Admin, Alive);

      claim(Admin, Nisynch);

      claim(Admin, Niagree);

      claim(Admin, Weakagree);

      }
```

```
role Broker

{

var OTPCD;

var ConnReqt;

var Data;

recv_8(User,Broker,OTPCD,pk(Broker));

recv_9(Publisher,Broker,ConnReqt,sk(Publisher));

macro data = (OTPCD,pk(Broker));

match(data,pk(Publisher));

macro correct = Approved;

not match (data,pk(Publisher));

not match (OTPCD,OTPCD);

macro notcorrect=Terminate;

recv_10(Publisher,Broker,{Data}pk(Broker));

send_11(Broker,Subscriber,{Data}pk(Subscriber));

       claim(Broker, Secret, Data);

      claim(Broker, Alive);

      claim(Broker, Nisynch);

      claim(Broker, Niagree);

      claim(Broker, Weakagree);


      }
```



```
role Publisher

{

fresh ConnReqt: Nonce;

fresh Data: Nonce;

send_9(Publisher,Broker,ConnReqt,sk(Publisher));

not match (ConnReqt,ConnReqt);

send_10(Publisher,Broker,{Data}pk(Broker));

       claim(Publisher, Secret, Data);

       claim(Publisher, Alive);

       claim(Publisher, Nisynch);

       claim(Publisher, Niagree);

       claim(Publisher, Weakagree);
}



role Subscriber
{

var Data;

recv_11(Broker,Subscriber,{Data}pk(Subscriber));

not match (Data,Data);

       claim(Subscriber, Secret, Data);

       claim(Subscriber, Alive);

       claim(Subscriber, Nisynch);

       claim(Subscriber, Niagree);

       claim(Subscriber, Weakagree);
}


       }
```

Based on the Scyther analysis Figure 25, it has been determined that all roles defined in the protocol are secure. This indicates that effective safeguards are in place to protect data signature, keys and authorization processes as set out in the Protocol. Given the role that authorization processes play in thwarting access, from potential attackers these findings hold immense value.



Scyther results : verify

Claim				Status		Comments
MQTTAuthorization	User	MQTTAuthorization,User2	Alive	Ok	Verified	No attacks.
		MQTTAuthorization,User3	Nisynch	Ok	Verified	No attacks.
		MQTTAuthorization,User4	Niagree	Ok	Verified	No attacks.
		MQTTAuthorization,User5	Weakagree	Ok	Verified	No attacks.
	Smartcontract	MQTTAuthorization,Smartcontract3	Secret Password	Ok	Verified	No attacks.
		MQTTAuthorization,Smartcontract4	Alive	Ok	Verified	No attacks.
		MQTTAuthorization,Smartcontract5	Nisynch	Ok	Verified	No attacks.
		MQTTAuthorization,Smartcontract6	Niagree	Ok	Verified	No attacks.
		MQTTAuthorization,Smartcontract7	Weakagree	Ok	Verified	No attacks.
	Admin	MQTTAuthorization,Admin2	Secret Password	Ok	Verified	No attacks.
		MQTTAuthorization,Admin3	Alive	Ok	Verified	No attacks.
		MQTTAuthorization,Admin4	Nisynch	Ok	Verified	No attacks.
		MQTTAuthorization,Admin5	Niagree	Ok	Verified	No attacks.
		MQTTAuthorization,Admin6	Weakagree	Ok	Verified	No attacks.
	Broker	MQTTAuthorization,Broker3	Secret Data	Ok	Verified	No attacks.
		MQTTAuthorization,Broker4	Alive	Ok	Verified	No attacks.
		MQTTAuthorization,Broker5	Nisynch	Ok	Verified	No attacks.
		MQTTAuthorization,Broker6	Niagree	Ok	Verified	No attacks.
		MQTTAuthorization,Broker7	Weakagree	Ok	Verified	No attacks.
	Publisher	MQTTAuthorization,Publisher2	Secret Data	Ok	Verified	No attacks.
		MQTTAuthorization,Publisher3	Alive	Ok	Verified	No attacks.
		MQTTAuthorization,Publisher4	Nisynch	Ok	Verified	No attacks.
		MQTTAuthorization,Publisher5	Niagree	Ok	Verified	No attacks.
		MQTTAuthorization,Publisher6	Weakagree	Ok	Verified	No attacks.
	Subscriber	MQTTAuthorization,Subscriber2	Secret Data	Ok	Verified	No attacks.
		MQTTAuthorization,Subscriber3	Alive	Ok	Verified	No attacks.
		MQTTAuthorization,Subscriber4	Nisynch	Ok	Verified	No attacks.
		MQTTAuthorization,Subscriber5	Niagree	Ok	Verified	No attacks.
		MQTTAuthorization,Subscriber6	Weakagree	Ok	Verified	No attacks.

Done.

Figure 25:Results of the security analysis conducted with Scyther

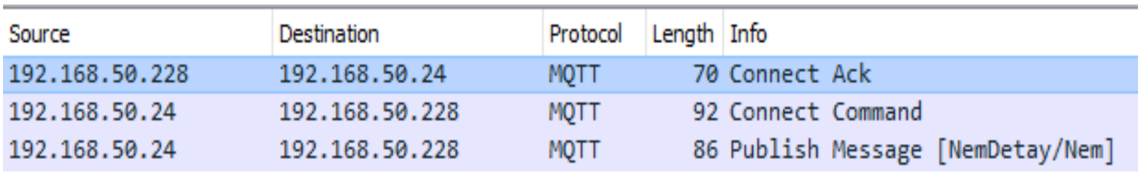
5. SECURITY MECHANISMS

The MQTT protocol, which was created for devices, with resources has become vulnerable to security issues due, to its simplicity and low processing power requirements. The low level of processing to transmit messages is the cause of these vulnerabilities. A number of studies have been carried out in the literature with a view to detecting and mitigating these vulnerabilities [69]. In general, when MQTT protocol is implemented with default settings, it lacks several security mechanisms. In order to ensure safety, configuration of settings is required. Even with active security settings, it cannot be claimed that MQTT applications provide complete security, as they remain susceptible to various attacks.

Additional security measures such as username, password, or client certificates should be used for user authentication in MQTT. However, client information such as client identifier, username, password, and client certificates can be collected or authentication information can be guessed, enabling unauthorized access to a rogue MQTT client. Malicious clients accessing the MQTT service using fake identities can publish unauthorized messages and subscribe to unauthorized messages. This can lead to the leakage of sensitive information, sending unauthorised control instructions for Internet of Things devices and causing damage to infrastructure or individuals using such devices. Furthermore, the messages of publishers and subscribers can be eavesdropped by unauthorised traders posing as legitimate brokers.

In 2019, Alaiz Moreton, in 2020, Ivan Vaccari, and most recently in 2021, Danish Javeed [70], created datasets for the detection of attacks in MQTT networks [71] [72]. Based on the findings derived from these datasets it has been observed that the MQTT network is susceptible, to forms of attacks. These include Denial of Service (DoS) Brute Force, Malformed, Flood, SlowIte and Man, in the Middle (MitM) attacks.

Network events can be examined using Wireshark [73]. During a network analysis, the sequence of operations in which a broker attempts to publish data to topics can be observed, as shown in Figure 26. When an MQTT transmission starts, Connect and Connect Ack messages can be observed, and when data transmission begins, Publish Message messages emerge. Upon receiving this message, the broker forwards it to the subscribers of the topic. Wireshark has the ability to capture and analyze network traffic allowing us to observe these operations. In addition this approach will greatly simplify the comprehension and transmission of data through MQTT. By utilizing Wireshark for network traffic analysis, events in an MQTT-based network can be examined, and security vulnerabilities or issues can be identified.



Source	Destination	Protocol	Length	Info
192.168.50.228	192.168.50.24	MQTT	70	Connect Ack
192.168.50.24	192.168.50.228	MQTT	92	Connect Command
192.168.50.24	192.168.50.228	MQTT	86	Publish Message [NemDetay/Nem]

Figure 26:MQTT connection analyses

Connect and Connect Ack are message types used in the MQTT protocol to facilitate communication between the client and the broker. The Connect message is used by the client to send a connection request to the broker. In this message, the client specifies its credentials, the version of the MQTT protocol, client identifier, and connection parameters. After receiving the Connect message, the broker examines the client's credentials and connection parameters for authentication.

The Connect Ack (Connect Acknowledge) message, on the other hand, is a response sent by the broker. The broker verifies the client and either approves or rejects the connection request after receiving the Connect message. The Connect Ack message informs the client about the connection status and may include additional connection parameters. If the connection is successful, the client can begin performing MQTT publications upon receiving this message.

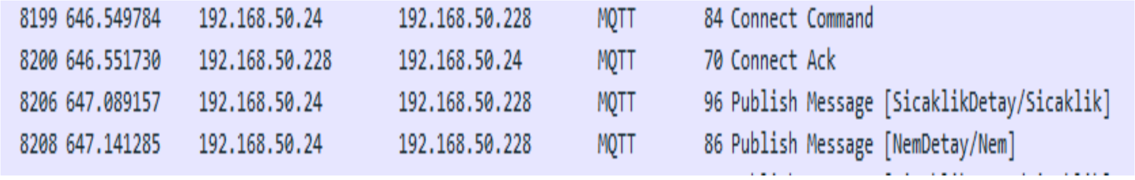
As suggested in the "System" section, higher levels of security can be achieved through the use of applications. This can be accomplished through two mechanisms: Authentication and Authorization. Authentication and authorization shall be the means by which one or more entities can prove identity to another entity, while authorisation refers to a mechanism for granting or rejecting access to services based on an individual's identity.

In addition, in order to increase security against hackers, a solution can be provided by combining technology, consensus algorithms and distributed architecture along with the use of blockchain public keys, efficient usage of smart contracts or integrated one time passwordsOTPs. The architecture of the blockchain allows this method to create a database, while at the same time allowing network members to agree by means of consensus algorithms. Secure authentication and encryption while protecting data confidentiality and integrity shall be ensured through the use of Digital Signatures and Public Keyes. By automatically Automating business procedures with their features, smart contracts offer the potential to increase security. Furthermore incorporating one time passwords (OTP) into the authentication process enhances security by generating a password for every session adding a level of defense, against potential attacks. The combination of a coherent solution and the user centered approach will result in an efficient strategy to counter different attacks. Any user wishing to publish or subscribe must first have access permission and possess OTP information, and they should also enter their private key information to decrypt the digital signature.

5.1. The Absence of Authentication Mechanisms for User Verification

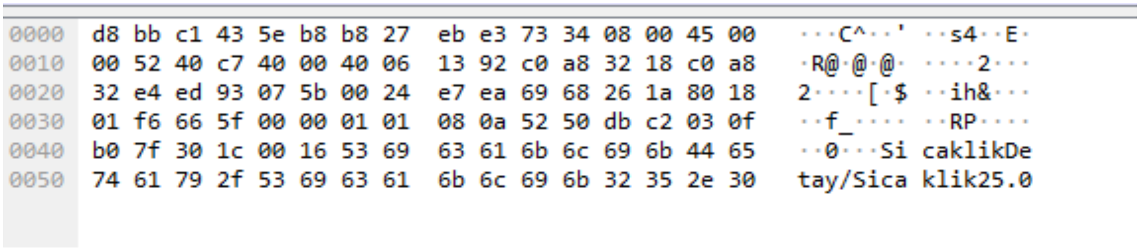
In situations where an authentication mechanism is not present, any client has the capability to publish and/or subscribe to any given topic, consequently creating a significant threat to data privacy. Data managed by brokers, or intermediaries, may encompass sensitive information, which can be accessed merely by supplying the corresponding topic name. The absence of identity verification measures and the lack of restrictions on data publication and subscription to registered organizations increase the risk.

As shown in Figures 27 and 28, it can be observed that MQTT publication starts on the network and, as demonstrated by malicious programmers, the simplest example such as accessing temperature values can be easily examined. This situation is not limited to temperature values but applies to any data in a factory. Considering the need for data protection and the importance of confidentiality in SCADA systems, this situation is an unacceptable drawback.



```
8199 646.549784   192.168.50.24    192.168.50.228   MQTT   84 Connect Command
8200 646.551730   192.168.50.228   192.168.50.24    MQTT   70 Connect Ack
8206 647.089157   192.168.50.24    192.168.50.228   MQTT   96 Publish Message [SicaklikDetay/Sicaklik]
8208 647.141285   192.168.50.24    192.168.50.228   MQTT   86 Publish Message [NemDetay/Nem]
```

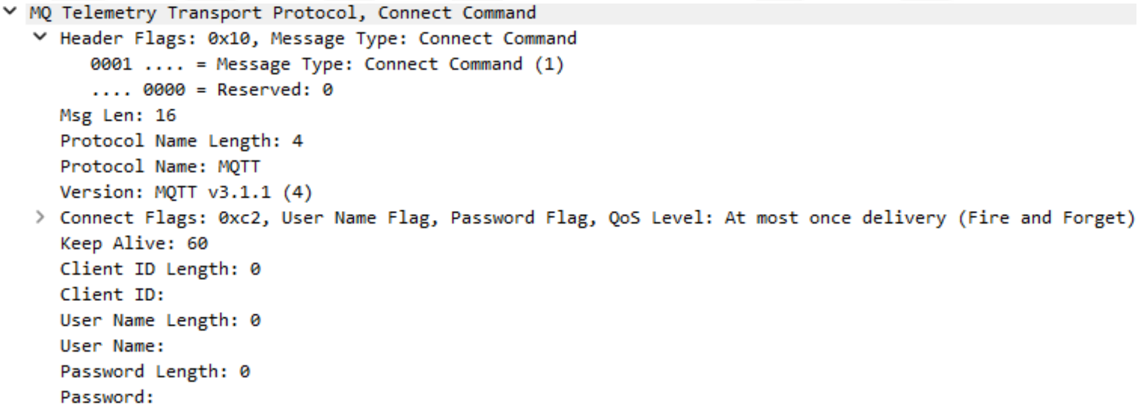
Figure 27:MQTT data transmission started



```
0000  d8 bb c1 43 5e b8 b8 27  eb e3 73 34 08 00 45 00   ···C^··' ··s4··E·
0010  00 52 40 c7 40 00 40 06  13 92 c0 a8 32 18 c0 a8   ·R@·@·@· ····2···
0020  32 e4 ed 93 07 5b 00 24  e7 ea 69 68 26 1a 80 18   2····[·$ ··ih&···
0030  01 f6 66 5f 00 00 01 01  08 0a 52 50 db c2 03 0f   ··f_···· ··RP····
0040  b0 7f 30 1c 00 16 53 69  63 61 6b 6c 69 6b 44 65   ··0···Si caklikDe
0050  74 61 79 2f 53 69 63 61  6b 6c 69 6b 32 35 2e 30   tay/Sica klik25.0
```

Figure 28:Easy access to data through network MQTT broadcast analysis

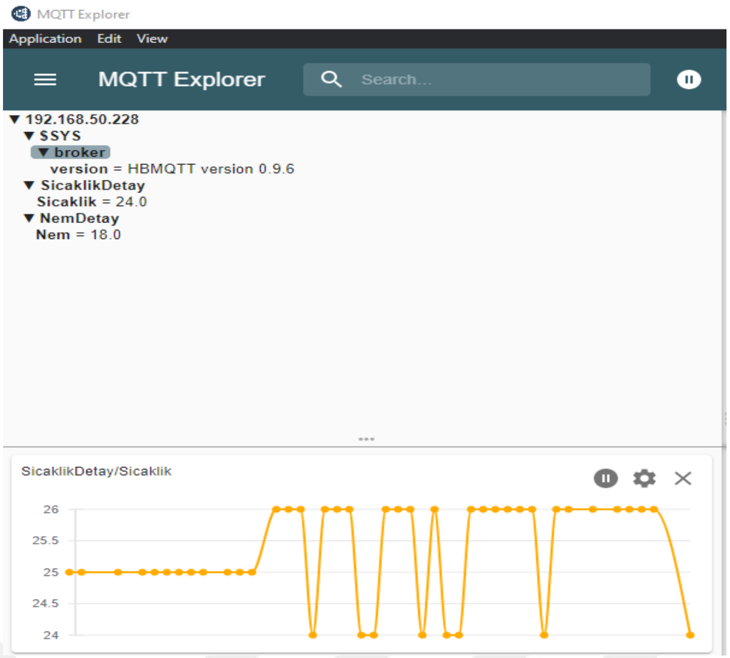
In Figure 29, another examination revealed the absence of a username and password mechanism for MQTT in the network. Not specifying a username and password in an MQTT publication where the port and IP address are automatically known is a widely adopted method worldwide [74]. It is evident how incorrect and detrimental this situation can be. Access to the system can be obtained within seconds.



```
v MQ Telemetry Transport Protocol, Connect Command
  v Header Flags: 0x10, Message Type: Connect Command
      0001 .... = Message Type: Connect Command (1)
      .... 0000 = Reserved: 0
    Msg Len: 16
    Protocol Name Length: 4
    Protocol Name: MQTT
    Version: MQTT v3.1.1 (4)
  > Connect Flags: 0xc2, User Name Flag, Password Flag, QoS Level: At most once delivery (Fire and Forget)
    Keep Alive: 60
    Client ID Length: 0
    Client ID:
    User Name Length: 0
    User Name:
    Password Length: 0
    Password:
```

Figure 29:Connection with a Null Username and Password

As a result of all these findings, it has been demonstrated that unauthorized access and monitoring of data can easily be achieved by establishing connections to any MQTT service (such as MQTT Explorer [75]). Based on the findings shown in Figure 30 it can be inferred that the system does not possess a security mechanism.



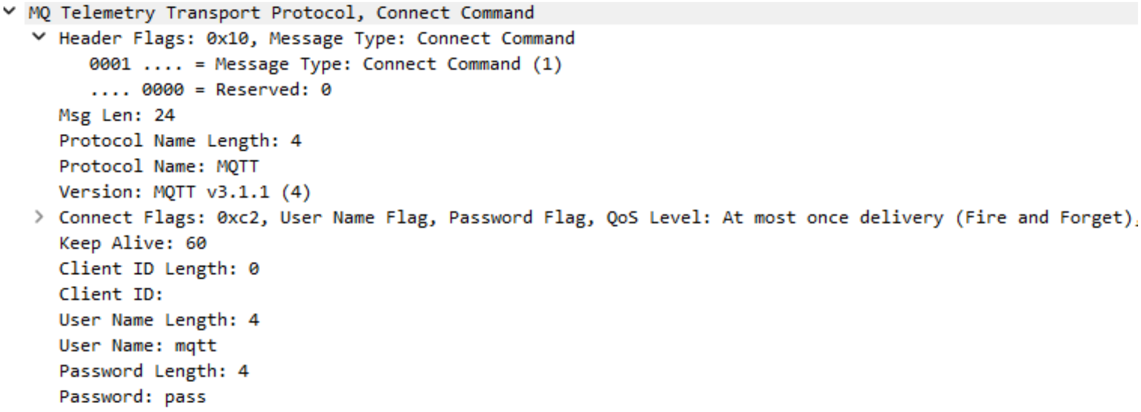
```
▼ 192.168.50.228
  ▼ $SYS
    ▼ broker
      version = HBMQTT version 0.9.6
  ▼ SicaklikDetay
    Sicaklik = 24.0
  ▼ NemDetay
    Nem = 18.0
```

Figure 30:Easy data access with MQTT explorer

5.2 Transmitting User Credentials in Plain Text within Connect Command Packets

The MQTT protocol is designed to be connection-oriented, and insufficient attention has been given to its security aspects. If an approved authentication mechanism is enabled, the data cannot be accessed by unauthorised users. In the second scenario, even if you create a username and password for MQTT connection authorization, your credentials are forwarded in plain text to allow an attacker to impersonate legitimate users that would be able to access their data using intercepted packets. In Figure 31, when examining the MQTT publication through the Wireshark application, it appears that the username is "mqtt" and the password is "pass"

There is a chance for the values to be manipulated in this scenario. A detailed connection command packet reveals the username and password in plain text, potentially exposing them to unauthorized access. The risk of data manipulation due to this vulnerability is increased.



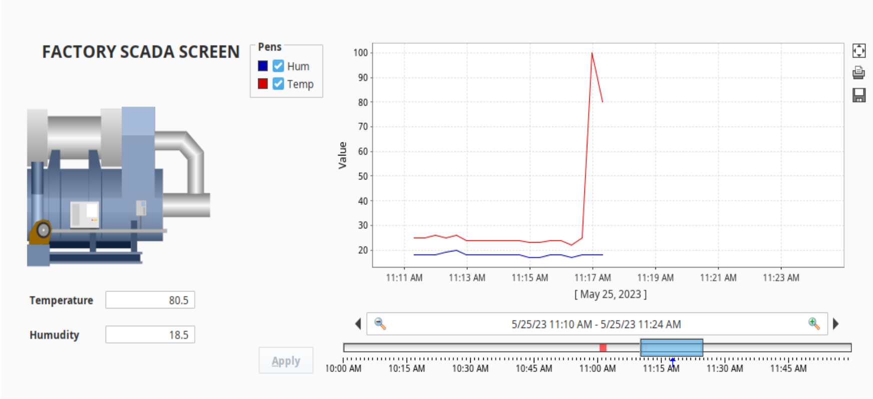
```
v MQ Telemetry Transport Protocol, Connect Command
  v Header Flags: 0x10, Message Type: Connect Command
      0001 .... = Message Type: Connect Command (1)
      .... 0000 = Reserved: 0
    Msg Len: 24
    Protocol Name Length: 4
    Protocol Name: MQTT
    Version: MQTT v3.1.1 (4)
  > Connect Flags: 0xc2, User Name Flag, Password Flag, QoS Level: At most once delivery (Fire and Forget),
    Keep Alive: 60
    Client ID Length: 0
    Client ID:
    User Name Length: 4
    User Name: mqtt
    Password Length: 4
    Password: pass
```

Figure 31:Exposed MQTT credentials in wireshark

A potential attack scenario example can be considered with a SCADA system, assuming the usage of Inductive Automation's [76] Ignition SCADA software. In this scenario, a SCADA operator is conducting a system review without being aware that a malicious programmer has gained unauthorized access to the system.

SCADA supervisory control and data acquisition systems are designed to track and monitor industrial processes, such as electricity transmission grids, production facilities or other vital infrastructure.

Once the attacker captures the connection packet and obtains the credentials, it is sufficient to impersonate a legitimate client. The broker doesn't even know about the attack. As shown in Figure 32, the attacker manipulates the “transformer's” temperature value in a malicious manner. As a result, continuous alarms are triggered in the system. Due to the lack of understanding of the emergencies and the root cause throughout the factory, there may be disruptions in production. As a result this leads to effects, on both the operational aspects resulting in tangible and intangible disadvantages.



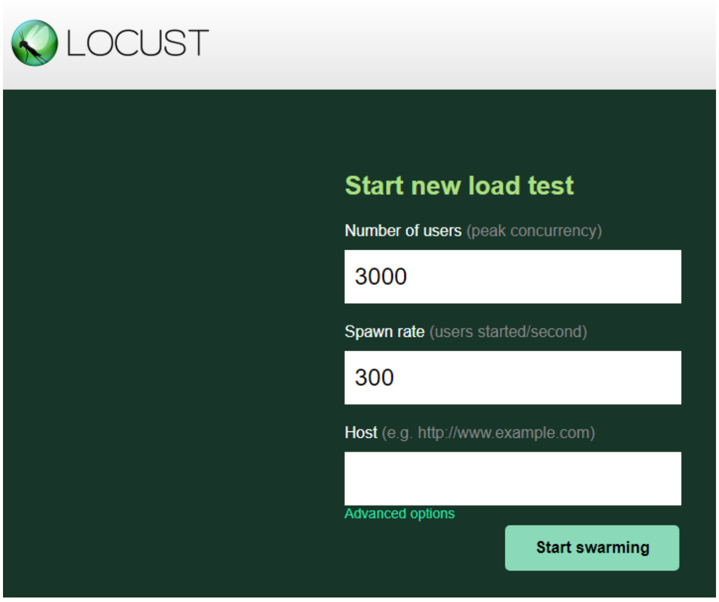
Temperature 80.5

Humudity 18.5

Figure 32: Manipulated scada data

DDoS and Sybil attacks can appear directly as MQTT publishers and both access the system by sending data intensively and enable manipulated data flow. Because from all these network analyses, it has now discovered how to access the Broker.

For example, in Figure 33 Dos Attack and Sybil attack scenarios were performed. Python plays a role, in the analysis process. Particularly, the Locust package, used for modeling attack scenarios and performance measurements, constitutes one of the cornerstones of our project. This package allows the possibility to carry out experiments which may be repeated.



Start new load test

Number of users (peak concurrency)

3000

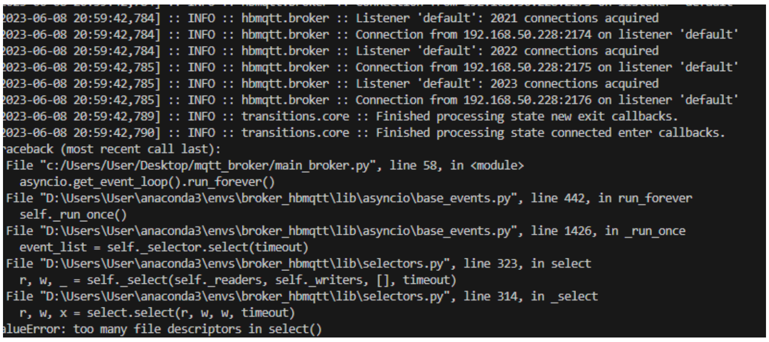
Spawn rate (users started/second)

300

Host (e.g. http://www.example.com)

Figure 33: Attack scenario

As seen here, the Broker system has automatically errored due to excessive network load, or as in Figure 34 it has been subject to manipulation resulting in non-genuine errors occurring on the system. Due to the excessive data access attempts on my website, the system eventually encountered a timeout error.



```
2023-06-08 20:59:42,784] :: INFO :: hbmqtt.broker :: Listener 'default': 2021 connections acquired
2023-06-08 20:59:42,784] :: INFO :: hbmqtt.broker :: Connection from 192.168.50.228:2174 on listener 'default'
2023-06-08 20:59:42,784] :: INFO :: hbmqtt.broker :: Listener 'default': 2022 connections acquired
2023-06-08 20:59:42,785] :: INFO :: hbmqtt.broker :: Connection from 192.168.50.228:2175 on listener 'default'
2023-06-08 20:59:42,785] :: INFO :: hbmqtt.broker :: Listener 'default': 2023 connections acquired
2023-06-08 20:59:42,785] :: INFO :: hbmqtt.broker :: Connection from 192.168.50.228:2176 on listener 'default'
2023-06-08 20:59:42,789] :: INFO :: transitions.core :: Finished processing state new exit callbacks.
2023-06-08 20:59:42,790] :: INFO :: transitions.core :: Finished processing state connected enter callbacks.
raceback (most recent call last):
 File "c:/Users/User/Desktop/mqtt_broker/main_broker.py", line 58, in <module>
   asyncio.get_event_loop().run_forever()
 File "D:\Users\User\anaconda3\envs\broker_hbmqtt\lib\asyncio\base_events.py", line 442, in run_forever
   self._run_once()
 File "D:\Users\User\anaconda3\envs\broker_hbmqtt\lib\asyncio\base_events.py", line 1426, in _run_once
   event_list = self._selector.select(timeout)
 File "D:\Users\User\anaconda3\envs\broker_hbmqtt\lib\selectors.py", line 323, in select
   r, w, _ = self._select(self._readers, self._writers, [], timeout)
 File "D:\Users\User\anaconda3\envs\broker_hbmqtt\lib\selectors.py", line 314, in _select
   r, w, x = select.select(r, w, w, timeout)
alueError: too many file descriptors in select()
```

Figure 34:Mqtt broker timeout after attack

SSL or TLS only introduces encryption to the recommended mechanism here. It has no impact on the Subscribe or Publish data transmission. Therefore, scaling and control mechanisms in IoT systems fail. Advancing this with smart contracts is much better. For authentication of user and device identities, the Blockchain can use Digital Signatories as well as Public and Private Keypairs. This will enable the identification of who has performed each transaction and prevent unauthorized users from accessing this system. When certain conditions are fulfilled, the Smart Contract may be executed on its own. For instance, a smart contract can determine whether a user has the authority to publish or subscribe to certain topics. Data protection is also ensured by the Blockchain. Data protection for transactions and data is possible with digital signatures and encryption techniques. The protection of the confidentiality of user data that are not authorised must also be ensured.

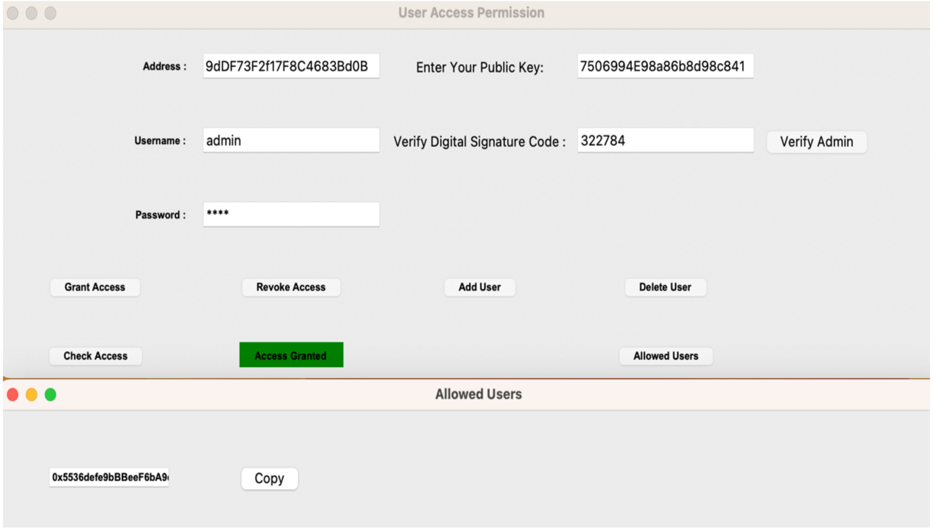
By employing Transport Layer Security(TLS) or Secure Sockets Layer (SSL) over port 8883 (the standard port used for secure MQTT connections), packets between the client and the broker are transmitted over a secure communication channel. This is not sufficient, however. A challenge in using this mechanism arises when the client frequently reconnects, as the overhead required for maintaining the session becomes significant. Therefore, it is not recommended to use this mechanism in cases of frequent reconnection.

The solution proposed shall include the use of blockchain technology as well as digital signature with user authentication mechanisms. In order to preserve the integrity of data exchanged and to prevent authentication procedures from going wrong, digitally signed documents play an essential role. Moreover the utilization of the ECDSA algorithm, for generating signatures can efficiently tackle concerns related to identity security. The security of that process is further enhanced by the immutable and transparent nature of the blockchain.

5.3 Blockchain Authentication Mechanism to Authenticate

A smart contract was created to control whether users have access permission to the MQTT broker, and this contract has a mapping structure that holds the information about which users can access which data.

Users are added solely by admin privileges using smart contract functions, as depicted in Figure 35. The desired username and password are determined accordingly. The dates on which users are granted access permission are also shared. As a result, you can easily scale your users for MQTT in mere seconds. This is not a feature typically found in MQTT Brokers. Now a permitted user is defined in the system. Users can be configured as desired. Admin user's identity is verified using digital signatures. An OTP (One-Time Password) that changes each time is generated using ECDSA. To authenticate the digital signature, the user needs to input the OTP and the correct public key known only to the admin user, as indicated in the visual. This shall make it possible to ensure a reliable authentication of the user's administration account, as well as prevent unauthorised access. Since the smart contract, ECDSA, OTP and MQTT are integrated with each other, every operation performed here automatically customizes the connection settings.



User Access Permission

Address : 9dDF73F2f17F8C4683Bd0B

Enter Your Public Key: 7506994E98a86b8d98c841

Username : admin

Verify Digital Signature Code : 322784

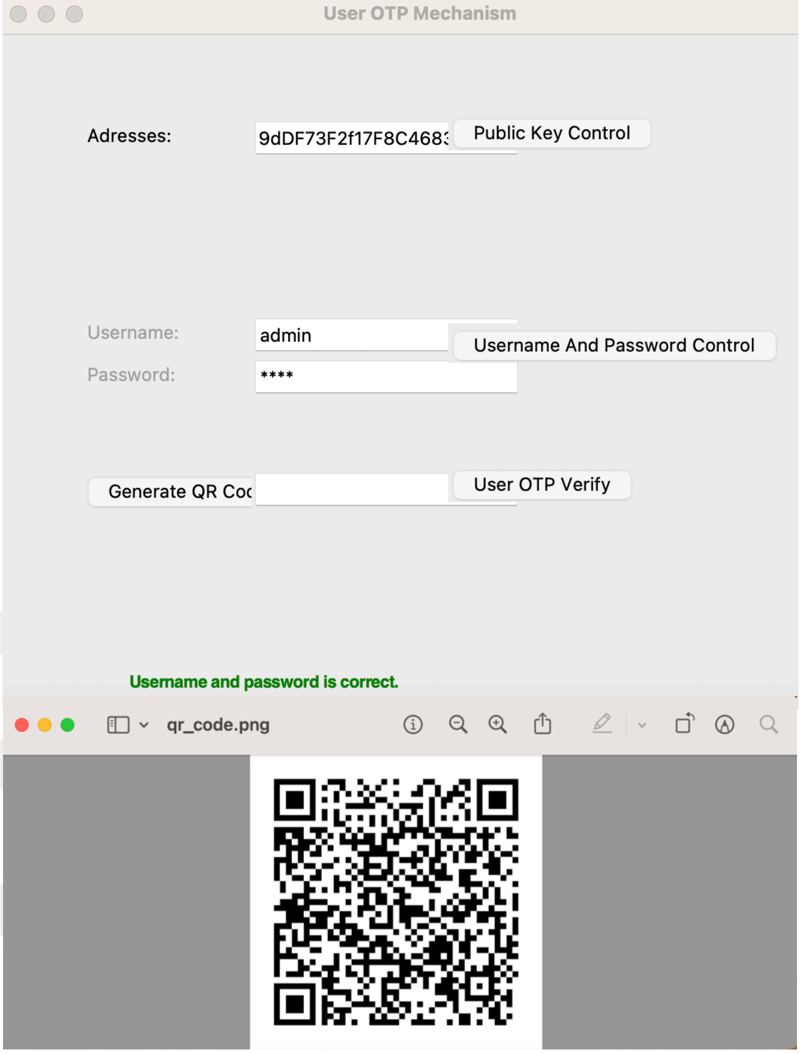
Password : ****

Allowed Users

0x5536defe9bBBeeF6bA9

Figure 35: User access permission

Afterwards, authorized users can connect to the broker by providing the required information accurately, as shown in Figure 36, and then defining the OTP code specific to their own Public Key for OTP password access. The OTP code consists of a constantly changing 6-digit password structure. This OTP information will be used when signing messages with digital signatures. Hence, this information is protected only with permissions granted to admin privileges.



Adresses: 9dDF73F2f17F8C4683

Username: admin

Password: ****

Username and password is correct.

qr_code.png

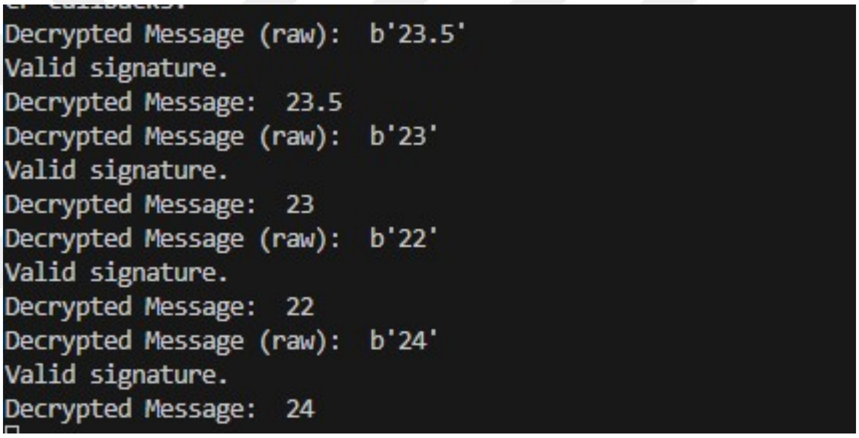
Figure 36: Obtaining OTP user password

After completing these operations, the first step required to access MQTT publication as a publisher or subscriber is accomplished. The access of a publisher is restricted by a digital signature generated using smart contracts and the Elliptic Curve Digital Signature Algorithm (ECDSA). This ensures that the publisher's identity and data security are authorised.

The waiting broker in this case is expecting a digital signature for confirmation. The incoming signed message should contain the access-permitted address and the OTP information specific to that address. Therefore, the publisher's message, apart from having access permission to publish, must also include a digitally signed verification of accurate information. Otherwise, the broker will not approve this message.

On the subscriber side, users with both access permissions and the ability to verify digital signatures decrypt the data. In this way, communication between the publisher and subscriber is protected by a robust encryption algorithm that requires both parties to possess their respective keys and authorization through smart contracts. Access control and data security of the MQTT broker is ensured by this approach.

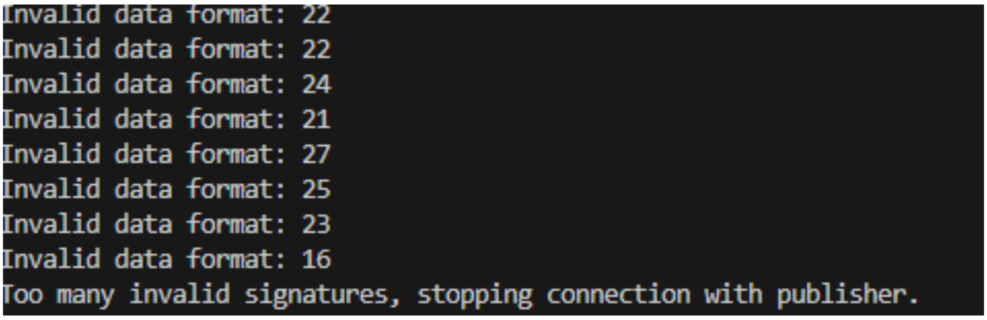
For each piece of data received by the MQTT broker, the broker can query the user's public key who wants to read this data to the blockchain. This query is directed to the smart contract in the blockchain. The smart contract receives the query and checks its internal mapping structure. If the mapping indicates that the user has permission to read the data, the smart contract returns a Figure 37 confirmation message.



```
Decrypted Message (raw):  b'23.5'
Valid signature.
Decrypted Message:  23.5
Decrypted Message (raw):  b'23'
Valid signature.
Decrypted Message:  23
Decrypted Message (raw):  b'22'
Valid signature.
Decrypted Message:  22
Decrypted Message (raw):  b'24'
Valid signature.
Decrypted Message:  24
```

Figure 37: User with MQTT broker permission

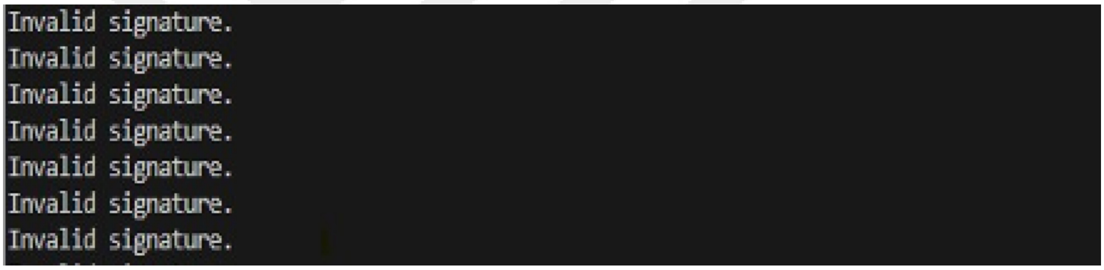
The MQTT broker checks the response from the smart contract. The user shall be given permission to read the information when receiving a confirmation message. The user's ability to view data is disabled when an error message appears. If an unauthorized user attempts to access the broker, the system, as shown in Figure 38, will sever its connection with the publisher by declaring an invalid signature. As a result, the system will be effectively prevented from being manipulated and attacked.



```
Invalid data format: 22
Invalid data format: 22
Invalid data format: 24
Invalid data format: 21
Invalid data format: 27
Invalid data format: 25
Invalid data format: 23
Invalid data format: 16
Too many invalid signatures, stopping connection with publisher.
```

Figure 38:User without MQTT permissionn

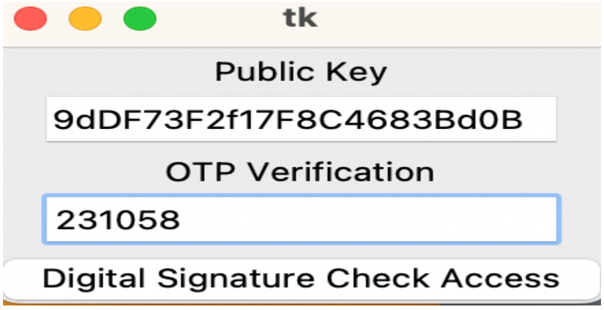
Furthermore, if publishers and subscribers fail to correctly validate their digital signatures with the system, their data access permissions will not be considered. Data access permissions will be disregarded when there is a mismatch with the expected public key or OTP during the validation of the digital signature, resulting in an 'invalid signature' response, as shown in Figure 39.



```
Invalid signature.
Invalid signature.
Invalid signature.
Invalid signature.
Invalid signature.
Invalid signature.
Invalid signature.
```

Figure 39: Access with an incorrect private key

To initiate the system's broker and subscription structure, it is required to be a registered user of the system again (Figure 40) and to correctly validate the digital signature generated with the user's unique OTP. This ensures that the desired publishing and subscribing to the broker are prevented unless the authentication process is successful. In case of providing incorrect information, the subscription and publishing access will be denied.

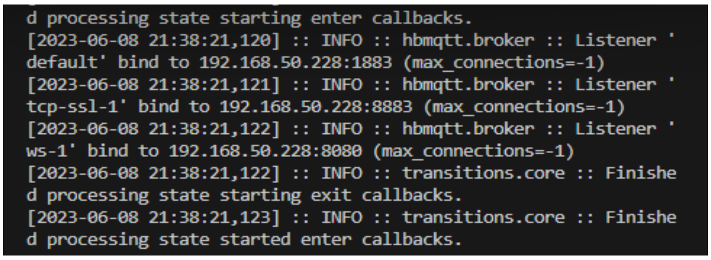


Public Key

9dDF73F2f17F8C4683Bd0B

OTP Verification

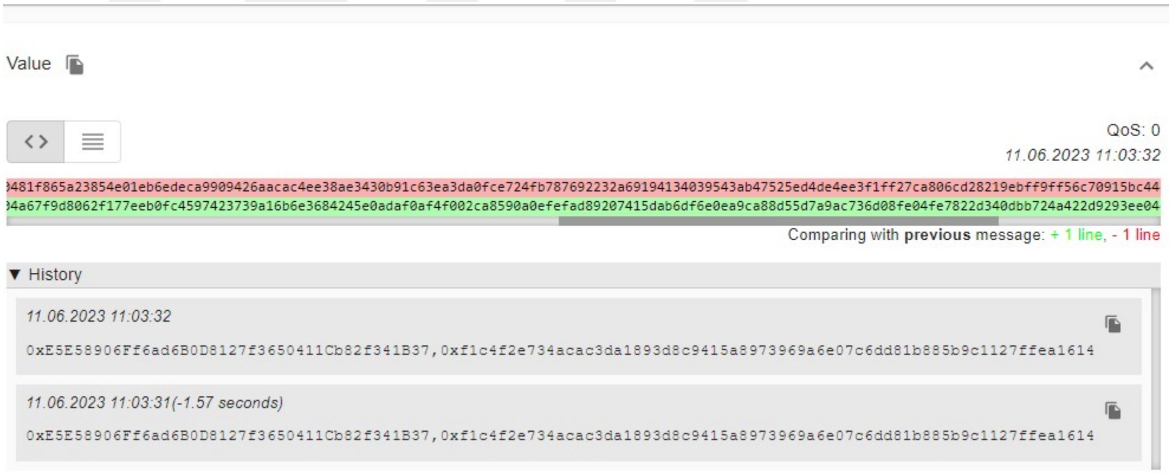
231058



```
d processing state starting enter callbacks.
[2023-06-08 21:38:21,120] :: INFO :: hbmqtt.broker :: Listener '
default' bind to 192.168.50.228:1883 (max_connections=-1)
[2023-06-08 21:38:21,121] :: INFO :: hbmqtt.broker :: Listener '
tcp-ssl-1' bind to 192.168.50.228:8883 (max_connections=-1)
[2023-06-08 21:38:21,122] :: INFO :: hbmqtt.broker :: Listener '
ws-1' bind to 192.168.50.228:8080 (max_connections=-1)
[2023-06-08 21:38:21,122] :: INFO :: transitions.core :: Finishe
d processing state starting exit callbacks.
[2023-06-08 21:38:21,123] :: INFO :: transitions.core :: Finishe
d processing state started enter callbacks.
```

Figure 40:User permission information for subscription

These methods, combined with smart contract, OTP and digital signature, result in a more secure system that resolves the previously mentioned issues. As shown in Figure 41, network traceability has been eliminated with the authentication mechanism. Complete confidentiality has been achieved. There is no access to data in any way. Smart contracts, OTPs and digital signatures to guarantee data security and authentication are a part of this process.



```
481f865a23854e01eb6edeca9909426aacac4ee38ae3430b91c63ea3da0fce724fb787692232a69194134039543ab47525ed4de4ee3f1ff27ca806cd28219ebff9ff56c70915bc44
4a67f9d8062f177eeb0fc4597423739a16b6e3684245e0adaf0af4f002ca8590a0efefad89207415dab6df6e0ea9ca88d55d7a9ac736d08fe04fe7822d340dbb724a422d9293ee04
```

Comparing with **previous** message: + 1 line, - 1 line

▼ History

11.06.2023 11:03:32

```
0xE5E58906Ff6ad6B0D8127f3650411Cb82f341B37,0xf1c4f2e734acac3da1893d8c9415a8973969a6e07c6dd81b885b9c1127ffea1614
```

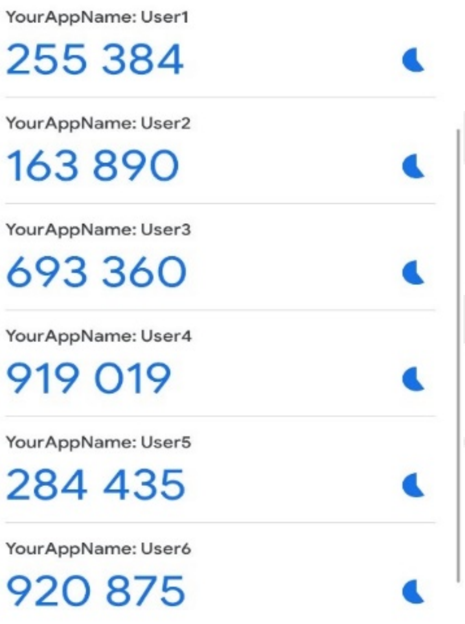
11.06.2023 11:03:31(-1.57 seconds)

```
0xE5E58906Ff6ad6B0D8127f3650411Cb82f341B37,0xf1c4f2e734acac3da1893d8c9415a8973969a6e07c6dd81b885b9c1127ffea1614
```

Figure 41:Encrypted MQTT data analysis

A decentralised and secured infrastructure for authentication of user identities is provided by the Blockchain Authentication Mechanism. The MQTT authentication process requires users to have unique identity credentials that match those registered on the Blockchain. This verification procedure shall ensure that unauthorised access is prevented and the system's communication environment has been secured. With this structure, unauthorized users are prevented from manipulating the data, making it impossible.

With the OTP security mechanism, a dynamic and unique password is available in order to ensure that no one can gain unauthorised access. Messages in digital signatures contain a unique OTP along with public keys. Only those with access permission and the ability to correctly validate the message with OTP can obtain access permission.It ensures the security of MQTT users and the Broker during the authentication process, allowing secure data publishing and subscription operations. As mentioned in Figure 42 an authorized user with permission can access their one-time passwords through the help of Google Authenticator by scanning the QR code upon request.



YourAppName: User1
255 384

YourAppName: User2
163 890

YourAppName: User3
693 360

YourAppName: User4
919 019

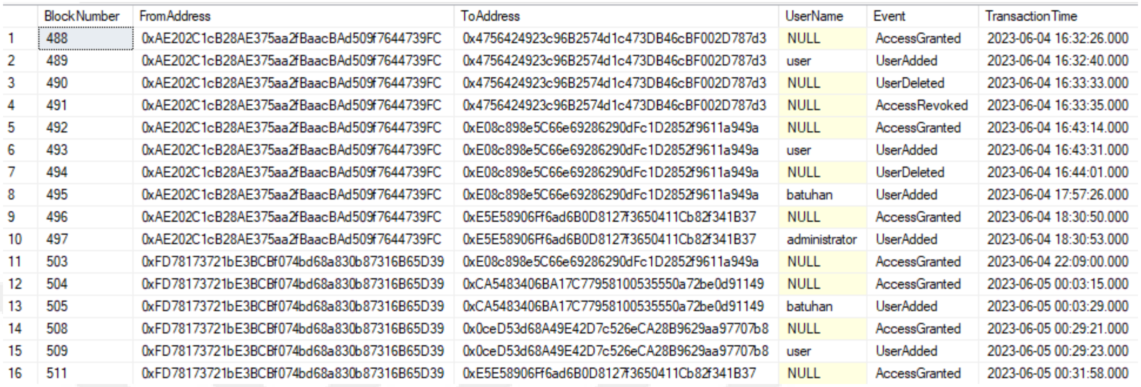
YourAppName: User5
284 435

YourAppName: User6
920 875

Figure 42:User OTP permission

Encryption Smart contracts operate without the need for a central authority or intermediary, providing greater freedom and independence in terms of trust compared to SSL. In order to improve reliability and speed of transactions, they shall be automatically introduced when certain preconditions have been fulfilled with respect to the elimination of errors by humans or intervention. Stored on the blockchain in a manner that is visible and verifiable by all participants, smart contracts enhance transparency and increase trustworthiness. Smart contracts, when they work in conjunction with Blockchain technology, guarantee data integrity which enables almost no change or manipulation of transactions so as to enhance security.

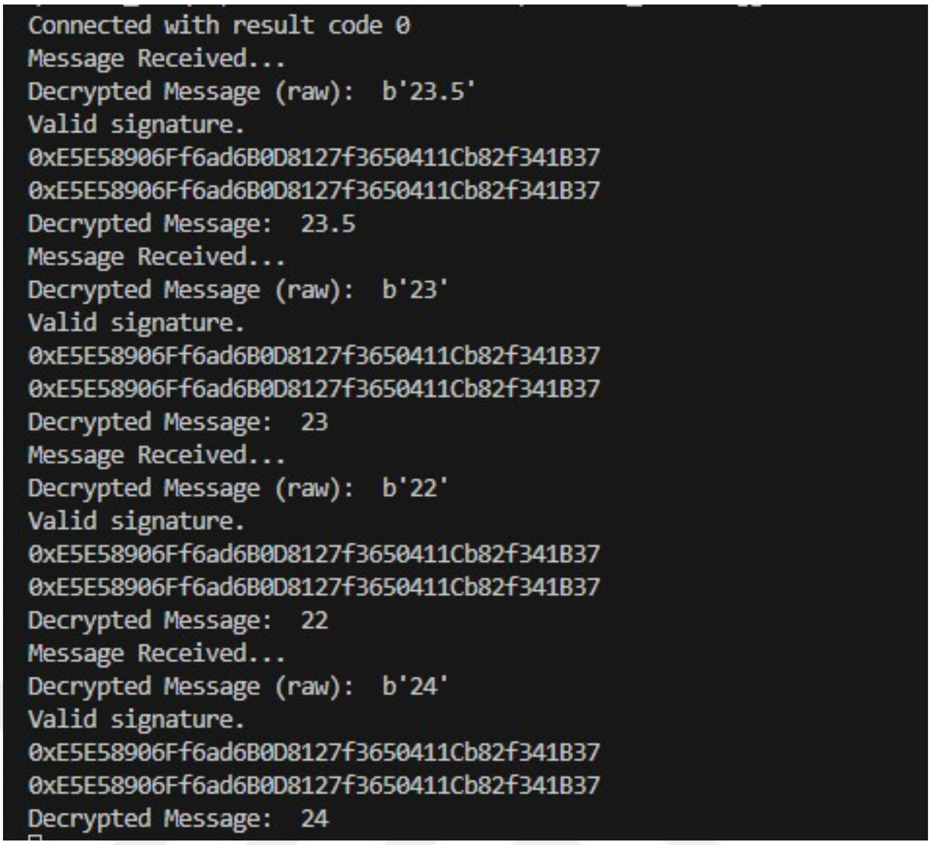
Encryption and decryption processes enable secure data sharing and allow only authorized users to access plain-text data. It is shared that unauthorized users are unable to access the data on the network. Because the data is now in the form of certificates rather than plain text. In Figure 43, it is also shared that every transaction is recorded.



	BlockNumber	FromAddress	ToAddress	UserName	Event	TransactionTime
1	488	0xAE202C1cB28AE375aa2fBaacBAd509f7644739FC	0x4756424923c96B2574d1c473DB46cBF002D787d3	NULL	AccessGranted	2023-06-04 16:32:26.000
2	489	0xAE202C1cB28AE375aa2fBaacBAd509f7644739FC	0x4756424923c96B2574d1c473DB46cBF002D787d3	user	UserAdded	2023-06-04 16:32:40.000
3	490	0xAE202C1cB28AE375aa2fBaacBAd509f7644739FC	0x4756424923c96B2574d1c473DB46cBF002D787d3	NULL	UserDeleted	2023-06-04 16:33:33.000
4	491	0xAE202C1cB28AE375aa2fBaacBAd509f7644739FC	0x4756424923c96B2574d1c473DB46cBF002D787d3	NULL	AccessRevoked	2023-06-04 16:33:35.000
5	492	0xAE202C1cB28AE375aa2fBaacBAd509f7644739FC	0xE08c898e5C66e69286290dFc1D2852f9611a949a	NULL	AccessGranted	2023-06-04 16:43:14.000
6	493	0xAE202C1cB28AE375aa2fBaacBAd509f7644739FC	0xE08c898e5C66e69286290dFc1D2852f9611a949a	user	UserAdded	2023-06-04 16:43:31.000
7	494	0xAE202C1cB28AE375aa2fBaacBAd509f7644739FC	0xE08c898e5C66e69286290dFc1D2852f9611a949a	NULL	UserDeleted	2023-06-04 16:44:01.000
8	495	0xAE202C1cB28AE375aa2fBaacBAd509f7644739FC	0xE08c898e5C66e69286290dFc1D2852f9611a949a	batuhan	UserAdded	2023-06-04 17:57:26.000
9	496	0xAE202C1cB28AE375aa2fBaacBAd509f7644739FC	0xE5E58906Ff6ad6B0D8127f3650411Cb82f341B37	NULL	AccessGranted	2023-06-04 18:30:50.000
10	497	0xAE202C1cB28AE375aa2fBaacBAd509f7644739FC	0xE5E58906Ff6ad6B0D8127f3650411Cb82f341B37	administrator	UserAdded	2023-06-04 18:30:53.000
11	503	0xFD78173721bE3BCBf074bd68a830b87316B65D39	0xE08c898e5C66e69286290dFc1D2852f9611a949a	NULL	AccessGranted	2023-06-04 22:09:00.000
12	504	0xFD78173721bE3BCBf074bd68a830b87316B65D39	0xCA5483406BA17C77958100535550a72be0d91149	NULL	AccessGranted	2023-06-05 00:03:15.000
13	505	0xFD78173721bE3BCBf074bd68a830b87316B65D39	0xCA5483406BA17C77958100535550a72be0d91149	batuhan	UserAdded	2023-06-05 00:03:29.000
14	508	0xFD78173721bE3BCBf074bd68a830b87316B65D39	0x0ceD53d68A49E42D7c526eCA28B9629aa97707b8	NULL	AccessGranted	2023-06-05 00:29:21.000
15	509	0xFD78173721bE3BCBf074bd68a830b87316B65D39	0x0ceD53d68A49E42D7c526eCA28B9629aa97707b8	user	UserAdded	2023-06-05 00:29:23.000
16	511	0xFD78173721bE3BCBf074bd68a830b87316B65D39	0xE5E58906Ff6ad6B0D8127f3650411Cb82f341B37	NULL	AccessGranted	2023-06-05 00:31:58.000

Figure 43:Transaction block record

A unique broker has been developed by bringing all of these together under an MQTT Broker structure. As shown in Figure 44, authorized users who request a connection and provide the correct credentials (such as the blockchain smart contract access permission username and password, the same user's public, digital signature and finally the OTP security code) are shared the permission that the data stream is allowed, provided that the appropriate access permissions match the public keys and the asymmetric encryption and digital signature are successfully verified.



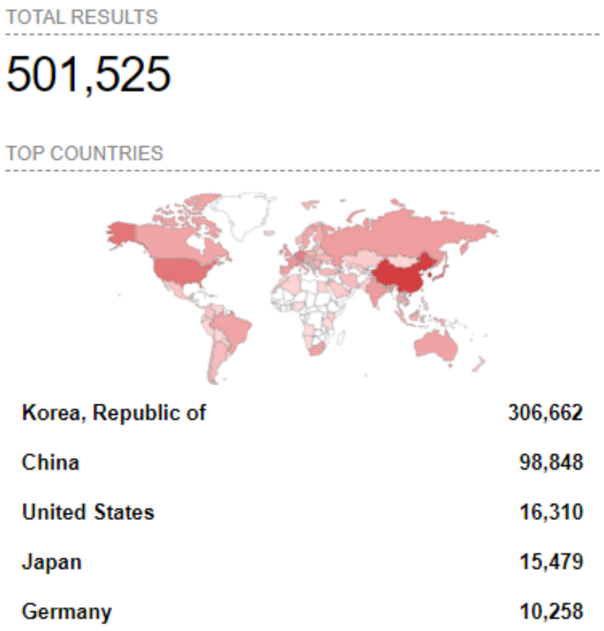
```
Connected with result code 0
Message Received...
Decrypted Message (raw):  b'23.5'
Valid signature.
0xE5E58906Ff6ad6B0D8127f3650411Cb82f341B37
0xE5E58906Ff6ad6B0D8127f3650411Cb82f341B37
Decrypted Message:  23.5
Message Received...
Decrypted Message (raw):  b'23'
Valid signature.
0xE5E58906Ff6ad6B0D8127f3650411Cb82f341B37
0xE5E58906Ff6ad6B0D8127f3650411Cb82f341B37
Decrypted Message:  23
Message Received...
Decrypted Message (raw):  b'22'
Valid signature.
0xE5E58906Ff6ad6B0D8127f3650411Cb82f341B37
0xE5E58906Ff6ad6B0D8127f3650411Cb82f341B37
Decrypted Message:  22
Message Received...
Decrypted Message (raw):  b'24'
Valid signature.
0xE5E58906Ff6ad6B0D8127f3650411Cb82f341B37
0xE5E58906Ff6ad6B0D8127f3650411Cb82f341B37
Decrypted Message:  24
```

Figure 44:User with access permission MQTT connection

The Blockchain Authentication Mechanism, digital signature and the integrated OTP security mechanism with Digital Signatures developed for the MQTT protocol play a significant role in ensuring security. The security vulnerabilities of MQTT protocols are addressed through these techniques, which provide a safe environment for communication. With this structure, unauthorized users are prevented from manipulating the data, making it impossible.

CONCLUSION AND FUTURE WORK

An initial observation made in practice is, for a general assessment the Shodan and Python API were used in the analysis. Shodan is a search tool that scans the Internet and uses different filters like protocol names and port numbers in order to locate devices which are freely available. When the results are examined, it shows the total number of publicly accessible devices using the MQTT protocol. Additionally, it displays the distribution of MQTT Broker hosting servers worldwide [74]. Currently, there are “501,525” MQTT Brokers in use worldwide, as shown in Figure 45.



TOTAL RESULTS

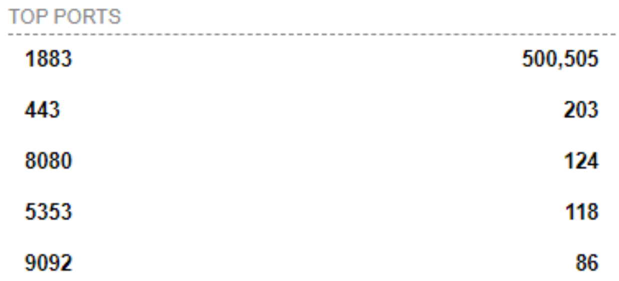
501,525

TOP COUNTRIES

Korea, Republic of	306,662
China	98,848
United States	16,310
Japan	15,479
Germany	10,258

Figure 45:Graph illustrating MQTT usage

%99.79 of the devices use port 1883 for communication via the MQTT Protocol. The remaining devices use HTTPS, Port 8081, and Port 8883 services and ports. For the purpose of identifying servers that allow unauthorised access, this information was used. The assessment provided in Figure 46 therefore shows that there is a strong need for a security system based on the ports to be exploited, while at the same time this system is vulnerable to different types of attacks.



TOP PORTS	
1883	500,505
443	203
8080	124
5353	118
9092	86

Figure 46:Distribution of most commonly used ports in MQTT connections

In this section, Using 500 devices from Port 1883 as a sample for an analysis. Here, the client attempts to subscribe to a topic called 'random' without specifying a username and password. When the Broker receives the connect message, it may perform some checks and then return a Connection code in the connect ack.

Figure 47 from the graph, it is observed that %65 of the servers allow unauthorized access to topics. That is, a client can publish or subscribe to any topic without credentials. The first line of defense, authentication, has not been enabled on most servers and can be used by attackers to carry out various attacks. Details such as server version, number of clients, message count, etc., are revealed. It can be used to steal all data passing through the Broker.

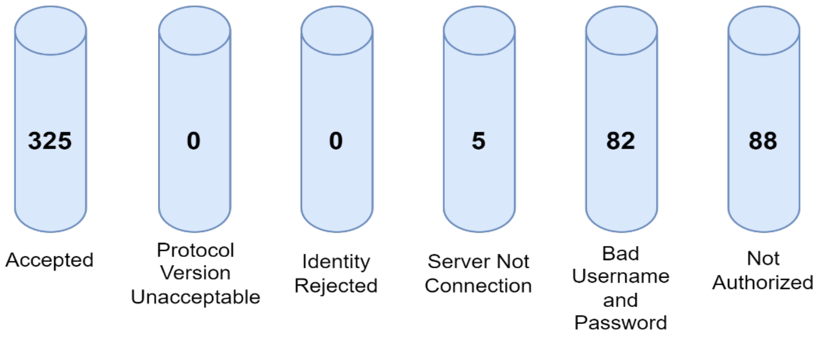


Figure 47:MQTT server connection results

Internet of Things IoT enables device capabilities to be increased by allowing them to connect with other connected devices and facilitating data exchange. With each passing year the Internet is witnessing an increase, in the number of devices joining its network. In our own way of life, this constant growth is bringing about a change. But the protection of such Internet enabled devices from cyber security vulnerabilities needs to be urgently addressed. Focusing on the MQTT protocol, which is extensively employed in IoT due to its low bandwidth requirements, efficiency, and reliability even in challenging conditions.

After carrying out experiments on the MQTT protocol we have come across vulnerabilities that could potentially be exploited by individuals, with ill intentions. Leveraging the Shodan web search engine and Python APIs, we have observed a lack of authentication mechanisms on the majority of MQTT Broker servers. Exploiting this situation means that an attacker who has unauthorised access to data or the leakage of unnecessary information can compromise a whole range of systems. Therefore, in order to address these vulnerabilities we have introduced robust security controls including the modification of Broker configuration files.

A conceptual proof encompassing the design and implementation of a blockchain-based identity authentication and authorization scheme has been presented. The solution given is based on the Ethereum blockchain.

The motivation behind this, lies in the inherent weakness of the local identity authentication mechanism provided by the protocol. This is due to its reliance on the simple transmission of a plaintext message containing a username and password from the client to the MQTT broker. However, considering the constraints posed by resource-limited devices, in order to implement robust identity authentication, lightweight mechanisms such as OTP, smart contracts, and digital signatures have been favored over TLS/SSL within the protocol.

The straightforward incorporation of the OTP mechanism into MQTT messaging has been observed in general studies to potentially leave certain security vulnerabilities exposed. With the solution of smart contracts and digital signatures, results can be obtained by overcoming the negativities that may occur. Users with access privileges through smart contracts will utilize OTP information in the transmission of digital signatures, thus mitigating these vulnerabilities. As a result, a more robust scheme is presented compared to the standard approach [65], [66].

The Elliptic Curve Digital Signature Algorithm (ECDSA) has been extensively. Compared in studies regarding the security of MQTT [77], [78].

However, it has been noted that none of these works have encountered an integration of ECDSA with OTP, as proposed within the framework of this thesis. Furthermore the combination of ECDSA, with contracts has been found to introduce new possibilities, for research. Based on the studys results it seems that ECDSA shows promise in meeting the security needs of Internet of Things devices. Particularly, the security mechanism proposed throughout this thesis has been designed with consideration of IoT device constraints and performance needs, aiming to achieve secure communication with reduced resource consumption. When it comes to security studying the comparison, between ECDSA and other safety algorithms in literature has provided us with insights, into their strengths and weaknesses. Nonetheless, research on internet of things security has been encouraged by the new avenues for integration into ECDSA with OTP and Smart Contracts.

A way to allow users the possibility of managing their identities and data without reliance on an authority is also part of our solution. Additionally we conducted an analysis to assess the performance of our systems. Based on the results of our study it appears that the approach we have adopted for offering and distributing identity verification in MQTT is proving to be successful. In addition we conducted a comparison of the CPU and memory utilization, between our method and SSL/TLS in relation, to storage needs. A better performance was achieved. It makes sure that the hardware doesn't create any performance issues or disrupt its functioning all while collaborating smoothly with the applications.

Ultimately, leveraging the inherent trust provided by blockchain, increased levels of accountability and forensic capabilities are achieved at no cost. Future efforts could involve extending the scope of the Smart Contract's identity verification capabilities and introducing our solution using a number of distributed Blockchain systems, e.g. CORDA or Iota. Furthermore it is still feasible to carry out investigations regarding the assessment of effectiveness, in consensus mechanisms.

REFERENCES

[1] S. Andy, B. Rahardjo and . B. Hanindhito, "Attack Scenarios and Security Analysis of Mqtt communication in IoT system", 4th International Conference on Electrical Engineering, Computer Science and Informatics, 2017, pp. 1-6, doi: 10.1109/EECSI.2017.8239179.

[2] A. R. Alkhafaje, A. M. A. Al-muqarm and A. H. Alwan, "Security and Performance Analysis of Mqtt Protocol with Tls in IoT Networks", 2021 4th International Iraqi Conference on Engineering Technology and Their Applications (IICETA), Najaf, Iraq, pp. 206-211, 2021, doi: 10.1109/IICETA51758.2021.9717495.

[3] A. S. W. Tang, J. H. Bong, Q. L. Teh, S. Sivalingam, S. Suet, Y. Chan, S. Y. Khoo and T. M. Nafy, "Authentication of IoT device with the enhancement of One-time Password", Journal of IT in Asia, 29-40, 2021, doi: :10.33736/jita.3841.2021.

[4] B. H. Çorak, M. Guzel, F. Yildirim and S. Ozdemir, "Comparative Analysis of IoT Communication Protocols", Conference: 2018 International Symposium on Networks, Computers and Communications, (ISNCC), Rome, Italy, pp.1-6, 2018, doi: 10.1109/ISNCC.2018.8530963.

[5] H. G. Hamid and Z. T. Alisa, "Survey on IoT application layer protocols", Baghdad, Iraq: Indonesian Journal of Electrical Engineering and Computer Science,Vol. 21, No. 3, pp. 1663-1672, 2021, doi: 10.13140/RG.2.2.11387.85283.

[6] O. M. T. Committee, “Mqtt Version 5.0", Oasis, Committee Specification, 02, 2018.

[7] L. Cruz-Piris, D. Rivera, I. Marsa-Maestre, E. e de la Hoz and J. R. Velasco, "Access Control Mechanism for IoT Environments Based on Modelling Communication Procedures as Resources", Sensors, vol. 18, no. 3, p. 917, Mar. 2018, doi: 10.3390/s18030917.

[8] R. Neisse, G. Baldini and G. Steri, "Enforcement of Security Policy Rules for the Internet of Things, Conference" , 2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), Larnaca, Cyprus, pp. 165-172, 2014, doi: 10.1109/WiMOB.2014.6962166.

[9] D. Soni and A. Makwana, "A Survey on Mqtt: A protocol of internet of things", 2017.

[10] K. Christidis, and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things" , IEEE Access , 2016, vol. 4, pp. 2292-2303, 2016, doi: 10.1109/ACCESS.2016.2566339.

[11] A. Dorri, S. Kanhere and R. Jurdak, "Blockchain in internet of things: Challenges and solutions", [https://arxiv.org/abs/1608.05187], (Last Access Date: August 28, 2023)

[12] N. Kshetri, "Can blockchain strengthen the internet of things?", in IT Professional, vol. 19, no. 4, pp. 68-72, 2017, doi: 10.1109/MITP. 2017.3051335.

[13] A. Ouaddah, A. Elkalam and A. Ouahman, "A new Blockchain-based access control framework for the Internet of Things, Secur. Commun. Networks", 2016, doi: 10.1002/sec.1748.

[14] A. P. Miguel, R. Perfecto, A. Seraffın, M. Antonio and D. Manuel, "Communication with resource-constrained devices through Mqtt for control education", IFAC-PapersOnLine, vol. 49, no. 6, pp. 150–155, 2016, 2016.

[15] T.Jaffey, "MQTT and CoAP, IoT Protocols", [https://eclipse.org/615 community/eclipsenewsletter/2014/febru-ary/article2.php], (Last Access Date: August 23, 2023)

[16] "HBMQTT", GitHub: [https://github.com/beerfactory/hbmqtt], (Last Access Date: August 23, 2023)

[17] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System" [https://bitcoin.org/bitcoin.pdf], (Last Access Date: August 23, 2023)

[18] I. Eyal, A. E. Gencer, . E. G. Sirer and . R. V. Renesse, "Bitcoin-NG: a scalable blockchain protocol, USA: 13th Usenix Conference on Networked Systems Design and Implementation", 2015.

[19] M. Mettler, "Blockchain technology in healthcare: The revolution starts here" IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom), Munich, Germany, pp. 1-3, 2016, doi: 10.1109/ /HealthCom.2016.7749510.

[20] N. Alilwit, "Authentication Based on Blockchain", IEEE 39th International Performance Computing and Communications Conference (IPCCC), pp. 1-6, 2020.

[21] X. He, S. Alqahtani, R. Gamble and M. Papa, "Securing over-the-air IoT firmware updates using blockchain", Proceedings of the International Conference on Omni-Layer Intelligent Systems, pp. 164-171, 2019, doi: 10.1145/3312614.3312649.

[22] X. He, R. Gamble and M. Papa, "A Smart Contract Grammar to Protect IoT Firmware Updates using Hyperledger Fabric", 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), Canada, 2019, pp. 0034-0042, doi: 10.1109/IEMCON. 2019.8936223.

[23] S. Sun, S. Chen, R. Du, W. Li and D. Qi, "Blockchain Based Fine-Grained and Scalable Access Control for IoT Security and Privacy", 2019 IEEE Fourth International Conference on Data Science in Cyberspace (DSC), Hangzhou, China, 2019, pp. 598-603, doi: 10.1109/DSC.2019.00097.

[24] U. Guin, P. Cui and A. Skjellum, "Ensuring Proof-of-Authenticity of IoT Edge Devices Using Blockchain Technology", 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), Canada, pp. 1042-1049, 2018, doi: 10.1109/Cybermatics_2018.2018.00193.

[25] G. Ali, N. Ahmad, Y. Cao, Q. E. Ali , F. Azim and H. Cruickshank, "Bcon: Blockchain based access Control across multiple conflict of interest domains", Journal of Network and Computer Applications, vol. 147, 2019, doi: 10.1016/j.jnca.2019.102440.

[26] A. El Kalam, A. Outchakoucht and H. Es-Samaali, "Emergence-based access control: New approach to secure the internet of things", Proceedings of the 1st International Conference on Digital Tools & Uses Congress, pp. 1-11, 2018, doi: 10.1145/3240117.3240136.

[27] B. Tang, H. Kang, J. Fan and R. Sandhu, "Iot passport: a blockchain-based trust framework for collaborative internet-of-things", Proceedings of the 24th Acm Symposium on Access Control Models and Technologies, pp. 83-92, 2019, doi: 10.1145/3322431.3326327.

[28] "Solidity Documentation", [http://solidity.readthedocs.org/en/latest/], (Last Access Date: August 23, 2023)

[29] S. E. Kafhali, C. Chahir, M. Hanini and K. Salah, "Architecture to manage Internet of Things Data using Blockchain and Fog Computing", Proceedings of the 4th International Conference on Big Data and Internet of Things, Association for Computing Machinery, New York, NY, USA, Article 32, 1–8, 2020.

[30] C. Dukkipati, Y. Zhang and L. Cheng, "Decentralized, blockchain based access control framework for the heterogeneous internet of things", Proceedings of the Third ACM Workshop on Attribute-Based Access Control, 2018, doi: 10.1145/3180457.3180458.

[31] S.-M. Choi, B. C. Kim, B.-H. Cho, K. W. Kang , K.-H. Choi, J.-T. Kim, S.-H. Lee, M.-S. Park, M.-K. Kim and K.-H. Cho, "Comparison of two motor subtype classifications in de novo Parkinson's disease", International Conferance on Advances in Computing and Communication Engineering, ICACCE,IEEE,vol. 54, pp-74-78, 2018.

[32] S. Sun, S. Chen, R. Du, W. Li and D. Qi, "Blockchain Based Fine-Grained and Scalable Access Control for IoT Security and Privacy", 2019 IEEE Fourth International Conference on Data Science in Cyberspace (DSC), China, 2019, pp. 598-603, doi: 10.1109/DSC.2019.00097.

[33] E. Kfoury and D. Khoury, "Distributed Public Key Infrastructure and PSK Exchange Based on Blockchain Technology", 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), Canada, 2018, pp. 1116-1120, doi: 10.1109/Cybermatics_2018.2018.00203.

[34] U. Javaid, M. Aman and B. Sikdar, "Blockpro: Blockchain based data provenance and integrity for secure iot environments", Proceedings of the 1st Workshop on Blockchain-Enabled Networked Sensor Systems, 2018, doi: 10.1145/3282278.3282281.

[35] A. M. Antonopoulos, "Mastering Bitcoin: Unlocking Digital Cryptocurrenciess", 1st ed. Sebastopol, USA: O'Reilly Media, Inc., 2014.

[36] "Eris Industries Documentation Blockchains", [https://docs.erisindustries.com/explainers/], (Last Access Date: August 23, 2023)

[37] "Understanding Public Key Cryptography", [https://learn.microsoft.com/en-us/previous-versions/tnarchive/aa998077v=exchg.65)?redirectedfrom =MSDN], (Last Access Date: August 23, 2023)

[38] L. Luu, V. Narayanan, K. Baweja, C. Zheng, S. Gilbert and P. Saxena, "A computationally-scalable byzantine consensus protocol for blockchains", IACR Cryptology ePrint Archive, 2015.

[39] A. Gervais, G. O. Karame, K. Wu, V. Glykantzis, H. Ritzdorf and S. Capkun, "On the security and performance of proof of work blockchains", Acm Sigsac Conference on Computer and Communications Security, 2016.

[40] S. King and S. Nadal, "PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake", 2012.

[41] "Announcing the Secure Hash Standard", [https://csrc.nist.gov/csrc/media/publications/fips/180/2/archive/2002-08-01/documents/fips180-2.pdf], (Last Access Date: August 23, 2023)

[42] "Hashcash-Bitcoin WiKi", [https://en.bitcoin.it/wiki/Hashcash], (Last Access Date: August 23, 2023)

[43] J.-P. Aumasson, L. Henzen, W. Meier and . R. C. W. Phan, "SHA-3 Proposal Blake", [https://131002.net/blake/blake.pdf], (Last Access Date: August 23, 2023)

[44] C. Percival, Tarsnap,"The Scrypt Key Derivation Function and Encryption Utility", [http://www.tarsnap.com/scrypt.html], (Last Access Date: August 23, 2023)

[45] Myriad."A Coin for Everyone", [http://myriadcoin.org/home], (Last Access Date: August 23, 2023)

[46] V.Buterin,"OnStake", [https://blog.ethereum.org/2014/07/05/stake/.], (Last Access Date: August 23, 2023)

[47] V. Buterin, "Slasher Ghost, and Other Developments in Proof of Stake", [https://blog.ethereum.org/2014/10/03/slasherghost-developments-proof-stake/], (Last Access Date: August 23, 2023)

[48] T. Swanson, "Consensus-as-a-service: A brief report on the emergence of permissioned, distributed ledger systems." [http://www.ofnumbers.com/2015/04/06/consensus-as-a-service-a-brief-report-on-the-emergence-oft-on-the-emergence-ofpermissioned-distributed-ledger-systems/], (Last Access Date: August 23, 2023).

[49] S. Maitra, V. P. Yanambaka, A. Abdelgawad, D. Puthal and K. Yelamarthi, "Proof-of-Authentication Consensus Algorithm:Blockchain-based IoT Implementation", 2020 IEEE 6th World Forum on Internet of Things (WF-IoT), New Orleans, LA, USA, 2020, pp. 1-2, doi: 10.1109/WF-IoT48130.2020.9221187.

[50] A. Fauzan, P. Sukarno and A. Arif Wardana, "Overhead Analysis of the Use of Digital Signature in MQTT Protocol for Constrained Device in the Internet of Things System", 3rd International Conference on Computer and Informatics Engineering, Yogyakarta, Indonesia, 2020, pp. 415-420, doi: 10.1109/IC2IE50715.2020.9274651.

[51] Y. Genç and E. Afacan, "Design and Implementation of an Efficient Elliptic Curve Digital Signature Algorithm (ECDSA)", IEEE International IOT, Electronics and Mechatronics Conference, Canada, 2021, pp. 1-6, doi: 10.1109/IEMTRONICS52119.2021.9422589

[52] H. C. Pöhls and B. Petschkuhn, "Towards compactly encoded signed IoT messages", IEEE 22nd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), Lund, Sweden, 2017, pp. 1-6, doi: 10.1109/CAMAD.2017.8031622.

[53] E.Foundation,"Ethereum Remix - Ethereum IDE", [https://remix.ethereum.org/.], (Last Access Date: August 23, 2023)

[54] T.Suite,"Ganache-Oneclickblockchain", [https://www.trufflesuite.com/ganache], (Last Access Date: August 23, 2023)

[55] MetaMask, "MetaMask - A crypto wallet & gateway to blockchain apps", [https://metamask.io/.], (Last Access Date: August 23, 2023)

[56] N. Kshetri, "Can Blockchain Strengthen the Internet of Things", IEEE IT Professional, vol. 19, no. 4, pp. 68-72, 2017, doi: 10.1109/MITP.2017.3051335.

[57] J. Scott and D. Spaniel, "Rise of the Machines: The Dyn Attack Was Just a Practice Run", Institute for Critical Infrastructure Technology, 2017.

[58] J. A. Jerkins, "Motivating a Market or Regulatory Solution to IoT Insecurity with the Mirai Botnet Code", Department of Computer Science and Information Systems, 2017, Las Vegas, NV, USA, 2017, pp. 1-5, doi: 10.1109/CCWC.2017.7868464.

[59] F. Taps, "Foundations for the Next Economic Revolution, Filament" [https://blockchainlab.com/pdf/Filament%20Foundations.pdf], (Last Access Date: August 23, 2023)

[60] R.P.Foundation,"RaspberryPi2ModelB", [https://www.raspberrypi.org/products/raspberry-pi-2-model-b/], (Last Access Date: August 23, 2023)

[61] L. Aosong Electronics Co., "DHT11 Digital Humidity and Temperature Sensor", [http://www.aosong.com/userguide/DHT11.pdf.], (Last Access Date: August 23, 2023)

[62] W. Liang Zhang, "Analysis of SSL Certificate Reissues and Revocations in the Wake of Heartbleed", 2018, pp. 109-116, doi: 10.1145/3176244.

[63] Prantl, Thomas & Iffländer, Lukas & Herrnleben, Stefan & Engel, Simon & Kounev, Samuel & Krupitzer, Christian, "Performance Impact Analysis of Securing MQTT Using TLS", Conference: ICPE '21: ACM/SPEC International Conference on Performance Engineering , 2021, pp. 241-248, doi: 10.1145/3427921.3450253.

[64] Abubakar, Mwrwan & Jaroucheh, Zakwan & Al-Dubai, Ahmed & Liu, Xiaodong, "Blockchain-based identity and authentication scheme for MQTT protocol", In 2021 The 3rd International Conference on Blockchain Technology, 2021, pp. 73-81, doi: 10.1145/3460537.3460549.

[65] Buccafurri, Francesco & Romolo, Celeste, "A Blockchain-Based OTP-Authentication Scheme for Constrainded IoT Devices Using MQTT", 2019, pp. 1-5, doi: 10.1145/3386164.3389095.

[66] F. Buccafurri, V. De Angelis, and R. Nardone, “Securing MQTT by Blockchain-Based OTP Authentication,” Sensors, vol. 20, no. 7, p. 2002, Apr. 2020, doi: 10.3390/s20072002.

[67] Bruno Blanchet, "ProVerif 2.04: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial," November 30, 2021.

[68] C. J. F. Cremers, "Scyther - Semantics and Verification of Security Protocols," [Phd Thesis 1 (Research TU/e / Graduation TU/e), Mathematics and Computer Science], 2006.

[69] C. &. D. N. Patel, "A Novel MQTT Security framework In Generic IoT Model", Procedia Computer Science, 171, 2020, doi: 10.1016/j.procs.2020.04.150.

[70] D. Javeed, T. Gao and M. Khan, "A Hybrid Deep Learning-Driven SDN Enabled Mechanism for Secure Communication in Internet of Things (IoT)", Sensors, 2021, vol. 21, no. 14, p. 4884, Jul. 2021, doi: 10.3390/s21144884.

[71] MQTTset, "a new dataset for MQTT", https://www.kaggle.com/ cnrieiit/mqttset], (Last Access Date: August 23, 2023)

[72] AWS, "CSE-CIC-IDS2018", [https://www.unb.ca/cic/datasets/ids-2018.html] , (Last Access Date: August 23, 2023)

[73] Wireshark, "Wireshark: Go Deep", [https://www.wireshark.org/], (Last Access Date: August 23, 2023)

[74] Shodan, "Shodan: World's First Search Engine for Internet-Connected Devices", [https://www.shodan.io], (Last Access Date: August 23, 2023)

[75] M. Explorer, "MQTT Explorer - MQTT Client and Testing Tool", [https://mqtt-explorer.com], (Last Access Date: August 23, 2023)

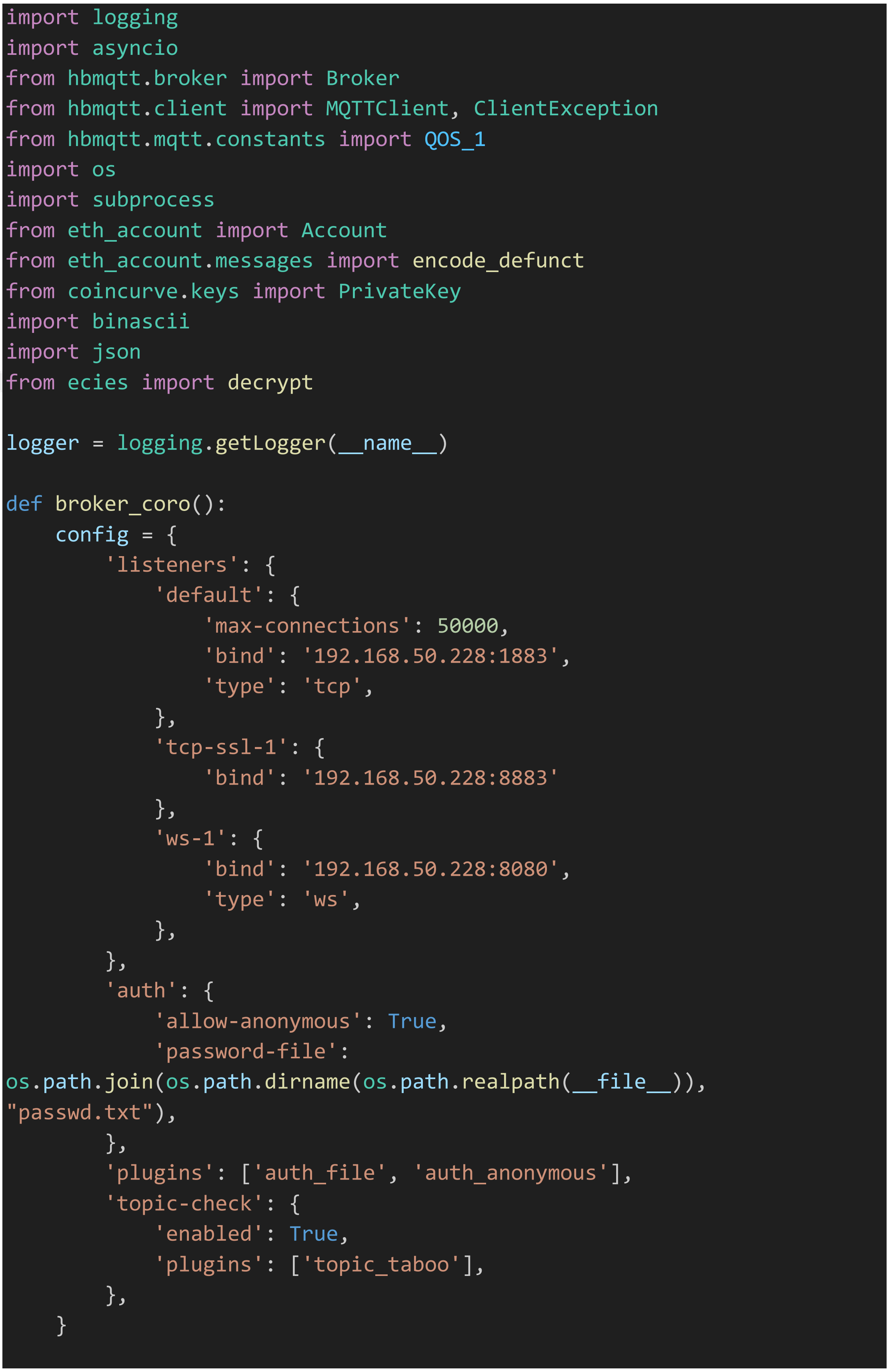
[76] I. Automation, "Ignition Scada", [https://inductiveautomation.com], (Last Access Date: August 23, 2023)

[77] P H. Hidayat, P. Sukarno and A. A. Wardana, "Overhead Analysis on the Use of Digital Signature in MQTT Protocol", 2019 International Conference on Electrical Engineering and Informatics (ICEEI)*,* 2019, Bandung, Indonesia, 2019, pp. 87-92, doi: 10.1109/ICEEI47359.2019.8988861.

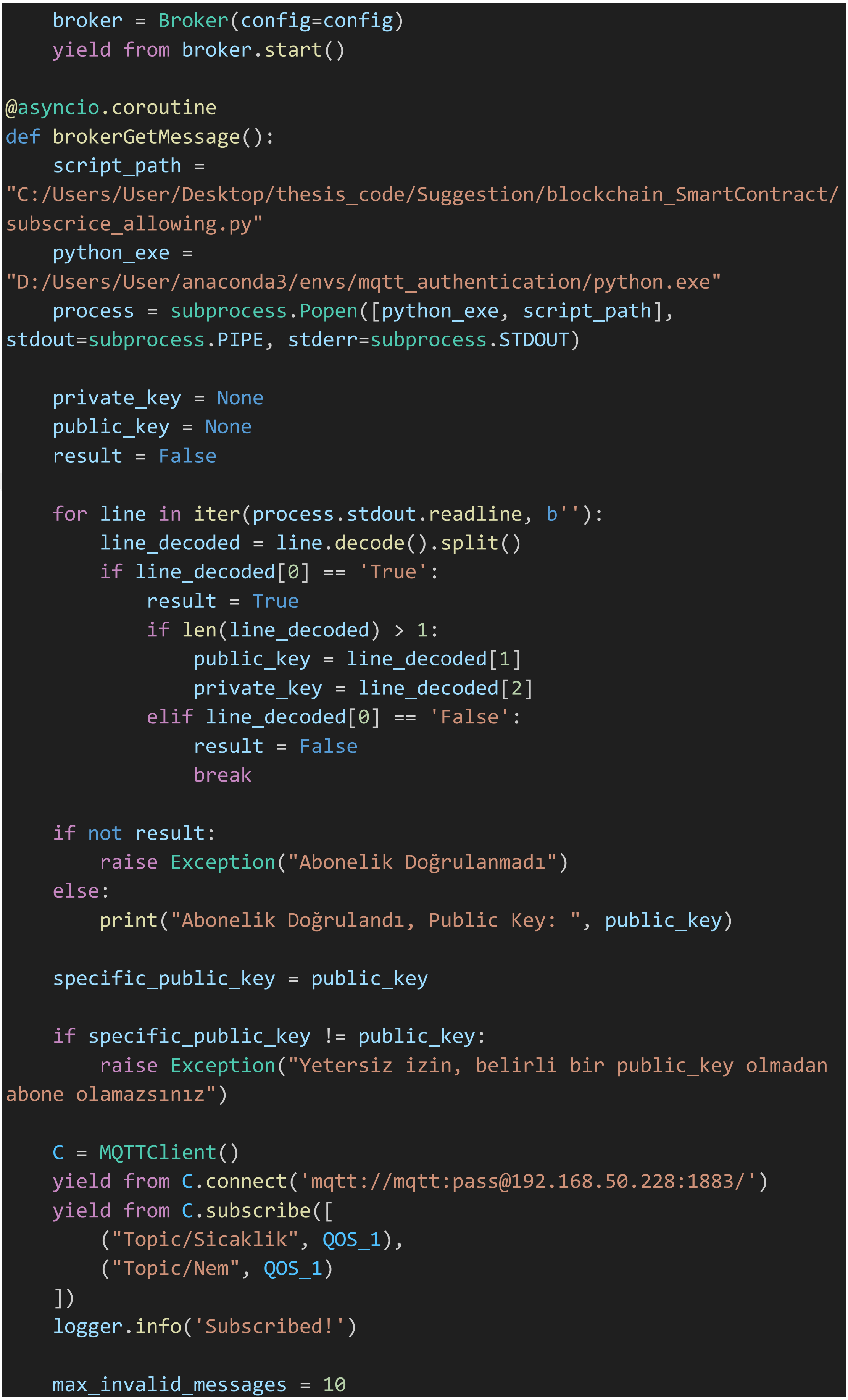
[78] Amanlou, Sanaz & Abu Bakar, Khairul Azmi, "Lightweight Security Mechanism over MQTT Protocol for IoT Devices", (IJACSA) International Journal of Advanced Computer Science and Applications, 2020, pp. 202-207.

APPENDIXES

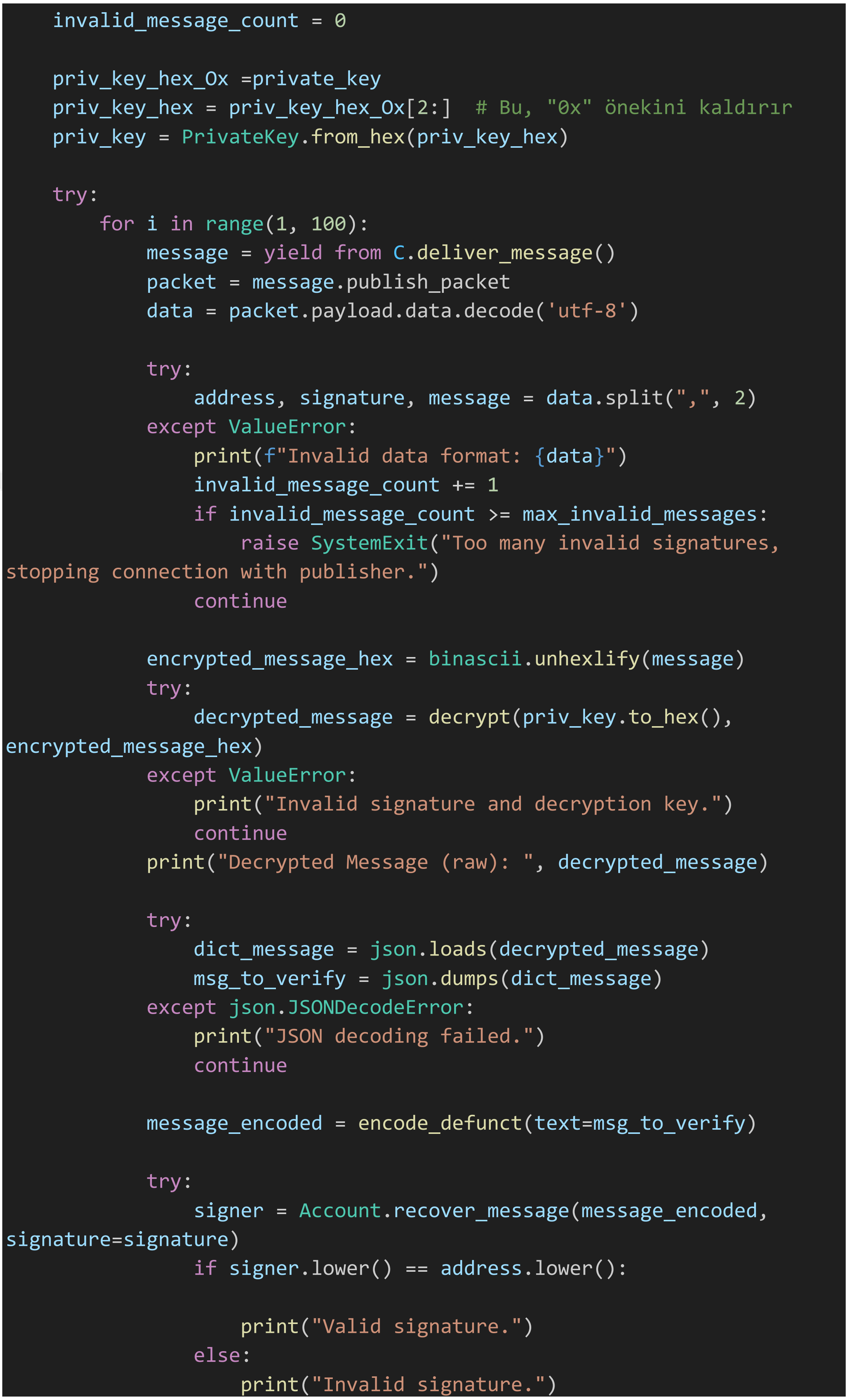
A: MQTT Broker Code



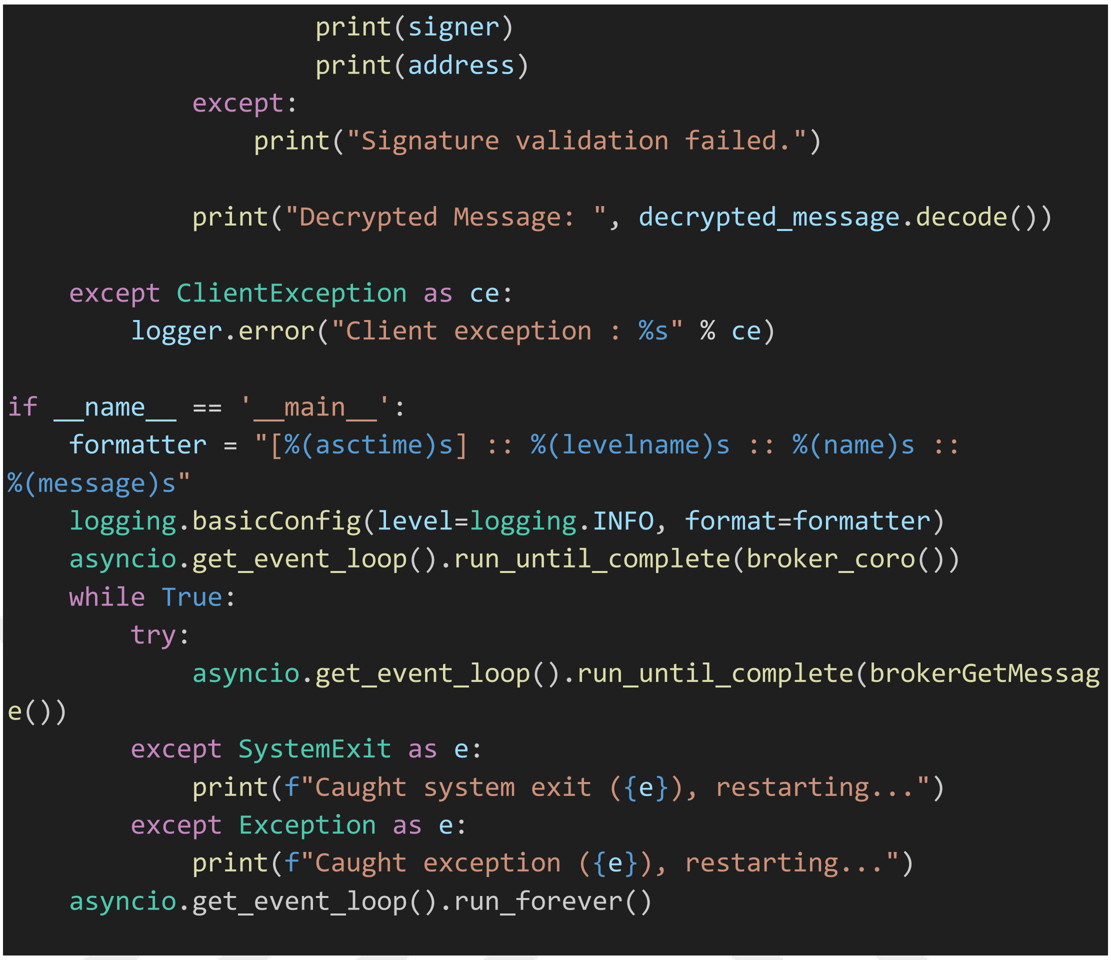
```
import logging
import asyncio
from hbmqtt.broker import Broker
from hbmqtt.client import MQTTClient, ClientException
from hbmqtt.mqtt.constants import QOS_1
import os
import subprocess
from eth_account import Account
from eth_account.messages import encode_defunct
from coincurve.keys import PrivateKey
import binascii
import json
from ecies import decrypt

logger = logging.getLogger(__name__)

def broker_coro():
    config = {
        'listeners': {
            'default': {
                'max-connections': 50000,
                'bind': '192.168.50.228:1883',
                'type': 'tcp',
            },
            'tcp-ssl-1': {
                'bind': '192.168.50.228:8883'
            },
            'ws-1': {
                'bind': '192.168.50.228:8080',
                'type': 'ws',
            },
        },
        'auth': {
            'allow-anonymous': True,
            'password-file':
os.path.join(os.path.dirname(os.path.realpath(__file__)),
"passwd.txt"),
        },
        'plugins': ['auth_file', 'auth_anonymous'],
        'topic-check': {
            'enabled': True,
            'plugins': ['topic_taboo'],
        },
    }
```



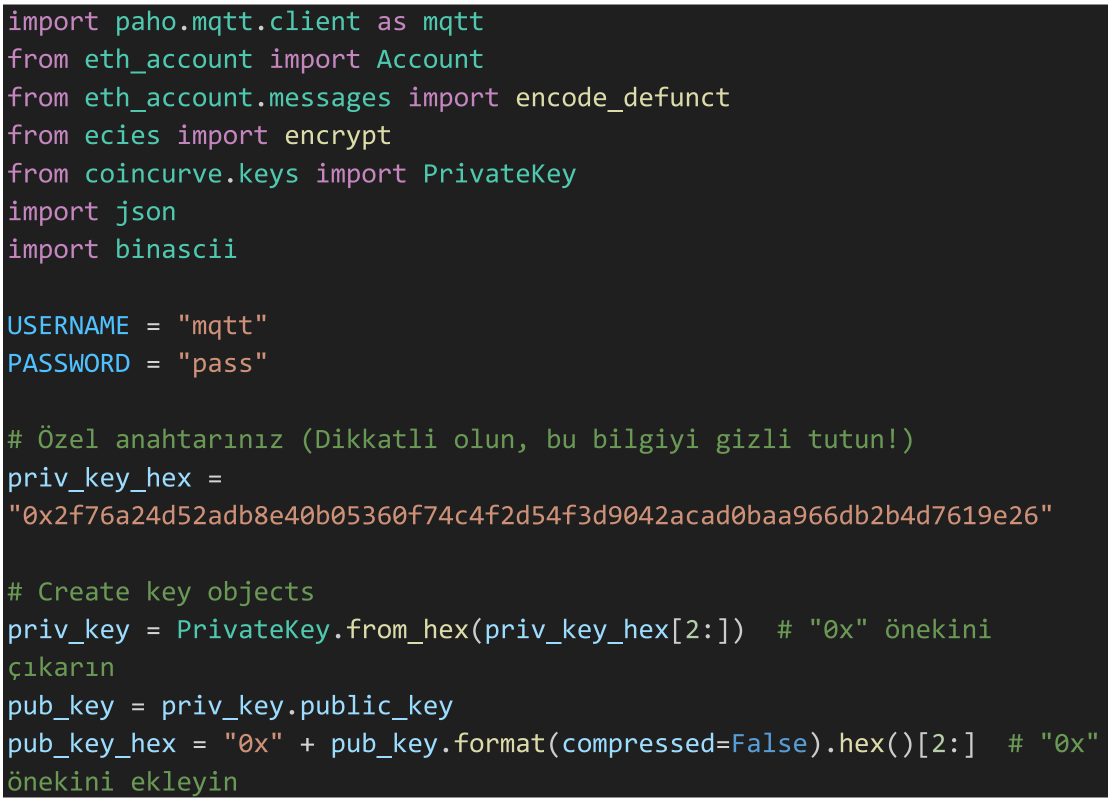
```
    broker = Broker(config=config)
    yield from broker.start()

@asyncio.coroutine
def brokerGetMessage():
    script_path =
"C:/Users/User/Desktop/thesis_code/Suggestion/blockchain_SmartContract/
subscrice_allowing.py"
    python_exe =
"D:/Users/User/anaconda3/envs/mqtt_authentication/python.exe"
    process = subprocess.Popen([python_exe, script_path],
stdout=subprocess.PIPE, stderr=subprocess.STDOUT)

    private_key = None
    public_key = None
    result = False

    for line in iter(process.stdout.readline, b''):
        line_decoded = line.decode().split()
        if line_decoded[0] == 'True':
            result = True
            if len(line_decoded) > 1:
                public_key = line_decoded[1]
                private_key = line_decoded[2]
            elif line_decoded[0] == 'False':
                result = False
                break

    if not result:
        raise Exception("Abonelik Doğrulanmadı")
    else:
        print("Abonelik Doğrulandı, Public Key: ", public_key)

    specific_public_key = public_key

    if specific_public_key != public_key:
        raise Exception("Yetersiz izin, belirli bir public_key olmadan
abone olamazsınız")

    C = MQTTClient()
    yield from C.connect('mqtt://mqtt:pass@192.168.50.228:1883/')
    yield from C.subscribe([
        ("Topic/Sicaklik", QOS_1),
        ("Topic/Nem", QOS_1)
    ])
    logger.info('Subscribed!')

    max_invalid_messages = 10
```



```
    invalid_message_count = 0

    priv_key_hex_Ox =private_key
    priv_key_hex = priv_key_hex_Ox[2:]  # Bu, "0x" önekini kaldırır
    priv_key = PrivateKey.from_hex(priv_key_hex)

    try:
        for i in range(1, 100):
            message = yield from C.deliver_message()
            packet = message.publish_packet
            data = packet.payload.data.decode('utf-8')

            try:
                address, signature, message = data.split(",", 2)
            except ValueError:
                print(f"Invalid data format: {data}")
                invalid_message_count += 1
                if invalid_message_count >= max_invalid_messages:
                    raise SystemExit("Too many invalid signatures,
stopping connection with publisher.")
                continue

            encrypted_message_hex = binascii.unhexlify(message)
            try:
                decrypted_message = decrypt(priv_key.to_hex(),
encrypted_message_hex)
            except ValueError:
                print("Invalid signature and decryption key.")
                continue
            print("Decrypted Message (raw): ", decrypted_message)

            try:
                dict_message = json.loads(decrypted_message)
                msg_to_verify = json.dumps(dict_message)
            except json.JSONDecodeError:
                print("JSON decoding failed.")
                continue

            message_encoded = encode_defunct(text=msg_to_verify)

            try:
                signer = Account.recover_message(message_encoded,
signature=signature)
                if signer.lower() == address.lower():

                    print("Valid signature.")
                else:
                    print("Invalid signature.")
```



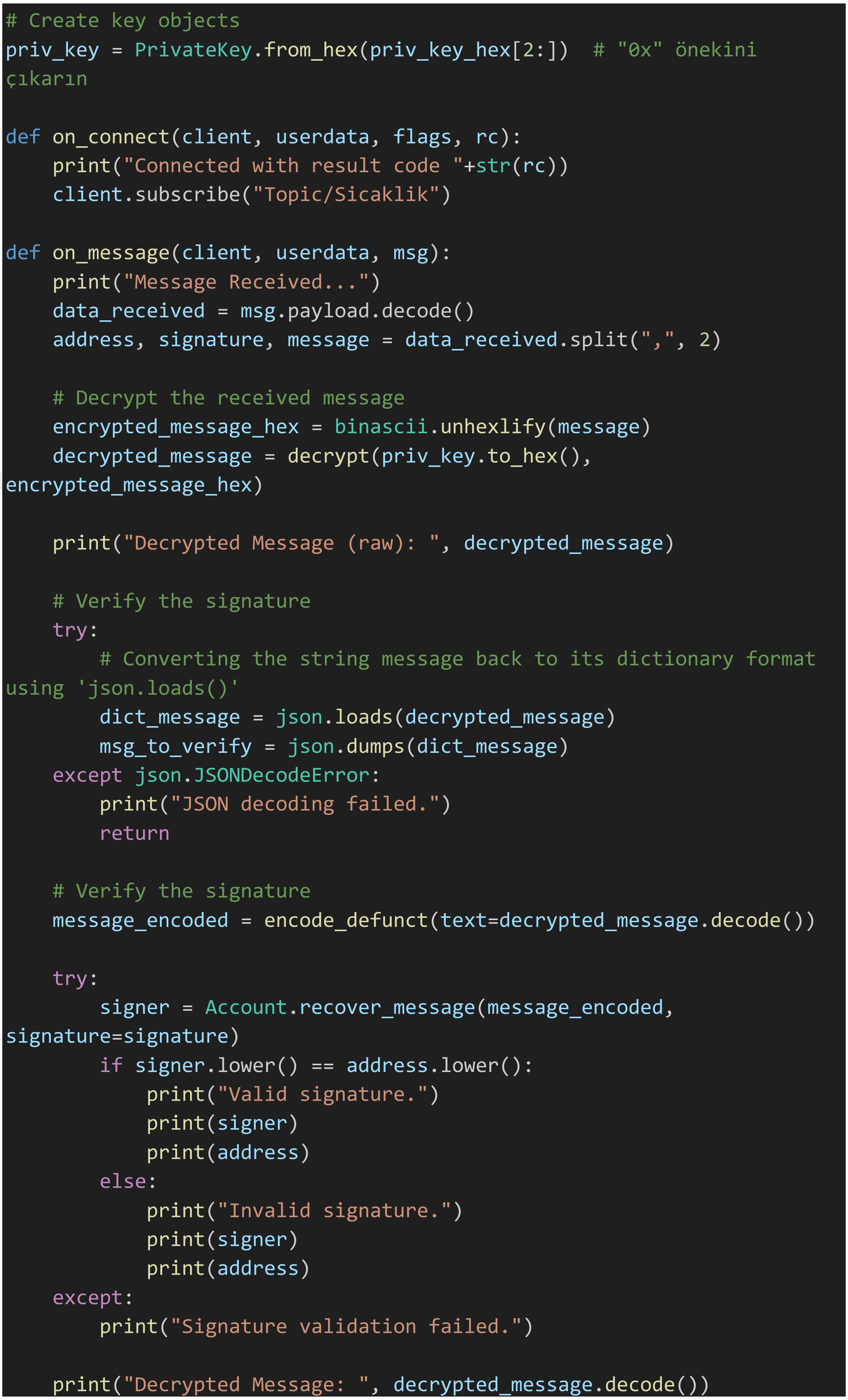
```
                    print(signer)
                    print(address)
            except:
                print("Signature validation failed.")

            print("Decrypted Message: ", decrypted_message.decode())

    except ClientException as ce:
        logger.error("Client exception : %s" % ce)

if __name__ == '__main__':
    formatter = "[%(asctime)s] :: %(levelname)s :: %(name)s :: 
%(message)s"
    logging.basicConfig(level=logging.INFO, format=formatter)
    asyncio.get_event_loop().run_until_complete(broker_coro())
    while True:
        try:
            asyncio.get_event_loop().run_until_complete(brokerGetMessag
e())
        except SystemExit as e:
            print(f"Caught system exit ({e}), restarting...")
        except Exception as e:
            print(f"Caught exception ({e}), restarting...")
    asyncio.get_event_loop().run_forever()
```



```
import paho.mqtt.client as mqtt
from eth_account import Account
from eth_account.messages import encode_defunct
from ecies import encrypt
from coincurve.keys import PrivateKey
import json
import binascii

USERNAME = "mqtt"
PASSWORD = "pass"

# Özel anahtarınız (Dikkatli olun, bu bilgiyi gizli tutun!)
priv_key_hex = 
"0x2f76a24d52adb8e40b05360f74c4f2d54f3d9042acad0baa966db2b4d7619e26"

# Create key objects
priv_key = PrivateKey.from_hex(priv_key_hex[2:])  # "0x" önekini 
çıkarın
pub_key = priv_key.public_key
pub_key_hex = "0x" + pub_key.format(compressed=False).hex()[2:]  # "0x" 
önekini ekleyin
```

```
print(pub_key_hex)

# Account nesnesini oluştur
account = Account.from_key(priv_key_hex)

client = mqtt.Client()
client.username_pw_set(USERNAME, PASSWORD)

client.connect('192.168.50.228', 1883)

while True:
    # Kullanıcıdan mesajı al
    user_input = input('Message : ')

    # Dijital imzalı mesajı oluştur
    message = {"message": user_input}
    # Dijital imzalı mesajı oluştur
    message_encoded = encode_defunct(text=user_input)
    signed_message = account.sign_message(message_encoded)
    signature = signed_message.signature.hex()

    # Mesajı şifrele
    replaced_message = user_input.replace(":", "#COLON#")
    encrypted_message = encrypt(pub_key_hex[2:],
replaced_message.encode())  # "0x" önekini çıkarın

    # Mesajı yayınla
    data_to_send =
f"{account.address},{signature},{binascii.hexlify(encrypted_message).de
code()}"
    client.publish("Topic/Sicaklik", data_to_send)
```

```
import paho.mqtt.client as mqtt
from ecies import decrypt
from coincurve.keys import PrivateKey
import binascii
from eth_account import Account
from eth_account.messages import encode_defunct
import json

USERNAME = "mqtt"
PASSWORD = "pass"

# Özel anahtarınız
priv_key_hex =
"0x2f76a24d52adb8e40b05360f74c4f2d54f3d9042acad0baa966db2b4d7619e26"
```

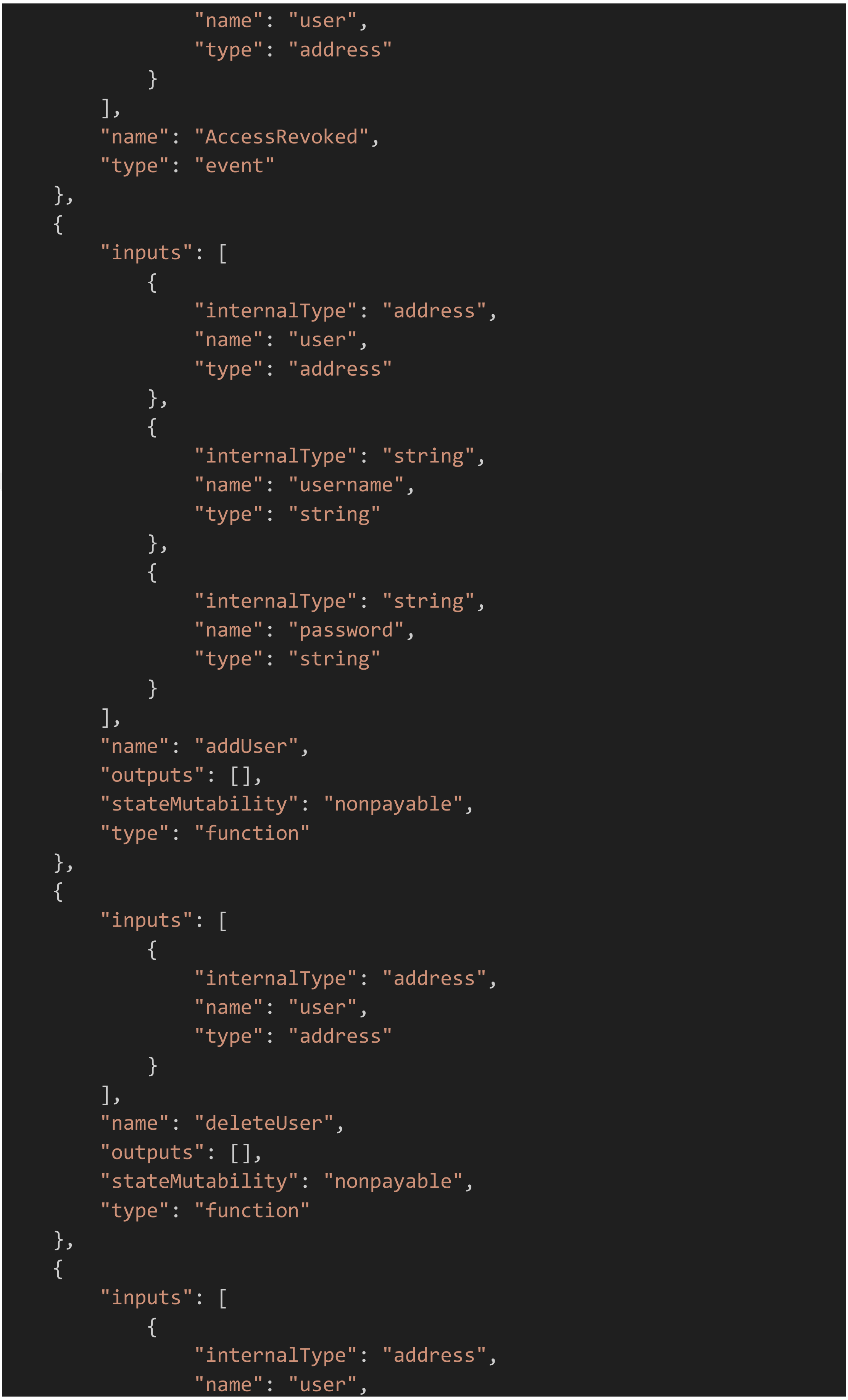


```
# Create key objects
priv_key = PrivateKey.from_hex(priv_key_hex[2:])  # "0x" önekini
çıkarın

def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))
    client.subscribe("Topic/Sicaklik")

def on_message(client, userdata, msg):
    print("Message Received...")
    data_received = msg.payload.decode()
    address, signature, message = data_received.split(",", 2)

    # Decrypt the received message
    encrypted_message_hex = binascii.unhexlify(message)
    decrypted_message = decrypt(priv_key.to_hex(),
encrypted_message_hex)

    print("Decrypted Message (raw): ", decrypted_message)

    # Verify the signature
    try:
        # Converting the string message back to its dictionary format
using 'json.loads()'
        dict_message = json.loads(decrypted_message)
        msg_to_verify = json.dumps(dict_message)
    except json.JSONDecodeError:
        print("JSON decoding failed.")
        return

    # Verify the signature
    message_encoded = encode_defunct(text=decrypted_message.decode())

    try:
        signer = Account.recover_message(message_encoded,
signature=signature)
        if signer.lower() == address.lower():
            print("Valid signature.")
            print(signer)
            print(address)
        else:
            print("Invalid signature.")
            print(signer)
            print(address)
    except:
        print("Signature validation failed.")

    print("Decrypted Message: ", decrypted_message.decode())
```

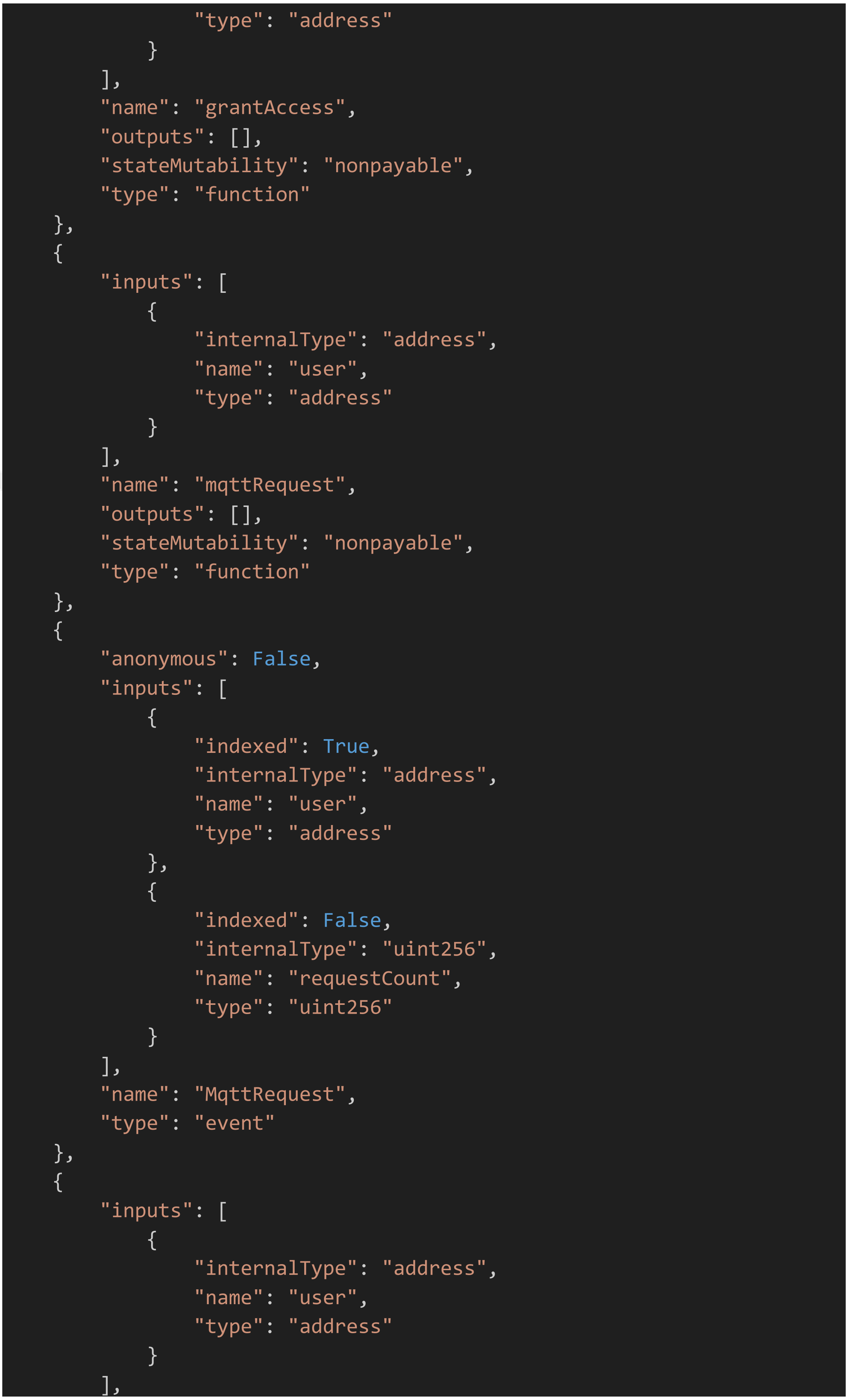
```

client = mqtt.Client()
client.username_pw_set(USERNAME, PASSWORD)

client.on_connect = on_connect
client.on_message = on_message

client.connect('192.168.50.228', 1883)

client.loop_forever()
```

B: Python Code: Integration of Broker and Smart Contract

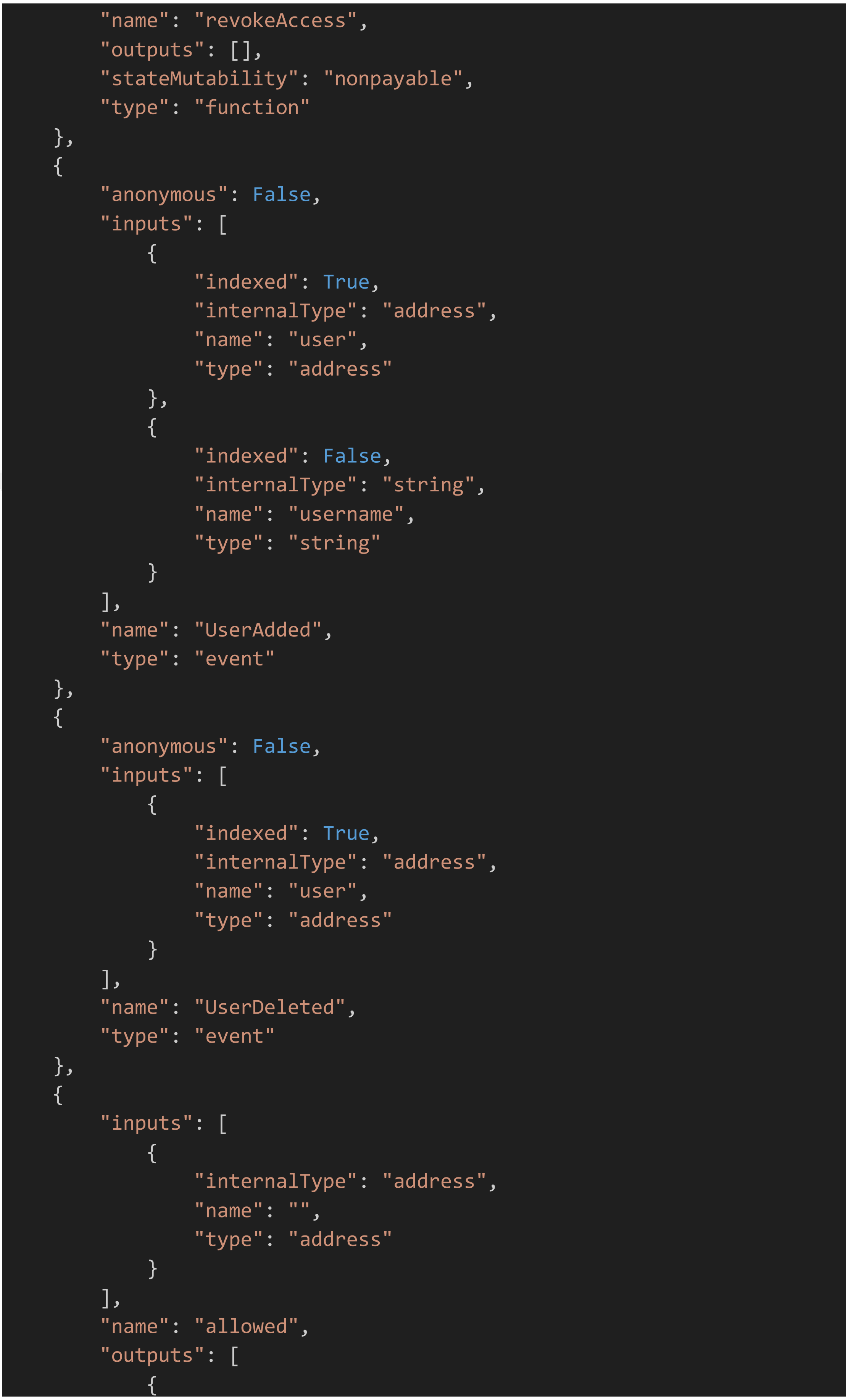
```
import tkinter as tk
import paho.mqtt.client as mqtt
from web3 import Web3
import pyotp
from threading import Thread
import subcribe
from datetime import datetime


ganache_url = "http://127.0.0.1:8545"
abi = [
    {
        "inputs": [],
        "stateMutability": "nonpayable",
        "type": "constructor"
    },
    {
        "anonymous": False,
        "inputs": [
            {
                "indexed": True,
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "AccessGranted",
        "type": "event"
    },
    {
        "anonymous": False,
        "inputs": [
            {
                "indexed": True,
                "internalType": "address",
```



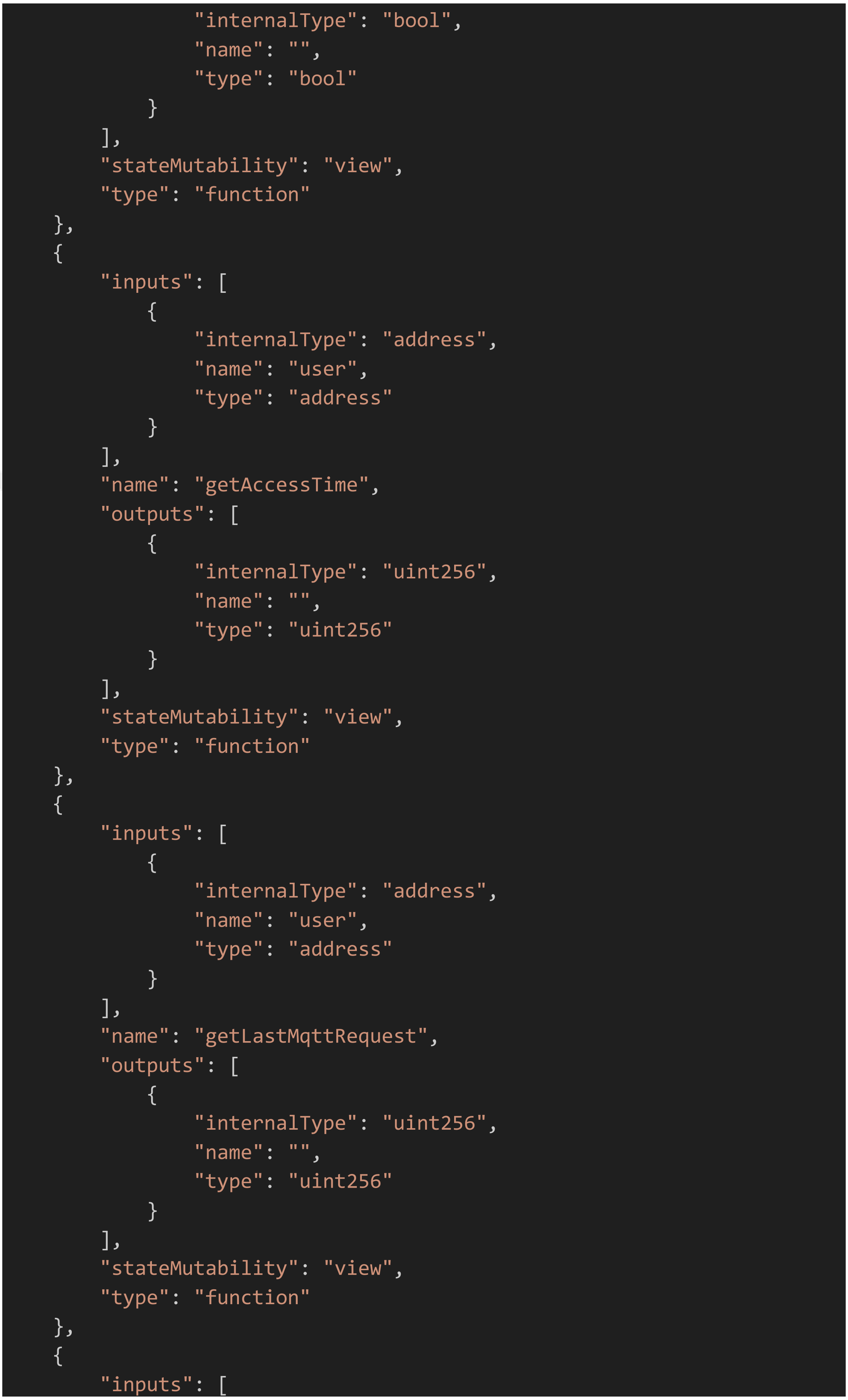
```
                "name": "user",
                "type": "address"
            }
        ],
        "name": "AccessRevoked",
        "type": "event"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            },
            {
                "internalType": "string",
                "name": "username",
                "type": "string"
            },
            {
                "internalType": "string",
                "name": "password",
                "type": "string"
            }
        ],
        "name": "addUser",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "deleteUser",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
```



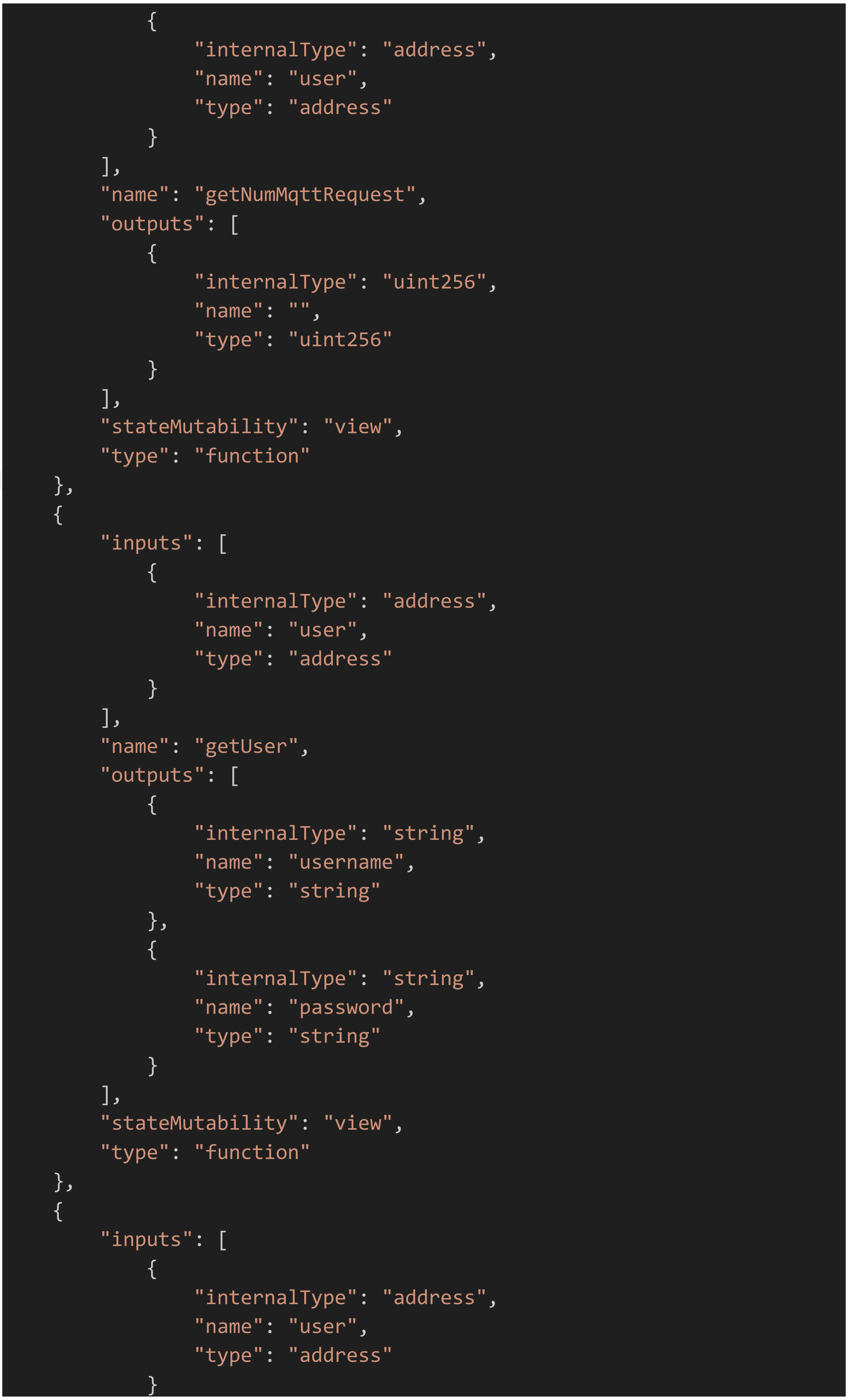
```
                "type": "address"
            }
        ],
        "name": "grantAccess",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "mqttRequest",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "anonymous": False,
        "inputs": [
            {
                "indexed": True,
                "internalType": "address",
                "name": "user",
                "type": "address"
            },
            {
                "indexed": False,
                "internalType": "uint256",
                "name": "requestCount",
                "type": "uint256"
            }
        ],
        "name": "MqttRequest",
        "type": "event"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
```



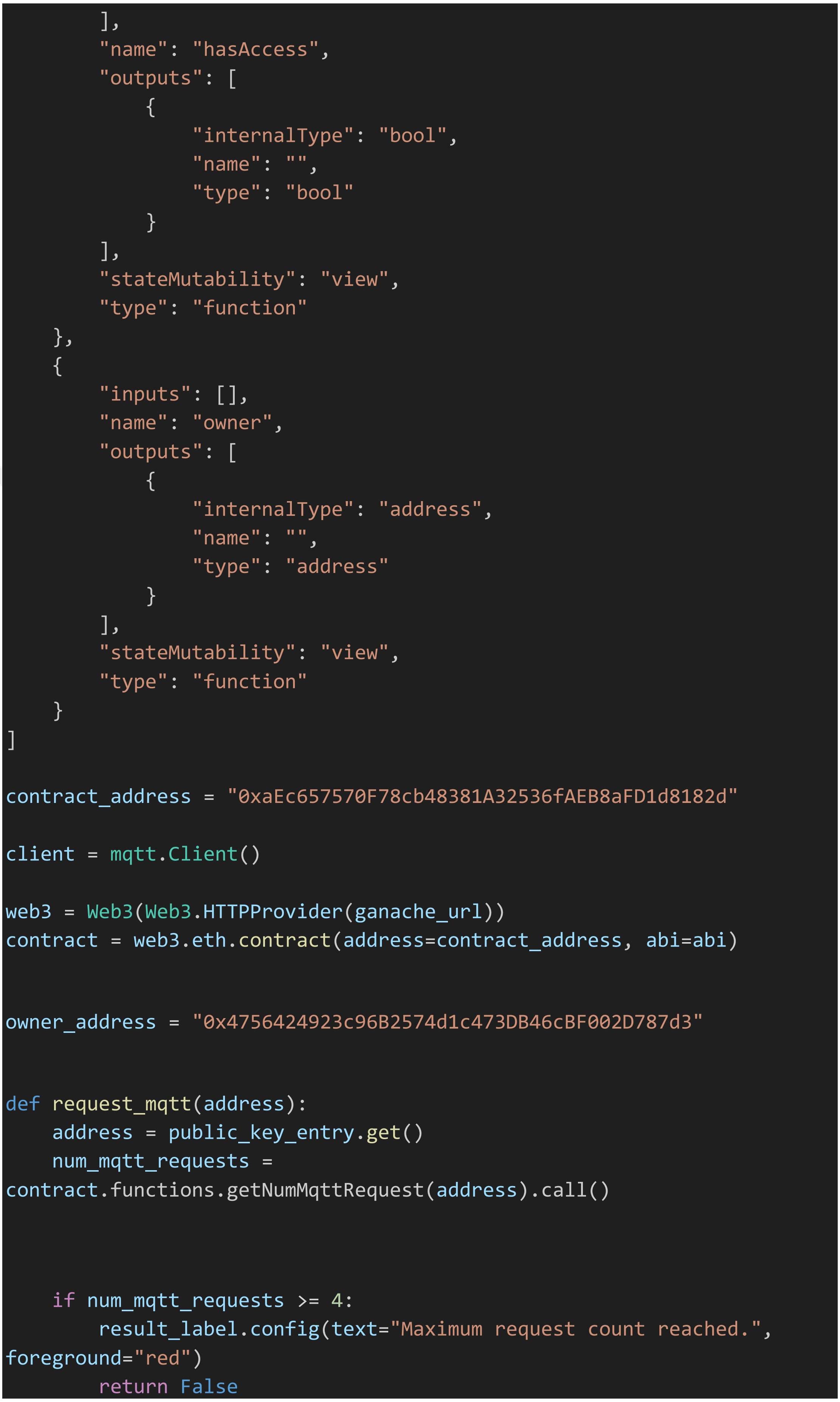
```
        "name": "revokeAccess",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "anonymous": False,
        "inputs": [
            {
                "indexed": True,
                "internalType": "address",
                "name": "user",
                "type": "address"
            },
            {
                "indexed": False,
                "internalType": "string",
                "name": "username",
                "type": "string"
            }
        ],
        "name": "UserAdded",
        "type": "event"
    },
    {
        "anonymous": False,
        "inputs": [
            {
                "indexed": True,
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "UserDeleted",
        "type": "event"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "",
                "type": "address"
            }
        ],
        "name": "allowed",
        "outputs": [
            {
```



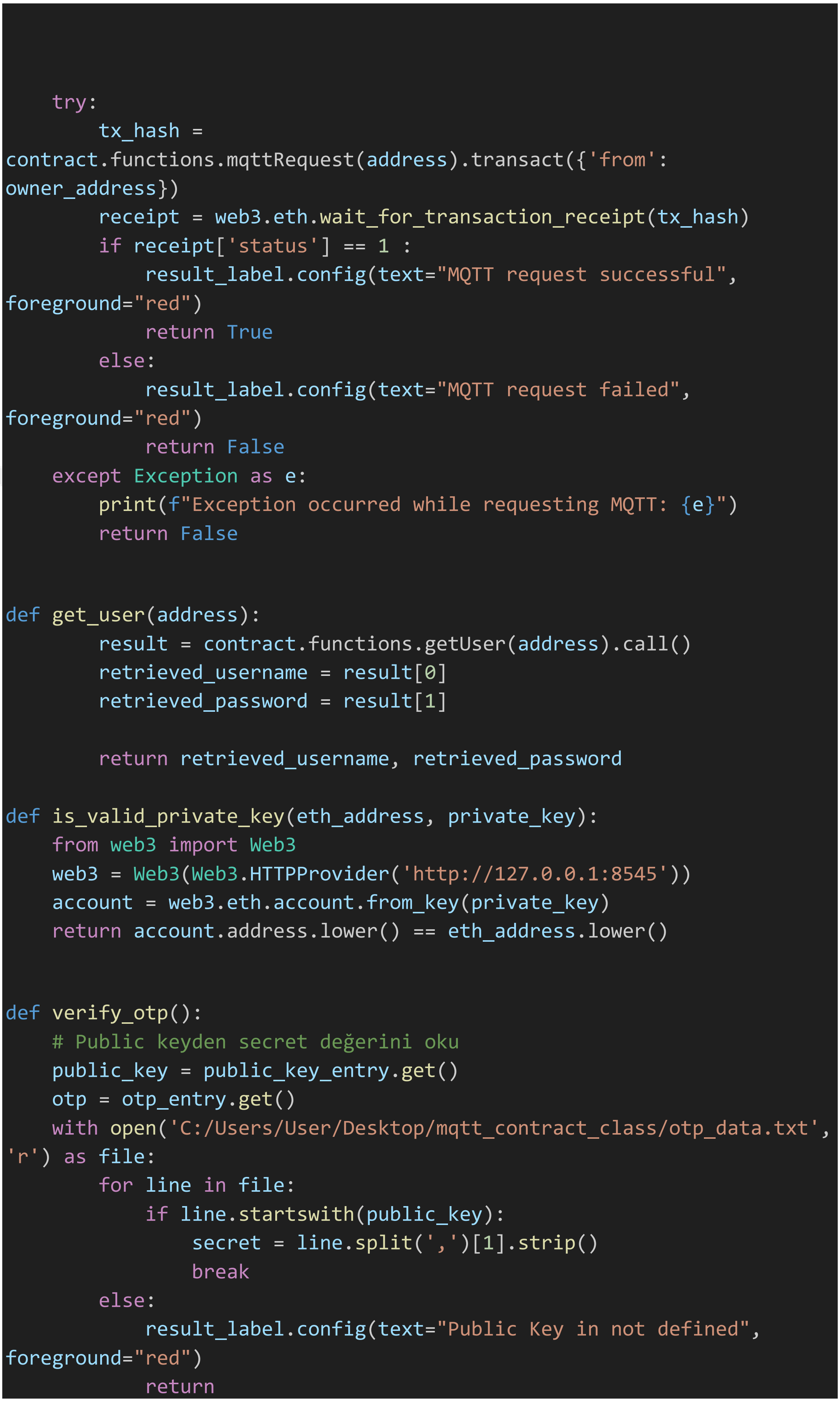
```
                "internalType": "bool",
                "name": "",
                "type": "bool"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "getAccessTime",
        "outputs": [
            {
                "internalType": "uint256",
                "name": "",
                "type": "uint256"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "getLastMqttRequest",
        "outputs": [
            {
                "internalType": "uint256",
                "name": "",
                "type": "uint256"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
```



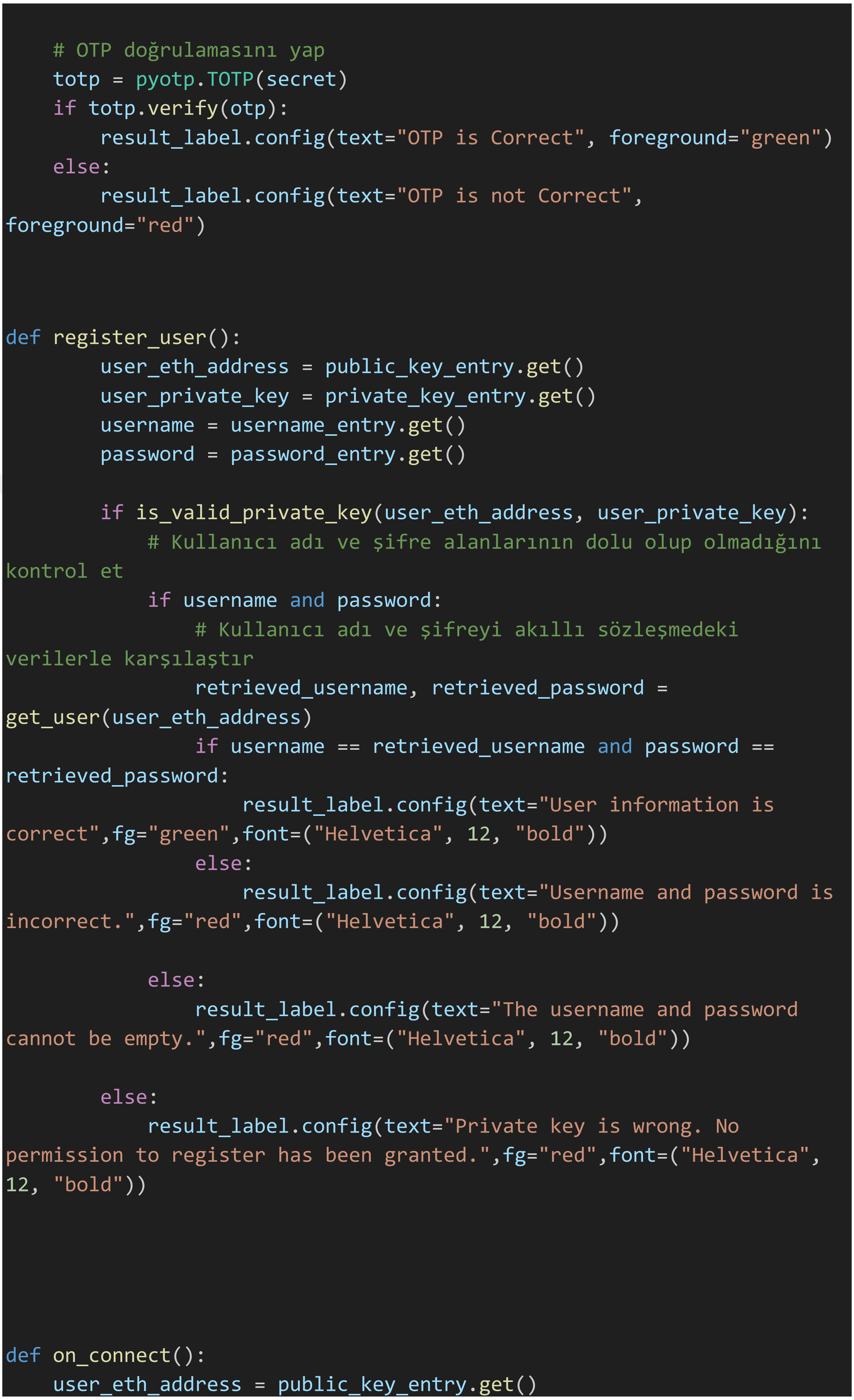
```
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "getNumMqttRequest",
        "outputs": [
            {
                "internalType": "uint256",
                "name": "",
                "type": "uint256"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "getUser",
        "outputs": [
            {
                "internalType": "string",
                "name": "username",
                "type": "string"
            },
            {
                "internalType": "string",
                "name": "password",
                "type": "string"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
```



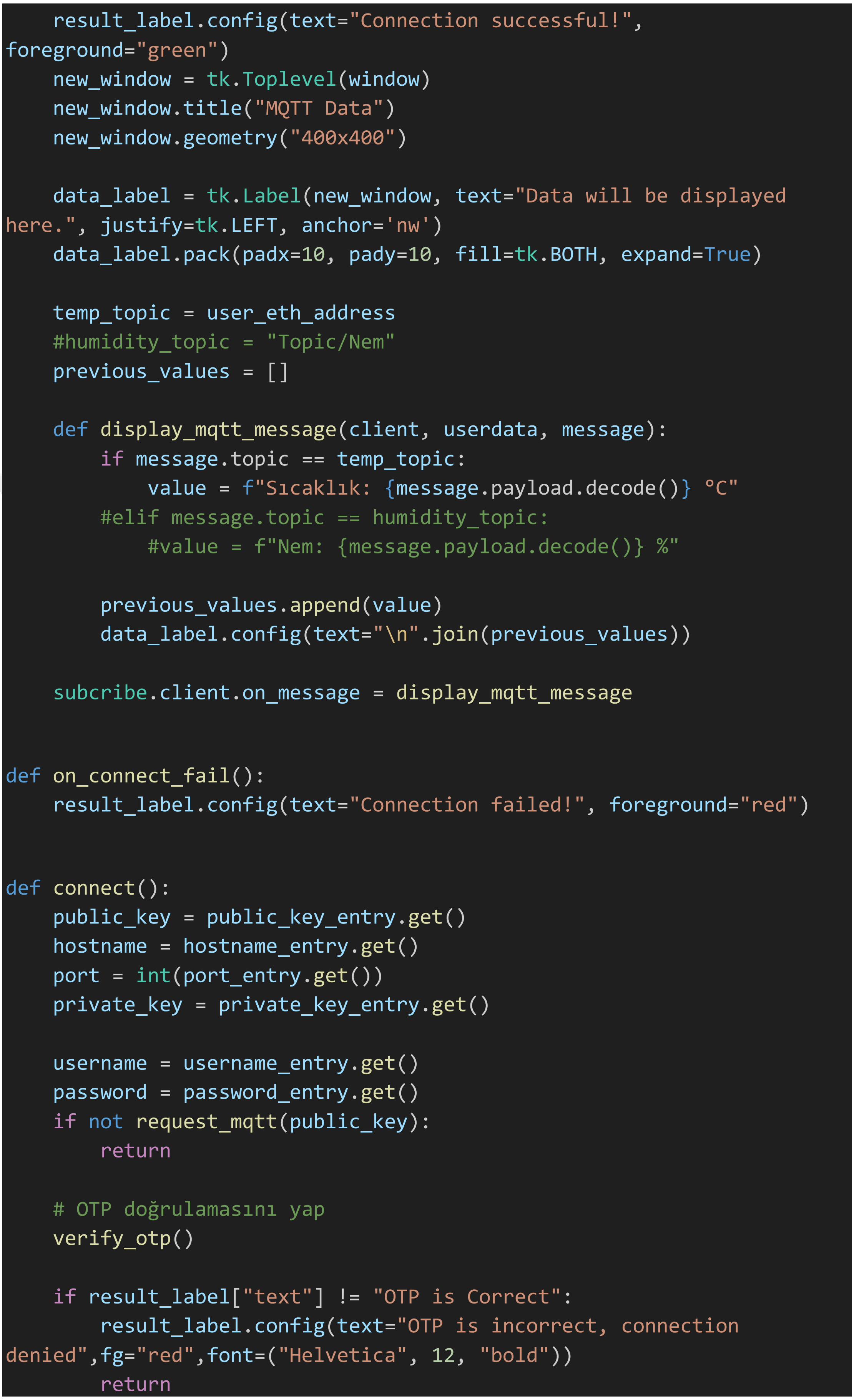
```
        ],
        "name": "hasAccess",
        "outputs": [
            {
                "internalType": "bool",
                "name": "",
                "type": "bool"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [],
        "name": "owner",
        "outputs": [
            {
                "internalType": "address",
                "name": "",
                "type": "address"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    }
]

contract_address = "0xaEc657570F78cb48381A32536fAEB8aFD1d8182d"

client = mqtt.Client()

web3 = Web3(Web3.HTTPProvider(ganache_url))
contract = web3.eth.contract(address=contract_address, abi=abi)


owner_address = "0x4756424923c96B2574d1c473DB46cBF002D787d3"


def request_mqtt(address):
    address = public_key_entry.get()
    num_mqtt_requests =
contract.functions.getNumMqttRequest(address).call()



    if num_mqtt_requests >= 4:
        result_label.config(text="Maximum request count reached.",
foreground="red")
        return False
```



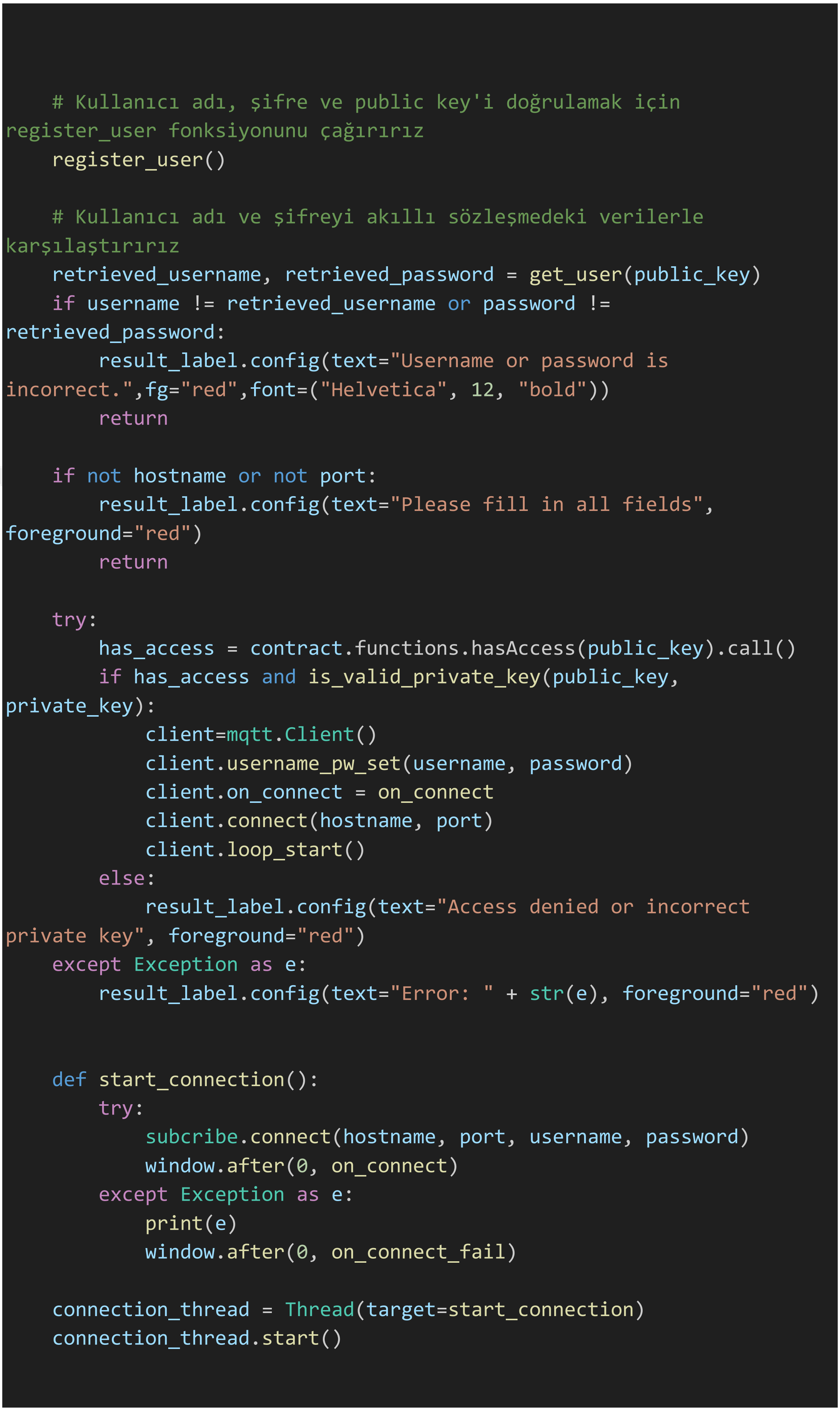
```
    try:
        tx_hash = 
contract.functions.mqttRequest(address).transact({'from': 
owner_address})
        receipt = web3.eth.wait_for_transaction_receipt(tx_hash)
        if receipt['status'] == 1 :
            result_label.config(text="MQTT request successful", 
foreground="red")
            return True
        else:
            result_label.config(text="MQTT request failed", 
foreground="red")
            return False
    except Exception as e:
        print(f"Exception occurred while requesting MQTT: {e}")
        return False


def get_user(address):
        result = contract.functions.getUser(address).call()
        retrieved_username = result[0]
        retrieved_password = result[1]

        return retrieved_username, retrieved_password

def is_valid_private_key(eth_address, private_key):
    from web3 import Web3
    web3 = Web3(Web3.HTTPProvider('http://127.0.0.1:8545'))
    account = web3.eth.account.from_key(private_key)
    return account.address.lower() == eth_address.lower()


def verify_otp():
    # Public keyden secret değerini oku
    public_key = public_key_entry.get()
    otp = otp_entry.get()
    with open('C:/Users/User/Desktop/mqtt_contract_class/otp_data.txt', 
'r') as file:
        for line in file:
            if line.startswith(public_key):
                secret = line.split(',')[1].strip()
                break
        else:
            result_label.config(text="Public Key in not defined", 
foreground="red")
            return
```



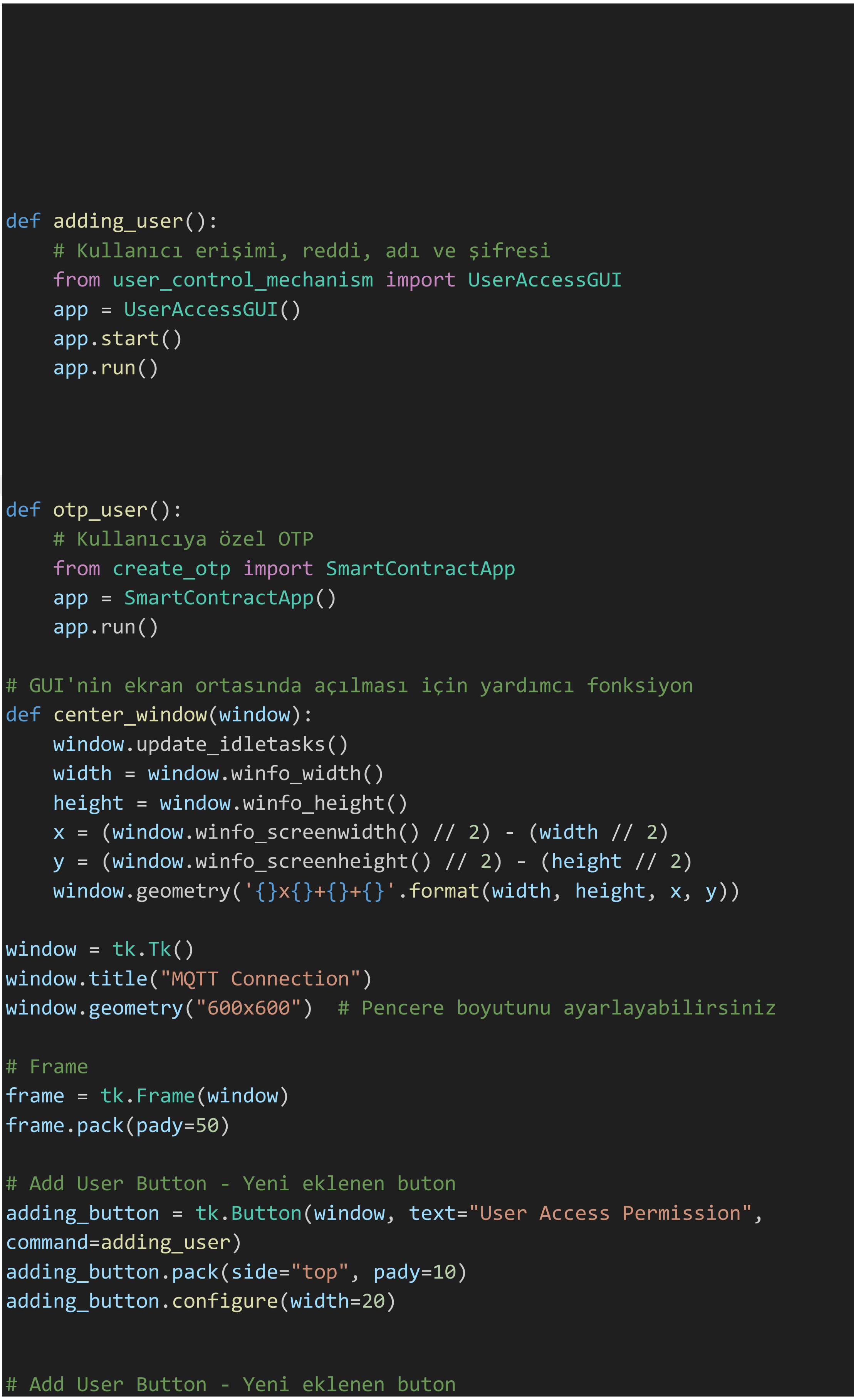
```
    # OTP doğrulamasını yap
    totp = pyotp.TOTP(secret)
    if totp.verify(otp):
        result_label.config(text="OTP is Correct", foreground="green")
    else:
        result_label.config(text="OTP is not Correct",
foreground="red")



def register_user():
        user_eth_address = public_key_entry.get()
        user_private_key = private_key_entry.get()
        username = username_entry.get()
        password = password_entry.get()

        if is_valid_private_key(user_eth_address, user_private_key):
            # Kullanıcı adı ve şifre alanlarının dolu olup olmadığını
kontrol et
            if username and password:
                # Kullanıcı adı ve şifreyi akıllı sözleşmedeki
verilerle karşılaştır
                retrieved_username, retrieved_password =
get_user(user_eth_address)
                if username == retrieved_username and password ==
retrieved_password:
                    result_label.config(text="User information is
correct",fg="green",font=("Helvetica", 12, "bold"))
                else:
                    result_label.config(text="Username and password is
incorrect.",fg="red",font=("Helvetica", 12, "bold"))

            else:
                result_label.config(text="The username and password
cannot be empty.",fg="red",font=("Helvetica", 12, "bold"))

        else:
            result_label.config(text="Private key is wrong. No
permission to register has been granted.",fg="red",font=("Helvetica",
12, "bold"))





def on_connect():
    user_eth_address = public_key_entry.get()
```



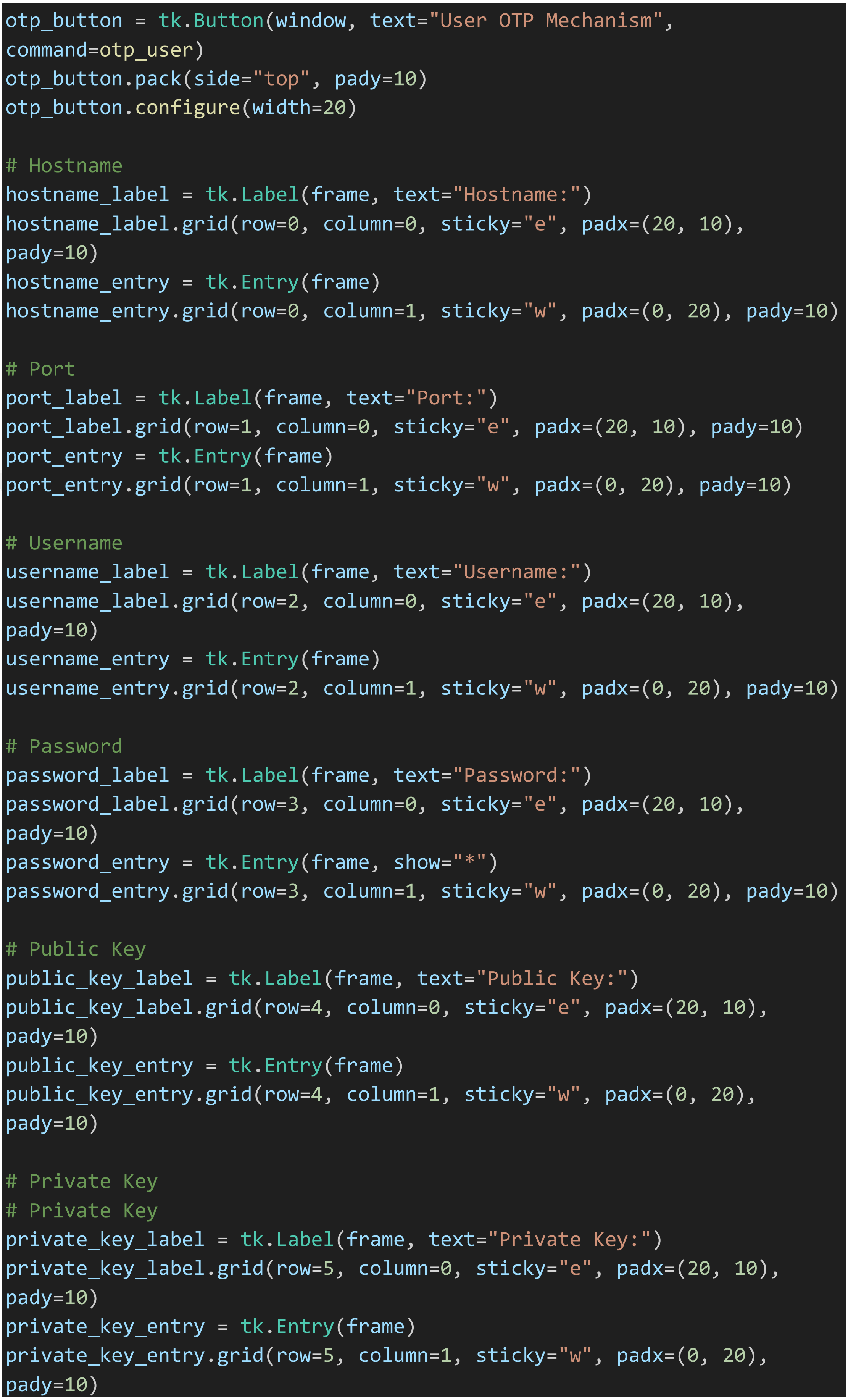
```
    result_label.config(text="Connection successful!",
foreground="green")
    new_window = tk.Toplevel(window)
    new_window.title("MQTT Data")
    new_window.geometry("400x400")

    data_label = tk.Label(new_window, text="Data will be displayed
here.", justify=tk.LEFT, anchor='nw')
    data_label.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)

    temp_topic = user_eth_address
    #humidity_topic = "Topic/Nem"
    previous_values = []

    def display_mqtt_message(client, userdata, message):
        if message.topic == temp_topic:
            value = f"Sıcaklık: {message.payload.decode()} °C"
        #elif message.topic == humidity_topic:
            #value = f"Nem: {message.payload.decode()} %"

        previous_values.append(value)
        data_label.config(text="\n".join(previous_values))

    subcribe.client.on_message = display_mqtt_message


def on_connect_fail():
    result_label.config(text="Connection failed!", foreground="red")


def connect():
    public_key = public_key_entry.get()
    hostname = hostname_entry.get()
    port = int(port_entry.get())
    private_key = private_key_entry.get()

    username = username_entry.get()
    password = password_entry.get()
    if not request_mqtt(public_key):
        return

    # OTP doğrulamasını yap
    verify_otp()

    if result_label["text"] != "OTP is Correct":
        result_label.config(text="OTP is incorrect, connection
denied",fg="red",font=("Helvetica", 12, "bold"))
        return
```



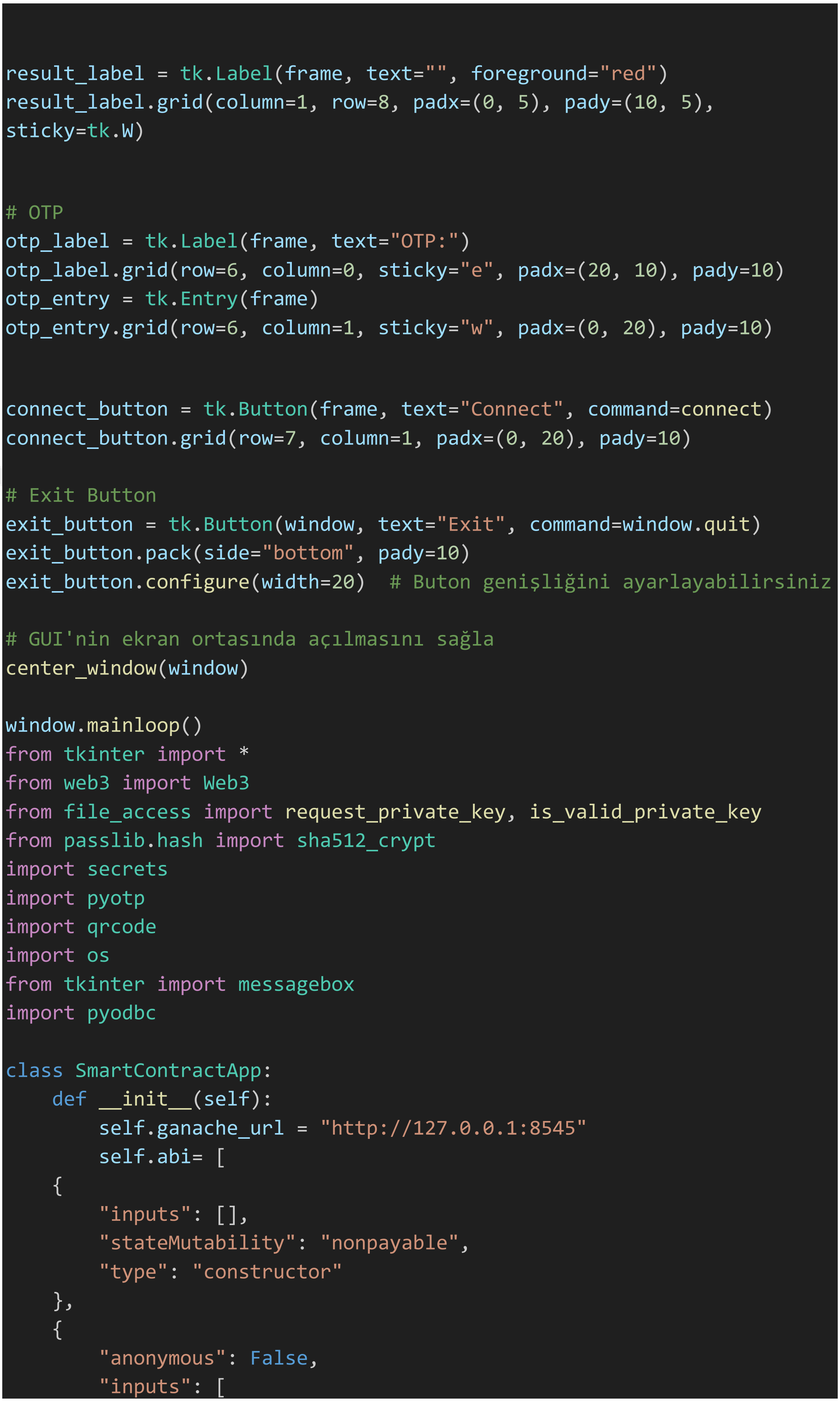
```
    # Kullanıcı adı, şifre ve public key'i doğrulamak için
register_user fonksiyonunu çağırırız
    register_user()

    # Kullanıcı adı ve şifreyi akıllı sözleşmedeki verilerle
karşılaştırırız
    retrieved_username, retrieved_password = get_user(public_key)
    if username != retrieved_username or password !=
retrieved_password:
        result_label.config(text="Username or password is
incorrect.",fg="red",font=("Helvetica", 12, "bold"))
        return

    if not hostname or not port:
        result_label.config(text="Please fill in all fields",
foreground="red")
        return

    try:
        has_access = contract.functions.hasAccess(public_key).call()
        if has_access and is_valid_private_key(public_key,
private_key):
            client=mqtt.Client()
            client.username_pw_set(username, password)
            client.on_connect = on_connect
            client.connect(hostname, port)
            client.loop_start()
        else:
            result_label.config(text="Access denied or incorrect
private key", foreground="red")
    except Exception as e:
        result_label.config(text="Error: " + str(e), foreground="red")


    def start_connection():
        try:
            subcribe.connect(hostname, port, username, password)
            window.after(0, on_connect)
        except Exception as e:
            print(e)
            window.after(0, on_connect_fail)

    connection_thread = Thread(target=start_connection)
    connection_thread.start()
```



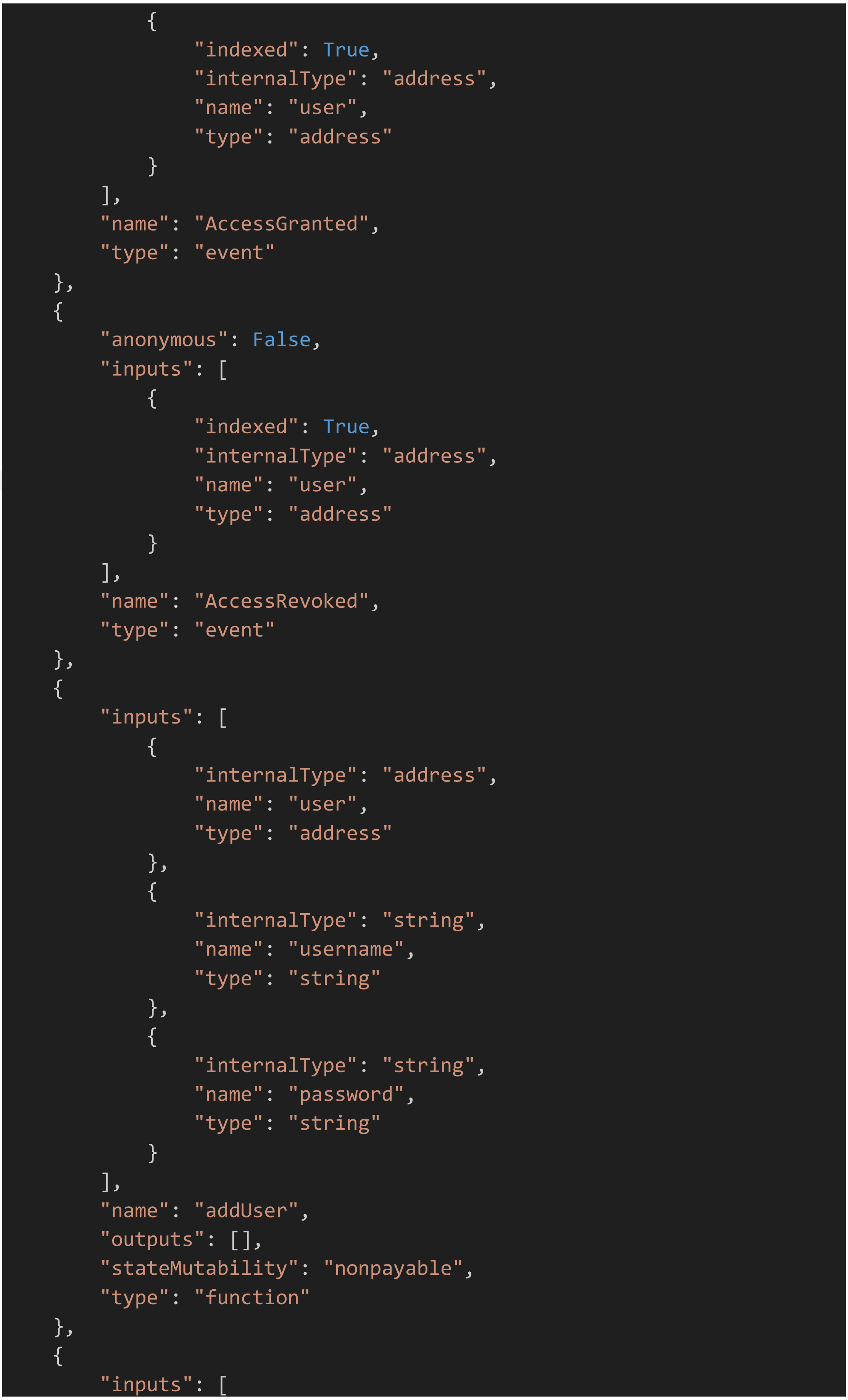
```python
def adding_user():
    # Kullanıcı erişimi, reddi, adı ve şifresi
    from user_control_mechanism import UserAccessGUI
    app = UserAccessGUI()
    app.start()
    app.run()




def otp_user():
    # Kullanıcıya özel OTP
    from create_otp import SmartContractApp
    app = SmartContractApp()
    app.run()

# GUI'nin ekran ortasında açılması için yardımcı fonksiyon
def center_window(window):
    window.update_idletasks()
    width = window.winfo_width()
    height = window.winfo_height()
    x = (window.winfo_screenwidth() // 2) - (width // 2)
    y = (window.winfo_screenheight() // 2) - (height // 2)
    window.geometry('{}x{}+{}+{}'.format(width, height, x, y))

window = tk.Tk()
window.title("MQTT Connection")
window.geometry("600x600")  # Pencere boyutunu ayarlayabilirsiniz

# Frame
frame = tk.Frame(window)
frame.pack(pady=50)

# Add User Button - Yeni eklenen buton
adding_button = tk.Button(window, text="User Access Permission",
command=adding_user)
adding_button.pack(side="top", pady=10)
adding_button.configure(width=20)


# Add User Button - Yeni eklenen buton
```



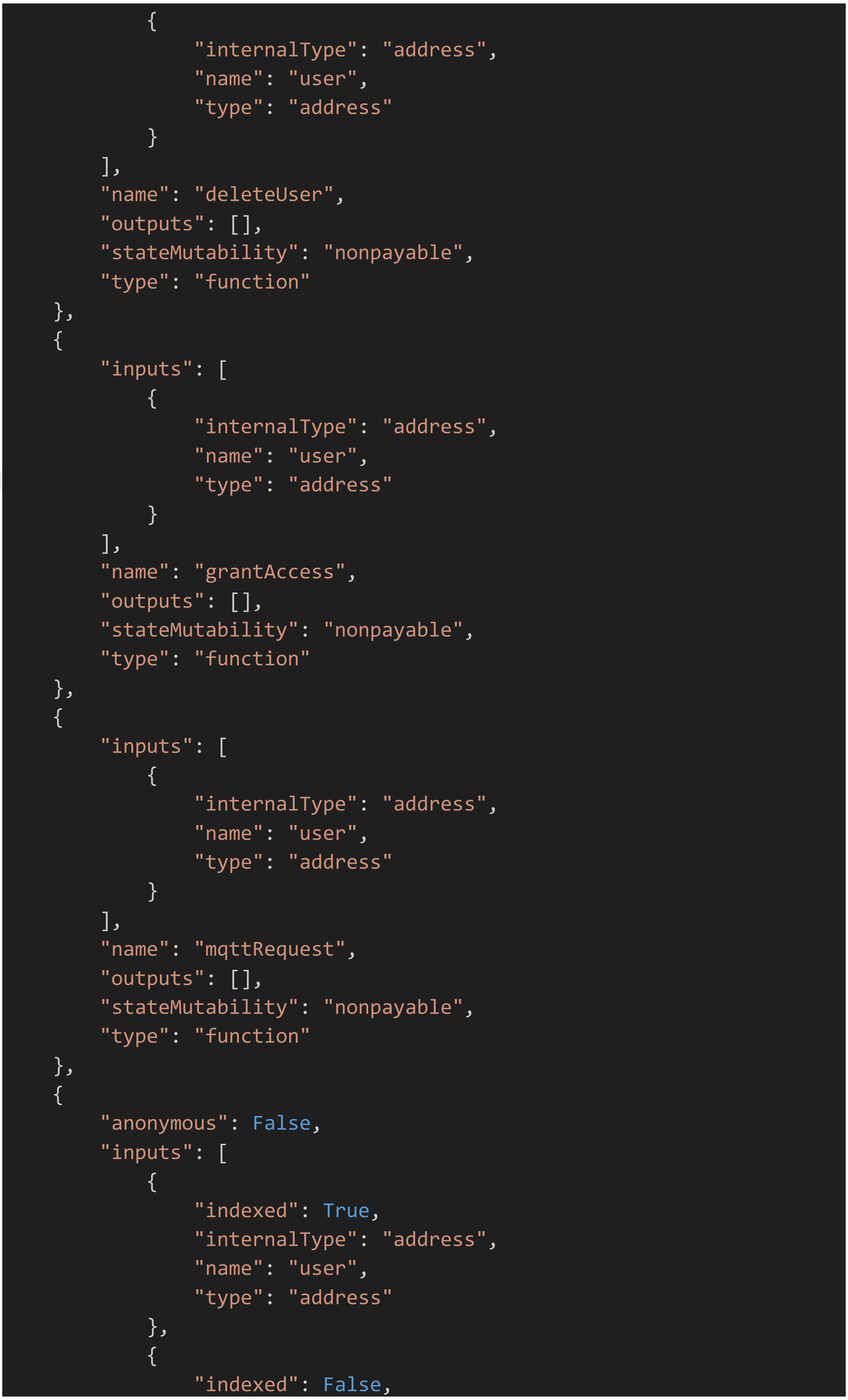
```
otp_button = tk.Button(window, text="User OTP Mechanism",
command=otp_user)
otp_button.pack(side="top", pady=10)
otp_button.configure(width=20)

# Hostname
hostname_label = tk.Label(frame, text="Hostname:")
hostname_label.grid(row=0, column=0, sticky="e", padx=(20, 10),
pady=10)
hostname_entry = tk.Entry(frame)
hostname_entry.grid(row=0, column=1, sticky="w", padx=(0, 20), pady=10)

# Port
port_label = tk.Label(frame, text="Port:")
port_label.grid(row=1, column=0, sticky="e", padx=(20, 10), pady=10)
port_entry = tk.Entry(frame)
port_entry.grid(row=1, column=1, sticky="w", padx=(0, 20), pady=10)

# Username
username_label = tk.Label(frame, text="Username:")
username_label.grid(row=2, column=0, sticky="e", padx=(20, 10),
pady=10)
username_entry = tk.Entry(frame)
username_entry.grid(row=2, column=1, sticky="w", padx=(0, 20), pady=10)

# Password
password_label = tk.Label(frame, text="Password:")
password_label.grid(row=3, column=0, sticky="e", padx=(20, 10),
pady=10)
password_entry = tk.Entry(frame, show="*")
password_entry.grid(row=3, column=1, sticky="w", padx=(0, 20), pady=10)

# Public Key
public_key_label = tk.Label(frame, text="Public Key:")
public_key_label.grid(row=4, column=0, sticky="e", padx=(20, 10),
pady=10)
public_key_entry = tk.Entry(frame)
public_key_entry.grid(row=4, column=1, sticky="w", padx=(0, 20),
pady=10)

# Private Key
# Private Key
private_key_label = tk.Label(frame, text="Private Key:")
private_key_label.grid(row=5, column=0, sticky="e", padx=(20, 10),
pady=10)
private_key_entry = tk.Entry(frame)
private_key_entry.grid(row=5, column=1, sticky="w", padx=(0, 20),
pady=10)
```



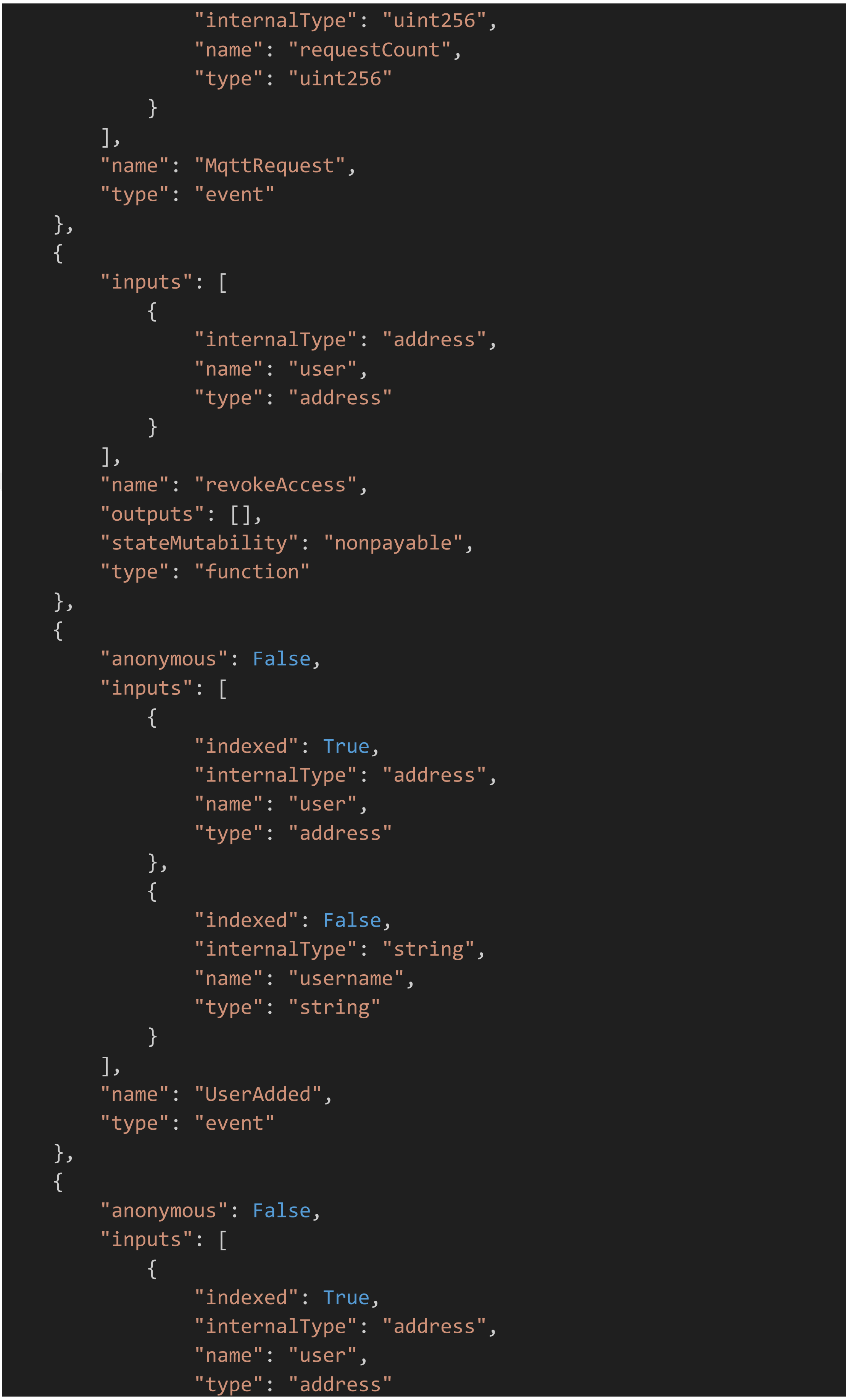
```
result_label = tk.Label(frame, text="", foreground="red")
result_label.grid(column=1, row=8, padx=(0, 5), pady=(10, 5),
sticky=tk.W)


# OTP
otp_label = tk.Label(frame, text="OTP:")
otp_label.grid(row=6, column=0, sticky="e", padx=(20, 10), pady=10)
otp_entry = tk.Entry(frame)
otp_entry.grid(row=6, column=1, sticky="w", padx=(0, 20), pady=10)


connect_button = tk.Button(frame, text="Connect", command=connect)
connect_button.grid(row=7, column=1, padx=(0, 20), pady=10)

# Exit Button
exit_button = tk.Button(window, text="Exit", command=window.quit)
exit_button.pack(side="bottom", pady=10)
exit_button.configure(width=20)  # Buton genişliğini ayarlayabilirsiniz

# GUI'nin ekran ortasında açılmasını sağla
center_window(window)

window.mainloop()
from tkinter import *
from web3 import Web3
from file_access import request_private_key, is_valid_private_key
from passlib.hash import sha512_crypt
import secrets
import pyotp
import qrcode
import os
from tkinter import messagebox
import pyodbc

class SmartContractApp:
    def __init__(self):
        self.ganache_url = "http://127.0.0.1:8545"
        self.abi= [
    {
        "inputs": [],
        "stateMutability": "nonpayable",
        "type": "constructor"
    },
    {
        "anonymous": False,
        "inputs": [
```



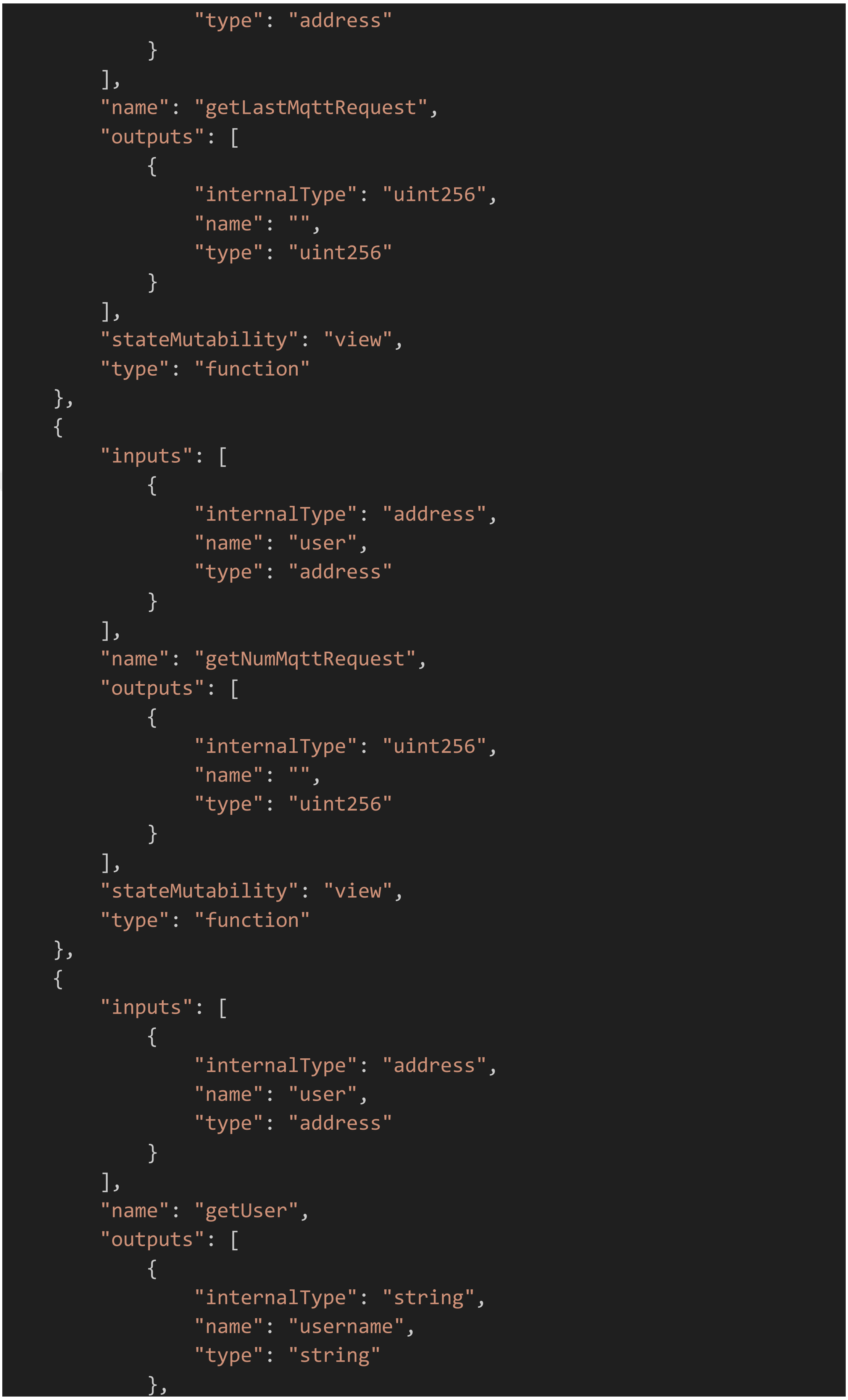
```
            {
                "indexed": True,
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "AccessGranted",
        "type": "event"
    },
    {
        "anonymous": False,
        "inputs": [
            {
                "indexed": True,
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "AccessRevoked",
        "type": "event"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            },
            {
                "internalType": "string",
                "name": "username",
                "type": "string"
            },
            {
                "internalType": "string",
                "name": "password",
                "type": "string"
            }
        ],
        "name": "addUser",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "inputs": [
```



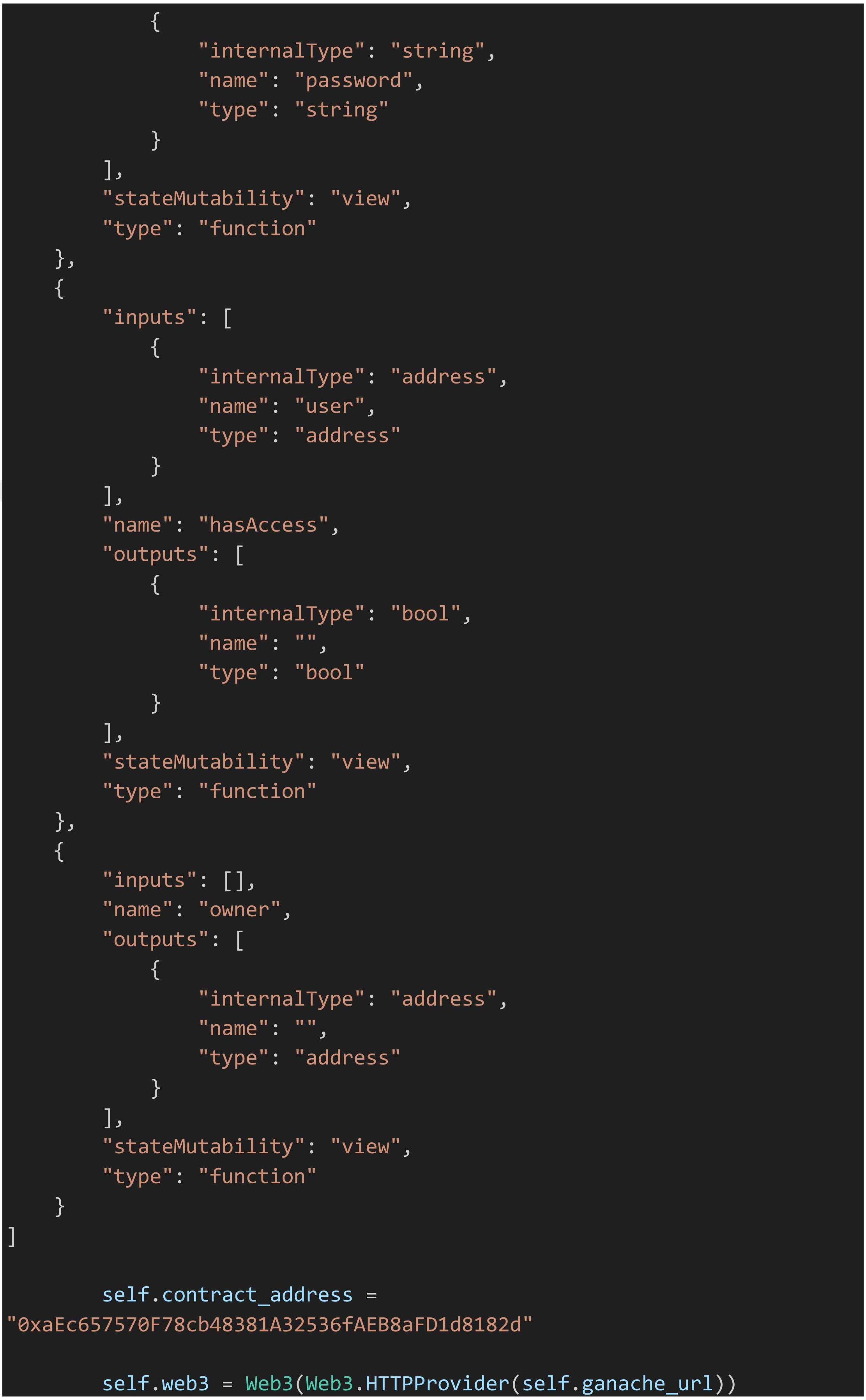
```
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "deleteUser",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "grantAccess",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "mqttRequest",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "anonymous": False,
        "inputs": [
            {
                "indexed": True,
                "internalType": "address",
                "name": "user",
                "type": "address"
            },
            {
                "indexed": False,
```



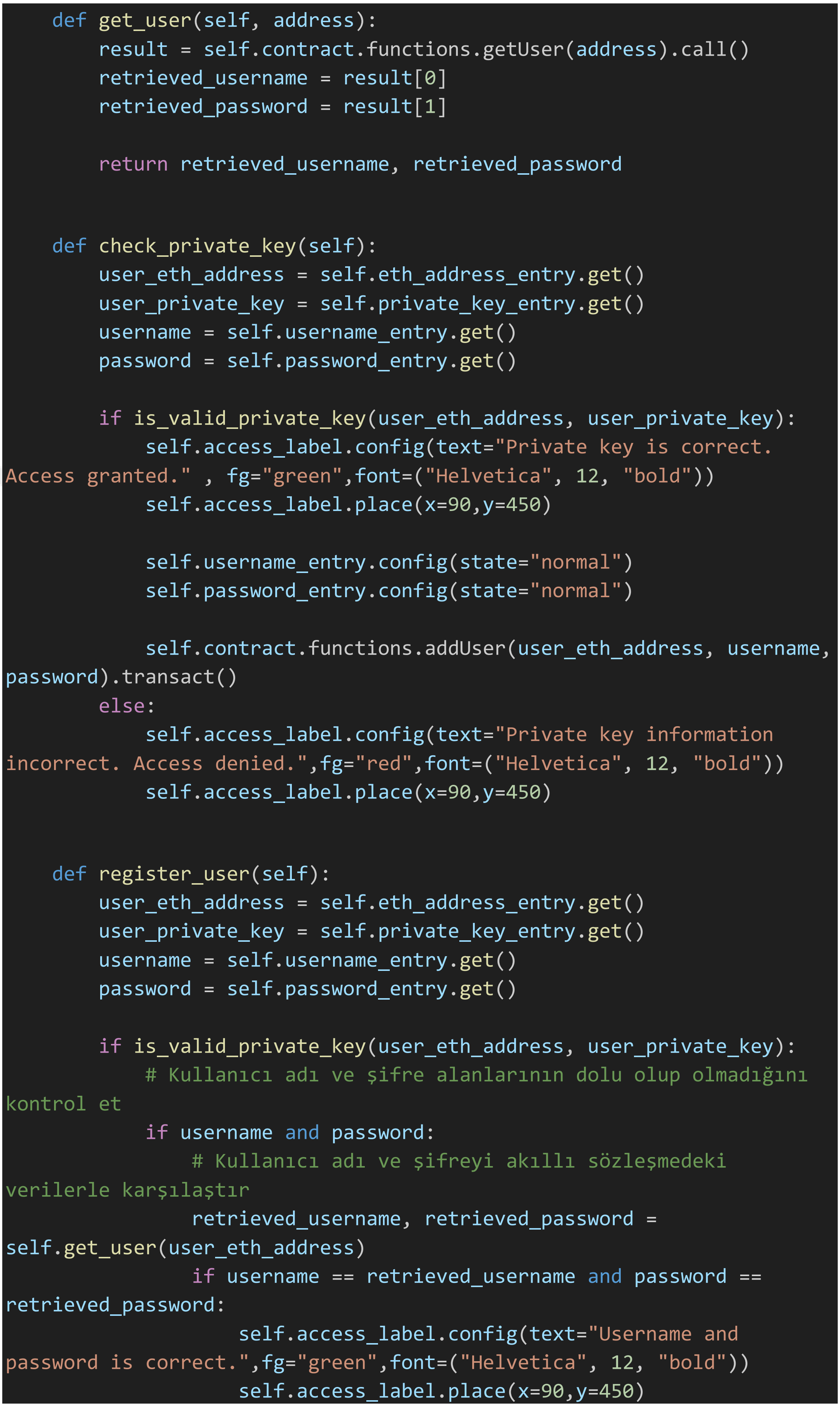
```
                "internalType": "uint256",
                "name": "requestCount",
                "type": "uint256"
            }
        ],
        "name": "MqttRequest",
        "type": "event"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "revokeAccess",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "anonymous": False,
        "inputs": [
            {
                "indexed": True,
                "internalType": "address",
                "name": "user",
                "type": "address"
            },
            {
                "indexed": False,
                "internalType": "string",
                "name": "username",
                "type": "string"
            }
        ],
        "name": "UserAdded",
        "type": "event"
    },
    {
        "anonymous": False,
        "inputs": [
            {
                "indexed": True,
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "UserDeleted",
        "type": "event"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "",
                "type": "address"
            }
        ],
        "name": "allowed",
        "outputs": [
            {
                "internalType": "bool",
                "name": "",
                "type": "bool"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "getAccessTime",
        "outputs": [
            {
                "internalType": "uint256",
                "name": "",
                "type": "uint256"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
```



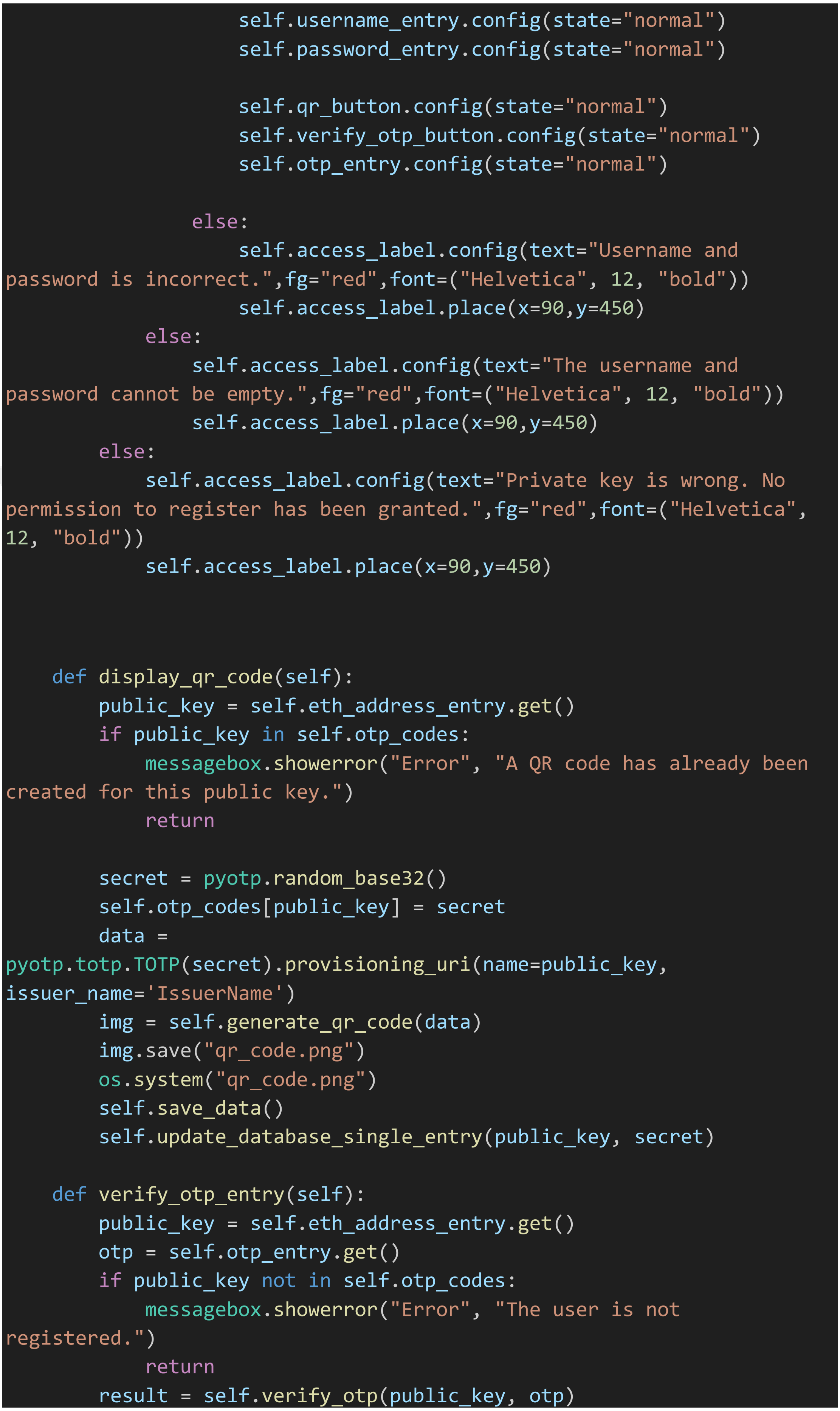
```
                "type": "address"
            }
        ],
        "name": "getLastMqttRequest",
        "outputs": [
            {
                "internalType": "uint256",
                "name": "",
                "type": "uint256"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "getNumMqttRequest",
        "outputs": [
            {
                "internalType": "uint256",
                "name": "",
                "type": "uint256"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "getUser",
        "outputs": [
            {
                "internalType": "string",
                "name": "username",
                "type": "string"
            },
```



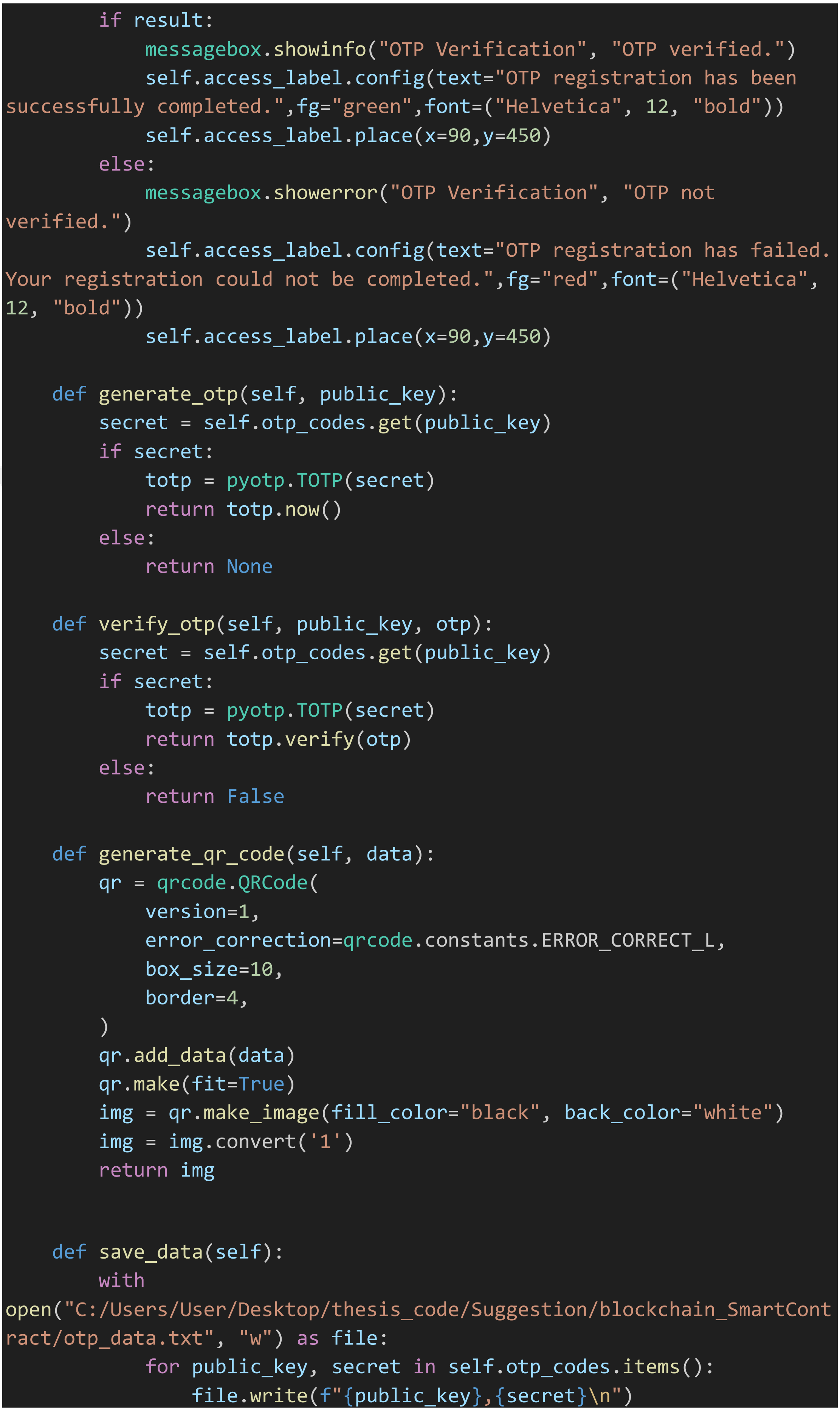
```
            {
                "internalType": "string",
                "name": "password",
                "type": "string"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "hasAccess",
        "outputs": [
            {
                "internalType": "bool",
                "name": "",
                "type": "bool"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [],
        "name": "owner",
        "outputs": [
            {
                "internalType": "address",
                "name": "",
                "type": "address"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    }
]

        self.contract_address = 
"0xaEc657570F78cb48381A32536fAEB8aFD1d8182d"

        self.web3 = Web3(Web3.HTTPProvider(self.ganache_url))
        self.contract =
self.web3.eth.contract(address=self.contract_address, abi=self.abi)

        self.otp_codes = {}

        self.window = Tk()
        self.window.title("User OTP Mechanism")
        self.window.geometry("550x500")

        self.create_widgets()

        # Load data
        self.load_data()

        self.window.mainloop()

    def create_widgets(self):
        self.eth_address_label = Label(self.window, text="Adresses:")
        self.eth_address_label.place(x=60, y=60)

        self.eth_address_entry = Entry(self.window)
        self.eth_address_entry.place(x=180, y=60)

        self.access_button = Button(self.window, text="Public Key
Control", command=self.check_access)
        self.access_button.place(x=320, y=55)

        self.access_label = Label(self.window, text="")
        self.access_label.place(x=60, y=90)

        self.private_key_label = Label(self.window, text="Private
Key:")
        self.private_key_label.place(x=60, y=120)

        self.private_key_entry = Entry(self.window, state="disabled")
        self.private_key_entry.place(x=180, y=120)

        self.check_button = Button(self.window, text="Private Key
Control", command=self.check_private_key, state="disabled")
        self.check_button.place(x=320, y=115)

        self.username_label = Label(self.window, text="Username:",
state="disabled")
        self.username_label.place(x=60, y=200)

        self.username_entry = Entry(self.window, state="disabled")
        self.username_entry.place(x=180, y=200)
        self.password_label = Label(self.window, text="Password:", 
state="disabled")
        self.password_label.place(x=60, y=230)

        self.password_entry = Entry(self.window, state="disabled", 
show="*")
        self.password_entry.place(x=180, y=230)

        self.private_key_entry.bind("<KeyRelease>", 
self.enable_check_button)

        self.register_button = Button(self.window, text="Username And 
Password Control", command=self.register_user, state="disabled")
        self.register_button.place(x=320, y=206)

        self.qr_button = Button(self.window, text="Generate QR Code", 
command=self.display_qr_code, state="disabled")
        self.qr_button.place(x=60, y=310)

        self.otp_entry = Entry(self.window, state="disabled", show="*")
        self.otp_entry.place(x=180, y=310)

        self.verify_otp_button = Button(self.window, text="User OTP 
Verify", command=self.verify_otp_entry, state="disabled")
        self.verify_otp_button.place(x=320, y=305)


        self.private_key_entry.bind("<KeyRelease>", 
self.enable_register_button)
        self.username_entry.bind("<KeyRelease>", 
self.enable_register_button)
        self.password_entry.bind("<KeyRelease>", 
self.enable_register_button)


    def update_database_single_entry(self, publickey, otpkey):
        # Veritabanı bağlantısı kurma
        server = "DESKTOP-UO8R7VH\SQL2022"
        database = "MQTT"
        conn = pyodbc.connect(f'Driver={{SQL 
Server}};Server={server};Database={database};Trusted_Connection=yes;')
        cursor = conn.cursor()

        # Veritabanına ekleme
        cursor.execute("INSERT INTO OTPInfo (PublicKey, OTPKey) VALUES 
(?, ?)", (publickey, otpkey))

        # Değişiklikleri onaylama ve bağlantıyı kapatma
        conn.commit()
        conn.close()

    def delete_database_single_entry(self, publickey):
        # Veritabanı bağlantısı kurma
        server = "DESKTOP-UO8R7VH\SQL2022"
        database = "MQTT"
        conn = pyodbc.connect(f'Driver={{SQL
Server}};Server={server};Database={database};Trusted_Connection=yes;')
        cursor = conn.cursor()

        # Veritabanından silme
        cursor.execute("DELETE FROM OTPInfo WHERE PublicKey = ?",
(publickey,))

        # Değişiklikleri onaylama ve bağlantıyı kapatma
        conn.commit()
        conn.close()


    def enable_check_button(self, event):
        if self.private_key_entry.get():
            self.check_button.config(state="normal")
            self.username_label.config(state="normal")
            self.password_label.config(state="normal")


    def enable_register_button(self, event):
        if self.private_key_entry.get() and self.username_entry.get()
and self.password_entry.get():
            self.register_button.config(state="normal")


    def check_access(self):
        user_eth_address = self.eth_address_entry.get()

        if self.contract.functions.hasAccess(user_eth_address).call():
            self.access_label.config(text="The user is registered in
the system. Access granted.", fg="green",font=("Helvetica", 12,
"bold"))
            self.access_label.place(x=90,y=450)
            self.private_key_entry.config(state="normal")
            self.check_button.config(state="normal")
            self.username_entry.config(state="disabled")
            self.password_entry.config(state="disabled")
        else:
            self.access_label.config(text="The user is not registered
in the system. Access denied.",fg="red",font=("Helvetica", 12, "bold"))
            self.access_label.place(x=90,y=450)
```



```
    def get_user(self, address):
        result = self.contract.functions.getUser(address).call()
        retrieved_username = result[0]
        retrieved_password = result[1]

        return retrieved_username, retrieved_password


    def check_private_key(self):
        user_eth_address = self.eth_address_entry.get()
        user_private_key = self.private_key_entry.get()
        username = self.username_entry.get()
        password = self.password_entry.get()

        if is_valid_private_key(user_eth_address, user_private_key):
            self.access_label.config(text="Private key is correct.
Access granted." , fg="green",font=("Helvetica", 12, "bold"))
            self.access_label.place(x=90,y=450)

            self.username_entry.config(state="normal")
            self.password_entry.config(state="normal")

            self.contract.functions.addUser(user_eth_address, username,
password).transact()
        else:
            self.access_label.config(text="Private key information
incorrect. Access denied.",fg="red",font=("Helvetica", 12, "bold"))
            self.access_label.place(x=90,y=450)


    def register_user(self):
        user_eth_address = self.eth_address_entry.get()
        user_private_key = self.private_key_entry.get()
        username = self.username_entry.get()
        password = self.password_entry.get()

        if is_valid_private_key(user_eth_address, user_private_key):
            # Kullanıcı adı ve şifre alanlarının dolu olup olmadığını
kontrol et
            if username and password:
                # Kullanıcı adı ve şifreyi akıllı sözleşmedeki
verilerle karşılaştır
                retrieved_username, retrieved_password =
self.get_user(user_eth_address)
                if username == retrieved_username and password ==
retrieved_password:
                    self.access_label.config(text="Username and
password is correct.",fg="green",font=("Helvetica", 12, "bold"))
                    self.access_label.place(x=90,y=450)
```

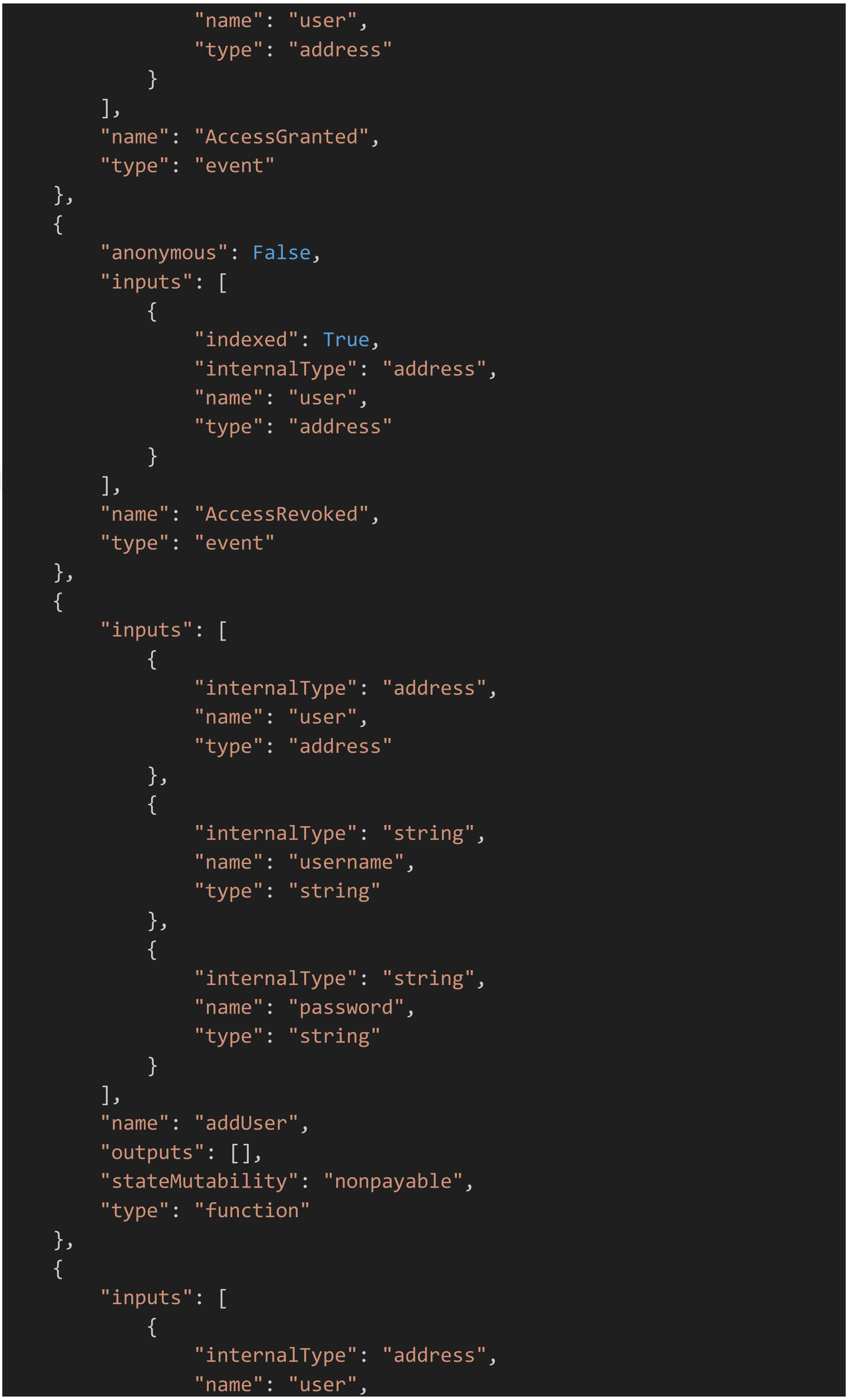


```
                    self.username_entry.config(state="normal")
                    self.password_entry.config(state="normal")

                    self.qr_button.config(state="normal")
                    self.verify_otp_button.config(state="normal")
                    self.otp_entry.config(state="normal")

                else:
                    self.access_label.config(text="Username and 
password is incorrect.",fg="red",font=("Helvetica", 12, "bold"))
                    self.access_label.place(x=90,y=450)
            else:
                self.access_label.config(text="The username and 
password cannot be empty.",fg="red",font=("Helvetica", 12, "bold"))
                self.access_label.place(x=90,y=450)
        else:
            self.access_label.config(text="Private key is wrong. No 
permission to register has been granted.",fg="red",font=("Helvetica", 
12, "bold"))
            self.access_label.place(x=90,y=450)



    def display_qr_code(self):
        public_key = self.eth_address_entry.get()
        if public_key in self.otp_codes:
            messagebox.showerror("Error", "A QR code has already been 
created for this public key.")
            return

        secret = pyotp.random_base32()
        self.otp_codes[public_key] = secret
        data = 
pyotp.totp.TOTP(secret).provisioning_uri(name=public_key, 
issuer_name='IssuerName')
        img = self.generate_qr_code(data)
        img.save("qr_code.png")
        os.system("qr_code.png")
        self.save_data()
        self.update_database_single_entry(public_key, secret)

    def verify_otp_entry(self):
        public_key = self.eth_address_entry.get()
        otp = self.otp_entry.get()
        if public_key not in self.otp_codes:
            messagebox.showerror("Error", "The user is not 
registered.")
            return
        result = self.verify_otp(public_key, otp)
```

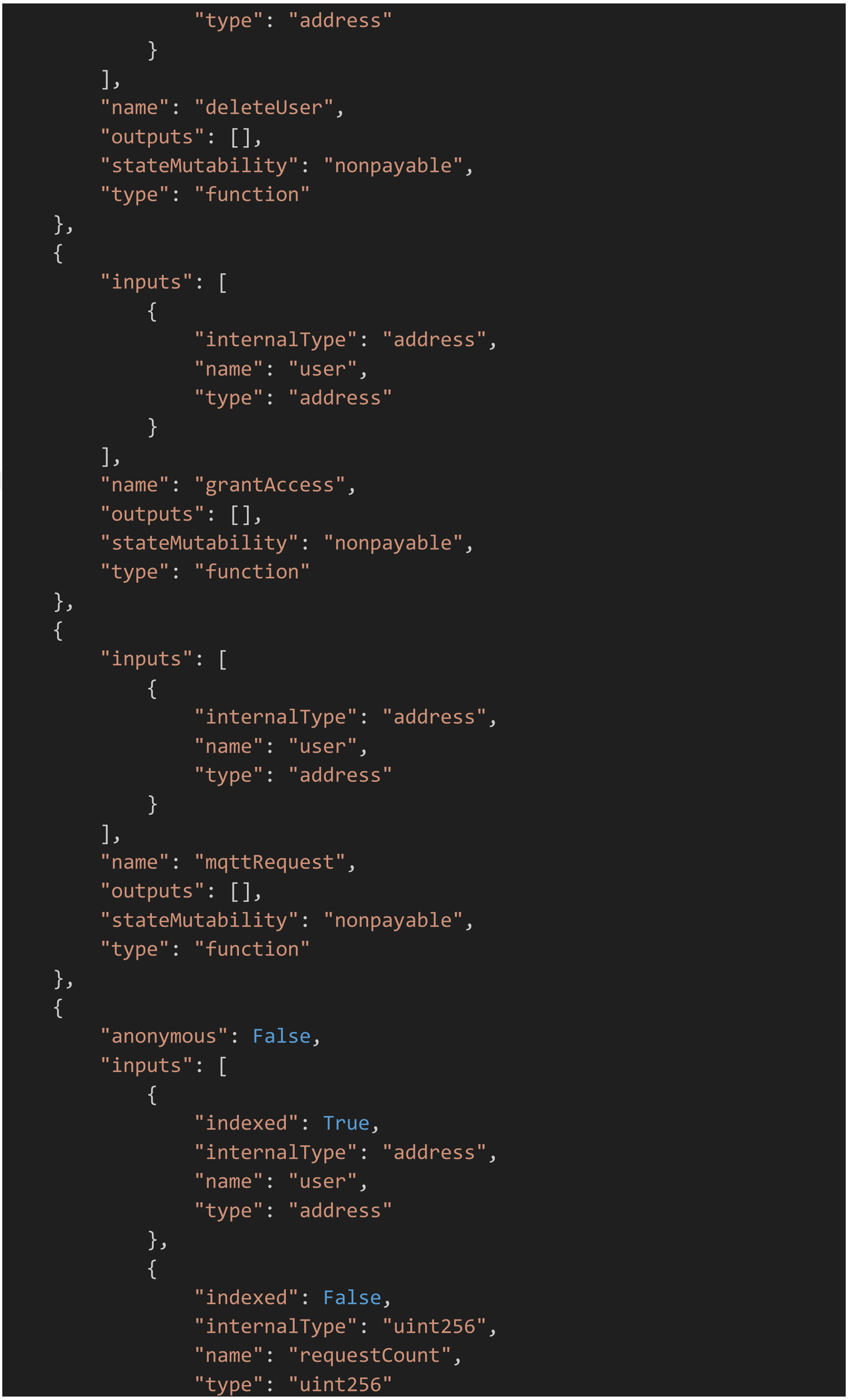


```
        if result:
            messagebox.showinfo("OTP Verification", "OTP verified.")
            self.access_label.config(text="OTP registration has been 
successfully completed.",fg="green",font=("Helvetica", 12, "bold"))
            self.access_label.place(x=90,y=450)
        else:
            messagebox.showerror("OTP Verification", "OTP not 
verified.")
            self.access_label.config(text="OTP registration has failed. 
Your registration could not be completed.",fg="red",font=("Helvetica", 
12, "bold"))
            self.access_label.place(x=90,y=450)

    def generate_otp(self, public_key):
        secret = self.otp_codes.get(public_key)
        if secret:
            totp = pyotp.TOTP(secret)
            return totp.now()
        else:
            return None

    def verify_otp(self, public_key, otp):
        secret = self.otp_codes.get(public_key)
        if secret:
            totp = pyotp.TOTP(secret)
            return totp.verify(otp)
        else:
            return False

    def generate_qr_code(self, data):
        qr = qrcode.QRCode(
            version=1,
            error_correction=qrcode.constants.ERROR_CORRECT_L,
            box_size=10,
            border=4,
        )
        qr.add_data(data)
        qr.make(fit=True)
        img = qr.make_image(fill_color="black", back_color="white")
        img = img.convert('1')
        return img


    def save_data(self):
        with 
open("C:/Users/User/Desktop/thesis_code/Suggestion/blockchain_SmartCont
ract/otp_data.txt", "w") as file:
            for public_key, secret in self.otp_codes.items():
                file.write(f"{public_key},{secret}\n")
    def load_data(self):
        try:
            with
open("C:/Users/User/Desktop/thesis_code/Suggestion/blockchain_SmartCont
ract/otp_data.txt", "r") as file:
                for line in file:
                    public_key, secret = line.strip().split(",")
                    self.otp_codes[public_key] = secret
        except FileNotFoundError:
            pass

if __name__ == '__main__':
    app = SmartContractApp()
    app.run()
```

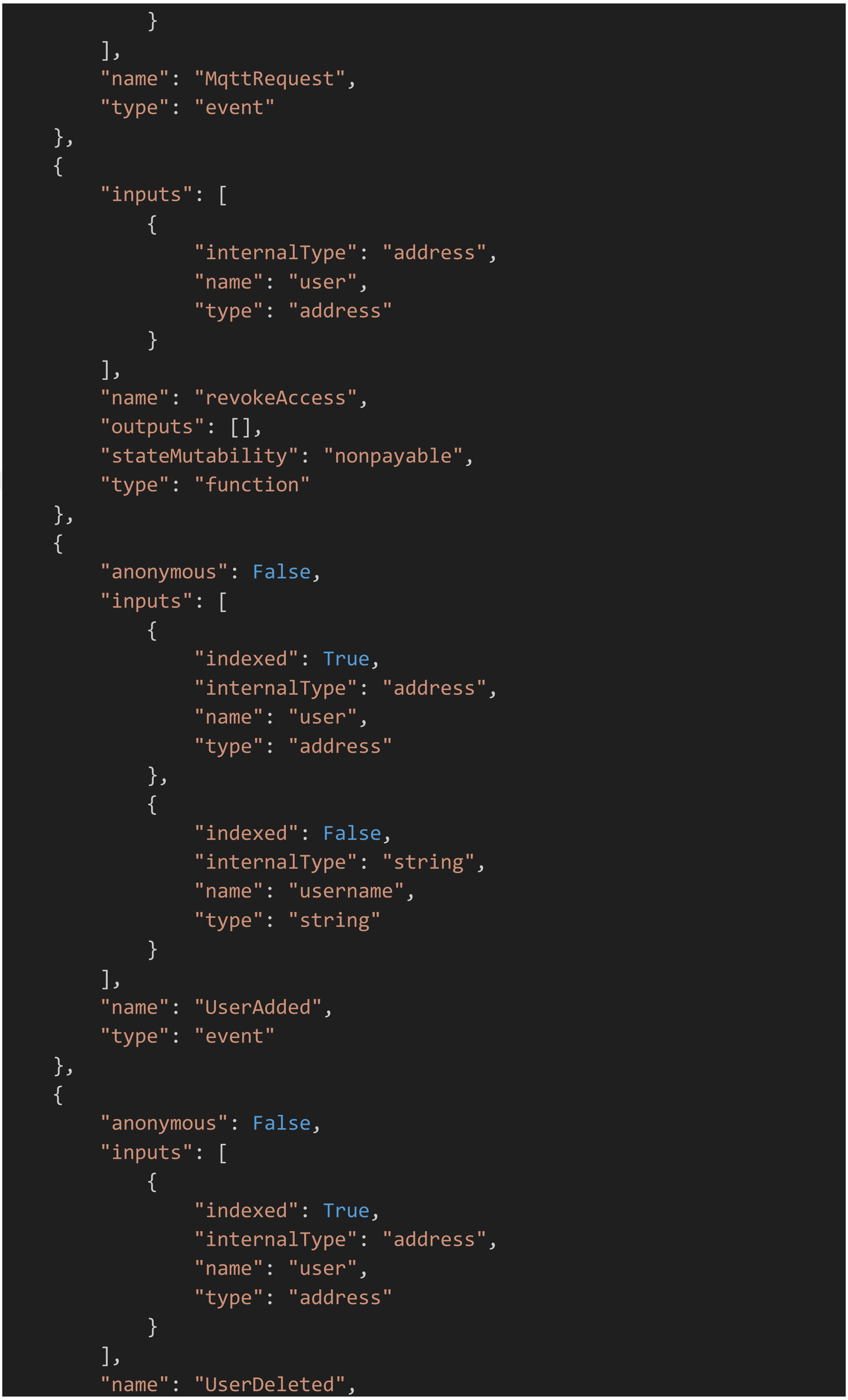
```
import tkinter as tk
from tkinter import messagebox
from web3 import Web3
from web3.auto import w3
from passlib.hash import sha512_crypt
from datetime import datetime as dt
import pyodbc
import datetime
import pytz


class UserAccessGUI:
    def __init__(self):
        self.root = tk.Tk()
        self.root.geometry('800x320')
        self.root.title('User Access Permission')

        self.web3 = Web3(Web3.HTTPProvider('http://localhost:8545'))

        self.ABI=[
    {
        "inputs": [],
        "stateMutability": "nonpayable",
        "type": "constructor"
    },
    {
        "anonymous": False,
        "inputs": [
            {
                "indexed": True,
                "internalType": "address",
```



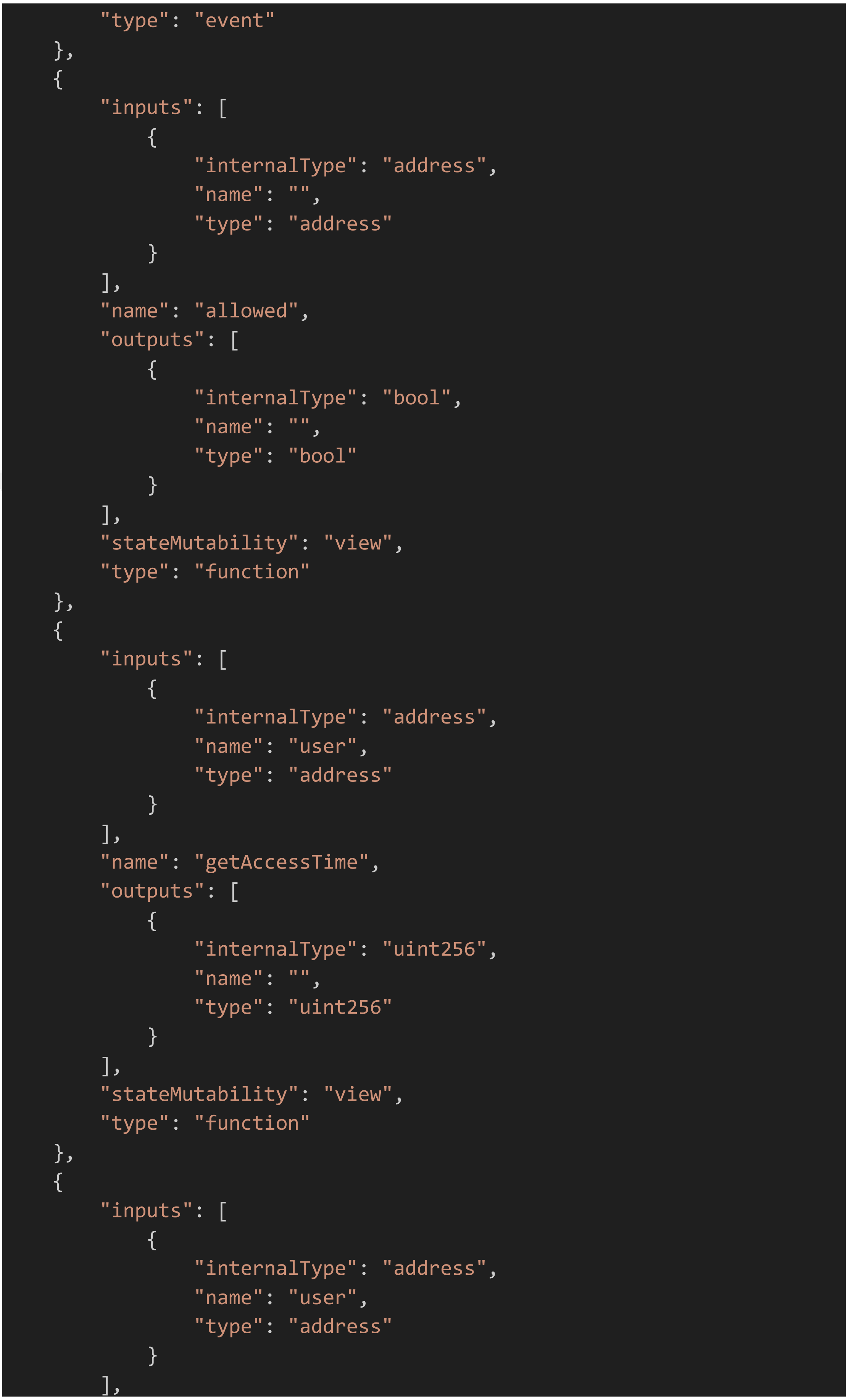
```
                "name": "user",
                "type": "address"
            }
        ],
        "name": "AccessGranted",
        "type": "event"
    },
    {
        "anonymous": False,
        "inputs": [
            {
                "indexed": True,
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "AccessRevoked",
        "type": "event"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            },
            {
                "internalType": "string",
                "name": "username",
                "type": "string"
            },
            {
                "internalType": "string",
                "name": "password",
                "type": "string"
            }
        ],
        "name": "addUser",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
```



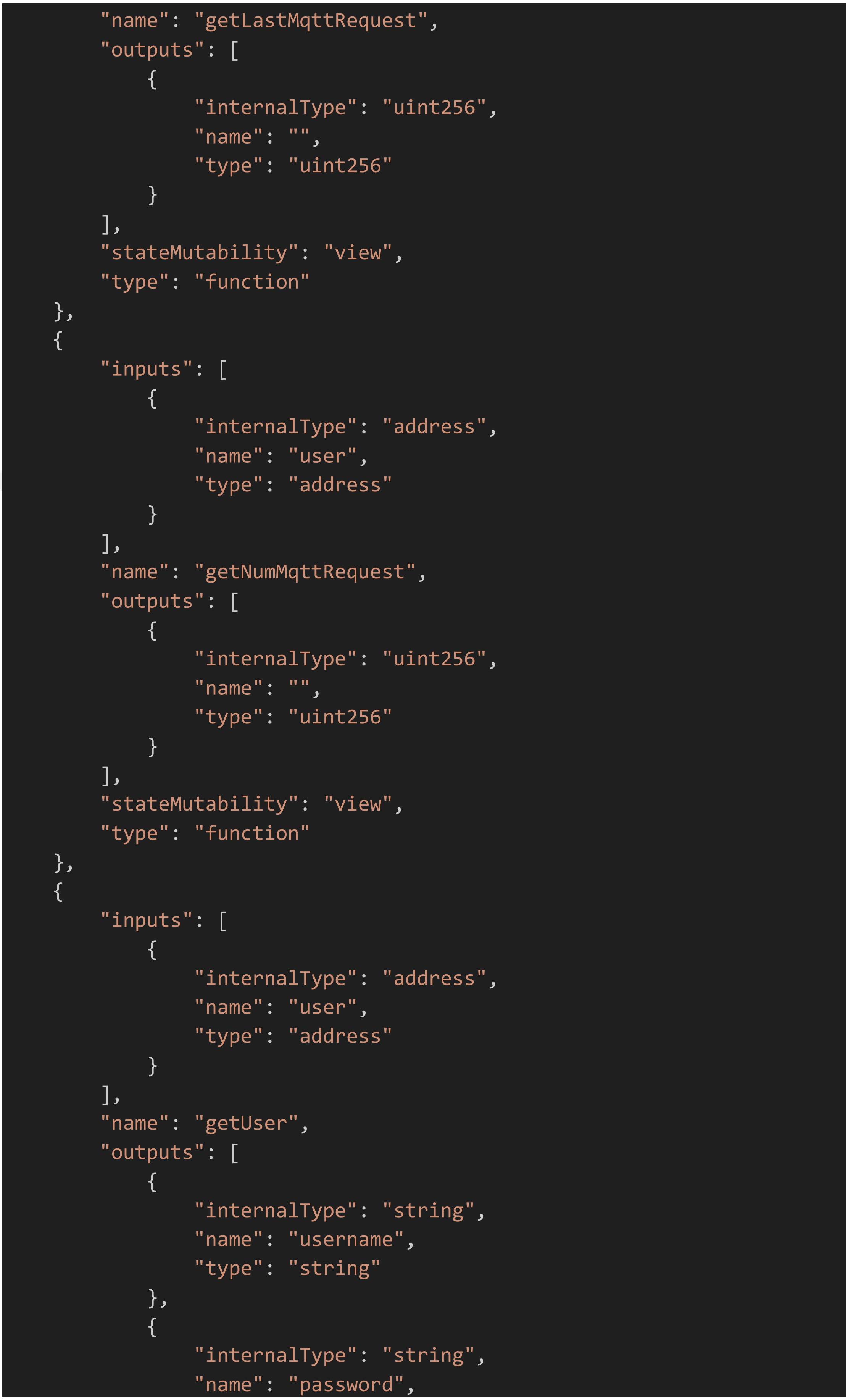
```
                "type": "address"
            }
        ],
        "name": "deleteUser",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "grantAccess",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "mqttRequest",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "anonymous": False,
        "inputs": [
            {
                "indexed": True,
                "internalType": "address",
                "name": "user",
                "type": "address"
            },
            {
                "indexed": False,
                "internalType": "uint256",
                "name": "requestCount",
                "type": "uint256"
```



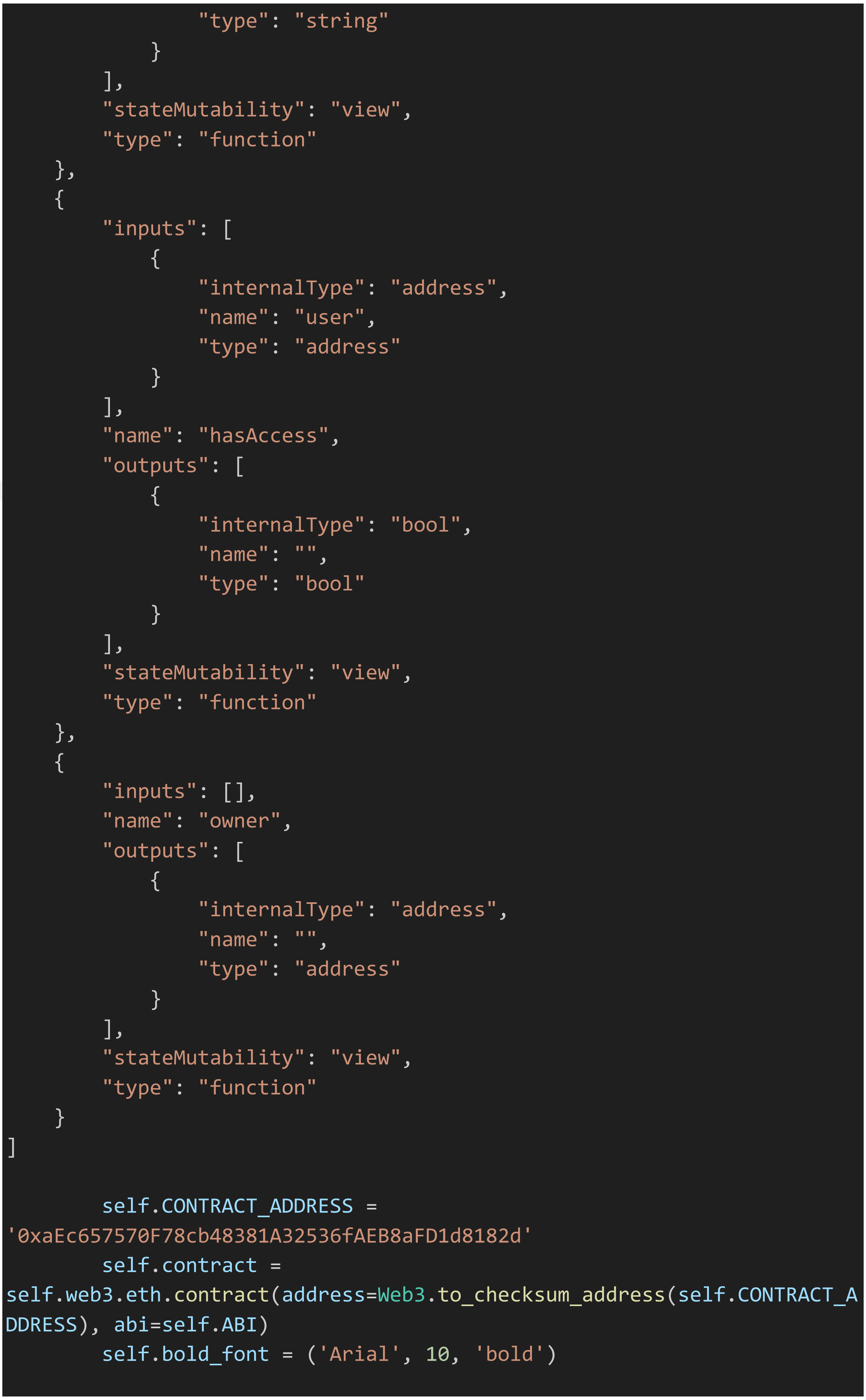
```
            }
        ],
        "name": "MqttRequest",
        "type": "event"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "revokeAccess",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "anonymous": False,
        "inputs": [
            {
                "indexed": True,
                "internalType": "address",
                "name": "user",
                "type": "address"
            },
            {
                "indexed": False,
                "internalType": "string",
                "name": "username",
                "type": "string"
            }
        ],
        "name": "UserAdded",
        "type": "event"
    },
    {
        "anonymous": False,
        "inputs": [
            {
                "indexed": True,
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "UserDeleted",
```



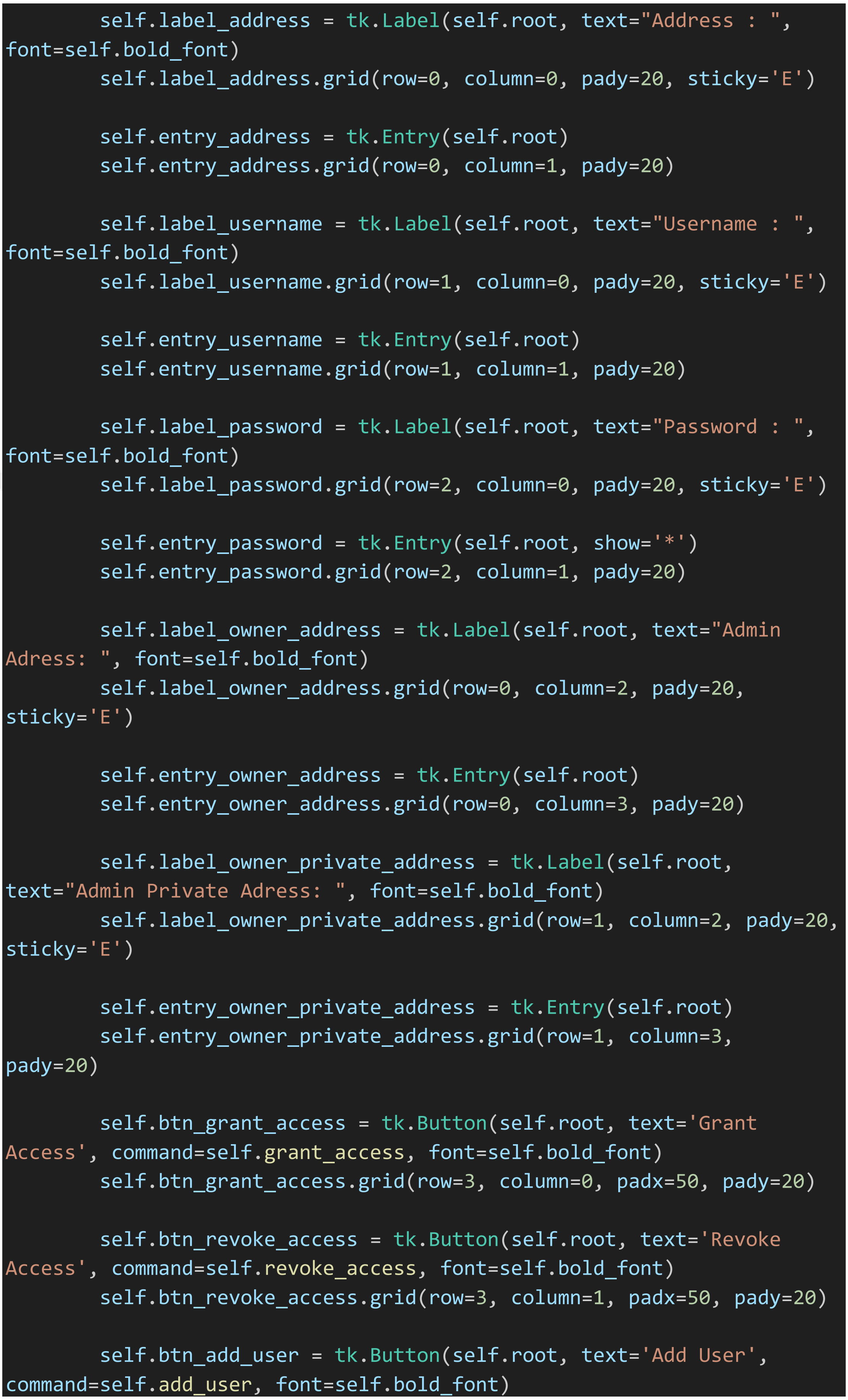
```
        "type": "event"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "",
                "type": "address"
            }
        ],
        "name": "allowed",
        "outputs": [
            {
                "internalType": "bool",
                "name": "",
                "type": "bool"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "getAccessTime",
        "outputs": [
            {
                "internalType": "uint256",
                "name": "",
                "type": "uint256"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
```



```
        "name": "getLastMqttRequest",
        "outputs": [
            {
                "internalType": "uint256",
                "name": "",
                "type": "uint256"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "getNumMqttRequest",
        "outputs": [
            {
                "internalType": "uint256",
                "name": "",
                "type": "uint256"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "getUser",
        "outputs": [
            {
                "internalType": "string",
                "name": "username",
                "type": "string"
            },
            {
                "internalType": "string",
                "name": "password",
```

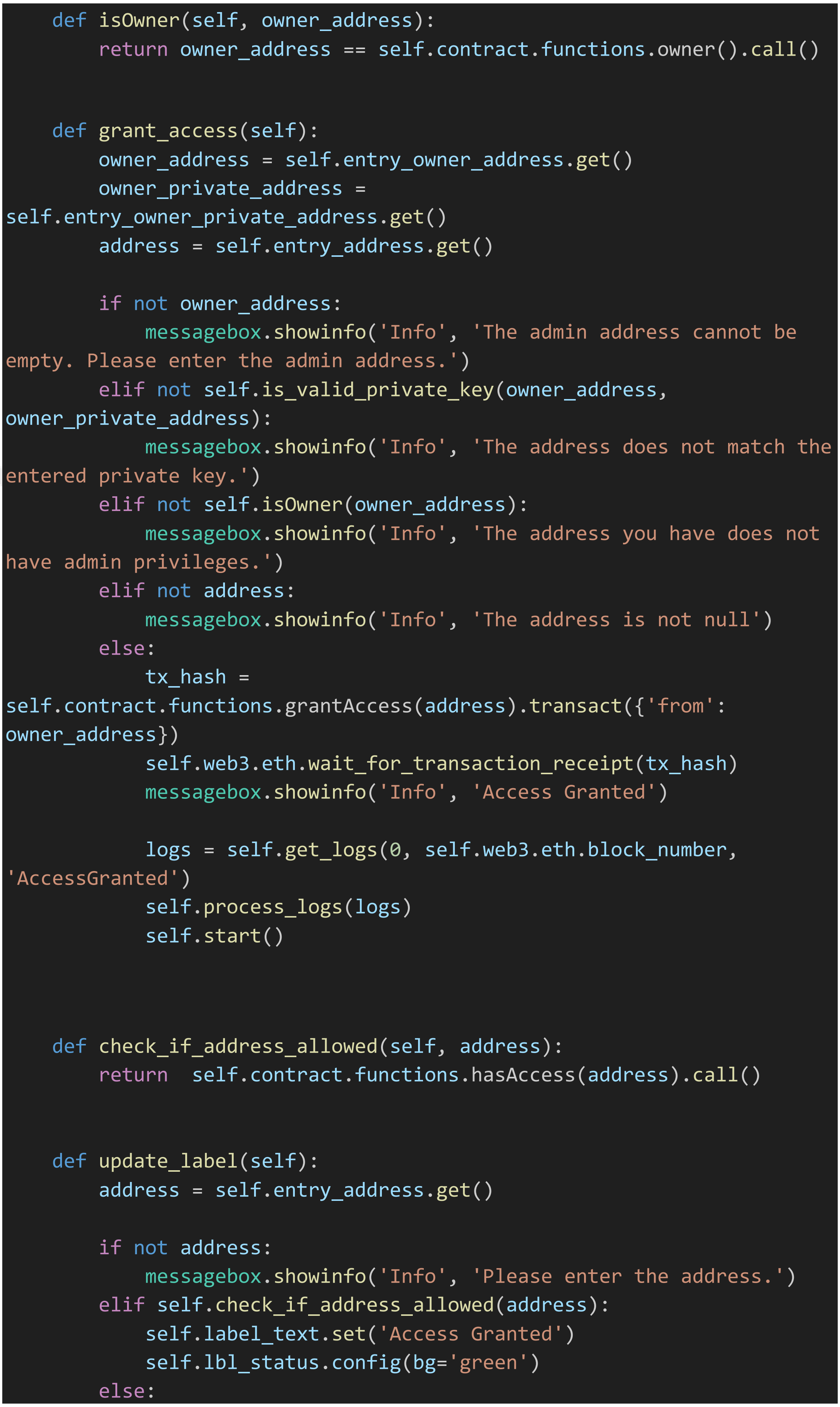


```
                "type": "string"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "user",
                "type": "address"
            }
        ],
        "name": "hasAccess",
        "outputs": [
            {
                "internalType": "bool",
                "name": "",
                "type": "bool"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [],
        "name": "owner",
        "outputs": [
            {
                "internalType": "address",
                "name": "",
                "type": "address"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    }
]

        self.CONTRACT_ADDRESS = 
'0xaEc657570F78cb48381A32536fAEB8aFD1d8182d'
        self.contract = 
self.web3.eth.contract(address=Web3.to_checksum_address(self.CONTRACT_A
DDRESS), abi=self.ABI)
        self.bold_font = ('Arial', 10, 'bold')
```

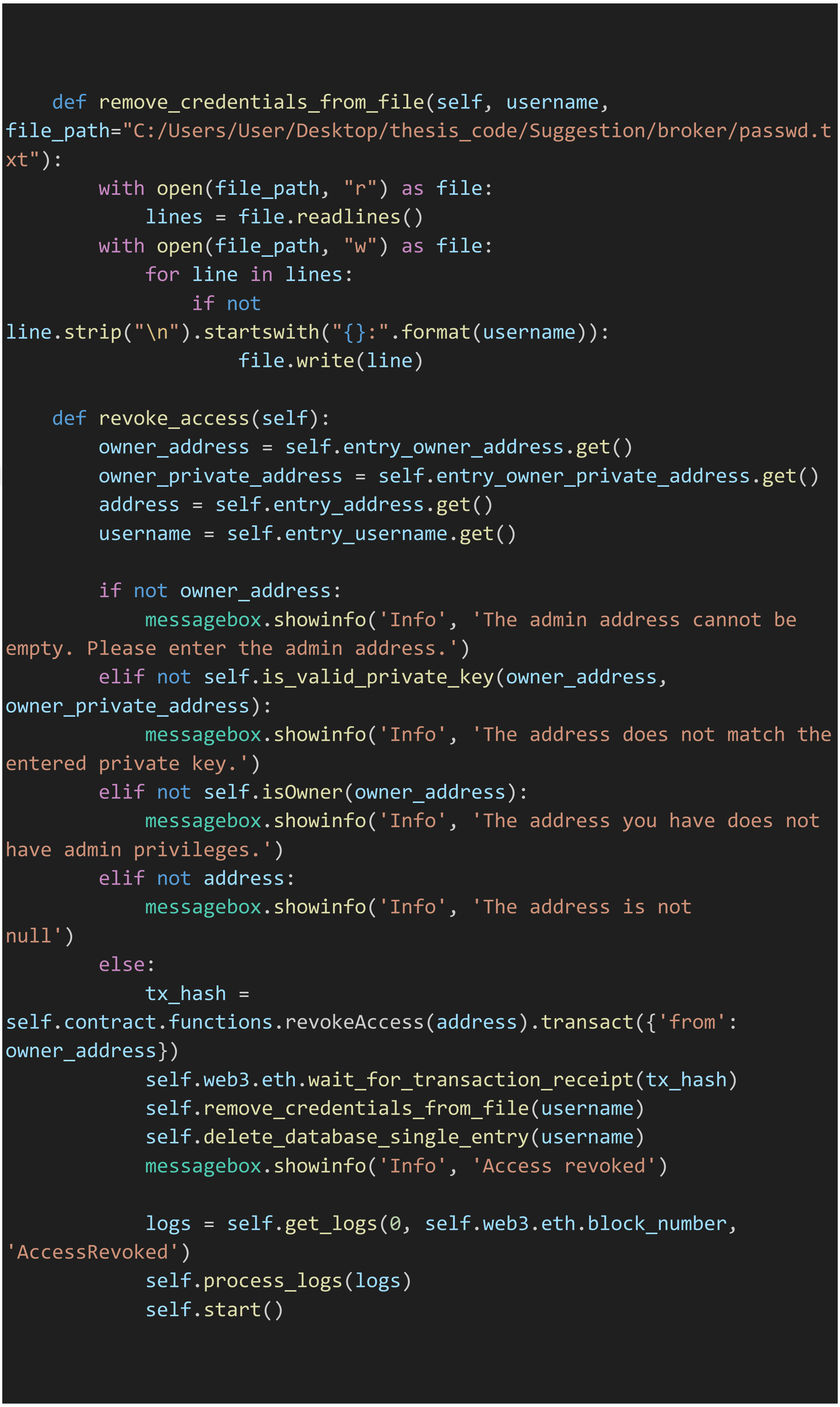


```
        self.label_address = tk.Label(self.root, text="Address : ",
font=self.bold_font)
        self.label_address.grid(row=0, column=0, pady=20, sticky='E')

        self.entry_address = tk.Entry(self.root)
        self.entry_address.grid(row=0, column=1, pady=20)

        self.label_username = tk.Label(self.root, text="Username : ",
font=self.bold_font)
        self.label_username.grid(row=1, column=0, pady=20, sticky='E')

        self.entry_username = tk.Entry(self.root)
        self.entry_username.grid(row=1, column=1, pady=20)

        self.label_password = tk.Label(self.root, text="Password : ",
font=self.bold_font)
        self.label_password.grid(row=2, column=0, pady=20, sticky='E')

        self.entry_password = tk.Entry(self.root, show='*')
        self.entry_password.grid(row=2, column=1, pady=20)

        self.label_owner_address = tk.Label(self.root, text="Admin
Adress: ", font=self.bold_font)
        self.label_owner_address.grid(row=0, column=2, pady=20,
sticky='E')

        self.entry_owner_address = tk.Entry(self.root)
        self.entry_owner_address.grid(row=0, column=3, pady=20)

        self.label_owner_private_address = tk.Label(self.root,
text="Admin Private Adress: ", font=self.bold_font)
        self.label_owner_private_address.grid(row=1, column=2, pady=20,
sticky='E')

        self.entry_owner_private_address = tk.Entry(self.root)
        self.entry_owner_private_address.grid(row=1, column=3,
pady=20)

        self.btn_grant_access = tk.Button(self.root, text='Grant
Access', command=self.grant_access, font=self.bold_font)
        self.btn_grant_access.grid(row=3, column=0, padx=50, pady=20)

        self.btn_revoke_access = tk.Button(self.root, text='Revoke
Access', command=self.revoke_access, font=self.bold_font)
        self.btn_revoke_access.grid(row=3, column=1, padx=50, pady=20)

        self.btn_add_user = tk.Button(self.root, text='Add User',
command=self.add_user, font=self.bold_font)
```

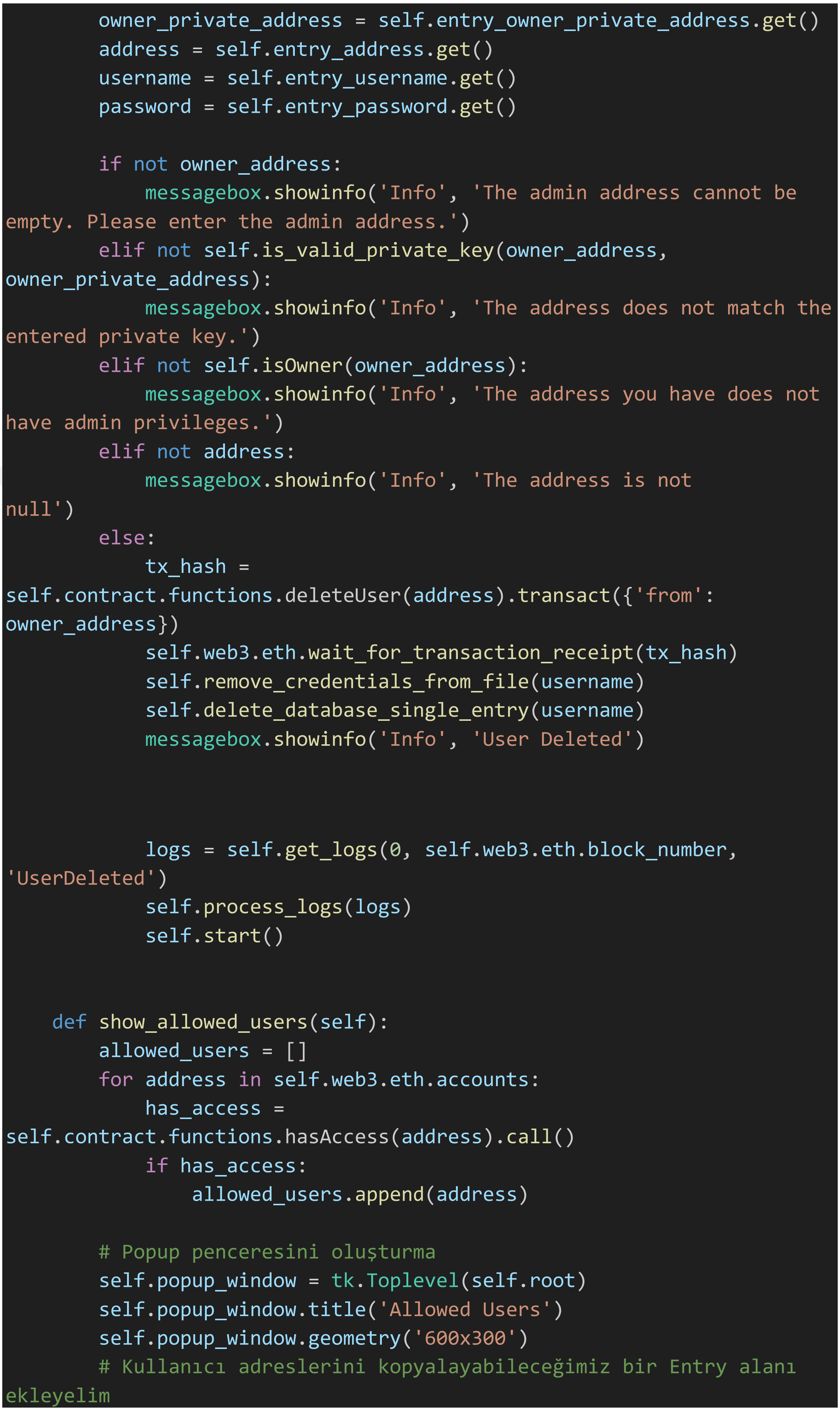
```
        self.btn_add_user.grid(row=3, column=2, padx=50, pady=20)

        self.btn_delete_user = tk.Button(self.root, text='Delete User',
command=self.delete_user, font=self.bold_font)
        self.btn_delete_user.grid(row=3, column=3, padx=50, pady=20)

        self.label_text = tk.StringVar()
        self.label_text.set('do you have user access?')
        self.lbl_status = tk.Label(self.root,
textvariable=self.label_text, font=self.bold_font)
        self.lbl_status.grid(row=8, column=1, padx=50, pady=20,
sticky='nsew')

        self.btn_check_access = tk.Button(self.root, text='Check
Access', command=self.update_label, font=self.bold_font)
        self.btn_check_access.grid(row=8, column=0, padx=50, pady=20)

        self.btn_allowed_users = tk.Button(self.root, text='Allowed
Users', command=self.show_allowed_users, font=self.bold_font)
        self.btn_allowed_users.grid(row=8, column=3, padx=50, pady=20)

        self.popup_window = None




    def check_address(self, func):
        def wrapper():
            address = self.entry_address.get()
            if not address:
                messagebox.showinfo('Info', 'Please enter an address
first')
            elif not self.contract.functions.hasAccess(address).call():
                messagebox.showinfo('Info', 'User does not have
access')
            else:
                func()

        return wrapper


    def is_valid_private_key(self, eth_address, private_key):
        from web3 import Web3
        web3 = Web3(Web3.HTTPProvider('http://127.0.0.1:8545'))
        account = web3.eth.account.from_key(private_key)
        return account.address.lower() == eth_address.lower()
```



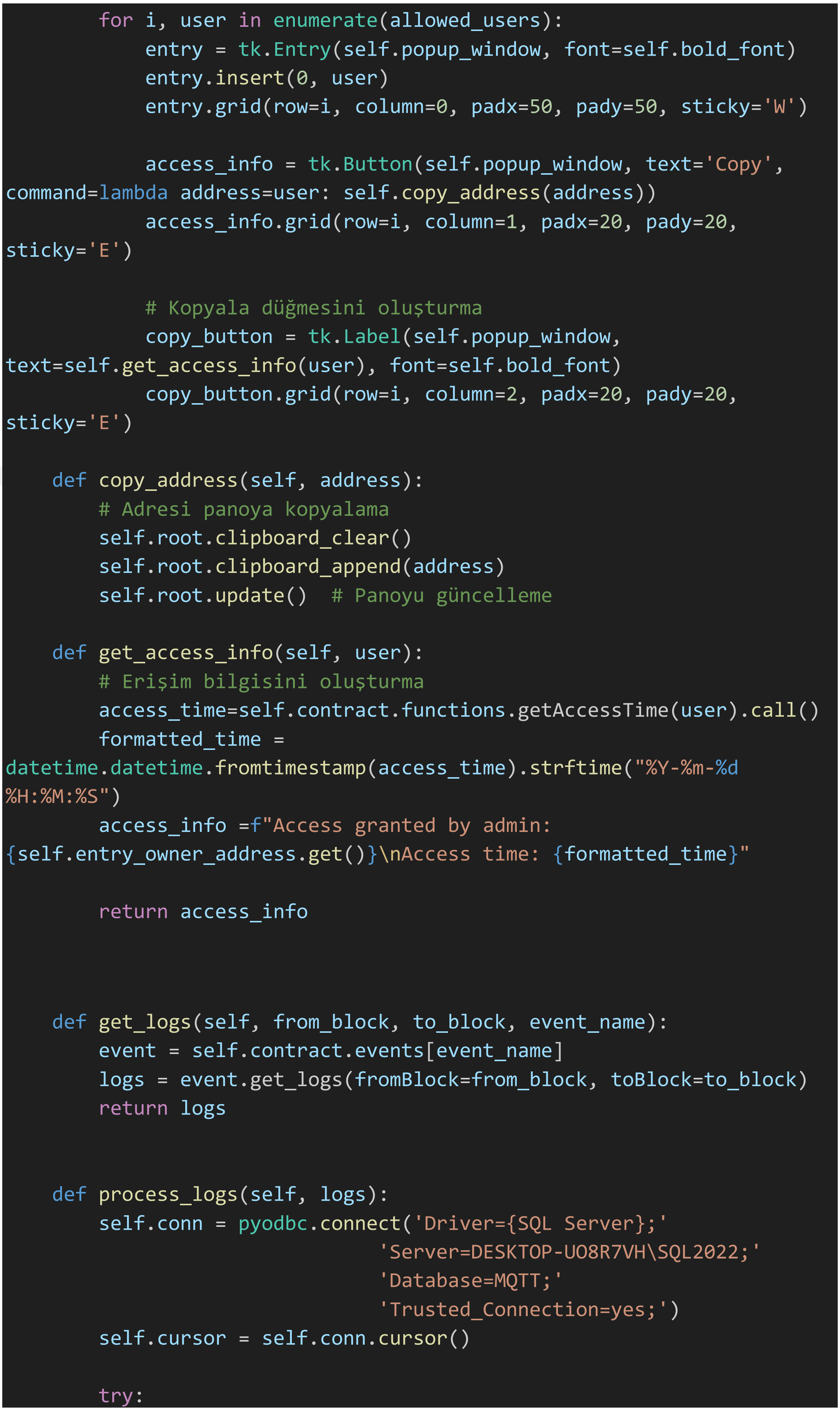
```
    def isOwner(self, owner_address):
        return owner_address == self.contract.functions.owner().call()


    def grant_access(self):
        owner_address = self.entry_owner_address.get()
        owner_private_address =
self.entry_owner_private_address.get()
        address = self.entry_address.get()

        if not owner_address:
            messagebox.showinfo('Info', 'The admin address cannot be
empty. Please enter the admin address.')
        elif not self.is_valid_private_key(owner_address,
owner_private_address):
            messagebox.showinfo('Info', 'The address does not match the
entered private key.')
        elif not self.isOwner(owner_address):
            messagebox.showinfo('Info', 'The address you have does not
have admin privileges.')
        elif not address:
            messagebox.showinfo('Info', 'The address is not null')
        else:
            tx_hash =
self.contract.functions.grantAccess(address).transact({'from':
owner_address})
            self.web3.eth.wait_for_transaction_receipt(tx_hash)
            messagebox.showinfo('Info', 'Access Granted')

            logs = self.get_logs(0, self.web3.eth.block_number,
'AccessGranted')
            self.process_logs(logs)
            self.start()



    def check_if_address_allowed(self, address):
        return  self.contract.functions.hasAccess(address).call()


    def update_label(self):
        address = self.entry_address.get()

        if not address:
            messagebox.showinfo('Info', 'Please enter the address.')
        elif self.check_if_address_allowed(address):
            self.label_text.set('Access Granted')
            self.lbl_status.config(bg='green')
        else:
            self.label_text.set('Access Denied')
            self.lbl_status.config(bg='red')




    def add_user(self):
        owner_address = self.entry_owner_address.get()
        owner_private_address = self.entry_owner_private_address.get()
        address = self.entry_address.get()
        username = self.entry_username.get()
        password = self.entry_password.get()

        # Şifreyi şifrele
        hashed_password =
sha512_crypt.using(rounds=5000).hash(password)
        # Kullanıcı adı ve şifreyi dosyaya kaydet


        if not owner_address:
            messagebox.showinfo('Info', 'The admin address cannot be
empty. Please enter the admin address.')
        elif not self.is_valid_private_key(owner_address,
owner_private_address):
            messagebox.showinfo('Info', 'The address does not match the
entered private key.')
        elif not self.isOwner(owner_address):
            messagebox.showinfo('Info', 'The address you have does not
have admin privileges.')
            return
        elif not address:
            messagebox.showinfo('Info', 'The address is not
null')
        else:
            tx_hash =
self.contract.functions.addUser(address,username,password).transact({'f
rom': owner_address})
            self.web3.eth.wait_for_transaction_receipt(tx_hash)
            self.save_credentials_to_file(username, hashed_password)
            self.update_database_single_entry(username,
hashed_password)
            messagebox.showinfo('Info', 'User Added')


            logs = self.get_logs(0, self.web3.eth.block_number,
'UserAdded')
            self.process_logs(logs)
            self.start()
```



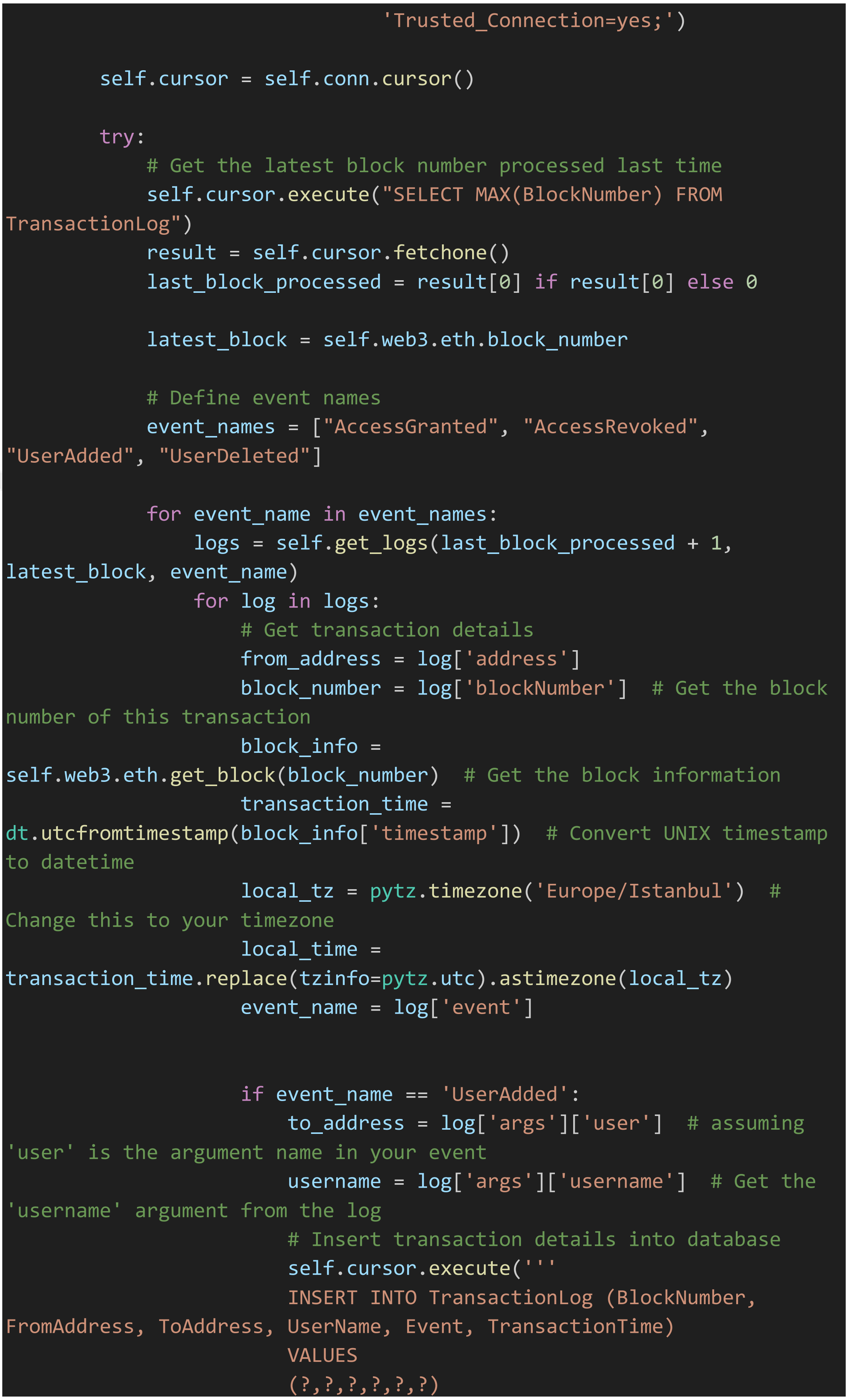
```
    def remove_credentials_from_file(self, username,
file_path="C:/Users/User/Desktop/thesis_code/Suggestion/broker/passwd.t
xt"):
        with open(file_path, "r") as file:
            lines = file.readlines()
        with open(file_path, "w") as file:
            for line in lines:
                if not
line.strip("\n").startswith("{}:".format(username)):
                    file.write(line)

    def revoke_access(self):
        owner_address = self.entry_owner_address.get()
        owner_private_address = self.entry_owner_private_address.get()
        address = self.entry_address.get()
        username = self.entry_username.get()

        if not owner_address:
            messagebox.showinfo('Info', 'The admin address cannot be
empty. Please enter the admin address.')
        elif not self.is_valid_private_key(owner_address,
owner_private_address):
            messagebox.showinfo('Info', 'The address does not match the
entered private key.')
        elif not self.isOwner(owner_address):
            messagebox.showinfo('Info', 'The address you have does not
have admin privileges.')
        elif not address:
            messagebox.showinfo('Info', 'The address is not
null')
        else:
            tx_hash =
self.contract.functions.revokeAccess(address).transact({'from':
owner_address})
            self.web3.eth.wait_for_transaction_receipt(tx_hash)
            self.remove_credentials_from_file(username)
            self.delete_database_single_entry(username)
            messagebox.showinfo('Info', 'Access revoked')

            logs = self.get_logs(0, self.web3.eth.block_number,
'AccessRevoked')
            self.process_logs(logs)
            self.start()
    def save_credentials_to_file(self, username, hashed_password,
file_path="C:/Users/User/Desktop/thesis_code/Suggestion/broker/passwd.t
xt"):
        with open(file_path, "a+") as file:
            file.seek(0)
            content = file.read()
            if content and not content.endswith("\n"):
                file.write("\n")
            file.write("{}:{}\n".format(username, hashed_password))



    def update_database_single_entry(self, user, passw):
        # Veritabanı bağlantısı kurma
        server = "DESKTOP-UO8R7VH\SQL2022"
        database = "MQTT"
        conn = pyodbc.connect(f'Driver={{SQL
Server}};Server={server};Database={database};Trusted_Connection=yes;')
        cursor = conn.cursor()

        # Veritabanına ekleme
        cursor.execute("INSERT INTO UserInfo (username, password)
VALUES (?, ?)", (user, passw))

        # Değişiklikleri onaylama ve bağlantıyı kapatma
        conn.commit()
        conn.close()


    def delete_database_single_entry(self, user):
        # Veritabanı bağlantısı kurma
        server = "DESKTOP-UO8R7VH\SQL2022"
        database = "MQTT"
        conn = pyodbc.connect(f'Driver={{SQL
Server}};Server={server};Database={database};Trusted_Connection=yes;')
        cursor = conn.cursor()

        # Veritabanından silme
        cursor.execute("DELETE FROM UserInfo WHERE username = ?",
(user,))

        # Değişiklikleri onaylama ve bağlantıyı kapatma
        conn.commit()
        conn.close()



    def delete_user(self):
        owner_address = self.entry_owner_address.get()
```



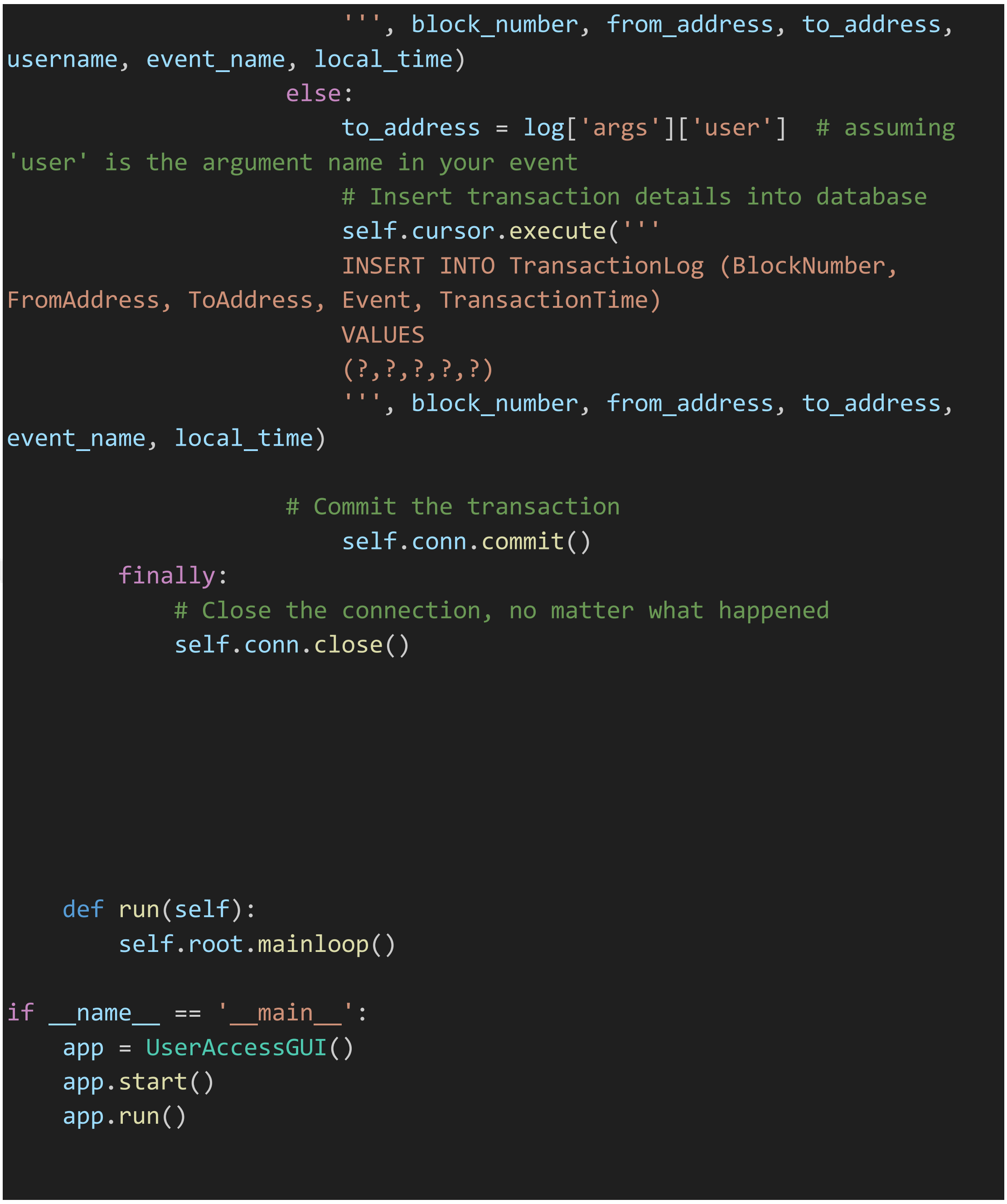
```
        owner_private_address = self.entry_owner_private_address.get()
        address = self.entry_address.get()
        username = self.entry_username.get()
        password = self.entry_password.get()

        if not owner_address:
            messagebox.showinfo('Info', 'The admin address cannot be
empty. Please enter the admin address.')
        elif not self.is_valid_private_key(owner_address,
owner_private_address):
            messagebox.showinfo('Info', 'The address does not match the
entered private key.')
        elif not self.isOwner(owner_address):
            messagebox.showinfo('Info', 'The address you have does not
have admin privileges.')
        elif not address:
            messagebox.showinfo('Info', 'The address is not
null')
        else:
            tx_hash =
self.contract.functions.deleteUser(address).transact({'from':
owner_address})
            self.web3.eth.wait_for_transaction_receipt(tx_hash)
            self.remove_credentials_from_file(username)
            self.delete_database_single_entry(username)
            messagebox.showinfo('Info', 'User Deleted')



            logs = self.get_logs(0, self.web3.eth.block_number,
'UserDeleted')
            self.process_logs(logs)
            self.start()


    def show_allowed_users(self):
        allowed_users = []
        for address in self.web3.eth.accounts:
            has_access =
self.contract.functions.hasAccess(address).call()
            if has_access:
                allowed_users.append(address)

        # Popup penceresini oluşturma
        self.popup_window = tk.Toplevel(self.root)
        self.popup_window.title('Allowed Users')
        self.popup_window.geometry('600x300')
        # Kullanıcı adreslerini kopyalayabileceğimiz bir Entry alanı
ekleyelim
```



```
        for i, user in enumerate(allowed_users):
            entry = tk.Entry(self.popup_window, font=self.bold_font)
            entry.insert(0, user)
            entry.grid(row=i, column=0, padx=50, pady=50, sticky='W')

            access_info = tk.Button(self.popup_window, text='Copy',
command=lambda address=user: self.copy_address(address))
            access_info.grid(row=i, column=1, padx=20, pady=20,
sticky='E')

            # Kopyala düğmesini oluşturma
            copy_button = tk.Label(self.popup_window,
text=self.get_access_info(user), font=self.bold_font)
            copy_button.grid(row=i, column=2, padx=20, pady=20,
sticky='E')

    def copy_address(self, address):
        # Adresi panoya kopyalama
        self.root.clipboard_clear()
        self.root.clipboard_append(address)
        self.root.update()  # Panoyu güncelleme

    def get_access_info(self, user):
        # Erişim bilgisini oluşturma
        access_time=self.contract.functions.getAccessTime(user).call()
        formatted_time =
datetime.datetime.fromtimestamp(access_time).strftime("%Y-%m-%d
%H:%M:%S")
        access_info =f"Access granted by admin:
{self.entry_owner_address.get()}\nAccess time: {formatted_time}"

        return access_info



    def get_logs(self, from_block, to_block, event_name):
        event = self.contract.events[event_name]
        logs = event.get_logs(fromBlock=from_block, toBlock=to_block)
        return logs


    def process_logs(self, logs):
        self.conn = pyodbc.connect('Driver={SQL Server};'
                                'Server=DESKTOP-UO8R7VH\SQL2022;'
                                'Database=MQTT;'
                                'Trusted_Connection=yes;')
        self.cursor = self.conn.cursor()

        try:
            for log in logs:
                # İşlem günlüğünü SQL veritabanına ekleme işlemi
                from_address = log['address']
                block_number = log['blockNumber']
                block_info = self.web3.eth.get_block(block_number)
                transaction_time =
dt.utcfromtimestamp(block_info['timestamp'])
                local_tz = pytz.timezone('Europe/Istanbul')  # Change
this to your timezone
                local_time =
transaction_time.replace(tzinfo=pytz.utc).astimezone(local_tz)
                event_name = log['event']

                # Check if log already exists in the database
                self.cursor.execute('''
                SELECT * FROM TransactionLog WHERE BlockNumber = ? AND
FromAddress = ? AND Event = ?
                ''', block_number, from_address, event_name)
                result = self.cursor.fetchone()
                if result:
                    continue  # if log exists, skip to the next log

                if event_name == 'UserAdded':
                    to_address = log['args']['user']
                    username = log['args']['username']
                    self.cursor.execute('''
                    INSERT INTO TransactionLog (BlockNumber,
FromAddress, ToAddress, UserName, Event, TransactionTime)
                    VALUES (?, ?, ?, ?, ?, ?)
                    ''', block_number, from_address, to_address,
username, event_name, local_time)
                else:
                    to_address = log['args']['user']
                    self.cursor.execute('''
                    INSERT INTO TransactionLog (BlockNumber,
FromAddress, ToAddress, Event, TransactionTime)
                    VALUES (?, ?, ?, ?, ?)
                    ''', block_number, from_address, to_address,
event_name, local_time)

                self.conn.commit()
        finally:
            self.conn.close()


    def start(self):
        self.conn = pyodbc.connect('Driver={SQL Server};'
                                'Server=DESKTOP-UO8R7VH\SQL2022;'
                                'Database=MQTT;'
```



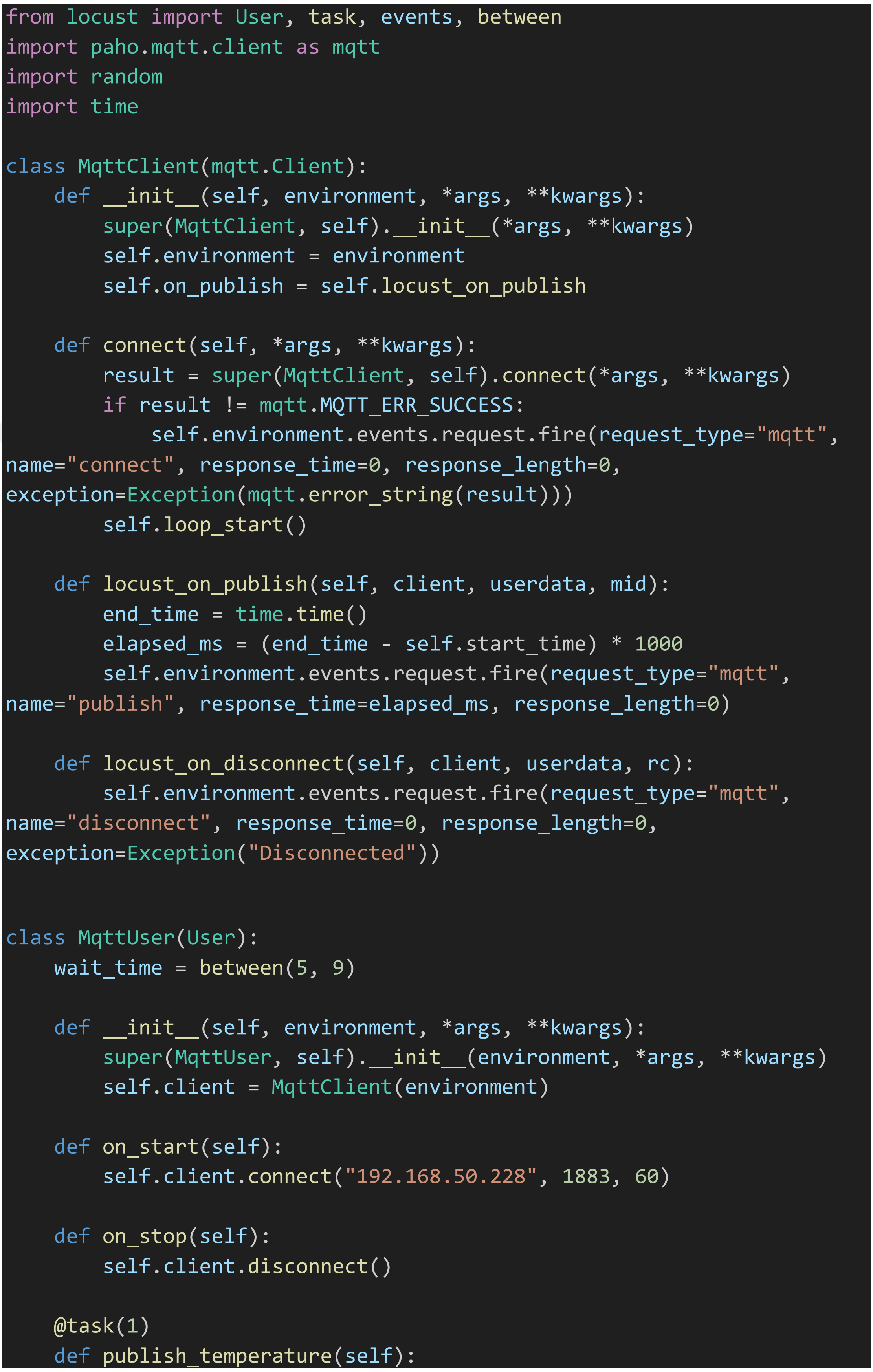
```
                                'Trusted_Connection=yes;')

        self.cursor = self.conn.cursor()

        try:
            # Get the latest block number processed last time
            self.cursor.execute("SELECT MAX(BlockNumber) FROM 
TransactionLog")
            result = self.cursor.fetchone()
            last_block_processed = result[0] if result[0] else 0

            latest_block = self.web3.eth.block_number

            # Define event names
            event_names = ["AccessGranted", "AccessRevoked", 
"UserAdded", "UserDeleted"]

            for event_name in event_names:
                logs = self.get_logs(last_block_processed + 1, 
latest_block, event_name)
                for log in logs:
                    # Get transaction details
                    from_address = log['address']
                    block_number = log['blockNumber']  # Get the block 
number of this transaction
                    block_info = 
self.web3.eth.get_block(block_number)  # Get the block information
                    transaction_time = 
dt.utcfromtimestamp(block_info['timestamp'])  # Convert UNIX timestamp 
to datetime
                    local_tz = pytz.timezone('Europe/Istanbul')  # 
Change this to your timezone
                    local_time = 
transaction_time.replace(tzinfo=pytz.utc).astimezone(local_tz)
                    event_name = log['event']


                    if event_name == 'UserAdded':
                        to_address = log['args']['user']  # assuming 
'user' is the argument name in your event
                        username = log['args']['username']  # Get the 
'username' argument from the log
                        # Insert transaction details into database
                        self.cursor.execute('''
                        INSERT INTO TransactionLog (BlockNumber, 
FromAddress, ToAddress, UserName, Event, TransactionTime)
                        VALUES 
                        (?,?,?,?,?,?)
```



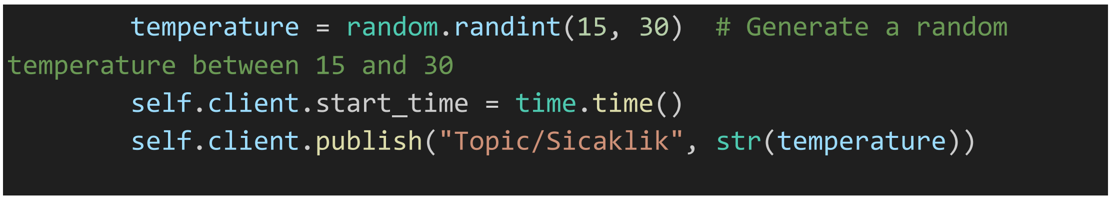
```
                        ''', block_number, from_address, to_address,
username, event_name, local_time)
                    else:
                        to_address = log['args']['user']  # assuming
'user' is the argument name in your event
                        # Insert transaction details into database
                        self.cursor.execute('''
                        INSERT INTO TransactionLog (BlockNumber,
FromAddress, ToAddress, Event, TransactionTime)
                        VALUES
                        (?,?,?,?,?)
                        ''', block_number, from_address, to_address,
event_name, local_time)

                    # Commit the transaction
                        self.conn.commit()
        finally:
            # Close the connection, no matter what happened
            self.conn.close()







    def run(self):
        self.root.mainloop()

if __name__ == '__main__':
    app = UserAccessGUI()
    app.start()
    app.run()
```

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract AccessControl {
    address public owner;
    mapping(address => bool) public allowed;
    mapping(address => string) private usernames;
    mapping(address => string) private passwords;
    mapping(address => uint256) private accessTimes;
    mapping(address => uint256) private requestCounts;
    mapping(address => uint256) private lastRequestTimestamps;

    // Event declarations
    event AccessGranted(address indexed user);
    event AccessRevoked(address indexed user);
    event UserAdded(address indexed user, string username);
    event UserDeleted(address indexed user);
    event MqttRequest(address indexed user, uint256 requestCount);

    constructor() {
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Only the owner can call this 
function.");
        _;
    }

    function grantAccess(address user) public onlyOwner {
        allowed[user] = true;
        accessTimes[user] = block.timestamp;
        emit AccessGranted(user);  // Trigger event
    }

    function revokeAccess(address user) public onlyOwner {
        allowed[user] = false;
        delete accessTimes[user];
        emit AccessRevoked(user);  // Trigger event
    }

    function hasAccess(address user) public view returns (bool) {
        return allowed[user];
    }

    function addUser(address user, string memory username, string 
memory password) public onlyOwner {
        usernames[user] = username;
        passwords[user] = password;
        emit UserAdded(user, username);  // Trigger event
    }

    function getUser(address user) public view returns (string memory
username, string memory password) {
        return (usernames[user], passwords[user]);
    }

    function deleteUser(address user) public onlyOwner {
        delete usernames[user];
        delete passwords[user];
        delete accessTimes[user];
        emit UserDeleted(user);  // Trigger event
    }

    function getAccessTime(address user) public view returns (uint256)
{
        return accessTimes[user];
    }

    function mqttRequest(address user) public {
        require(allowed[user], "Access denied.");

        uint256 currentTime = block.timestamp;
        require(requestCounts[user] < 5, "Maximum request count
reached.");
        require(currentTime >= accessTimes[user] && currentTime <
(accessTimes[user] + 1 hours), "Request allowed only during a specific
hour.");

        requestCounts[user]++;
        lastRequestTimestamps[user] = currentTime;
        emit MqttRequest(user, requestCounts[user]);
    }

    function getNumMqttRequest(address user) public view returns
(uint256) {
        return requestCounts[user];
    }

    function getLastMqttRequest(address user) public view returns
(uint256) {
        return lastRequestTimestamps[user];
    }
}
```

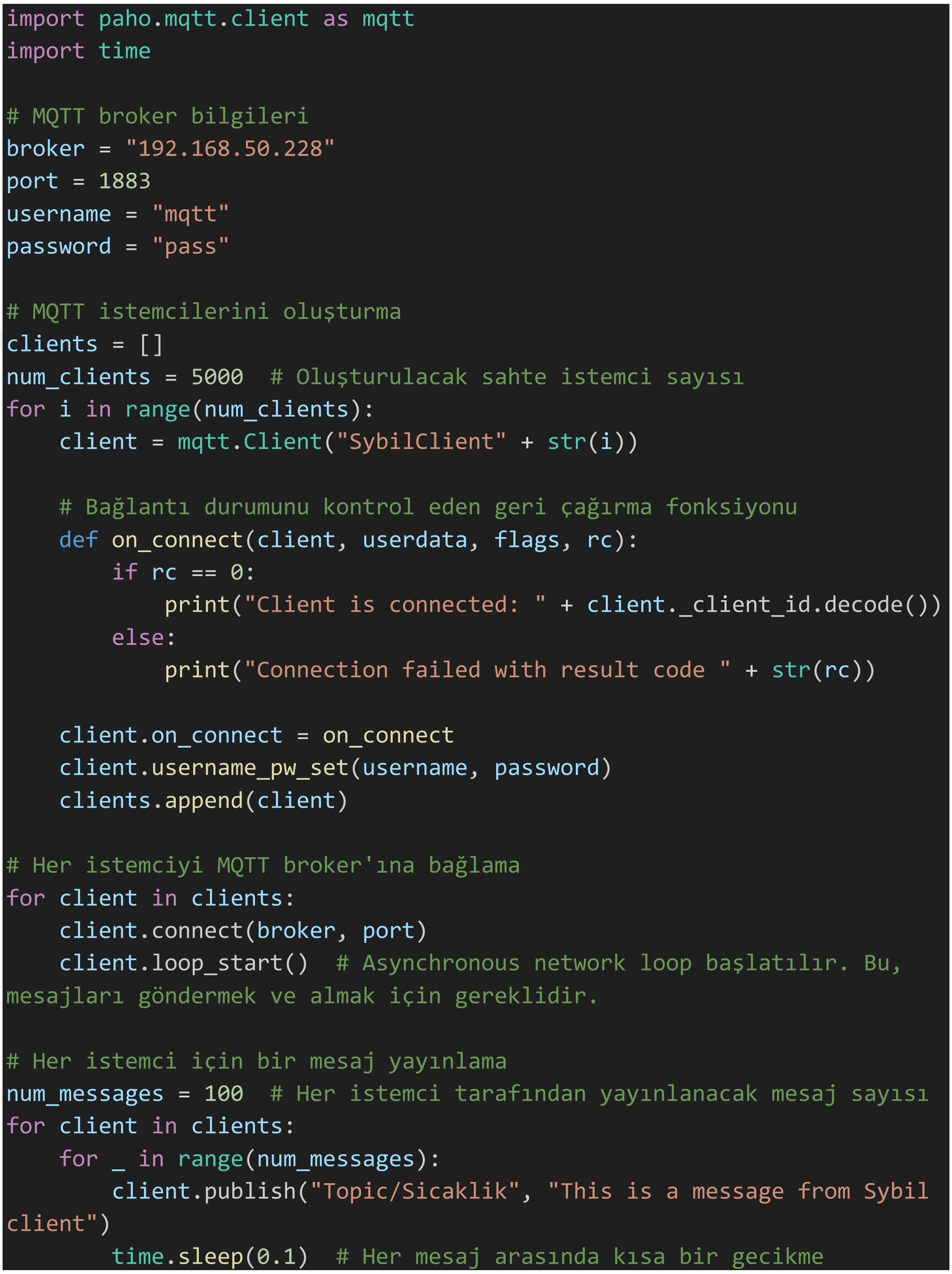
C: MQTT Python Attacks



```
from locust import User, task, events, between
import paho.mqtt.client as mqtt
import random
import time

class MqttClient(mqtt.Client):
    def __init__(self, environment, *args, **kwargs):
        super(MqttClient, self).__init__(*args, **kwargs)
        self.environment = environment
        self.on_publish = self.locust_on_publish

    def connect(self, *args, **kwargs):
        result = super(MqttClient, self).connect(*args, **kwargs)
        if result != mqtt.MQTT_ERR_SUCCESS:
            self.environment.events.request.fire(request_type="mqtt",
name="connect", response_time=0, response_length=0,
exception=Exception(mqtt.error_string(result)))
        self.loop_start()

    def locust_on_publish(self, client, userdata, mid):
        end_time = time.time()
        elapsed_ms = (end_time - self.start_time) * 1000
        self.environment.events.request.fire(request_type="mqtt",
name="publish", response_time=elapsed_ms, response_length=0)

    def locust_on_disconnect(self, client, userdata, rc):
        self.environment.events.request.fire(request_type="mqtt",
name="disconnect", response_time=0, response_length=0,
exception=Exception("Disconnected"))


class MqttUser(User):
    wait_time = between(5, 9)

    def __init__(self, environment, *args, **kwargs):
        super(MqttUser, self).__init__(environment, *args, **kwargs)
        self.client = MqttClient(environment)

    def on_start(self):
        self.client.connect("192.168.50.228", 1883, 60)

    def on_stop(self):
        self.client.disconnect()

    @task(1)
    def publish_temperature(self):
```



```
        temperature = random.randint(15, 30)  # Generate a random 
temperature between 15 and 30
        self.client.start_time = time.time()
        self.client.publish("Topic/Sicaklik", str(temperature))
```



```
import paho.mqtt.client as mqtt
import time

# MQTT broker bilgileri
broker = "192.168.50.228"
port = 1883
username = "mqtt"
password = "pass"

# MQTT istemcilerini oluşturma
clients = []
num_clients = 5000  # Oluşturulacak sahte istemci sayısı
for i in range(num_clients):
    client = mqtt.Client("SybilClient" + str(i))

    # Bağlantı durumunu kontrol eden geri çağırma fonksiyonu
    def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print("Client is connected: " + client._client_id.decode())
        else:
            print("Connection failed with result code " + str(rc))

    client.on_connect = on_connect
    client.username_pw_set(username, password)
    clients.append(client)

# Her istemciyi MQTT broker'ına bağlama
for client in clients:
    client.connect(broker, port)
    client.loop_start()  # Asynchronous network loop başlatılır. Bu, 
mesajları göndermek ve almak için gereklidir.

# Her istemci için bir mesaj yayınlama
num_messages = 100  # Her istemci tarafından yayınlanacak mesaj sayısı
for client in clients:
    for _ in range(num_messages):
        client.publish("Topic/Sicaklik", "This is a message from Sybil 
client")
        time.sleep(0.1)  # Her mesaj arasında kısa bir gecikme
```