Blockchain-based decentralized identity system: Design and security analysis

Bilel Zaghdoudi
Sorbonne University
LIP6
Paris, France
bilel.zaghdoudi@lip6.fr

Gewu BU
Clermont Ferrand University
LIMOS
France
gewu.bu@uca.fr

Maria Potop-Butucaru
Sorbonne University
LIP6
Paris, France
maria.potop-butucaru@lip6.fr

Serge Fdida
Sorbonne University
LIP6
Paris, France
serge.fdida@lip6.fr

***Abstract*—This paper presents a novel blockchain-based decentralized identity system (DID), tailored for enhanced digital identity management in Internet of Things (IoT) and device-to-device (D2D) networks. The proposed system features a hierarchical structure that effectively merges a distributed ledger with a mobile D2D network, ensuring robust security while streamlining communication. Central to this design are the gateway nodes, which serve as intermediaries, facilitating DID registration and device authentication through smart contracts and distributed storage systems. A thorough security analysis underscores the system's resilience to common cyber threats and adherence to critical principles like finality and liveness.**

Keywords*—*Decentralized identity, D2D, Blockchain, Decentralized Storage, Smart contract

I. Introduction

In the dynamic realm of technology, the implementation of blockchain-based services stands as a transformative force, poised to revolutionize the landscape of decentralized applications for devices and the Internet of Things (IoT). Internet of Things devices are often limited in terms of computational and energy resources due to their design goals and constraints. Due to these limitations, IoT devices often require specialized software and algorithms tailored for resource-constrained environments. Optimization techniques, lightweight protocols, and efficient algorithms are used to ensure effective operation within the available resources while meeting the desired functionality and security requirements of IoT applications. The importance of incorporating blockchain into the fabric of these applications cannot be overstated, as it introduces a level of security, transparency, and efficiency that is paramount in our interconnected world.

The provision of blockchain-based services offers a unique opportunity to empower and expedite the implementation of decentralized applications for devices and IoT in a modular and versatile manner. By breaking down the complexities associated with bespoke development, these services serve as building blocks, enabling developers to leverage essential functionalities seamlessly. This modular approach not only accelerates the development process but also fosters a more adaptable and scalable ecosystem for a diverse range of applications.

As we delve into specific examples of blockchain-based services, the significance becomes clear. Smart contracts, for instance, automate and secure agreements between devices, reducing the need for intermediaries and enhancing the efficiency of interactions. Immutable data storage on the blockchain ensures the integrity and reliability of data generated by devices, fostering a trustworthy foundation for various applications. Decentralized consensus mechanisms mitigate the risks associated with centralized points of failure, contributing to the resilience of the entire network.

One example of service that emerges as a linchpin in this transformative landscape is the *decentralized identity* service based on blockchain. As the backbone of secure and tamper-resistant identification for devices, this service not only safeguards against unauthorized access but also empowers users with greater control over their digital identities.

Decentralized identity systems and self-sovereign identity (SSI), especially their blockchain-based implementations, are increasingly important in digital identity management. Numerous surveys and studies [15] - [25] have explored these systems' technological, security, governance, and user-centric aspects. These works highlight the balance between blockchain's immutability and transparency and SSI's focus on autonomy and privacy. They analyze the challenges and benefits of decentralized identity systems, including their potential to revolutionize traditional identity management and give users more control over their data. Furthermore, these studies examine the implications for regulatory compliance, scalability, and integration with existing digital infrastructures.

Recent research efforts have increasingly focused on exploring the benefits and practical applicability of decentralized identities and self-sovereign identities (SSIs) within the realm of the Internet of Things (IoT). This burgeoning interest stems from the growing recognition of the potential that decentralized identity models, underpinned by technologies such as blockchain and distributed ledgers, hold in addressing core IoT challenges. In [26] for example, the authors present a detailed analysis of the Self-Sovereign Identity (SSI) concept and its application in the context of the Internet of Things (IoT). It contrasts existing identity approaches like cloud-based accounts and digital certificates with emerging SSI standards such as Decentralized Identifiers (DIDs) and Verifiable Credentials (VCs). The authors argue that SSI, with its owner-centric, privacy-aware, and decentralized approach, offers a viable and attractive option for the secure identification of IoT devices and users. While the paper discusses the theoretical security aspects of SSI, it does not delve into a detailed study of the security of the proposed model, such as vulnerability assessments or resistance to specific types of cyber-attacks, which is crucial in the context of IoT security. In [27] the authors propose an innovative approach for implementing self-sovereign identity (SSI) in IoT networks, particularly focusing on the challenges posed by constrained networks. The authors propose a new DID method tailored for IoT networks, alongside a novel serialization mechanism for the DID document. Additionally, the paper introduces a binary message envelope for secure communication. This study focuses on the practical implications of implementing SSI in constrained IoT networks, offering a significant reduction in the size of identity metadata and security overhead. Although the authors emphasize reducing security overhead and ensuring secure communication, this study lacks a detailed examination of the model's resilience against various cyberattacks and vulnerabilities.

Furthermore, in [28] the authors focus on providing an in-depth overview of various approaches to self-sovereign identity (SSI) and its application in IoT environments. It discusses the evolution of digital identity, emphasizing the importance of SSI in the context of increasing privacy concerns and the proliferation of IoT devices. The authors compare different SSI infrastructures, highlighting their strengths and weaknesses. The paper also examines existing SSI solutions, such as uPort [1], Sovrin [2], and others [3]- [14]. Although this research provides a detailed comparison of the current SSI infrastructures and their relevance to IoT, it has a significant drawback in terms of security. The authors discuss the cryptographic foundations of SSI and the use of blockchain technology to ensure safe identity management. However, they do not conduct a thorough analysis of the security of their particular attack model.

To the best of our knowledge, we are the first to propose a blockchain-based hierarchical architecture specifically designed to support device-to-device communication and propose a secure decentralized identity service. The originality of our work stands in proposing an end-to-end security proof for our decentralized identity service against a formally defined attacker. The rest of the paper is organized as follows: Section II presents the system architecture, Section III presents the detailed workflow of our decentralized identity service, and Section IV proposes the formal security proofs for our service and finally Section VI concludes the paper.

II. Blockchain-based system architecture

Our architecture integrates a dynamic, decentralized network, incorporating a hierarchical structure with both an upper-layer distributed ledger system and a mobile Device-to-Device (D2D) network layer (see Figure 1). Two main functional entities compose our architecture:

- Devices are the main component of the network. A device is an entity or physical object that has a unique identifier, an embedded system, and the ability to transfer data over a network.
- The IoT gateways that act as Blockchain nodes. These gateways maintain the connection between the devices and the Blockchain. These nodes verify and validate transactions corresponding to the device's requests.

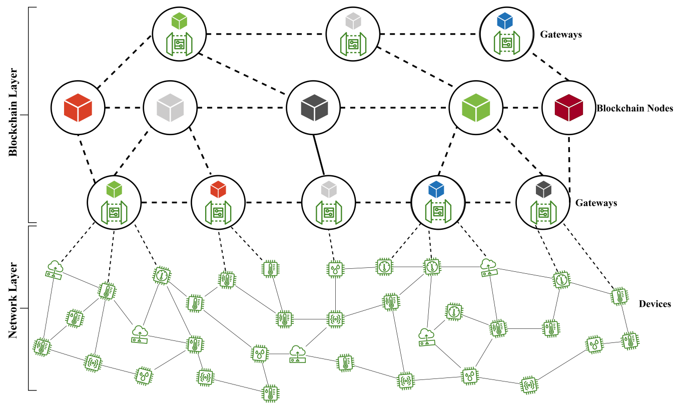

Fig. 1. Hierarchical architecture

Two groups of participant nodes in the blockchain need to be addressed separately in our architecture:

1) Nodes that maintain the consensus algorithm of the blockchain, can be called Miners in traditional blockchains or Members of the Election Committee in voting-based blockchains. They are the ones who ensure the proper functioning of the chosen blockchain in the architecture
2) Gateway nodes are spread at the edge of the upper blockchain network and are the intermediary medium to link the blockchain layer to the D2D network layer. They follow a process of registering their DID during system initialization.

Note that a blockchain participant may be both a miner and a gateway node, or neither at all.

Both Gateways and D2D nodes are equipped with asymmetric cryptography keys, which enable secure and private communication. These keys consist of a public key and a private key, which are uniquely generated for each device. Furthermore, the private key is securely stored within each device, safeguarding it from unauthorized access. It is utilized for decrypting received encrypted data and for digitally signing messages. By leveraging these cryptographic capabilities, including encryption, digital signatures, and hashing, each device within the network can ensure secure and trustworthy communication, protect sensitive information, verify the authenticity of data exchanges, and be associated with a unique identifier.

Another pivotal component of our architecture is the decentralized storage system, which is intricately linked to the upper layer of the architecture via gateway nodes. These nodes serve as critical junction points, enabling seamless communication and data transfer between the decentralized storage system and the rest of the architecture. This system offers a fundamental storage primitive for our architecture, essentially acting as the backbone for data storage and management. It also maintains a public-private key pair and provides a signature of retrieved data using this key pair. By leveraging the decentralized nature of this storage system, we ensure enhanced security, scalability, and redundancy in data handling, which is crucial for maintaining the integrity and efficiency of the entire architecture. The integration of this system through gateway nodes not only simplifies data accessibility but also bolsters the overall resilience and robustness of our architectural framework.

III. Decentralized Identity Service

Our proposal introduces a Decentralized Identifier (DID) service that offers two key functionalities: device DID registration and device DID authentication. The first functionality, device DID registration, enables devices to acquire unique decentralized identifiers. The second functionality, device DID authentication, provides a mechanism for verifying the identity of these devices. Together, these functionalities form the core of our DID service, facilitating secure and efficient identity management for devices in a decentralized environment.
Gateways are at the heart of our DID service and function as controllers. They possess the ability to generate Decentralized Identifier (DID) documents, conduct transactions, and interact with the decentralized storage system for storing and retrieving these documents. When a node joins the system for the first time, it will try to establish a connection with at least one trusted gateway node to register a network-wide unique DID. Once a node has successfully registered its DID, other nodes can verify its legitimacy based on this node's DID. Nodes use their DIDs to participate in the various communications in the system. Since the DID is stored in the upper layer distributed ledger along with its metadata in connected distributed storage at the time of node registration, and cannot be tampered with, this unique DID will replace the traditional certificate system to verify the identification of the node.
Additionally, a smart contract is implemented, fulfilling the essential roles of both a resolver and a secure, unalterable registry for DIDs. This smart contract provides the functionality to map DIDs with their corresponding data and upholds a reliable registry, guaranteeing the accuracy and legitimacy of decentralized identity data.

A. Register a device's DID

In the following, we will present a detailed, step-by-step description of Figure 2, which illustrates the DID registration flow.

1) The DID registration: A device requests the gateway to register its DID by sending its public key, the verification method, identity metadata if needed, and its signature.
2) At this stage, the gateway playing the controller role in the DID standard architecture formulates the DID based on the chosen verification method and the public key. The specific format and syntax may vary depending on the method.
3) Create a DID Document: Construct a DID document for the IoT device. The DID document contains metadata and cryptographic material associated with the DID. It typically includes the public key, verification method, and service endpoints. The structure and content of the DID document depend on the data received from the device.
4) Store the DID Document: A decentralized storage system is used to store the DID document. This can be achieved by uploading the document to the decentralized storage system gateway and obtaining its unique identifier (the document hash) and the decentralized storage system public key.
5) Register the DID: Call the appropriate function in the smart contract to register the DID and link it to the DID document hash. This step establishes the connection between the DID and its corresponding document and finalizes the process of identity creation.
6) The Smart contract DID registration Response: Upon the DID registration, the smart contract responds to the requesting gateway with the created DID and the registration transaction hash.
7) The Gateway DID registration Response: The gateway forwards the created DID and the decentralized storage system public key after signing the message with its private key to the requesting device as the final step of the DID creation.

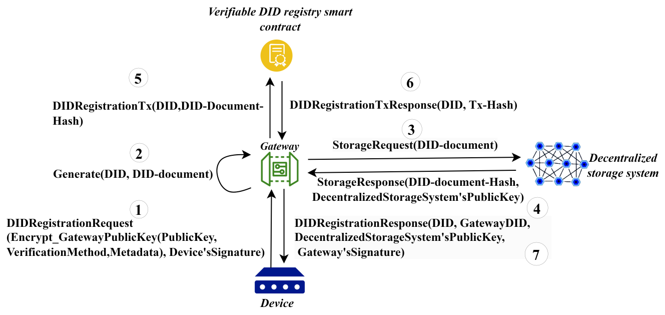
Verifiable DID registry smart contract
5
DIDRegistrationTx(DID,DID-Document-Hash)
6
DIDRegistrationTxResponse(DID, Tx-Hash)
3
StorageRequest(DID-document)
2
Generate(DID, DID-document)
Gateway
Decentralized storage system
StorageResponse(DID-document-Hash, DecentralizedStorageSystem'sPublicKey)
1
DIDRegistrationRequest (Encrypt_GatewayPublicKey(PublicKey, VerificationMethod,Metadata), Device'sSignature)
4
DIDRegistrationResponse(DID, GatewayDID, DecentralizedStorageSystem'sPublicKey, Gateway'sSignature)
7
Device

Fig. 2. DID registration flow

B. Authenticate a device

In the subsequent section, we will provide a thorough, sequential breakdown of Figure 3, illustrating the DID authentication flow.

1) Obtain the Device's DID: Retrieve the specific DID associated with the device that you want to authenticate. This can be obtained from the device itself or from a trusted source that provides the device's DID.
2) Start the authentication process by sending a request to the gateway containing the DIDtoAuthenticate, the device's DID, and the device's signature.
3) Resolve the DID: The gateway uses the device's DID to fetch the associated device DID document identifier (document hash) by querying the resolver smart contract.
4) Fetch the DID Document: Use the obtained device document identifier to retrieve the associated device DID document. This document contains the necessary information to authenticate the device.
5) The gateway extracts the public key from the document and verifies the device's signature.
6) The gateway repeats the same process for the DID-toAuthenticate to get its document.
7) As the final DID document response, the gateway responds to the device with a message containing the DIDtoAuthenticate document and the decentralized storage system signature.
8) When receiving the final DID Document response message, the device starts by verifying the decentralized storage system signatures.

9) Extract the Verification Method: Within the fetched DIDtoAuthenticate document, the device identifies the verification method. The verification method specifies the cryptographic algorithm and key material required for authentication.
10) Retrieve the Public Key: Extract the public key that will be used for the authentication process.
11) Verify Authentication: Use the public key to verify the signature provided by the device to authenticate. This process ensures that the device possesses the private key corresponding to the public key and can provide valid authentication proof.

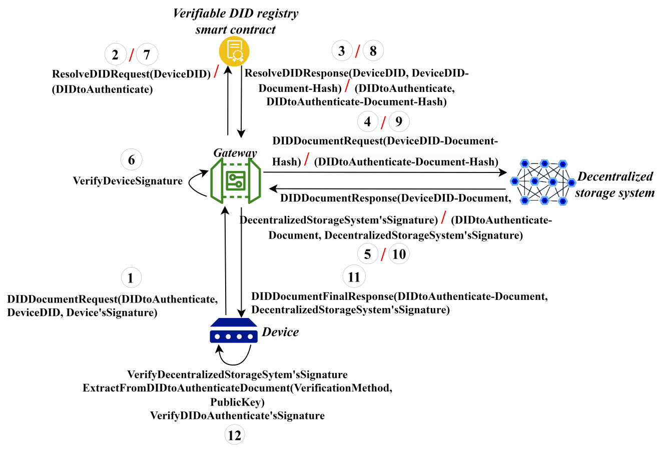
Verifiable DID registry smart contract
2 / 7
ResolveDIDRequest(DeviceDID) / (DIDtoAuthenticate)
3 / 8
ResolveDIDResponse(DeviceDID, DeviceDID-Document-Hash) / (DIDtoAuthenticate, DIDtoAuthenticate-Document-Hash)
4 / 9
DIDDocumentRequest(DeviceDID-Document-Hash) / (DIDtoAuthenticate-Document-Hash)
6
VerifyDeviceSignature
Gateway
Decentralized storage system
DIDDocumentResponse(DeviceDID-Document, DecentralizedStorageSystem'sSignature) / (DIDtoAuthenticate-Document, DecentralizedStorageSystem'sSignature)
5 / 10
1
DIDDocumentRequest(DIDtoAuthenticate, DeviceDID, Device'sSignature)
11
DIDDocumentFinalResponse(DIDtoAuthenticate-Document, DecentralizedStorageSystem'sSignature)
Device
VerifyDecentralizedStorageSytem'sSignature
ExtractFromDIDtoAuthenticateDocument(VerificationMethod, PublicKey)
VerifyDIDoAuthenticate'sSignature
12

Fig. 3. Authenticate device flow

IV. Security Analysis

In the following, we propose the security proof of our DID service. First, we define the adversary model and the relevant security hypotheses.

A. Formal Notations and Adversary Model

The following notations and definitions are borrowed from Dolev, et al. [29]. A node A holds X as known information is denoted by:

$$I_A = \{X\}$$

Sending message from A to B with content X is denoted by:

$$A - B : (X)$$

when this is a broadcast message from A, destination will not be mentioned :

$$A : (X)$$

The Checking operation of the legitimacy, performed by B, for a received message is denoted by:

$$Checking_B((X), I_B)$$

The checking operation verifies whether a message is legitimate by comparing it with locally available information.

Encryption / Decryption operations on a particular content X with key K is denoted by:

$$E(K, X)$$

We define an adversary node Z with the following capabilities:

- Z can intercept the message sent from A to B during the transmission.
- Z can modify the message received from a node and re-transfer it.
- Z can generate and propagate any type of message that may appear in the system
- When required, Z may or not perform any operation, then give arbitrary result or response.
- Z can update I_Z, when intercepting useful information.
- Z can try to decrypt encrypted message by using information from I_Z at any time.

In terms of attack patterns, the defined adversary Z is an attacker who can initiate *Adaptive Chosen Message Attacks (A-CMA)* [30]. In this case, the adversary can not only select a set of messages and obtain their corresponding messages (ciphertexts / plaintexts), but also select subsequent messages based on the results of the previous set of messages.

B. System model and security hypothesis

Our system is a dynamic and decentralized network, consisting of a large finite yet unbounded set of mobile devices.

We consider that in our system, nodes communicate via messages, and wireless communication links are designed to prevent data loss.

In our model, we distinguish between three types of messages: a signed message, a signed item part of a message, and a non-signed message. The difference between these types lies in the scope and verification of the cryptographic signature applied to the message or its components. In the following, we present a detailed explanation of each type of message:

- Signed Message: This refers to an entire message that has been digitally signed, i.e., $sign_A$ represents the signature of a complete message signed by node A. The signature covers every part of the message, ensuring its integrity and authenticity. This is common in scenarios where the entire content of the message needs to be protected against tampering and impersonation. The list of signed messages exchanged during the DID service workflows is as follows: DIDRegistrationRe0quest, DIDRegistrationTx, DIDRegistrationResponse, DIDDocumentRequest.
- Signed item part of a message: In this case, only a specific part or item within the message is signed, rather than the entire message, i.e., $sign_A(item)$ represents the signature of a part or an item of a message signed by a node A. This might be used in scenarios where only certain critical elements of

the message need integrity and authenticity protection. The messages with a signed item exchanged during the DID service workflows are DIDDocumentResponse, DIDDocumentFinalResponse.

- Non-Signed Message: A non-signed message is a message that does not have any digital signature associated with it. The list of non-signed messages exchanged during the DID service workflows is as follows: StorageRequest, DIDRegistrationTxResponse, ResolveDIDRequest, ResolveDIDResponse, DIDDocumentRequest, StorageResponse.

We list below the security hypotheses.

1) The distributed ledger system at the upper layer tolerates the attacker previously defined.
2) Connected distributed storage system satisfies the properties of *Atomicity*, *Consistency*, *Isolation* and *Durability* (ACID properties recalled below. This distributed storage system tolerates attackers previously defined.
3) All Gateways are connected to the distributed storage system via an interface. Through that interface, the communication between gateways and distributed storage system satisfies the properties of *Confidentiality*, *Integrity*, and *Availability* (CIA properties recalled below), which tolerates attackers previously defined.
4) Each device and gateway node possesses a unique pair of asymmetric cryptography keys (public and private keys) for secure communication. The private key is securely stored within each device.
5) Both Gateway and D2D nodes can compute local DIDs via public keys, establishing a mono-mapping of public keys to DIDs.
6) Public-private key paires based RSA encryption scheme used in the system is secure under *Adaptive Chosen Message Attacks*.
7) Public-private key paires based digital signature used in the system satisfies *Existential Unforgeability* under *Adaptive Chosen Message Attacks* [30], which means the adversary cannot succeed in forging the signature of any message.
8) Hash function, $hash()$, used in the system, is considered one-way and unbreakable.
9) A whitelist of trusted gateway nodes, with their public keys and DIDs, is provided to joining devices during initialization for secure connection establishment.
10) The network includes firewall-like middleware to prevent physical attacks like flooding and DoS attacks.
11) The distributed storage system maintains a public-private key pair and provides a signature of retrieved data using this key pair.

The chosen upper-layer distributed storage should satisfy the atomicity, consistency, isolation, and durability properties listed below:

- Atomicity: all operations in a transaction, either all complete or none of them.
- Consistency: The integrity of the database (storage) will not be corrupted before the transaction begins and after the transaction ends.
- Isolation: The storage system prevents data inconsistency due to the execution concurrently of multiple transactions.
- Durability: once a transaction completes, changes to the data are permanent and will not be lost.

Through a dedicated interface, gateway nodes interact with the distributed storage system satisfying the security properties of confidentiality, integrity, and availability listed as follows:

- Confidentiality: ensuring that information is transmitted and stored confidentially so that unauthorized users do not reveal the contents of the information.
- Integrity: ensuring the correctness and consistency of data throughout its life cycle, either in transmission or in storage.
- Availability: when one needs to operate through the storage systems, information, and services must remain available.

C. DID Service Security proofs

Let us begin with a basic Lemma 1.

Lemma 1. *A node can verify the integrity of a signed item (e.g., message, message field, message content etc.) by using the corresponding public key and verification function.*

Proof: The lemma directly follows from *Hypothese 7*. ■

Next, we prove the security of the DID registration and authentication process.

*1) **DID Registration**:* In DID registration, the D2D node attempts to communicate with the gateway nodes in its whitelist, with encryption. Upon receiving the message, the gateway node interacts with the distributed ledger and distributed storage to register its DID, which will be eventually sent back to the requesting node as confirmation. We consider a DID registration scheme to be secure if it satisfies the following two properties:

P1 Any correct DID registration request will eventually be accepted by the DID service.

P2 Any node invoking DID registration will eventually receive a correct DID registration response.

A correct DID registration request or response is a message that respects the format defined in Section III. And a correct DID registration response should be sent by a whitelisted gateway node and its integrity should be verified by the receiving node.

For the first property $P1$, we recall that according to the hypothesis, newly joined D2D node A is pre-installed with a whitelist containing several trusted gateway nodes and a public-private key pair for asymmetric encryption.

DID registration requirement consists of the body of the requirement and signature of message signed with A's secret key. Node A will try to send the requirement to some of the gateway nodes in its whitelist, to increase the potential for messages to be accepted by the DID service. Let Dst be one of these chosen destination gateways.

$$I_A = \{..., K_{pub}(A), K_{sct}(A), whiteList_A, K_{pub}(Dst), ...\}$$
$$A - Dst : (X||sign_A)$$

where X contains A's public key as well as related metadata and verification method, $verMD$. X is encrypted with the public key of the expected destination gateway node Dst:

$$X = E(K_{pub}(Dst), K_{pub}(A)||verMD||metaData)$$

and $sgin_A$ is the result of encryption of the hash of X by the $K_{sct}(A)$

$$sign_A = E(K_{sct}(A), hash(X))$$

According to our hypotheses, wireless communication links are not lossy. Therefore, any DID registration request are eventually received by a whitelisted destination gateway node.

We then give two Lemmas 2 and 3 to show the first property $P1$ in DID registration holds.

Lemma 2. *If a DID registration request is received by a whitelisted gateway, and it is encrypted by the public key of that gateway node, then the integrity of the DID registration request can be verified by that gateway.*

Proof: Since the DID registration request is signed by the requesting node A, when another node B receives this message, it needs only to obtain the public key of A, which allows it to perform the integrity check according to the Lemma 1.

In order to obtain the public key of A, $K_{pub}(A)$, the node receiving this message will try to use its own secret key to decrypt the message content X' and obtain $K'_{pub}(A)$ from it. Note, however, a node can only decrypt the content correctly and obtain the correct $K_{pub}(A)$ and related information, if it is the destination gateway Dst. Thus the destination gateway node can verify the integrity of the DID registration request according to Lemma 1.

$$X = E(K_{pub}(Dst), K_{pub}(A)||verMD||metaData)$$
$$A - Dst : (X||sign_A)$$
$$I_{Dst} = \{..., K_{sct}(Dst), X', sign'_A, ...\}$$
$$(K'_{pub}(A)||verMD'||metaData') == E(K_{sct}(Dst), X')$$
$$K'_{pub}(A) == K_{pub}(A)$$

Since the public key used to encrypt the content of this message is the public key of Dst, it is impossible for any other node Z to correctly decrypt the content of the message without the private key of Dst, and hence for any other receiver nodes, it is impossible for them to obtain the correct $K_{pub}(A)$, and hence they will not be able to pass the integrity check.

$$I_Z = \{..., K_{sct}(Z), X', sign'_A, ...\}$$
$$(K'_{pub}(A)||verMD'||metaData') == E(K_{sct}(Z), X')$$
$$K'_{pub}(A) \neq K_{pub}(A)$$

■

Lemma 3. *Any correct DID registration request will eventually be accepted by the DID service.*

Proof: According to Lemma 2, the integrity of a DID registration request can be verified by a whitelisted destination gateway node. Therefore based on the communication assumptions, there must be a correctly formatted DID registration request whose integrity is verified that is correctly received by a whitelisted destination gateway node. Then the gateway can obtain the correct public key of requesting node and all related information: verification method and metadata, and perform the correct registration process. At this point we can say that the DID registration request has been correctly accepted by the proposed DID service. Therefore the first property $P1$ holds. ■

For the second property $P2$, since once a gateway G in the whitelist of the requesting D2D node has correctly received the DID registration request, it will honestly execute the process described in Section III to complete the DID registration. Since all interactions with the distributed storage and the smart contract in distributed ledger layer are performed by a whitelisted gateway node, a correct DID registration response therefore must be generated at that whitelisted gateway node.

$$I_G = \{..., DID_G, DID_A, K_{sct}(G), K_{pub}(sys), ...\}$$
$$X = DID_G||DID_A||K_{pub}(sys)$$
$$sign_G = E(K_{sct}(G), hash(X))$$
$$G - A : (X||sign_G)$$

where $K_{pub}(sys)$ is the public key of a public-secret key pair maintained by the distributed storage system. This public key is received at step 4 of DID registration by that whitelisted gateway node. It is used in the DID authentication process.

However, it is notable that a gateway node attempting to register a DID, will first check whether the DID has already been registered to prevent data redundancy. Therefore, if a malicious gateway node is the first to receive the DID registration request of A, and tries to perform a malicious operation in the upper distributed system with DID of A, it is likely to cause an impact on the registration process of A, for example it can modify the metadata or verification method of DID, then create and store wrongly DID document to the distributed storage system. But in fact, with the following Lemma 4, we can see that the malicious gateway node is not able to interrupt the correct DID registration process.

Lemma 4. *If a DID registration request from a correct node dev is received by a malicious D2D node / gateway then the malicious D2D node / gateway cannot decrypt the DID registration request or any related information. The malicious D2D node/gateway cannot execute correctly the DID registration process.*

Proof: We know that only gateway nodes can interact with the distributed system and execute the DID registration according to the description in Section III, therefore if the malicious node that receives the DID registration request is a D2D node, it cannot do anything except tamper with the message or the content of message, and then forward it to other nodes. Then, according to Lemma 2, we know that the tampering with DID registration requests will eventually be detected by verifying the message integrity. Therefore, in this proof, we only consider that the malicious node that receives the DID registration request is a gateway node.

By receiving a DID registration request from A, a malicious gateway node Z tries to use the wrong information to disrupt the registration process of the requesting D2D node. Z will try to get the public key of A, then it computes the DID of A from its public key. Let the function $did()$ be the algorithm for computing DID from a public key. With this function, from one public key, only one unique DID can be obtained.

$$X = E(K_{pub}(Dst), K_{pub}(A)||verMD||metaData)$$
$$A - Dst : (X||sign_A)$$
$$I_Z = \{..., X', sign'_A, K_{pub}(Z), K_{sct}(Z)...\}$$
$$K'_{pub}(A)||verMD'||metaData'_A \leftarrow E(K_{sct}(Z), X')$$
$$DID'_A = did(K'_{pub}(A))$$

Since Z is a malicious gateway node, it will not be on A's whitelist. Therefore Z will never be able to get the correct public key of A by decrypting the content of received message. So it can't compute the correct DID_A either. That means, Z will not be able to create an error DID document with correct DID_A to preform the DID registration and to interrupt the registration process of other honest gateway node.

Note that a malicious gateway node can indeed request the registration with fictitious DIDs, but it cannot predict and be aware of the DIDs that need to be registered.

■

From previous lemma it follows that malicious nodes/gateways cannot store in the distributed ledger or distributed storage a wrong DID.

Lemmas 5, 6 and 7 below prove the second property $P2$.

Lemma 5. *When DID registration request is accepted by the DID service, a DID registration response from a whitelisted gateway node will eventually be sent to the DID registration requesting node.*

Proof: According to Lemma 3, the DID registration request message must eventually be received by a whitelisted gateway node, which means accepted by DID service. According to Lemma 4, malicious nodes cannot disrupt the correct DID registration process. Therefore the receiving whitelisted gateway node must perform the DID registration operation correctly and generate a correct response message and send it back to the requesting node. Also according to our network model in Section IV-B, we know that any message will eventually be sent to any node. So we can say that a DID registration response from a whitelisted gateway node will be eventually received by the DID registration requesting node.

■

Lemma 6. *If a DID registration response is received by the requesting node from one of its whitelisted gateway nodes, then the integrity of the DID registration response can be verified by that requesting node.*

Proof: According to Lemma 5, the requesting node A will eventually receive a DID registration response from a gateway node in its whitelist. To verify the integrity of this response, A only needs to obtain the DID of the sending gateway node from the response, and search for the corresponding public key of that gateway node from its whitelist.

$$X = DID_G||DID_A||K_{pub}(sys)$$
$$G - A : (X||sign_G)$$
$$I_A = \{..., X', sign'_G, whiteList_A, ...\}$$
$$DID'_G \leftarrow X$$
$$K'_{pub}(G) \leftarrow whiteList_A(DID_G)$$

By now, node A has obtained the public key of the signing node, the content to be verified and the signature for the content part. Thus according to Lemma 1, node A can verify the integrity of the DID registration response.

■

Lemma 7. *Any node invoking DID registration request will eventually receive a correct DID registration response.*

Proof: According to Lemma 5 and 6, A can eventually receive a response from a whitelisted gateway node whose integrity can be verified. Therefore as long as A does not receive a DID registration response from a whitelisted gateway that passes the integrity verification, it can resend the DID registration request until it receives a response from a whitelisted gateway whose integrity is verified. Since the whitelisted gateway node is a honest node, it is sufficient to verify the integrity of its response to confirm that the content of DID registration response is correct. Hence the second property $P2$ holds. ■

Finally according to Lemma 3 and 7, both $P1$ and $P2$ of DID registration are proven to be correct and secure. Therefore we say that the whole DID registration process is correct and secure.

According to Lemma 4 and 7 the DID registration process is secure and correct and will only be performed by whitelisted honest gateway nodes, so once a DID from a requesting D2D node has been registered, it must be correctly stored in the distributed ledger system and distributed storage system.

Note 1. *Note that if a node correctly accomplished the registration process then all the registration information must have been stored correctly in distributed ledger and distributed storage system.*

*2) **DID Authentication**:* In DID authentication, when a registered node has obtained a DID of another node (which can be done from online or offline), it can request a gateway node to verify if the owner of the DID is a registered node in the system or not, and to obtain its public key as well as the verification method for future signature verifying. Thus through DID authentication, a node can verify the source and integrity of the received message. We say that a DID authentication process is secure and correct if it satisfies the following two properties:

P1 Any correct DID authentication request will eventually be accepted by the DID service.

P2 Any node invoking DID authentication will eventually receive a correct DID authentication response.

A correct DID authentication request is a message that respects the format defined in Section III. And a correct DID authentication response is a message containing the requesting DID document, the integrity of that DID document is preserved.

For the first property $P1$, we recall that when a registered node A wishes to verify the registration of a DID, DID_{req} and get the corresponding verification method for its signature, it creates a DID authentication request and tries to send it to at least a gateway node through one-hop or multi-hop communication mentioned before, including the DID of A, DID_A, queried DID, DID_{req}, alongside A's signature of DID authentication request:

$$\begin{aligned}
&I_A = \{...DID_A, DID_{req}, K_{pub}(A), K_{sct}(A), ...\}\\
&X = DID_A||DID_{req}\\
&sign_A = E(K_{sct}(A), hash(X))\\
&A : (X||sign_A)
\end{aligned}$$

Note 2, Lemma 8 and 9 prove that the first property $P1$ in DID authentication holds.

Note 2. *According to our hypotheses, a DID authentication request is eventually correctly received by at least one honest gateway node.*

Lemma 8. *If a DID authentication request is received by a gateway node, then the integrity of the DID authentication request can be verified by that gateway node.*

Proof: When a DID authentication request sent by a D2D node A is received by a gateway node G, that G can verify its integrity to determine if the request has been tampered with.

In order to verify the integrity of DID authentication request, G needs to obtain the public key of A and the corresponding verification method. Node G can first obtain the hash value of the DID document corresponding to A's DID by interacting with the smart contract according to the DID authentication process described in Section III. With this hash value, G can then proceed to obtain the complete DID document from the distributed storage system, and finally, to obtain the public key of A and the corresponding verification method through the DID document.

$$\begin{aligned}
&X = DID_A||DID_{req}\\
&A : (X||sign_A)\\
&I_G = \{..., X', sign'_A, ...\}\\
&I_{sys} = \{..., K_{pub}(sys), K_{sct}(sys), ...\}\\
&h'_{didDoc}(A) \leftarrow getDL(DID'_A)\\
&didDoc'_A||sign'_{sys}(didDoc'_A) \leftarrow getDS(h'_{didDoc}(A))\\
&sign'_{sys}(didDoc'_A) = E(K_{sct}(sys), hash(didDoc'_A))\\
&(K'_{pub}(A)||verMD') \leftarrow didDoc_A
\end{aligned}$$

where $getDL()$ and $getDS()$ are two functions from distributed system. These two functions can be invoked by a gateway to retrieve data from distributed ledgers via smart contract and distributed storage systems, respectively. $didDoc_A$ and $h_{didDoc}(A)$ are the DID document of the sender's DID and its hash value, respectively. $sign_{sys}(didDoc)$ is the signature of retrieved DID document, $didDoc$, signed by distributed storage system in step 5 of DID authentication.

Note that according to Note 1, as long as node A has completed DID registration, the stored information such as the public key, verification method and other metadata obtained from its DID document must be correct.

At this point, G has obtained the public key of A, the verification method, and the corresponding signature. Therefore according to Lemma 1, node G can verify the integrity of this DID authentication request.

■

Lemma 9. *Any correct DID authentication request will eventually be accepted by the DID service.*

Proof: Finally, according to Note 2 and Lemma 8, and similar to Lemma 3, in the case where A can try to retransmit, a DID authentication request that passes integrity verification will eventually be correctly received by at least one honest gateway node. These honest gateway node then performs DID authentication correctly. We say, at this point, the DID authentication requests has been received correctly by the DID service. The first property $P1$ therefore holds.

■

For the second property $P2$, when an honest gateway node G tries to reply to a DID authentication request, it first checks the distributed ledger and distributed storage system to see if the DID to be verified has been registered (through the getDL() and getDS() functions). Then, G sends the obtained complete DID document to node A along with the corresponding signature signed by the distributed storage system.

$$I_G = \{..., DID_{req}, DID_G, K_{pub}(G), K_{sct}(G), ...\}$$
$$h_{didDoc}(req) \leftarrow getDL(DID_{req})$$
$$didDoc_{req}||sign_{sys}(didDoc_{req}) \leftarrow getDS(h_{didDoc}(req))$$
$$sign_{sys}(didDoc_{req}) = E(K_{sct}(sys), hash(didDoc_{req}))$$
$$G - A : (didDoc_{req}||sign_{sys}(req))$$

And for malicious node Z, not only malicious gateway nodes, it can reply to X' like anything arbitrarily.

$$I_Z = \{..., X', K_{pub}(Z), K_{sct}(Z), ...\}$$
$$Z - A : (X')$$

Lemma 10. *When DID authentication request is accepted by DID service, at least one DID registration response from a honest gateway node will eventually be sent to the DID authentication requesting node.*

Proof: According to Lemma 9, we know that eventually the DID authentication request will be received correctly by at least one honest gateway node. And that honest gateway node must perform correctly the DID authentication process and try to reply to the requesting node correctly. During which other nodes may also try to reply to the requesting node with arbitrary responses.

Therefore according to our network model in Section IV-B, all these DID authentication responses, including the one from the honest gateway node, will eventually be received by the requesting node.

■

Lemma 11. *If a DID authentication response is received by the requesting node, then the integrity of the DID document contained in the DID authentication response can be verified by the requesting node.*

Proof: When a DID authentication response is finally received by the requesting node A according to Lemma 10, A can verify the integrity of the DID document in this response.

To verify the integrity of the content of DID authentication response, i.e., the DID document, the receiving node A only needs to know the public key to decrypt the signature of ID-document signed by the distributed storage system, $K_{pub}(sys)$. And according to Lemma 7, we know that any registered node will correctly receive a correct DID registration response, which contains the public key of the distributed storage system. Therefore, for any registered node, it knows $K_{pub}(sys)$.

$$I_A = \{..., didDoc_{req}, sign_{sys}(req), K_{pub}(sys), ...\}$$
$$K_{pub}(sys) \leftarrow I_A$$

At this point, the node A can verify the integrity of the received DID document according to Lemma 1.

■

Lemma 12. *Any node invoking DID authentication will eventually receive a correct DID authentication response.*

Proof: Similar to Lemma 7, according to Lemma 10 and 11, we know that a response sent by an honest gateway node will eventually be received by the requesting node. And the integrity of its content can be verified. Therefore, as long as the node A does not receive a response that the integrity of the content of that response has been verified, it can re-submit DID authentication requests. Until it receives a response that the integrity of the contents of that response is verified. Since the content is signed by the distributed storage system, once the integrity of the content in a response is verified, A can confirm that it has received the correct DID authentication response contenting the DID document of the DID to be verified.

■

Finally, according to Lemma 9 and 12, both two properties of DID authentication are proven to be correct and secure. Therefore, the DID authentication process is correct and secure.

D. Authentication with DID Service

At this point, we have shown that both DID registration and verification are correct and secure. Therefore, we can abstract these two processes into two functions, $didReg()$ and $didVer()$. A node can achieve DID registration and verification through these two functions.

Note that performing DID authentication does not, by itself, verify the integrity of a received message during the communication with others D2D node. Rather, it will return the information necessary to verify the integrity of a received message, i.e., the correct public key of the sender of the message, and the corresponding verification method.

The following example demonstrates that through DID service, nodes can authenticate and verify the integrity of communications without the need for third-party certificates.

Two nodes A and B that are expected to communicate with each other register their DIDs with the DID service:

$$A : didReg(K_{pub}(A)||verMD_A||metaData_A)$$
$$B : didReg(K_{pub}(B)||verMD_B||metaData_B)$$

Later A signs and sends its DID along with the contents of the message to B.

$$X = DID_A||data$$
$$sign_A = E(K_{sct}(A), hash(X))$$
$$A - B : (X||sign_A)$$

When B receives this message, it first extracts the DID of A from the message and then performs DID authentication process with this DID. By verifying the obtained DID document from corresponding DID authentication response,

B can finally obtain A's public key and its corresponding verification method.

$$DID_A \leftarrow X$$
$$didDoc_A = didVer(DID_A)$$
$$(K_{pub}(A)||verMD) \leftarrow didDoc_A$$

So far according to Lemma 1, the node B has obtained the public key of A, corresponding verification method, and the corresponding signature of A. Thus, it can verify the integrity of this message sent by A.

V. Performance Evaluation

The decentralized identity (DID) creation times were measured across two public blockchain configurations: Fantom Testnet, Etherlink Testnet. For private blockchains, we utilized the Hyperledger Besu client version 24.5.1, configured with two different consensus mechanisms: IBFT2.0 and QBFT. Each configuration operated with a block period of 2 seconds, deployed on two virtual machines (VMs) running Linux Ubuntu 22.04, with specifications of 8 vCPUs, 80 GB of storage, and 16 GB of memory. The data was plotted to provide a visual representation of the performance of each configuration, with average values indicated by dashed lines in distinct colours.

The implementation, deployment, and testing of the DID Proof of Concept (PoC) were carried out using an HP EliteBook 850 G8 Notebook laptop. The laptop was configured with Windows 11 Professional Version 23H2, an 11th Gen Intel(R) Core(TM) i7-1185G7 3.00GHz processor, and 64 GB of memory. Linux Ubuntu 24.04 was installed over the Windows Subsystem for Linux version 2.1.5.0. JavaScript, specifically npm 10.7.0 and Node.js 20.15.0, was chosen as the programming language for the APIs, while Solidity 0.8.0 was used for implementing and compiling the DID registry contract. Additional tools and versions used include express version 4.19.2, Docker version 27.0.3, IPFS Version 0.29.0, kubo 0.29.0-3f0947b, and Postman v10.24.26.

Based on the performance results for various signature algorithms, it is evident that both signing and verifying signatures of messages incur very low time costs, thereby having a minimal impact on the overall time required to create a Decentralized Identifier (DID). For instance, using the 11th Gen Intel(R) Core(TM) i7-1185G7 3.00GHz processor, the time to sign a message with ECDSA is approximately 596.98 microseconds, while verification takes about 695.83 microseconds. Even more efficient algorithms, such as the one discussed in the paper [34], show signing times of around 29.18 microseconds and verification times of about 44.67 microseconds in batch mode on the same processor. These extremely short durations demonstrate that the computational overhead of cryptographic operations for signatures is negligible, ensuring that the process of creating and managing DIDs remains efficient and responsive. This underscores that modern processors, like the Intel i7-1185G7, further reduce the significance of cryptographic computation times, thereby facilitating the swift creation and verification of DIDs without any noticeable delays.

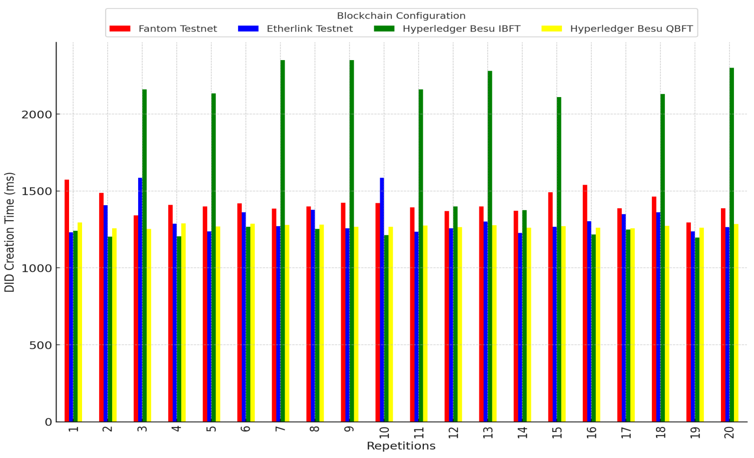

Fig. 4. DID Creation Time

The chart 4 illustrates the DID creation times for each blockchain configuration across multiple instances. The configurations are represented as follows: Fantom Testnet: Red, Etherlink Testnet: Blue, Hyperledger Besu IBFT: Green, Hyperledger Besu QBFT: Yellow. Each configuration's average DID creation time is also shown 5 in the corresponding colour.

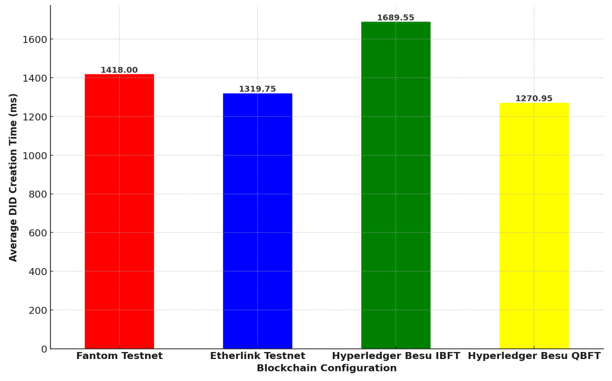

Fig. 5. DID Average Creation Time per Blockchain

A. Observations

1) Fantom Testnet: The DID creation time for the Fantom Testnet varied between approximately 1340 ms and 1573 ms. The average DID creation time is about 1442 ms. The red bar shows relatively consistent performance with moderate variability.

2) Etherlink Testnet: The DID creation time for the Etherlink Testnet ranged from around 1231 ms to 1586 ms. The average DID creation time is approximately 1341 ms. The blue bar also indicates consistent performance with a similar range of variability as the Fantom Testnet.

3) Hyperledger Besu IBFT 2.0: The DID creation time for Hyperledger Besu IBFT exhibited more variability, ranging from 1203 ms to 2160 ms. The average DID creation time is about 1428 ms. The green bar shows some fluctuations, indicating periods of higher latency.

4) Hyperledger Besu QBFT: The DID creation time for Hyperledger Besu QBFT was more stable, ranging from 1253 ms to 1295 ms. The average DID creation time is approximately 1272 ms. The yellow bar indicates the most consistent performance with the least variability among the configurations.

B. Discussion

The analysis reveals several key insights into the performance of different blockchain configurations in terms of DID creation time:

- **Consistency and Stability**: Hyperledger Besu QBFT demonstrated the most consistent performance with minimal variability, making it a potentially reliable choice for applications requiring predictable DID creation times. Fantom Testnet and Etherlink Testnet also showed good performance with moderate variability, suitable for scenarios where slight variations in DID creation time are acceptable.
- **Variability**: Hyperledger Besu IBFT exhibited the most variability in DID creation times, which may impact applications requiring consistent performance.
- **Average Performance**: Hyperledger Besu QBFT had the lowest average DID creation time, followed closely by Etherlink Testnet, making them efficient options for rapid DID creation.

This comparative analysis of DID creation times across four blockchain configurations provides valuable insights into their performance characteristics. Hyperledger Besu QBFT stands out for its consistency, while Etherlink Testnet offers the lowest average creation time. These findings can guide the selection of blockchain platforms for implementing decentralized identity systems, depending on the specific performance requirements of the application. Further research could explore the scalability and resilience of these configurations under varying network conditions.

VI. Conclusion

We propose a secure distributed identity service dedicated to device-to-device networks. The security features of our service combines classical cryptographical tools with the recent blockchain technology. The originality of our work steams in proposing formal security proofs for the entire workflow of our service. It should be noted that our DID service is tolerant to adversaries capable of performing adaptive chosen message attacks and therefore it is tolerant to classical attacks on the communication system such as *Authorization violation*, *Eavesdropping*, *Masquerade and forgery* and *Modification attack* (see [31] and [32] for more details). Moreover our solution is resilient to *Replay Attacks* since the replay of any DID registration or DID verification request / response does not have any impact on the DID registration or DID authentication process, except for a DoS-like physical attack.

Our DID service can be used in the implementation of other secure services such as localization, clustering or routing. Our future work aims at designing of a secure middleware orchestrating these services.

References

[1] C. Lundkvist, R. Heck, J. Torstensson, Z. Mitton, and M. Sena. (2016). Uport: A Platform for Self Sovereign Identity.

[2] A. Tobin and D. Reed, "The inevitable rise of self-sovereign identity," Sovrin Found., vol. 29, no. 2016, p. 18, 2016.

[3] Travel Identity of the Future, ShoCard, Cupertino, CA, USA, 2016.

[4] D. van Bokkem, R. Hageman, G.Koning, L. Nguyen, and N. Zarin," Self-sovereign identity solutions: The necessity of blockchain technology," 2019, arXiv:1904.12816.

[5] A. E. Panait, R. F. Olimid, and A. Stefanescu, "Identity management on blockchain-Privacy and security aspects," in Proc. Romanian Acad. A, Math. Phys. Tech. Sci. Inf. Sci., 2020, vol. 21, no. 1, pp. 45-52.

[6] S. Y. Lim, P. T. Fotsing, A. Almasri, O. Musa, M. L. M. Kiah, T. F. Ang, and R. Ismail, "Blockchain technology the identity management and authentication service disruptor: A survey," Int. J. Adv. Sci. Eng. Inf. Tech., vol. 8, pp. 1735-1745, Sep. 2018.

[7] M. Takemiya and B. Vanieiev, "Sora identity: Secure, digital identity on the blockchain," in Proc. IEEE 42nd Annu. Comput. Softw. Appl. Conf., vol. 2, Oct. 2018, pp. 582-587.

[8] O. Dib and K. Toumi, "Decentralized identity systems: Architecture, challenges, solutions and future directions," Ann. Emerg. Technol. Comput., vol. 4, no. 5, pp. 19-40, Dec. 2020.

[9] M. Shuaib, N. H. Hassan, S. Usman, S. Alam, S. Bhatia, A. Mashat, A. Kumar, and M. Kumar, "Self-sovereign identity solution for blockchain-based land registry system: A comparison," Mobile Inf. Syst., vol. 2022, pp. 1-1117, Apr. 2022.

[10] M. Ali, J. Nelson, R. Shea, and M. J. Freedman, "Blockstack: A global naming and storage system secured by blockchains," in Proc. USENIX Annu. Tech. Conf. (USENIX ATC), 2016, pp. 181-194.

[11] S. Y. Lim, P. T. Fotsing, A. Almasri, O. Musa, M. L. M. Kiah, T. F. Ang, and R. Ismail, "Blockchain technology the identity management and authentication service disruptor: A survey," Int. J. Adv. Sci. Eng. Inf. Tech., vol. 8, pp. 1735-1745, Sep. 2018.

[12] A. Poikola, K. Kuikkaniemi, and H. Honko, "MyData-A Nordic Model for human-centered personal data management and processing," Work. Paper, 2015.

[13] A. Giaretta, S. Pepe, and N. Dragoni, "UniquID: A quest to reconcile identity access management and the Internet of Things," in Proc. Int. Conf. Objects, Compon., Models Patterns, 2019, pp. 237-251.

[14] M. T. Quasim, M. A. Khan, F. Algarni, A. Alharthy, and G. M. M. Alshmrani, Blockchain Frameworks. Cham, Switzerland: Springer, Mar. 2020.

[15] Houtan, B., Hafid, A. S., & Makrakis, D. (2020). A survey on blockchain-based self-sovereign patient identity in healthcare. IEEE Access, 8, 90478-90494.

[16] Shuaib, M., Alam, S., Alam, M. S., & Nasir, M. S. (2021). Self-sovereign identity for healthcare using blockchain. Materials Today: Proceedings.

[17] Gordon, W. J., & Catalini, C. (2018). Blockchain technology for healthcare: facilitating the transition to patient-driven interoperability. Computational and structural biotechnology journal, 16, 224-230.

[18] Soltani, R., Nguyen, U. T., & An, A. (2021). A survey of self-sovereign identity ecosystem. Security and Communication Networks, 2021, 1-26.

[19] Liu, Y., He, D., Obaidat, M. S., Kumar, N., Khan, M. K., & Choo, K. K. R. (2020). Blockchain-based identity management systems: A review. Journal of network and computer applications, 166, 102731.

[20] Kuperberg, M. (2019). Blockchain-based identity management: A survey from the enterprise and ecosystem perspective. IEEE Transactions on Engineering Management, 67(4), 1008-1027.

[21] Gilani, K., Bertin, E., Hatin, J., & Crespi, N. (2020, September). A survey on blockchain-based identity management and decentralized privacy for personal data. In 2020 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS) (pp. 97-101). IEEE.

[22] Mühle, A., Grüner, A., Gayvoronskaya, T., & Meinel, C. (2018). A survey on essential components of a self-sovereign identity. Computer Science Review, 30, 80-86.

[23] Lim, S. Y., Fotsing, P. T., Almasri, A., Musa, O., Kiah, M. L. M., Ang, T. F., & Ismail, R. (2018). Blockchain technology the identity management and authentication service disruptor: a survey. International Journal on Advanced Science, Engineering and Information Technology, 8(4-2), 1735-1745.

[24] Zhu, X., & Badr, Y. (2018). Identity management systems for the internet of things: a survey towards blockchain solutions. Sensors, 18(12), 4215.

[25] Rathee, T., & Singh, P. (2022). A systematic literature mapping on secure identity management using blockchain technology. Journal of King Saud University-Computer and Information Sciences, 34(8), 5782-5796.

[26] G. Fedrecheski, J. M. Rabaey, L. C. P. Costa, P. C. Calcina Ccori, W. T. Pereira and M. K. Zuffo, ”Self-Sovereign Identity for IoT environments: A Perspective,” 2020 Global Internet of Things Summit (GIoTS), Dublin, Ireland, 2020, pp. 1-6, doi: 10.1109/GIOTS49054.2020.9119664.

[27] Fedrecheski, G., Costa, L. C., Afzal, S., Rabaey, J. M., Lopes, R. D., & Zuffo, M. K. (2022). A low-overhead approach for self-sovereign identity in IoT. In Global IoT Summit (pp. 265-276). Cham: Springer International Publishing.

[28] Kulabukhova, N., Ivashchenko, A., Tipikin, I., & Minin, I. (2019). Self-sovereign identity for iot devices. In Computational Science and Its Applications–ICCSA 2019: 19th International Conference, Saint Petersburg, Russia, July 1–4, 2019, Proceedings, Part II 19 (pp. 472-484). Springer International Publishing.

[29] Dolev, D., & Yao, A. (1983). On the security of public key protocols. IEEE Transactions on information theory, 29(2), 198-208.

[30] Jonathan Katz and Yehuda Lindell. 2014. Introduction to Modern Cryptography, Second Edition (2nd. ed.). Chapman & Hall/CRC.

[31] Stallings, W., & Brown, L. (2015). Computer security: principles and practice. Pearson.

[32] Security Architecture for Open Systems Interconnection (OSI) for CCITT Applications, T. S. S. International Telecommunication Union, Geneva, Switzerland, 1991.

[33] B. G. Kim, Y. -S. Cho, S. -H. Kim, H. Kim & S. S. Woo, ”A Security Analysis of Blockchain-Based Did Services,” in IEEE Access, vol. 9, pp. 22894-22913, 2021, doi: 10.1109/ACCESS.2021.3054887.

[34] Bernstein, Daniel J., Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. ”High-speed high-security signatures.” Journal of cryptographic engineering 2, no. 2 (2012): 77-89.