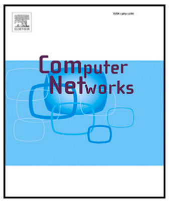
Computer Networks 219 (2022) 109434

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



An offline mobile access control system based on self-sovereign identity standards

Alexander Enge, Abylay Satybaldy *, Mariusz Nowostawski

Department of Computer Science, NTNU, Gjovik, Norway

ARTICLE INFO

Keywords:
Self-sovereign identity
Secure communication
Access control

ABSTRACT

Self-sovereign identity (SSI) is a new paradigm to digital identity management that is built on decentralized technologies and can exist without centralized third-parties for managing the identity data. Within the SSI model, a digital identity wallet enables a user to establish relationships and interact with third parties in a secure and trusted manner. At present, the existing SSI solutions rely on the Internet connection for carrying out the necessary operations such as messaging and credential verification. However, there are many places the Internet may not be accessible and other means for communication is needed. The objective of this paper is to design a proof-of-concept that would allow for secure, trustworthy, and decentralized peer-to-peer communication without the need for any external networking infrastructure. For this, we investigate a particular case involving DIDComm and Bluetooth Low Energy (BLE). We identify requirements for the architecture and propose an architectural framework that allows two entities to securely communicate and exchange verifiable credentials. Furthermore, we look at a specific use case, namely, how offline access control can be enabled within SSI between two mobile devices. We present and evaluate the implementation of offline access control system based on the proposed architecture. Through this research, and experimentation we can conclude that this approach has the potential to enable a wide range of interesting use cases and can be integrated into existing digital identity wallet solutions to extend the capabilities of offline messaging in a secure and decentralized manner that goes beyond the current models that rely on the Internet connectivity.

1. Introduction

Wireless technology has become an integral aspect of everyday life, with Wi-Fi and Bluetooth being two of the most extensively utilized wireless communication technologies. Bluetooth is included in the vast majority of today's smartphones, smartwatches, bracelets and other portable devices. Furthermore, the Internet of Things (IoT) is developing rapidly, and the Bluetooth Low Energy (BLE) protocol has proven to be an effective option for seamlessly communicating between low-power devices [1]. The number of communicating devices is predicted to grow in conjunction with the Internet of Things.

The ability to communicate securely among peers is an essential component of any communication system. Individuals or organizations nowadays frequently rely on some central authority or intermediate third party to build a trusted and secure communication route between unrelated devices. The issue is that establishing a trusted and secure communication channel between unrelated individuals or organizations is often difficult with today's methods. Despite the development of various messaging systems for delivering peer-to-peer communications, most solutions rely on a centralized identity management system for device authentication and operate as silos or federations, preventing interoperability across many current and future systems.

Self-sovereign identity (SSI) is a new paradigm that promises a more decentralized approach compared to existing identity systems. In this new model, individuals take ownership of their digital identities and have full control over their personal information. This information is typically carried around by the user in the digital identity wallet on their mobile device. A digital wallet, in the context of SSI, is a software application and encrypted database that stores credentials, keys, and other secrets necessary for self-sovereign identity management to operate. The model enables the user to have data sovereignty and complete control as well as data portability. A user can establish relationships and interact with third parties in a trusted manner.

One of the principles of SSI is to allow people to interact in the digital world with the same freedom and capacity for trust as they do in the physical/offline world. There is a broad interest from many individuals and organizations around the world in the area of decentralized identity systems and there exist many proposals and contributions [2–7]. However, the use of offline communication in SSI without the need

* Corresponding author.
E-mail address: abylay.satybaldy@ntnu.no (A. Satybaldy).

https://doi.org/10.1016/j.comnet.2022.109434
Received 17 August 2022; Received in revised form 15 October 2022; Accepted 24 October 2022
Available online 28 October 2022

for the Internet access is largely unexplored, especially since many existing SSI implementations depend on an Internet connection for their operations and sending messages between parties, and there is little research on how SSI could be applied in an offline setting without the need for external infrastructure.

There are many situations where we need to be able to exchange messages or prove our identity without having the Internet access. For example, the vast majority of Africa does not have access to the Internet [8], in conflict areas or after natural disasters there might be no Internet connectivity and in some use-cases, e.g. large gatherings such as concerts or ad-hoc demonstrations may not have sufficient Internet infrastructure to support the communication needs. There are also possible scenarios where you might need to verify a digital credential stored in your wallet in the remote locations where the Internet access is simply not available at all.

This research is primarily motivated by the new technological advancements in the fields of self-sovereign identity and decentralized technology. Self-sovereign identity and supporting technologies have enabled opportunities that were previously not possible to achieve. It is therefore interesting to explore how self-sovereign identity could be deployed in an offline scenario to exchange messages between digital wallets without requiring an active Internet connection.

As a result, the topic we want to investigate in this paper is offline messaging and mobile access control in the context of self-sovereign identity. This study is concerned with a specific feature of self-sovereign identity, particularly, how credentials can be transferred wirelessly and verified with another party without relying on any other network infrastructure, specifically, without the Internet connectivity.

Our proposal is based on the open SSI standards such as Decentralized Identifiers (DIDs) [9] and Verifiable Credentials (VCs) [10]. A shared protocol is needed for two entities to create a connection and communicate with one another. With the rise of DIDComm [11], a new transport-agnostic messaging protocol based on the concept of decentralized identifiers that enables secure, trustworthy, and decentralized communication between agents in a self-sovereign identity ecosystem, some interesting alternatives to existing systems are now possible. Because this protocol is transport-agnostic, it can use BLE to transfer the data without depending on a certain transport-specific protocol. Furthermore, BLE is already supported by billions of devices, making it a suitable candidate for transmitting data between devices. Therefore, integrating these two technologies for transmitting messages can be desirable in certain situations, particularly BLE combined with DIDComm can provide a secure manner of sharing information directly between peers in close proximity.

The main contributions of this paper is the proposal of the architectural design and development of a proof-of-concept application for a direct peer-to-peer communication system that is based on the self-sovereign identity standards and does not rely on any centralized servers or external infrastructure. The proof-of-concept consists of a mobile application (digital wallet) that enables interoperable and secure DIDComm BLE messaging between two devices, and an Android library that enables two devices to send DIDComm messages to each other. A mobile application and the library have been created in order to issue, present and verify credentials over BLE via DIDComm messaging protocol using Presentation Exchange protocol [12] to demonstrate the mobile access control use case.

This work is an extended version of our paper, presented at IEEE COMPSAC conference [13], where we proposed an architectural design for offline peer-to-peer messaging between two devices. In this paper, we significantly expanded the design by integrating verifiable credentials on top of the protocol and developed and evaluated a proof-of-concept application for the mobile access control use case. The performance, security, and interoperability aspects of the application are evaluated and discussed. The paper concludes with valuable insights into the limitations and capabilities of the system. This work should provide a better understanding of how BLE credential exchange works and whether such a method is ready for production use in current SSI systems.

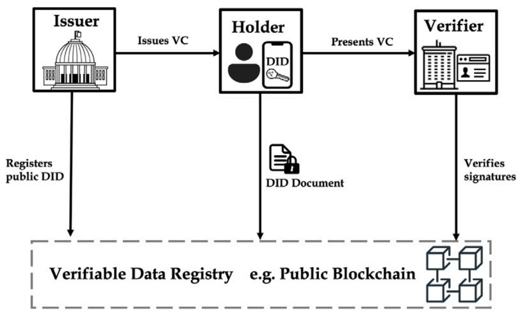


Fig. 1. Self-Sovereign Identity Architecture.

2. Background

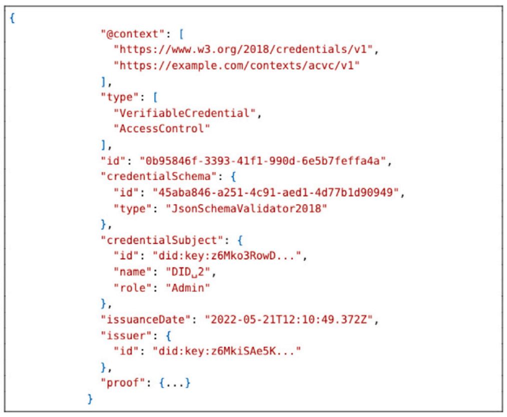
In this section, we will provide the background knowledge on the main concepts and technologies that are needed to understand the work.

2.1. Self-sovereign identity and decentralized technologies

The foundation concepts of SSI were officially brought to life when the Credentials Community Group was created under the umbrella of the international organization, the World Wide Web Consortium (W3C). The group defined two fundamental standards for the development of a new decentralized identity architecture: Decentralized Identifiers (DIDs) [9] and Verifiable Credentials (VCs) [10]. Decentralized identifiers are a new type of identifier that is decentralized, resolvable and cryptographically verifiable. A DID has the format of an URI scheme and is in the form `did:<DID method>:<method-specific identifier>`. The DID method describes how the DID is resolved into its DID document and the CRUD actions[1] are done on the document. This document contains information associated with a DID, including cryptographic public keys, authentication suites, and service endpoints. The data format of DID documents can be serialized as JSON or JSON-LD (JavaScript Object Notation for Linked Data) [14]. In order to be resolvable to DID documents, DIDs are typically recorded on an underlying system or network of some kind. Regardless of the specific technology used, any such system that supports recording DIDs and returning data necessary to produce DID documents is called a *verifiable data registry* as shown in Fig. 1. Examples include distributed ledgers, decentralized file systems, various database systems, peer-to-peer networks, and other forms of trusted data storage. This design eliminates dependence on centralized registries for identifiers as well as centralized certificate authorities for key management — which are the standard patterns in hierarchical PKI (Public Key Infrastructure). Each identity owner may serve as its own root authority — an architecture referred to as DPKI (Decentralized PKI).

To date, various DID methods have been created and implemented for different use cases [15]. A `did:key` [16] and `did:peer` [17] are examples of peer DID methods that can be used independent of any central source of truth and can be resolved off-chain. They are suitable for most private relationships between people, organizations, and things. The specification for `did:peer` method is maintained by the Decentralized Identity Foundation (DIF) [18]. A simpler version of peer DID is the `did:key`, which encodes the public key directly into the identifier string to generate the DID document in a deterministic process. The `did:peer` can be seen as more versatile than `did:key` since they enable extra features such as defining service endpoints for communication and CRUD activities to alter the document, key rotation

[1] Create, read, update, and delete (CRUD) are the four basic operations of persistent storage.



```
{
            "@context": [
              "https://www.w3.org/2018/credentials/v1",
              "https://example.com/contexts/acvc/v1"
            ],
            "type": [
              "VerifiableCredential",
              "AccessControl"
            ],
            "id": "0b95846f-3393-41f1-990d-6e5b7feffa4a",
            "credentialSchema": {
              "id": "45aba846-a251-4c91-aed1-4d77b1d90949",
              "type": "JsonSchemaValidator2018"
            },
            "credentialSubject": {
              "id": "did:key:z6Mko3RowD...",
              "name": "DID␣2",
              "role": "Admin"
            },
            "issuanceDate": "2022-05-21T12:10:49.372Z",
            "issuer": {
              "id": "did:key:z6MkiSAe5K..."
            },
            "proof": {...}
          }
```

Fig. 2. Example of a Verifiable Credential in JSON-LD format.

without having to replace the DID. It is essential to select a method that supports peer-to-peer systems that do not rely on the Internet. As a result, we will rely on the did:peer and did:key in this project because they are built for peer-to-peer relationships and do not require any additional infrastructure. In this paper, we refer to `did:peer` and `did:key` as peer DIDs for simplicity sake as either can be used to establish peer connections.

A Verifiable Credential (VC) is made up of three main parts. The first is the credential metadata, which consists of information that describes the credential such as credential type, who issued the credential, when it was issued, and when it expires, as well as a context property that permits an agreed-upon understanding of the credential and its structure and can be processed by JSON-LD. Second, the credential can contain statements about the credential subject in the form of one or more claims expressed as property-value pairs in the credential. Last but not least, it contains proof(s) that enables the credential to be cryptographically verifiable using digital signatures. A verifiable credential can be serialized in JSON or JSON-LD, with the proof format being JSON Web Token (JWT) or Linked Data [19]. Fig. 2 shows an example of a VC in JSON-LD format. A Verifiable Presentation (VP) contains data that can be cryptographically verified and is commonly used to encapsulate one or more VCs. In addition, the proof on the VP is often used for authenticating the holder.

A digital wallet is the software or hardware that is responsible for securely storing identity data and cryptographic content. In the context of SSI solutions, this includes storing VCs, DIDs and the associated cryptographic keys. An agent acts on behalf of the user and can communicate with other agents to do various actions [20], it typically accesses the digital wallet for storing and retrieving information to perform cryptographic operations. Depending on the usage, these actions can be programmed to be executed automatically by the agent or manually by the user. Furthermore, the agent can operate on an edge device or in the cloud. When an agent communicates using the DIDComm messaging protocol, the agent can act according to three roles: relay, mediator, or edge agent. The relay and mediator's job is to send a message from one edge agent to another edge agent. A relay agent can be used to change the transport medium, whereas a mediator has more sophisticated logic and can also perform routing and cryptographic operations [21]. This approach, for example, allows a sender device to transmit a message using HTTP as the transport protocol by first sending it to the relay, which then sends it to the final destination device over BLE.

2.2. DIDComm

DIDComm is a novel DID-based asynchronous end-to-end encrypted communication and messaging protocol maintained by DIF. It creates a secure communication channel between agents controlled by entities, which can be people, organizations or things. This protocol has various distinguishing characteristics that set it apart from other secure communication protocols. As described in the specification, DIDComm is designed to be transport-agnostic, flexible, and secure. Several organizations have contributed to its implementation by releasing libraries in a variety of programming languages [4]. DIDComm is also used to establish the methodology and underlying basis for higher-level protocols to be implemented on top of, inheriting DIDComm's properties. The *presentation exchange* protocol, for example, allows a verifier to request credentials from a holder [12]. This protocol will be covered in greater depth later.

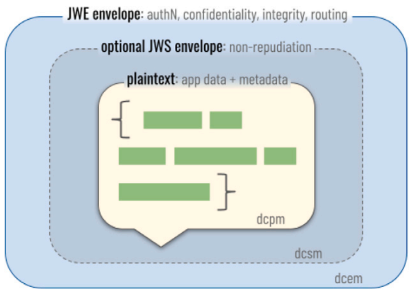

JWE envelope: authN, confidentiality, integrity, routing
optional JWS envelope: non-repudiation
plaintext: app data + metadata
dcpm
dcsm
dcem

Fig. 3. The three different DIDComm messages formats [11].

The DIDComm specification defines three message formats: DIDComm plaintext message, DIDComm signed message, and DIDComm encrypted message. The messages that are transmitted are typically encoded as JSON and follow the JSON Web Message (JWM) [22] specification. In comparison to an encrypted or signed message, plaintext does not guarantee security or integrity. The DIDComm signed message is a plaintext message with a digital signature that ensures message integrity and follows the JSON Web Signature (JWS) [23] format. The DIDComm encrypted message is a plaintext message that has been encrypted, following the JSON Web Encryption (JWE) [24] format, to prevent the content of the message from being revealed to unauthorized parties and provide integrity guarantees. Fig. 3 depicts the three different DIDComm message formats.

DIDComm supports a variety of encryption algorithms that relies on a subset of the JSON Web Algorithms (JWA) [25] defined algorithms for the purpose of interoperability. It employs Public Key Authenticated Encryption for JOSE (ECDH-1PU) algorithm [26] for authenticated sender encryption (*authcrypt*) and Key Agreement with Elliptic Curve Diffie–Hellman Ephemeral Static (ECDH-ES) algorithm [27] for anonymous sender encryption (*anoncrypt*). Both types provide end-to-end encryption (E2EE) of messages to the sender, but only the authcrypt achieves sender authentication.

Self-sovereign identity standards and DIDComm have been used in a variety of applications, including decentralized factorization [28], a learning framework for healthcare [29], federated machine learning tasks [30], and vehicle identity management [31]. In addition, the Sovrin Foundation SSI in IoT Task Force [32] has looked into business opportunities and IoT use cases in the context of SSI. They exemplify the use of peer DIDs and DIDComm with HTTPS as the transport layer in the paper.

2.3. Bluetooth Low Energy

In 2010, the Bluetooth Special Interest Group (SIG) officially announced the Bluetooth 4.0 specifications along with the Bluetooth Low Energy (BLE) protocol [33]. BLE is a low-power wireless communication technology that is based on the Bluetooth Classic protocol. It operates at 2.4 GHz and is commonly used in IoT and low-powered devices.

According to the Bluetooth specifications, two BLE devices can communicate using one of two interaction patterns. The advertisement and observer roles are defined in the first mode. This mode does not require the devices to connect; instead, the advertiser broadcasts a message, which the observers can retrieve by listening to the advertisement channel.

In the other mode, the two devices can establish a connection with each other. Depending on who initiates the connection with the other device, a device can represent one of two roles: peripheral or central. The peripheral is responsible for advertising itself to other devices. It does so by periodically sending out advertisement packets at regular intervals. The central then scans for these advertisement packets to find available peripherals that it may initiate a connection with. The communication channel can be secured and encrypted by pairing the devices. The data packets are encrypted with Advanced Encryption Standard (AES), and while this encryption is considered secure, attackers can exploit different BLE vulnerabilities during key exchange [34].

BLE defines two profiles that an application will interact with to perform operations. These are the Generic Access Profile (GAP) and Generic Attribute Profile (GATT). The GAP essentially defines the parameters required to enable a connection or pairing with another device. While GATT defines the attributes used for transmitting data between devices once the connection is established. Profiles define the set of services and characteristics that the device supports. The Maximum Transmission Unit (MTU) is the maximum amount of data that can be transmitted at once in a packet.

A universal unique identifier (UUID) is a unique number used to distinguish between all the attributes. These UUIDs are communicated by the peripheral in order for the central to know which services are available. A UUID can be either public (16 bit) or vendor specific (128 bit). The Bluetooth standards describe the public UUIDs, whereas the vendor specifics are custom UUIDs for services defined by the various vendors. A 6-byte address (BD ADDR) is used to uniquely identify a BLE device. To avoid tracking, the device can utilize random addresses that are either static (do not change) or private (change on a regular basis). Paired devices that support Bluetooth V4.2 and higher can resolve the private address on the link layer.

3. Related work

Existing research into offline communication and credential verification in a self-sovereign identity ecosystem is rather limited and under-explored.

Authentication and authorization using DIDs and VCs has been studied by several researchers. Lux et al. [35] combine OpenID Connect (OIDC) with VCs using a scheme that adheres to the OIDC specification's claims. This technique enables user login by presenting verifiable credentials to the OIDC provider, who authenticates the user. Fotiou et al. [36], on the other hand, proposed a capability-based access control technique that employed verifiable credentials as an access token to access protected resources using the OAuth 2.0 authorization flow.

Lagutin et al. [37] proposed a method where they moved the computational power required for the cryptographic operations on DIDs and VCs. Instead of performing the operations on the IoT device itself, it is moved to an external OAuth authorization server that then authenticates the user on behalf of the IoT device. This is achieved by proving to the authorization server that the user has a credential that it trusts. After that, the server generates an access token that is given to the user and can be shown to the IoT device in order to gain access. An SSI-based access control based on the attribute-based model and the use of privacy-preserving techniques including selective disclosure and range proofs is proposed by Belchior et al. [38]. Their approach relies on blockchain technology for authentication and does not address offline access control.

Abraham et al. [3] proposed and evaluated a method for achieving revocation and verification of VCs without requiring an active Internet connection. This works by continuously generating new timestamped access tokens against a credential revocation registry in order to attest the validity of a credential. These access tokens are used to demonstrate to an offline verifier that the credential was not revoked at the time of token attestation and is still valid. The verifier can then utilize this information to see if the period between presenting it and attestation is within the permitted range. A trusted network of nodes signs the access tokens using a multi-signature mechanism. When offline, verifiers can assess the validity of the multi-signature using a trust store, which contains a list of all the trusted nodes' public keys signed by the network. Their research is focused on the mechanisms involved in offline credential revocation, as well as the evaluation of the credential exchange and verification stage.

Fedrecheski et al. [39] argue that self-sovereign identity can significantly enhance the security and privacy of IoT. In addition, they highlight issues that constrained devices must overcome related to the required computation power for asymmetric cryptography processing, data overhead, DID resolution without Internet access, traceability, and limited software for IoT devices. As a solution to resolving DID without Internet access they suggest using a local cache of trusted DIDs that is managed by the device itself or a gateway. Alternatively, peer DIDs could be securely exchanged and used instead. Their study compares several approaches to digital identity, focusing on data models, and discusses the benefits and challenges of SSI in the IoT.

Bartolomeu et al. [40] studies SSI in the context of industrial IoT. They mention that it has many use cases, but still faces technical difficulties related to lack of widespread adoption and immaturity. Furthermore, Grabatin et al. [41] looks into how devices can be managed in a wireless ad-hoc mesh network (WANET) based on LoRa by using SSI. Their solution was benchmarked in terms of data overhead, time on-air, computation time and power consumption. Although some of these are more difficult to quantify precisely than others, whilst others can be calculated directly, they came to the conclusion that the $secp192k1$ elliptic curve algorithm provides an acceptable overhead in real-world mesh networks. For being interoperable at the communication protocol level, they plan to implement DIDComm into their solution.

Cäsar et al. [42] conducted a survey on BLE in terms of security and privacy, considering the various versions and providing an overview of known weaknesses and attacks. Barua et al. [1] provides a comprehensive survey of security and privacy threats of BLE devices in the IoT environment. It includes a threat model and discusses the various attacks and mitigation strategies.

Furthermore, Tosi et al. [43] provides a comprehensive review of the Bluetooth Low Energy protocol and its performance. It looks into throughput, maximum sensors that can be connected, power consumption, latency and maximum range. They found that although the theoretical limit of data throughput in BLE is around 230 kbps, the analyzed applications reached a limit of around 100 kbps. Also, the maximum number of sensors connected is usually lower than 10, but is dependent on the connection parameters, network architecture and the device characteristics. Both latency and power consumption is largely dependent on many different factors. Lastly, they found that the maximum range is up to a few ten meters, but depends on radio power.

In January 2020, Albrecht et al. [44] conducted a security analysis of a popular end-to-end encrypted messaging platform called Bridgefy. The platform relies on Bluetooth amongst other technologies for sending messages between peers. The analysis revealed that it offered no authentication mechanism, poor confidentiality protections and users could be tracked by constructing social graphs. This was made possible because both the identifiers for sender and receiver are sent across the mesh network in plain-text. It supports direct one-to-one messaging over a BLE connection or using a Bluetooth mesh network. When sending messages over the mesh network, it can either be sent to a specific receiver or broadcasted to all nearby users within range. If both parties are online, it can also be sent over the Internet. In response to their research, the developers of Bridgefy switched to Signal protocol in October 2020 which is a secure instant messaging protocol.

Schoolfield [45] implemented a proof-of-concept messaging application for sending authenticated and encrypted messages over a BLE connection. They provided implementation library and application for Android. A performance test was conducted on the solution and a transfer speed of 1 KB/s was achieved when sending a file between two BLE connected Android devices.

A draft specification for using BLE as the transport layer for DIDComm has been released by DIF [46]. The specification is not complete yet and lacks technical details. It provides recommendations such as using an agreed-upon service UUID during advertising to identify devices that support the DIDComm protocol which we followed in our work.

Although the aforementioned studies cover techniques and evaluation for sending messages over BLE, they do not address some of the concerns related to self-sovereign identity and offline access control. This work seeks to fill in some of the missing gaps identified in the previous works above. It offers a proof-of-concept for an offline access control system that details each step of the message exchange process and empirically evaluates the results. Aspects such as feasibility, interoperability, and security are also discussed.

4. Requirements

In this section we start with an overview of the use case that will be investigated, followed by the identified requirements for the system. For mapping out the requirement and to better understand the requirements of the system, relevant specifications, documentations and papers have been reviewed. Moreover, the self-sovereign identity properties are taken into account when making decisions about the requirements. It is also influenced by requirements identified in other existing SSI solutions and Bluetooth messaging solutions. Lastly, the requirements and application was developed over several iterations following an evolutionary prototyping approach [47].

4.1. Use case

A use case is presented in order to better understand the system's requirements. For this, a common scenario for IoT devices will be employed, which is to perform authentication and authorization [48]. The case under study in this project will look more closely at how two devices can perform offline authentication and authorization using SSI principles and the enabling technologies discussed in the background section. Moreover, the goal of this use case is to evaluate and demonstrate how devices can use SSI in a peer-to-peer fashion to perform messaging and access control in a secure and trustworthy manner using the designed architecture without relying on external infrastructure, proprietary software, a single service provider, centralized servers, or federation. Furthermore, unless otherwise specified, it is assumed that BLE is used to transfer messages between devices, that the devices do not have active Internet access when performing message and credential exchange, authentication, or authorization, and that all data is stored off-chain and locally on the peer devices. The messaging between the devices and the access control system are two elements of the case covered in this paper.

First, the system must facilitate messaging between two mobile devices without requiring the Internet access. It involves both sending and receiving a message from device A to device B. The BLE protocol will be used for this, as already mentioned, this technology is widely used to enable wireless communication between IoT devices. Although there are other end-to-end encrypted secure messaging protocols [49], the protocol that will be employed here is DIDComm, which is commonly used in existing SSI frameworks for agent-to-agent communication.

For sending a message from one device to another, two roles are defined: sender and recipient. The sender is responsible for creating and encrypting messages for the intended recipient. The sender then establishes the connection with the recipient device and begins sending the message. When the message has been entirely transferred, the message receiver can decrypt and read it. The mobile access control system that will be demonstrated will leverage the messaging capabilities described here to securely send and receive messages via BLE.

Access control system is defined by the National Institute of Standards and Technology (NIST) as "a set of procedures and/or processes, normally automated, which allows access to a controlled area or to information to be controlled, in accordance with pre-established policies and rules" [50].

There are various access control models for presenting access control policies, and one common model is the role-based access control model (RBAC), in which the permissions granted to a subject are determined by the assigned role [51]. In this use case we will focus on the RBAC model. Because a VC can be used to make verifiable claims about a subject, we can use these credentials to assign a particular role to a subject. For example a subject of a VC can have the role of an administrator giving permissions to enter restricted areas that non-administrators should not have access to.

The case under study will use the Decentralized Identifiers and Verifiable Credential data model. As seen by several papers [35–37], these standards have been used in existing authentication and authorization protocols. The following is a brief description of how access control can be accomplished among the three roles (issuer, holder and verifier). The issuer is responsible for issuing VC that include the rights provided in the form of claims. The holder responsibility is to hold the VC obtained from the issuer and present them to the verifier upon request during access control. After that, the verifier can verify the VC to see if the holder has the necessary permissions to gain access.

The aim of the access control system is to give users access that is consistent with the access policies. The system is extended to show how to use SSI for access control when using BLE as the transport layer under the established assumptions. This access control process is divided into four steps, each of which is discussed in detail in Section 5 about system architecture and design.

4.2. Functional requirements

This section outlines the functional requirements of the system. They are divided into two parts: messaging and access control. The messaging part focuses on the setup of connections and how messages are securely transported using BLE, whilst the access control part will make use of the messaging capabilities to build an access control system.

Messaging.

- A device should be able to define a service endpoint in the DID document that other devices can use to discover the BLE device associated with this DID.
- A device should be able to manually or automatically initiate and establish a BLE connection with another device.
- Allow for the pairing of two BLE devices before sending messages.
- Allow two devices to agree on common connection parameters and MTU before sending messages.
- A device should support both the peripheral and central BLE roles.
- A device should have the ability to both send and receive messages over BLE.
- There should be no constraints on the message size transferred (i.e., large files or messages can be sent).
- Allow two devices to securely establish peer-to-peer connection that will be used for encrypted communications and application-layer encryption.
- A device should be able to display and use peer-to-peer connections that have already been established for future interactions with that peer.

- Support for the three types of DIDComm message formats (none, anoncrypt, authcrypt).
- Any of the three DIDComm message formats can be sent and received over a BLE.
- A device should be able to display messages that have been retrieved and sent in a chat.

Access control.

- Able to create and issue verifiable credentials and presentations.
- Able to request proofs from another device in a standard format.
- Able to verify the proofs on verifiable credentials and presentations.
- Able to display verifiable credentials and presentations that have been received and issued.
- Grant or deny access for a device based on the defined access policies, the role the device is assigned, and the outcome of the verification of the verifiable credential.

4.3. Non-functional requirements

The section describes the non-functional requirements that the system must satisfy. It should also align with properties of SSI and the performance that BLE can provide.

- The system must be extendable, interoperable, and capable of running on a wide range of platforms.
- The system must provide high availability and be able to handle any errors that occur in a way that the system does not crash.
- The system must be decentralized and work independently of any third-party, central authority or external infrastructure.
- The system must support offline access.
- The system must provide security and privacy-preserving features that adhere to privacy regulations.
- The system must be efficient and provide minimal latency when performing the access control procedure and transferring messages.

5. System architecture

In this section, we first present an overview of the proposed offline mobile access control system and subsequently give a detailed description of the individual components of the designed architecture and protocols. The proposed architecture is based on the use case and the identified requirements presented in Section 4. To adhere to the separation of concerns principle, the design takes a modular approach.

5.1. Components

A high-level overview of the system is depicted in Fig. 4. The architecture is composed of multiple components that work together. It displays two mobile devices, using an application, that communicate with each other in a peer-to-peer manner over BLE. The application consists of five main components: (1) Application view, (2) Services, (3) Edge agent and wallet, (4) DIDComm BLE library, (5) BLE Android API. The DIDComm BLE library is responsible for handling the BLE connections and transport of DIDComm messages at the lower layers. This library uses the BLE Android API. Further on, the edge agent is concerned about performing actions on behalf of the user and it uses the DIDComm BLE library for operations such as sending and receiving messages over BLE. Next, the services are the functionality provided by the application and implements higher-level protocols. It makes use of the edge agent and the view to provide functionality such as establishing a DID connection, message exchange, and access control. Following the separation of concerns, it also makes the application more maintainable and easier to test since the view and agent are not directly coupled. Finally, the view is the graphical interface that the user is interacting with to perform actions and displaying information. The next section focuses on the interactions between these components and the different data flows associated. For the purpose of showing how the applications can be integrated into the wider SSI ecosystem, the high-level architecture demonstrates that the applications can be connected through the Internet in order to establish connection with cloud agents, utilize a public blockchain for storing or resolving DIDs, or using other online resources. However, further on in the paper, it is assumed that the Internet is not accessible and the BLE connection is used for device-to-device communication and for transferring messages.

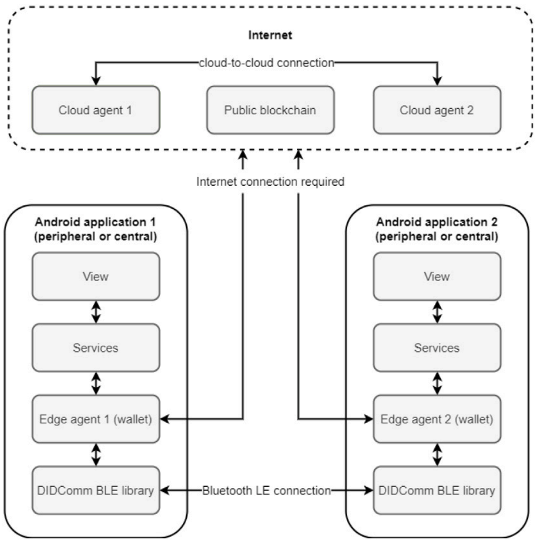


Fig. 4. A diagram showing the system's high-level architecture.

5.2. Protocols

This section describes each protocol used in the application for achieving secure messaging and the access control use case. It is broken into: out-of-band message, BLE connection establishment, device messaging (sending and retrieval), DID connection establishment and presentation exchange. Then the access control process is presented which includes all the aforementioned protocols. When we use the term flow instead of protocol, we refer to the flow of information in the protocol.

The protocols will further be modeled as a set of the involved roles, states and message types. Each of the message types that are sent between the devices is described for each protocol. Although both devices in the system can act as either the peripheral and central under the different protocols, each protocol is being described from the perspective of the peripheral and central BLE roles depending on which device first initiated the connection in the protocol. An example of this can be illustrated when sending and receiving a ping. The sender of the ping connects to the device in which the ping should be sent to and therefore the sender becomes the central, and receiver of the ping is the peripheral. However, when the receiver of the ping wants to send a response, it will also connect with the device that should receive the ping response message and thus the ping receiver becomes the central and the original sender of the ping becomes the peripheral. The reason for this is because each device that wants to send a message needs to establish a connection with the other device that should receive the message. However, more details on this are provided when discussing each of the respective protocols.

The protocols that work with DIDComm messaging and are used in the implementation are: Basic Message,[2] Trust Ping,[3] Out-of-Band,[4] DID Exchange[5] and Present Proof.[6] Minor changes were made to some of these in order to accommodate for the specific use case, and they will be discussed when it applies. In the next following sections, each of the protocols and the data flows used in the solution is described.

5.3. Out-of-band message

The out-of-band (OOB) message is used to bootstrap the connection establishment between the two devices. We assume that each of the devices that want to establish a DID connection already has created their own DID (using the did:key method). Moreover, when two devices are first presented to each other, they do not have a BLE connection, and they do not know each other's DIDs or the public key material required for encryption. It is important to note that a connection between two devices in the system can exist on two levels. The BLE connection that is established between two devices, and the DID connection that can exist between two peer DIDs. The BLE connection is responsible for providing a secure wireless communication channel where the messages can be sent from one device to another, while the DID connection is responsible for enabling the devices to have a secure and trusted peer-to-peer communication that allows messages to be end-to-end encryption (E2EE) and authenticated on the application-level.

The out-of-band protocol has two roles: sender and receiver. However, since BLE and DIDComm define different roles, and when also considering which device that will initiate the BLE connection, the sender is also the peripheral, and the receiver is the central. The role that a device has in the system during the different data flows depends on whether looking at it from the perspective of DIDComm or BLE. The protocols will mostly be described from the perspective of the peripheral and central when looking at the data flow in order to highlight the use of BLE as the technology for sending and receiving the message. This contrast is emphasized in Sections 5.4 and 5.5, which address the BLE connection establishment and device messaging, respectively.

The protocol begins with the peripheral device creating an connection invitation using a OOB message that includes information such as a self-generated BLE Service UUID that the receiver of the OOB uses to identify the device to which it should send any response to the OOB, as well as the sender DID, which the receiver can use for encrypting the response. Fig. 5 depicts the OOB data flow between peripheral and central, whereas establishing the BLE connection is covered in Section 5.4.

The OOB message can be represented as binary data using *Base64-url* encoding and contained in a URL. There are several ways to share this OOB message, e.g. included in QR-code or NFC-tag. In the implementation, NFC-tags can for example be used to hold the OOB invitation message. We will focus on the case where the OOB is presented as a QR-code on the peripheral device's screen. The central will then use a camera to scan this QR-code and obtain the *Base64-url* encoded OOB message. It then decodes it and reads the payload of the message. This payload includes a DID and a self-generated BLE Service UUID for identifying the peripheral.

Following that, the central begins the discovery process by scanning for devices. The BLE Service UUID obtained from the OOB message can

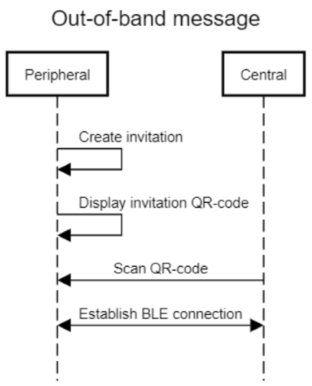


Fig. 5. Overview of the out-of-band message data flow between peripheral and central.

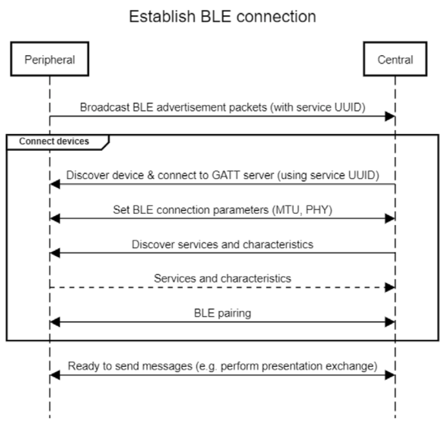

Establish BLE connection
Peripheral
Central
Broadcast BLE advertisement packets (with service UUID)
Connect devices
Discover device & connect to GATT server (using service UUID)
Set BLE connection parameters (MTU, PHY)
Discover services and characteristics
Services and characteristics
BLE pairing
Ready to send messages (e.g. perform presentation exchange)

Fig. 6. Overview of the BLE connection establishment flow.

then be used to filter out other devices and find the peripheral device that sent the OOB. The next step is to connect the devices, which is addressed in the following section.

5.4. BLE connection establishment

This section describes the steps involved in connecting two devices using BLE and the OOB message. A BLE session allows the connected devices to exchange data over the channel and begins when the connection is established and ends when one of the devices disconnects. An overview of the flow is depicted in Fig. 6.

After the peripheral has advertised its BLE service UUID and the central has found it using the OOB message, the central can connect to the peripheral. The Maximum Transmission Unit (MTU) and Physical Layer (PHY) of the channel are then set based on the capability of the BLE devices in order to achieve higher throughput. Then services provided by the peripheral are discovered and received by the central. The devices may optionally start the pairing process if needed. After that, the BLE connection is ready for messages to be exchanged using the read and write characteristics, and is covered in the next flow.

5.5. Device messaging

Despite the fact that the system was built to allow communication to flow both ways between peripheral and central devices, the system only uses the central to peripheral flow when sending messages from one device to another. This is because when multiple messages are sent back and forth between the devices in a BLE session, if one of the devices loses connection, there is no mechanism for the peripheral to contact the central device. Therefore, an alternative was used where

[2] https://didcomm.org/basicmessage/2.0/
[3] https://identity.foundation/didcomm-messaging/spec/#trust-ping-protocol-20
[4] https://github.com/hyperledger/aries-rfcs/tree/main/features/0434-outofband
[5] https://github.com/hyperledger/aries-rfcs/tree/main/features/0023-did-exchange
[6] https://github.com/hyperledger/aries-rfcs/tree/main/features/0454-present-proof-v2

one device establishes a connection with the other device when sending a message, and both act as either the peripheral and central depending on who sends the message. Nevertheless, both sending messages from the peripheral to central, and central to peripheral are covered below.

A message that is sent from peripheral to central is denoted as $M_p \rightarrow M_c$, while sending a message from central to peripheral is denoted as $M_c \rightarrow M_p$. To symbolize that a message is encrypted, a superscript is used, such as $M_c^e \rightarrow M_p^e$, meaning that the message is encrypted from central to peripheral. If not otherwise specified, it is assumed that the DIDComm messages transported over the established BLE channel are using JSON encoding and encrypted using the authcrypt method.

Send message from peripheral to central. Sending a message from peripheral to central, $M_p \rightarrow M_c$, is performed as follows. First, the peripheral and central needs to have an established BLE connection as described in the above flow. Then the peripheral begins by creating a message M_p. This is a message in the DIDComm plaintext format and JSON encoding. After that, the message is split into message packets of size MTU. This is the payload size that can be transferred at once as specified by the Bluetooth specifications. The packets are then added to a message queue, waiting to be transmitted. A loop will iterate through all the queued message packets until all is sent. On each iteration, the read characteristics are updated with the data of the packet and a notification is sent to the central. When the central receives those notifications, it will append the packet to a buffer. After all the message packets for a specific message are sent by the peripheral, it will update the read characteristics to the value "EOM" and send a notification, indicating to the central that the entire message has been transmitted. The central can then read the message from the buffer. Furthermore, after the DID connection establishment is completed, the messages can be encrypted to the other party $M_p^e \rightarrow M_c^e$. This is achieved by the peripheral first encrypting the message using the public key that belongs to the central, before splitting it into message packets.

Send message from central to peripheral. This flow is similar to the $M_p \rightarrow M_c$, however there are some differences when sending a message from central to peripheral $M_c \rightarrow M_p$. Again, the peripheral and central first need to have an established BLE connection. The central then creates a message M_c, splits the message into message packets of size MTU and then added to the queue. Then a loop iterates through all the packets. Differently from the peripheral to central flow, on each iteration the central will instead write to the peripheral write characteristics with the value of the message packet. Also, on each write, the peripheral will append those packets to a buffer. Then, when the central has no more packets to send, the central will write to the peripherals write characteristic with the value "EOM" to indicate end of message. An overview of this flow can be seen in Fig. 7. Also, the central can encrypt the message to the peripheral $M_c^e \rightarrow M_p^e$ using the public key that belongs to the peripheral after DID connection establishment. The DID connection establishment flow is described in the next section.

Now that the different ways for sending a message over BLE between devices have been explained, we will focus on the type of messages being sent and the higher-level functionality such as how a secure peer-to-peer connection is established through a series of DIDComm messages exchanged between the devices.

Sending a basic message. Despite the fact that it is not used in the access control flow, the basic message[7] protocol has been integrated into the application to provide messaging and chat functionality between the devices, and is later used as a baseline for testing the performance when sending and receiving messages over BLE. It is a stateless protocol using one message type with the following URI "https://didcomm.org/basicmessage/2.0/message". Two roles are defined: sender and receiver. The message that is to be conveyed from sender to receiver is contained within a "content" attribute in the body of an DIDComm message.

[7] https://didcomm.org/basicmessage/2.0/

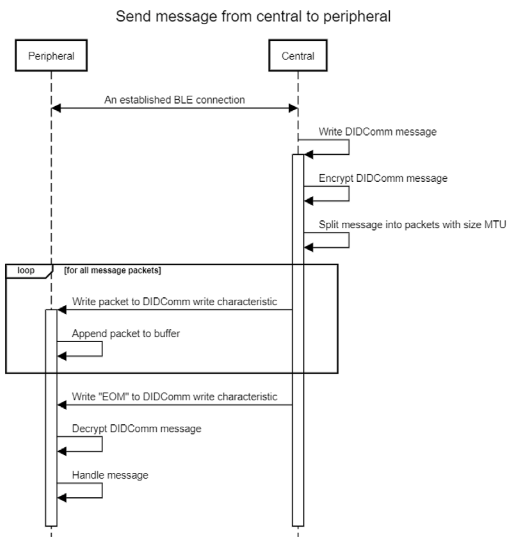

Send message from central to peripheral
Peripheral
Central
An established BLE connection
Write DIDComm message
Encrypt DIDComm message
Split message into packets with size MTU
loop
[for all message packets]
Write packet to DIDComm write characteristic
Append packet to buffer
Write "EOM" to DIDComm write characteristic
Decrypt DIDComm message
Handle message

Fig. 7. Overview of the message exchange data flow from central to peripheral.

Sending a ping and receiving a ping response. The core DIDComm V2 specifications define the trust ping protocol.[8] It is used to check the other agent's connectivity, responsiveness, and security. The ping is analogous to the one used in networking, but also inherits the properties of DIDComm messages. The protocol has two message types: the ping message and the ping response message.

5.6. DID connection establishment

A DID connection provides a cryptographically secure direct peer-to-peer link between the edge agents of the respective devices. This allows the devices to securely exchange encrypted and authenticated messages in a trustful way. The DID Exchange[9] protocol is used to exchange the DIDs between two peers in order to establish the DID connection.

In order to establish a DID connection, the devices must first exchange DID documents and the public keys contained within them. This DID connection can be established in one of two ways. When an existing DID connection is not used, the initial step is for the central to send a request message to the peripheral. A reference to the OOB is included in this request message, along with the DID document for the DID that the central wants to use for this connection. The message is then sent to the peripheral using the message exchange flow described previously in Section 5.5. After receiving this message, the peripheral responds by sending its own DID document to the central. Lastly, the central will send a complete message to the peripheral to acknowledge that the response message was received. The second option is to reuse a previously established DID connection. It begins when the central sends the peripheral a reuse message. The peripheral will respond with a reuse accepted message if the reuse is accepted. Fig. 8 depicts an overview of the data flows when not reusing an existing connection and when reusing an existing connection.

5.7. Presentation exchange

This protocol is concerned with requesting the proofs and makes use of the DIF Presentation Exchange [12] specification for defining the

[8] https://identity.foundation/didcomm-messaging/spec/#trust-ping-protocol-20

[9] https://github.com/hyperledger/aries-rfcs/tree/main/features/0023-did-exchange

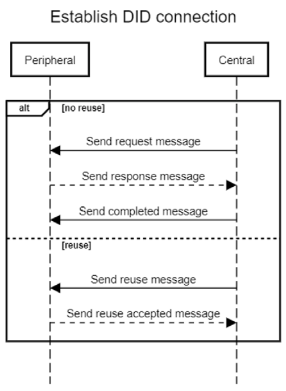


Fig. 8. Overview of the DID connection establishment flow using Bluetooth Low Energy for sending the DIDComm messages.

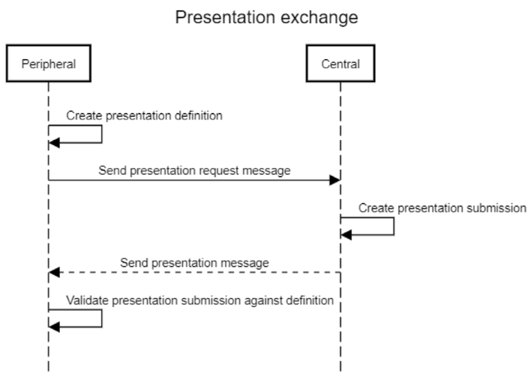

Presentation exchange
Peripheral
Central
Create presentation definition
Send presentation request message
Create presentation submission
Send presentation message
Validate presentation submission against definition

Fig. 9. Overview of the Presentation Exchange flow using Bluetooth Low Energy for sending the DIDComm messages.

data format for requesting VCs and certain claims from the prover. We use the term prover instead of holder in the context of presenting proofs to a verifier. The specification defines a way for verifiers to request holders for proofs that meet certain requirements. Its goal is to provide a standard way for a verifier to request proof from a holder, and it specifies two data formats for doing so: a presentation definition that a verifier can use to describe the proofs that should be submitted, and a presentation submission that a holder can use to include the proofs that are in accordance with the presentation definition.

The presentation definition is sent from the verifier to the prover and will include the required credential claims and proofs that the holder needs to submit. The definition data model can include a set of input descriptors that describes what information is expected from the prover and the purpose. Also, the input descriptor may include a constraint property that contains fields. The field specifies a JSONPath for selecting a target value from the input, as well as a filter for filtering the data provided by the JSONPath evaluation using JSON Schema.

A presentation submission contains the credential claims and proofs that fulfill the requirements set forth by the presentation definition. The data model specifies the *descriptor_map* object that contain an array of *input_descriptor_ mapping object* where each object must have an id that maps to the input descriptor of the presentation definition, a format property for specifying the claim format, and a path expressed as a JSONPath for selecting the input claim.

Moreover, the specification does not impose any restriction on the credential format and how the presentation definition or submission is transported. The DIDComm based present-proof protocol as specified in an Hyperledger Aries RFC [52] has been used for requesting and presenting proofs using the presentation definition and presentation submission. Here, the definition and presentation submission is appended to the 'attachment' field of the DIDComm message.

In this described data flow, the peripheral will act as the verifier and the central will act as the prover. An overview of the flow is depicted in Fig. 9. It first requires that a presentation definition is created by the peripheral. The peripheral then includes this presentation definition inside a presentation request message and sends it to the central. When the request is received by the central, it generates a presentation submission. On submitting this, it will send a presentation message to the peripheral that includes the presentation submission along with any required verifiable credentials. The peripheral will then validate the presentation submission against the definition and check proofs to determine its validity.

6. Implementation

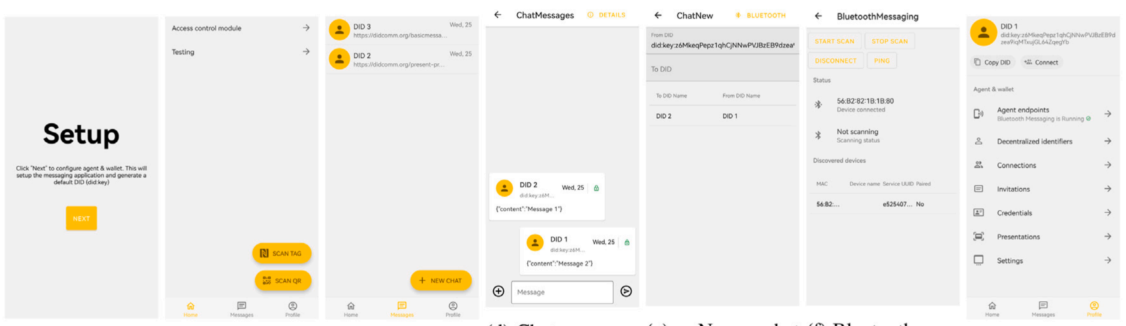
In this section we describe our approach to implement a proof-of-concept (PoC) application for Android that integrates the DIDComm messaging protocol and Bluetooth Low Energy protocol. The focus is on the BLE transport layer and the presentation exchange flow that is used for access control, as stated in the use case. It aims to provide developers with an overview of the implementation and technology utilized to provide the given functionality. The implementation strives to be as straightforward and adaptable as feasible while still achieving the desired functionality. The full source code for the application is available at GitHub.[10]

The PoC application has mainly two parts that will be described separately:

- **The React Native application**. This is the mobile digital identity wallet application that the user is interacting with and provides the graphical user interface. After assessing several SSI frameworks [53–55], Veramo was selected as the framework for building the agent. The Veramo framework is open-source, have active development and community, and support the W3C VC/DID standards. Each fundamental agent functionality in the PoC application is implemented as a Veramo plugin such as identifier management, key management, DID method provider and resolver (for did:key), credential signing and verification.[11] Those plugins do not handle any functionality related to the BLE transport or higher-level protocol message handling. However, they manage the involved cryptography such as encryption and decryption of messages, and also the signing and verification of credentials. A reference implementation of DIDComm [56] has been used and is integrated into a Veramo plugin. This implementation provides the functionality needed to pack and unpack DIDComm messages. Packing a DIDComm message will encrypt the plaintext message using the specified mode. On the other hand, unpacking the DIDComm message will decrypt the message into a plaintext message. The application relies on the Native Modules [57] provided by React Native for making native calls to the Android DIDComm BLE library.
- **The Android DIDComm BLE library**. A custom library was made to provide the BLE functionality needed for the application to exchange DIDComm messages between devices. The library provides the React Native application with functionality for conducting the various Bluetooth operations, such as establishing a BLE connection with another device, writing and reading operations on the GATT server in order to send the messages. It also handles splitting larger messages into message packets and appends them to a queue, waiting to be transferred. This library provides an abstraction on top of the Android Native BLE API [58] for serving the React Native application with an API that provides functionality for sending and receiving DIDComm messages.

[10] https://github.com/alexanderenge/DIDCommunication-using-BLE

[11] https://veramo.io/docs/veramo_agent/plugins/



(a) Setup screen (b) Home tab (c) Messages tab (d) Chat messages screen (e) New chat screen (f) Bluetooth messaging (g) Profile screen

Fig. 10. Application screens.

Storage and state management. The application stores information about the agent settings, chat messages, current protocol states, as well as the decentralized identifiers, cryptographic keys, established connections, verifiable credentials, and presentations. The storing of cryptographic keys and identifiers is handled by the Veramo agent framework and uses a SQLite database. Redux is selected as the tool for managing state within the application.

6.1. Application screens

The proof-of-concept mobile application integrates the above-mentioned flows and includes multiple screens as well as a navigation bar with three primary tabs: Home, Messages, and Profile. Although not all screens were necessary to demonstrate the given use case, they were useful for testing and presenting some of the functionality of the system. Nonetheless, a brief description of some of the screens is provided, followed by the section that covers each stage of the access control process in the system.

Home. If the application is being used for the first time, the user is presented with a setup screen, as shown in Fig. 10(a). Clicking the 'Next' button will set up the agent and wallet with a default DID (did:key), and then redirect to the home screen. A button on the home screen opens the access control module panel, which is used to perform access control and is detailed in more depth in Section 7 about access control process. A button for accessing the camera and reading QR-codes, as well as another for scanning NFC-tags, are all on the same screen. This allows the content of the OOB message to be read in order to bootstrap the establishment of BLE and DID connection with another device. Fig. 10(b) depicts the home tab.

Messages. The message tab displays the chats (Fig. 10(c)) that exist between a DID connection, and each chat can be opened to display the chat messages from and to the specific DID (Fig. 10(d)). From the chat messages screen, encrypted DIDComm messages can be sent to the particular DID over BLE. A new chat can also be created by selecting an existing DID connection (Fig. 10(e)). The functionality provided here is primarily based on the establishment of a DID connection covered in Section 5.6 to establish a trusted peer-to-peer connection for securely sending messages, and the messaging exchange flow to transmit the messages from one device to the other as covered in Section 5.5. The Bluetooth messaging screen (Fig. 10(f)) can be used to manually start scanning for BLE devices and connect to discovered devices, in addition to sending a simple ping message containing a string message to test connectivity and the messaging functionality. This ping message is not a DIDComm message since the DID of the connected device is still unknown because DIDs have not been exchanged.

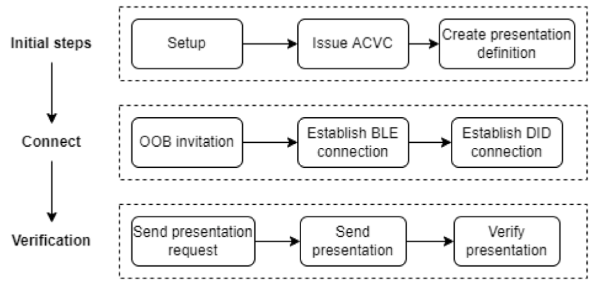


Fig. 11. The overall steps involved in the access control system.

Profile. From this tab, the user can navigate to their agent endpoints, identifiers, connections, invitations, credentials, presentations and settings (Fig. 10(g)).

The agent endpoints screen is used to set up the agent's BLE service endpoint, which is used by other devices to identify the device when scanning for it. Starting and stopping the BLE advertisements can be done from this screen. However, in order to be discovered by other BLE devices and receive messages, the BLE advertisement must be active.

The identifiers screen consist of a list of the created DIDs, as well as the options for creating new DIDs using the did:key method. When clicking on a specific DID, it redirects to a screen that views the DID document of that DID, the service endpoints of that DID, and the possibility to give the DID a name that will be shared when establishing a DID connection with another device. An invitation to establish a connection with the DID will be created when clicking the 'connect' button. This invitation is displayed on the device as a QR-code. Writing the message to an NFC-tag can be done by clicking the 'share' button. Another device can then scan this QR-code or NFC-tag to establish a DID connection with this device. The connections screen shows a list of existing DID connections and the respective DIDs used in the peer-to-peer relationship.

Moreover, the invitations screen keeps track of the state of current OOB invitations, and whether they have been accepted or not. The credentials screen lists the verifiable credentials that have been issued or received from another issuer. It also contains a button that, when clicked, redirects to a list of existing schemas that can be used as the template for issuing new credentials. The presentations screen provides information about the created presentation definitions used for requesting proofs and the submitted presentations. It allows creating new presentation definitions that can be used in presentations requests sent from the verifier to the prover. Lastly, the settings tab lets the user configure the agent, such as whether to reuse existing connections for an invitation from a DID that there exists an connection with, whether invitations should be automatically accepted, or multiple use of the same OOB invitation is allowed.

```
{
  "@context": {
    "@version": 1.1,
    "@protected": true,
    "AccessControl": "https://example.com/contexts/AccessControl",
    "name": "https://schema.org/name",
    "role": "https://schema.org/roleName"
  }
}
```

Fig. 12. Example of schema for the access control verifiable credential.

7. Access control process

This section describes the access control process. The information flows according to the covered protocols in Section 5.2. It makes use of the previous protocols, except basic message and the trust ping protocol. The different stages are presented along with the corresponding screens that are displayed within the developed application.

The necessary functionality required to perform the role of an issuer, holder and verifier is integrated into a single mobile application as one unit. The function of the issuer is to manage and issue the credentials to the holder that specifies the access rights. During the access control process, the function of the verifier is to request and verify access rights. While, the prover uses the verifiable credentials as proof of access rights. Within the proof-of-concept, there have been identified three main phases: setup, credential issuance and verification. An illustration of the workflow for performing the access control process can be seen in Fig. 11. The following paragraphs will describe these phases, focusing on access control between two devices.

The application setup phase consists of first creating a DID using the did:key method and this DID is used as the default DID for further interactions with other agents and devices. This will create a key-pair consisting of a public and private key, and then use the public key as part of the method-specific identifier for the DID. The exact procedure is specified by the DID method. Key material is then safely stored within the digital wallet for protection. After this, the application generates a random BLE Service UUID and adds it alongside the DID document as a service endpoint. This finishes the setup phase and BLE advertising could be started using the BLE Service UUID that was generated earlier. This setup phase will be performed on both devices involved in the access control such that each device has their own DID.

The next phase consists of issuing the access control verifiable credentials. This credential is based on a predefined schema that should be understood by all participating parties in order to be interoperable. It contains the granted permissions as attributes within the VC. In the current solution, an example schema for access control was created for testing purposes and is seen in Fig. 12 below. Following this schema, a verifiable credential can then be created, including the name and role attributes as specified by the schema, and digitally signed by the issuer. Then the credential is sent to the credential subject. After this, the verification step can take place.

The input for the verification step is the presentation definition used for requesting proofs, while the output is access granted or denied. The verifier will check to see if the prover's presentation submission is in accordance with the presentation definition. If the presentation submission is valid, it will grant access with the specified role. However, if it is not valid, it will deny access. This verification step also includes determining whether the data structure is as expected, validating the proofs and checking the credential status.

A proof of holder identity and that the credentials are indeed owned by the holder that presents them, and not some third-party that got hold of them, can be proved with the subject-holder bindings as shown in the specifications when creating a verifiable presentation [59]. This creates a link between the subject and claims held in the verifiable credentials using digital signatures.

Inside the access control module screen, an invitation to present proof can be shown for others to scan. To do so, the user must first choose a presentation definition that will be used in the presentation exchange and validation stage. The entire process goes through four different steps: select a presentation definition (step 1) → display invitation as QR-code (step 2) → await proof (step 3) → validate (step 4). We assume that the access control verifiable credentials have already been distributed to those who require them as proof of access rights, and that the verifier has established the presentation definition. These presentation definitions are stored on the verifier device.

In step 1, the verifier decides on the presentation definition that will be used (Fig. 13(a)). This lets the verifier choose the proofs that must be submitted in order to be granted access. When a definition is selected, the 'start access control module' button can be clicked to generate an OOB invitation message, then it redirects to the invitation screen which displays the invitation as a QR-code (Fig. 13(b)), step 2.

In step 2, any holder of access control verifiable credentials that want access can scan this QR-code. This will first establish the BLE connection between the devices if not already existing and a DID connection. A presentation request message containing the presentation definition is then sent from the verifier to the prover. In step 3, when the prover receives the presentation request, the verifier sees an awaiting proof screen (Fig. 13(c)), while the prover sees a screen asking if they want to submit or cancel the presentation request (Fig. 13(d)). The prover will try to match the presentation definition against the credentials held in the wallet in order to find candidate credentials that can be used to generate a presentation submission. On submit, the presentation submission is sent to the verifier.

In step 4, the verifier validates the proofs against the presentation definition, as well as verifying that the credentials have not been tampered with and are issued by a trusted issuer, to determine if access should be granted or denied. Figs. 13(e) and 13(f) show the access granted and denied screens.

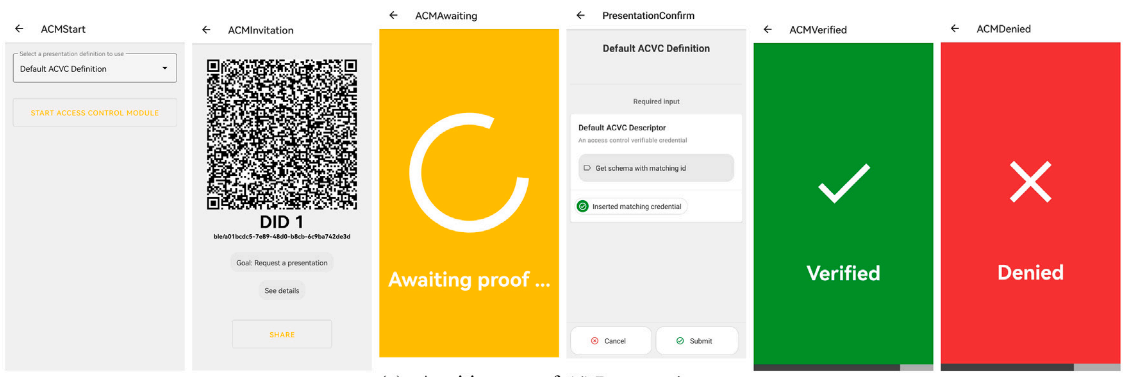
8. Experimental results

After describing the functionality and the specific implementation of proof-of-concept, this section presents the evaluation setup and test bed infrastructure. The overall evaluation will refer to the application use case presented earlier, however a discussion on how the system can be applied to other cases is also given along with benefits and challenges.

The methods for each of the various tests that aim to capture the performance of the system as a whole are presented individually along with the experimental result. When doing performance testing, numerous factors must be considered, such as the time required for cryptographic computation, network latency, and other variables that may affect the outcome. All of this adds to the intricacy of the procedure. The performance is mainly measured in the duration it takes for each step to complete in the aforementioned protocol flows. Also, the performance will vary depending on the encryption scheme applied and BLE connection parameters.

The goal is not to capture the precise BLE data-transfer rate as this is already done in several papers [43], but instead focusing on capturing and quantifying overall performance under different scenarios specific to our use case. It also provides an overview of the extra payload in bytes produced when encryption is applied to the messages, as seen by the difference between content size and message payload. The message size of the different message types when encrypted has been measured. Lastly, the measures are taken in such a way that it captures the duration it takes to send messages over the network, and the duration spent on computation such as encrypting, decrypting and verifying.

During the experiment, two BLE capable smartphone devices that can interact with one another were used as the hardware and when benchmarking the developed application. Furthermore, the two devices are kept in close proximity to one another during the experiments and are not paired. Table 1 depicts the hardware specifications of these two mobile devices. For the rest of the report, the two devices used for performance testing will be referred to as device 1 and device 2. Within


ACMStart
Select a presentation definition to use
Default ACVC Definition
START ACCESS CONTROL MODULE
ACMInvitation
DID 1
ble/a01bcdc5-7e89-48d0-b8cb-6c9ba742de3d
Goal: Request a presentation
See details
SHARE
ACMAwaiting
Awaiting proof ...
PresentationConfirm
Default ACVC Definition
Required input
Default ACVC Descriptor
An access control verifiable credential
Get schema with matching id
Inserted matching credential
Cancel
Submit
ACMVerified
Verified
ACMDenied
Denied

(a) Start screen (b) Invitation screen (c) Awaiting proof screen (d) Presentation confirm screen (e) Verified screen (f) Denied screen

Fig. 13. Access control process.

Table 1
Hardware specifications for the two devices used in performance testing.

	Device 1	Device 2
Device name:	HUAWEI P30 Pro	HUAWEI Mate 20 Pro
Model:	VOG-L29	LYA-L29
Build number:	12.0.0.132	11.0.0.200
EMUI version:	12.0.0	11.0.0
Android version:	10	10
Processor:	Huawei Kirin 980	Huawei Kirin 980
RAM:	6.0 GB	6.0 GB
Resolution:	2340 × 1080	3120 × 1440

Table 2
Performance results for the BLE connection establishment.

	Device 1	
	$S_{1=startscan}$	
	$M_{1=founddevice}$	$M_{2=discoveredservices}$
Min:	76	1355
Max:	1415	2940
Average:	212	1691
Stdev:	220	238

the presentation exchange protocol, device 1 is assigned the verifier role, whereas device 2 is assigned the prover role.

While conducting the performance tests, a production build of the React Native application was used. All performance measurements are carried out with the react-native-performance library.[12] Each test is run through the application's user interface, and the results are then saved in a database for later retrieval and analysis. In the following paragraphs, the different testing procedures and metrics used will be described.

Measurements were done using four separate tests, each with its own set of parameters. These tests are: (1) BLE connection establishment, (2) Device messaging, (3) DID connection establishment, and (4) Presentation exchange. An MTU of 509 is used when transferring the message payload over BLE. Each test contains a number of measurements. Also a test can have different scenarios such as different content length, payload. Each test scenario is repeated 100 times and then the results are averaged for all measurements. All measurements are recorded using four separate metrics: min duration (over 100 measurements), max duration (over 100 measurements), average duration (over 100 measurements), standard deviation (over 100 measurements).

Performance measurements are taken on both devices 1 and 2. Each measurement reports the duration from a start marker and until the measurement is taken. These are denoted using the symbols **S** and **M**, where **S** is the start marker that sets the start time for the measurement **M**, and **M** is then the time duration from the start marker **S**. The duration for all measurements taken is specified in milliseconds (ms). A subscript is used to differentiate between each start marker and measurement.

8.1. BLE connection establishment

We first consider the scenario when the sender device (central) is not connected to the receiver device (peripheral) and perform tests to identify how long it takes for the BLE connection establishment. Between each test, the message sender disconnects from the connected device and waits 5 s before running the next test.

In this test, device 1 acts as the message sender. The start marker (S_1) is placed right before the device begins scanning for other BLE devices. It first measures the duration it takes for device 1 to find device 2 (M_1). Then it measures the duration it takes until the services have been discovered (M_2). Both of these measurements (M_1, M_2) start from when device 1 starts scanning for other devices (S_1). After the service has been discovered, messages can be sent to the device.

Results. An overview of the results can be seen in Table 2. It is observed that the average duration from start scanning to when the device is found (M_1) is 212 ms, having a minimum duration of 76 ms and a maximum of 1415 ms. The standard deviation is 220 ms. When the services have been discovered on the device (M_2), the total average duration is 1691 ms with a minimum of 1355 ms and maximum of 2940 ms. The standard deviation here is 238 ms.

8.2. Device messaging

This test will serve as a baseline for the other tests and covers the three different DIDComm packing modes (none, anoncrypt and authcrypt) described in Section 2.2 using five different content lengths for each packing mode. In this test, three measurements have been taken for each of the packing modes and content lengths in order to capture time taken to transmit a message from one device to another device via BLE. In this test, device 1 acts as the message sender and device 2 acts as the message receiver of that message. It assumes that the sender device (central) is connected to the receiver device (peripheral). The measurements along with the associated starting mark are described as follows.

On device 1, the start marker (S_2) is placed right before packing the DIDComm message. The first measurement considers the duration from the start marker (S_2) and until before transporting it across BLE (M_3). This measure includes the time taken to pack the message and

[12] https://www.npmjs.com/package/react-native-performance

Table 3
Performance results for messaging between two BLE devices.

Packing	Content	Payload	Device 1		Device 2
			$S_{2=send-message}$		$S_{3=handle-message}$
			$M_{3=start-sending}$	$M_{4=sent-message}$	$M_{5=unpacked-message}$
none	10	328	1	69	2
	100	418	1	71	1
	1000	1318	1	143	2
	10000	10318	2	774	7
	100000	100318	10	7133	38
anoncrypt	10	951	97	225	170
	100	1071	98	293	170
	1000	2271	101	375	170
	10000	14271	108	1651	183
	100000	134271	200	11763	295
authcrypt	10	1105	198	400	244
	100	1225	201	370	242
	1000	2425	201	456	244
	10000	14425	214	1849	248
	100000	134425	295	11653	343

encrypt it, but excludes the duration taken to transmit the message via BLE. The next measurement also includes the time it takes to send an entire message over BLE (M_4). Both of these measurements (M_3, M_4) start from before the packing of the message (S_2). The device 2 uses a start mark (S_3) in the message handler right before unpacking the received message. From this start marker (S_3), a measurement is then taken after finishing unpacking the message (M_5).

The basic message is used as the message type for this test. Each basic message has some overhead which includes the attributes for the messages. Therefore, to get the exact payload for each message, the equation will be $contentlength + overhead = messagepayload$. The content length is specified as the number of characters in the 'content' field of the message body that is sent. The message payload is specified as the total number of bytes in the message after the message has been packed. The Table 3 depicts this relationship between content length (shown as content) and message payload (shown as payload) for each packing mode.

Results. An overview of the results is shown in Table 3. On the sender device, it can be observed from M_3 that the average elapsed time before transporting the message is shortest when packing mode is none, then comes anoncrypt, and authcrypt takes the longest time with 295 ms when using a content length of 100,000 characters. When also considering the time when the message is sent (M_4), anoncrypt spends 11,763 ms, while authcrypt spends 11,653 ms. For unpacking the message on the receiver device (M_5), authcrypt using a content length 100,000 takes the longest with 343 ms.

Data transfer speed rate, including the duration it takes to encrypt the message, can be calculated with the following formula: speed = (message payload)/(duration of measurement M_4). We then get the following results for the different payloads when using a content length of 100,000:

- Packing mode is none = $\frac{100318\text{ bytes}}{7133\text{ ms}}$ = 14.1 KB/s
- Packing mode is anoncrypt = $\frac{134271\text{ bytes}}{11763\text{ ms}}$ = 11.4 KB/s
- Packing mode is authcrypt = $\frac{134425\text{ bytes}}{11653\text{ ms}}$ = 11.5 KB/s

In order to calculate the transfer speed to send the message without including the time spent on encryption, we subtract the duration of M_3 since that is the duration elapsed before sending the message over BLE. Data transfer speed rate is then calculated as follows: speed = (message payload)/(duration of measurement M_4 - duration of measurement M_3):

- none = $\frac{100318\text{ bytes}}{(7133-10)\text{ ms}} = \frac{100318\text{ bytes}}{7123\text{ ms}}$ = 14.1 KB/s
- anoncrypt = $\frac{134271\text{ bytes}}{(11763-200)\text{ ms}} = \frac{134271\text{ bytes}}{11563\text{ ms}}$ = 11.6 KB/s
- authcrypt = $\frac{134425\text{ bytes}}{(11653-295)\text{ ms}} = \frac{134425\text{ bytes}}{11358\text{ ms}}$ = 11.8 KB/s

8.2.1. *Sending and receiving a ping response over BLE*

This test compliments the baseline test presented above by also including a response message sent from the receiver device. It assumes that both sender devices (central) are connected to the other receiver device (peripheral). From the ping sender, this means that the device has a connection with the ping receiver, and also that the ping receiver which now becomes a sender when responding to the ping, has a connection with the ping sender. Meaning that both devices act in the peripheral and central role simultaneously.

The start marker (S_4) for device 1 is placed right before constructing the trust ping message object and therefore also before encryption is applied. From this start marker (S_4), two measurements have been taken. First, a measurement that measures the duration when the ping is sent (M_6), which then includes the time spent from start sending the ping, constructing the message, then to completely having sent the entire ping message over BLE. Another measurement is then taken when a ping response is received from the other device (M_7).

Results. The results for this test are shown in Table 4. For the ping sender device, the average duration elapsed when sent ping (M_6) is 383 ms, while when a ping response has been received (M_7) the duration is 1210 ms. During the 100 test runs, the lowest duration was 324 ms, highest was 456 ms, and a standard deviation of 30 ms for M_6, whereas M_7 resulted in 1108 ms as the lowest duration, 1392 ms as the maximum, and a standard deviation of 44 ms.

8.3. *DID connection establishment over BLE*

This test measures the time it takes to establish a DID connection over BLE. In this test, device 1 acts as the inviter and device 2 acts as the invitee. It covers the two scenarios: no connection reuse and connection reuse. During the testing, the invitation messages were sent over BLE to the other device, and not using OOB.

8.3.1. *No connection reuse*

This test covers the scenario when not reusing the DID connection. On device 1, the start marker (S_5) is placed right before the invitation object is created. Then, measurement M_8, measures the duration until completely having sent the entire invitation message over BLE. It included the time encrypting the invitation message to device 1. Measurement M_9 includes the duration elapsed when handling the request message. Then, measurement M_{10} is the duration before sending the response, while measurement M_{11} is the duration when the response has been sent over BLE. Lastly, measurement M_{12} measures the duration when handling the complete message and includes the total duration for the DID connection flow when no connection reuse is utilized.

Table 4
Performance results for sending and receiving a ping between two BLE devices.

Device	Start marker	Measurement	Min	Max	Avg.	Stdev
1	$S_{4=send-ping}$	$M_{6=sent-ping}$	324	456	383	30
		$M_{7=received-ping-response}$	1108	1392	1210	44

Table 5
Performance results for establishing a DID connection with no reuse.

Device	Start marker	Measurement	Min	Max	Avg.	Stdev
1	$S_{5=send-invitation}$	$M_{8=sent-invitation}$	96	217	142	18
		$M_{9=handle-request}$	876	1363	1095	101
		$M_{10=send-response}$	901	1392	1127	102
		$M_{11=sent-response}$	1420	1861	1674	105
		$M_{12=handle-complete}$	2290	2994	2560	151
2	$S_{6=handle-invitation}$	$M_{13=send-request}$	31	75	45	6
		$M_{14=sent-request}$	524	1005	729	107
		$M_{15=handle-response}$	1544	1977	1776	102
		$M_{16=send-complete}$	1552	1987	1786	102
		$M_{17=sent-complete}$	1918	2640	2203	156

On device 2, the start marker (S_6) is placed right before handling the invitation. The first measurement M_{13}, measures the duration before sending a connection request in response to the invitation, and measurement M_{14} is the duration when it has been sent over BLE. Measurement M_{15} is the duration when handling the response message. Lastly, measurement M_{16} is the duration before sending the connection complete message and measurement M_{17} is the duration when the message is sent over BLE.

Results. The performance results when a DID connection is not used are shown in Table 5. First, on device 1 (inviter), the duration of the measurements starts from the send marker (S_5). The duration when the invitation is sent (M_8) are 142 ms, 1095 ms on handling the request message (M_9), 1127 ms when start sending the response (M_{10}) and 1674 ms when the response is sent (M_{11}), then the total duration when handling the complete message (M_{12}) is 2560 ms.

For device 2 (invitee), the measurements start from when handling the received invitation (S_6). On send request (M_{13}), the duration is 45 ms and when the request is sent (M_{14}) the duration is 729 ms. On handling the response (M_{15}) the duration is 1776 ms, whereas the duration is 1786 ms when sending the complete message (M_{16}) and 2203 ms when sent (M_{17}).

8.3.2. Connection reuse

Differently from the previous test, this test considers the case of when a DID connection is reused. On device 1, the start marker (S_5) is placed right before the invitation object is created. Then, measurement M_{18}, is the duration when having sent the invitation message. Measurement (M_{19}) is the duration when handling the reuse message received. Measurement M_{20} includes the duration before sending the reuse accepted message whereas measurement M_{21} is the duration when the reuse accepted message has been sent over BLE.

On device 2, the start marker (S_6) is placed right before handling the invitation. It includes three measurements. First, measurement M_{22} is the duration before sending the reuse message. Second, measurement M_{23} is the duration when the reuse message has been sent over BLE. Lastly, measurement M_{24} is the duration when handling the reuse accepted message.

Results. An overview of the test results for the scenario when reusing a DID connection are shown in Table 6. The measurements taken on device 1 (inviter) start from the send invitation mark (S_5). When the invitation is sent (M_{18}), the average duration is 152 ms, whereas on handle reuse (M_{19}) the duration is 831 ms. The duration is 861 ms when send reuse accepted (M_{20}) and 1235 ms when sent (M_{21}). For device 2 (invitee), the measurements use handle invitation (S_6) as the start marker. Here, the average duration is 52 ms when send reuse

Table 6
Performance results for establishing a DID connection with reuse.

Device	Start marker	Measurement	Min	Max	Avg.	Stdev
1	$S_{5=sendinvitation}$	$M_{18=sentinvitation}$	77	215	152	22
		$M_{19=handlereuse}$	748	925	831	42
		$M_{20=sendreuseaccepted}$	776	959	861	43
		$M_{21=sentreuseaccepted}$	1102	1359	1235	52
2	$S_{6=handleinvitation}$	$M_{22=sendreuse}$	29	108	52	9
		$M_{23=sentreuse}$	375	541	459	36
		$M_{24=handlereuseaccepted}$	1205	1469	1324	55

Table 7
The average reported performance for presenting and verifying proofs over BLE. M_{25}=sent presentation request, M_{26}=handle presentation, M_{27}=validated presentation submission, M_{28}=accepted invitation, M_{29}=verified presentation, M_{30}=send presentation, M_{31}=sent presentation.

Reuse	Device 1					Device 2	
	$S_{7=sendpresentationrequest}$			$S_{8=createinvitation}$		$S_{9=handlepresentationrequest}$	
	M_{25}	M_{26}	M_{27}	M_{28}	M_{29}	M_{30}	M_{31}
No	425	1994	2486	2707	5217	17	1134
Yes	513	2093	2589	1359	4032	19	1138

(M_{22}) and 459 ms when sent (M_{23}), and 1324 ms when handling the reuse accepted message (M_{24}).

8.4. Presentation exchange over BLE

This test reports on the performance results for the presentation exchange over BLE, including the DID connection establishment flow, using the two scenarios. These scenarios are whether DID connection reuse is used or not. In this test, device 1 acts as the verifier and device 2 acts as the prover. When running the tests, all steps that would require interactions by the user are performed automatically by the system, i.e both invitations and presentations requests are accepted automatically such that no user involvement is needed during the testing.

On device 1, the start marker (S_8) is placed right before creating the invitation. Then measurement M_{28} is the duration from the start marker and until the invitation has been accepted. Measurement M_{29} is the duration when the presentation has been verified and covers the entire flow used in the access control process. The two scenarios of whether DID connection is reused or not have been tested. On the other hand, the start marker (S_7) is used when measuring only the presentation exchange flow part and is placed right before sending the presentation request message, it excludes the duration taken for the DID connection establishment. Then measurement (M_{25}) is the duration when the request has been sent over BLE. Measurement (M_{26}) is the duration when handling the received presentation from the other device. In addition, measurement (M_{27}) reports the duration when the presentation submission contained in the presentation has been verified.

On device 2, the start marker (S_9) is placed right before handling the presentation request. Then measurement M_{30} is taken before sending the presentation. It therefore includes the time to handle the presentation request. Then measurement M_{31} is the duration when the presentation has been sent over BLE to the verifier.

Results. An overview of the results is shown in Table 7. On device 1 and when DID connection reuse is not used, it is observed that the average duration from sending the presentation request (S_7) is 425 ms on sent presentations request (M_{25}), 1994 ms on handle presentation (M_{26})

and 2486 ms when validated presentation submission (M_{27}). When looking at the duration from the created invitation marker (S_8), the duration on accepted invitation (M_{28}) is 2707 ms and 5217 ms when verified presentation (M_{29}). On the other hand when reuse is used, from presentation request marker (S_7), the average duration on sent presentation request (M_{25}) is 513 ms, 2093 ms on handle presentation (M_{26}) and 2589 ms when validated presentation submission (M_{27}). From the created invitation marker (S_8), the duration is 1359 ms (M_{28}) on accepted invitation and 4032 ms when verified presentation (M_{29}).

For device 2, the measurements start from handle presentation request (S_9). When no reuse is used, the average duration is 17 ms on send presentation (M_{30}) and 1134 ms when the presentation is sent (M_{31}). When reuse is used, the send presentation (M_{30}) has a duration of 19 ms and 1138 ms when sent (M_{31}).

9. Discussion

In this section, a general discussion of the proposed proof-of-concept and the performance is given.

Performance. When we look at all of the performance data (Section 8), we can see that the time it takes to establish the BLE connection between the two devices is the most time-consuming process. This phase spends on average 212 ms before the device is found and 1691 ms when the services on the GATT server have been discovered and messages are ready to be sent. Once the connection has been established, a 1000-byte message payload was transmitted to the other device in 459 ms on average, including the time it takes to encrypt the message using authcrypt, using a MTU of 509 bytes. Different BLE connection parameters could have been used to get different results. If low energy consumption is a requirement, the time taken to establish the BLE connection would have been longer since the performance was tested on Android with a low latency, high power mode when advertising.

For sending the messages, a packing mode of none was used as the baseline for testing the performance when sending unencrypted DIDComm messages. When the packing mode is set to none, meaning that no encryption is applied, the overhead is 318 bytes for all different content lengths. On the other hand, when anoncrypt and authcrypt is used the message payload for a basic message with content length 10 is 951 and 1105 bytes, respectively. We can therefore see that the message payload relative to the content length increases more when using encryption than when the messages are not encrypted. As seen in Table 3, the anoncrypt and authcrypt algorithms are relatively similar in terms of the produced message payload after packing the message for each of the content lengths.

When sending messages over BLE, the ratio between content length and message payload is important because higher payloads require more message packets to be sent. Message payloads of fewer than 509 bytes could be sent in a single message packet because the MTU between the two devices was set to 509 during testing. This is evidenced by the fact that sending a message with a payload less than 509 takes about the same amount of time.

The various differences seen across a few measurements that should report similar results may be due to other underlying factors lower down the stack or in the operating system that causes the process to take less or more time in some circumstances. We had no influence over these factors because they are handled by the Android operating system and the React native framework on which the app is based such as a resource reallocation. Comparing the data transfer speed when sending an encrypted DIDComm message, excluding the duration to encrypt it, over BLE using anoncrypt and authcrypt gives reasonable results when comparing it to other applications [38,44,45]. As seen in BLE, although the theoretical data throughput is around 230 kbps, the analyzed applications reached a limit of around 100kbps [43]. Our result reported the data transfer rates for the anoncrypt to be 92.8kbps, and 94.4kbps for authcrypt. When comparing the two cases of reusing a DID connection against not reusing a DID connection, we can see that the overall time to finish the DID connection establishment from sending the invitation takes on average 2560 ms when not reusing the connection and 1235 ms when reusing the connection. Therefore, reusing an existing DID connection takes substantially less time.

When using no reuse and reuse connections, the time to complete the presentation exchange from sending the request to the presentation submission is validated takes on average 2489 ms and 2589 ms, respectively. Because the presentation request is sent after the DID connection is established, the reported results should be similar, but this could be due to some underlying factors. When comparing the time from creating the invitation to the time it takes to have verified the presentation, the two scenarios produce different results. This is because the process includes the establishing of a DID connection. When not reusing a DID connection, the total duration is 5217 ms, and when reusing the DID connection, the time is 4032 ms.

The overall transfer speed of which the messages are sent depends on a variety of factors, including message payload size, data overhead, packet loss, whether packet acknowledgment is used, and interference during transmission. For achieving higher throughput in BLE devices, five methods can be applied [60]: First, increasing the connection interval parameters allows faster transfer. Second, when performing operations on the attributes and sending messages, commands and notifications can be used instead of request and indications. Third, increasing the MTU causes more data to be transmitted in each packet therefore achieving higher throughput. Fourth, take advantage of the Data Packet Length Extension (DLE) which reduces overhead during transmission. Last, enable 2M PHY on devices that support Bluetooth 5.0 and above. Some of these methods are used in the proposed system. The effect of this, however, will vary based on the BLE device used, as the capabilities enabled by each device may vary. Also the message data needed to be transmitted could be reduced by applying different compression and encoding techniques.

Security. There are many components related to the system that concerns the security and privacy. As covered in the background section, both BLE and DIDComm already have security and privacy features built into their protocols for allowing secure messaging and communication. These individual features are not analyzed in this paper, but we instead focus on the combination of BLE and DIDComm and how it affects the security as a whole during the initial connection establishment and message exchange under different scenarios. The steps that are considered are from the beginning of creating an OOB message, and until the DIDs have been securely exchanged in a trusted manner. Only direct one-to-one messaging between devices over BLE is included, and both transport layer security and application layer security is discussed.

Before or after the inviter shares the OOB invitation message, it starts to advertise a service UUID. This makes the inviter device discoverable to other devices. Another device can then find and connect with the inviter's device by utilizing the information obtained from the OOB. The OOB invitation message is a plaintext DIDComm message without any encryption applied and is used for bootstrapping connections. Moreover, the OOB message contains a BLE service UUID encoded as part of the invitation in a service endpoint attribute field which looks like: "ble/84a4fc0f-a310-4629-885d-83d95ebc58e3". This service UUID with a "ble/" prefix will be referred to as the BLE service UUID. A connecting device can then filter for available BLE devices found during scanning which has the same advertised service UUID as the BLE service UUID, and once a matching device is found, it initiates the BLE connection.

Service endpoints can cause privacy leakage as it is used to identify a device. In addition to be included in OOB messages, service endpoints are also included in the DID documents shared during DID connection and is needed by other devices as a way of specifying how a DID with associated device or agent can be reached. The broad disclosure of service endpoints may provide a unique fingerprint that can be used to correlate multiple identifiers in use by a single party. In the proposed system the service UUID being used as the service endpoint for a DID is created during the application's setup phase and is considered static; however, it may be rotated afterwards to avoid tracking and for privacy purposes. In this case, because the service UUID is required to identify the device associated with a specific DID (as stored in the service endpoint), the new service UUID would need to be shared with existing DID connections in order to link the new BLE device address to the DID.

A MITM attack could occur during the connection process since there is no method to prohibit other devices from advertising with the same BLE service UUID. This implies that the device could connect to a device other than the one that originally created the OOB. To prevent tampering with the data, the exchange of DID and related documents must take place via a BLE secure connection. Because the inviter's DID and public key are also contained in the OOB message, any response to this invitation can be encrypted to this DID using the included public key.

An authentication system can have been employed to counteract these spoofing attempts. During pairing, BLE has several methods for authenticating the other device. Also, once the DID connection is established, a key verification mechanism, such as manually comparing the public keys using a hash or QR-code, could have been utilized to validate and authenticate the other device [49]. We will not go into detail of the privacy and security of BLE which has already been analyzed to great extent [1,42].

When the BLE connection is established, the DIDs can be exchanged over this BLE connection. This process consists of two cases. First, if the devices have not been connected before, they need to exchange public keys. The key exchange is performed when the DID documents are shared between the parties using the DID connection exchange protocol as described earlier in order to establish a secure DID connection. This will leave each peer knowing each other's did:key and the public key which can be used to encrypt the messages during device messaging at the application layer. Second, if the public keys have already been exchanged, the devices can choose to reuse the DID connection. After the Bluetooth connection is established and the DID exchange protocol has successfully completed, then they become each other's contact, and messages can be securely exchanged between them using end-to-end encryption (E2EE) in a trusted manner providing both message encryption and authentication. E2EE protects content from being read by anyone who does not have the private key required for decryption. Having encryption at the application level is recommended as one of the security controls defined in the National Institute of Standards and Technology (NIST) Bluetooth Security Guide [61].

Another security and privacy-preserving method is DID rotation. A DID rotation can be performed in order to change from one DID to another DID. Although it is a supported DIDcomm feature, it has not been implemented in the proof-of-concept application. One could, for example, perform a DID rotation during the DID exchange protocol because the OOB invitation message is sent unencrypted and therefore the DID used in the invitation should not be considered private.

In addition, regularly rotating the keys is advised for minimizing risk in the event of a key compromise. Any updates to the cryptographic keys in the DID document is handled by the corresponding DID method. However, because did:key does not support updates to the DID document without changing the DID itself, a DID rotation is required if keys must be updated. An alternative approach is to use the did:peer method which support updates to the DID document.

This paragraph provides an explanation on how a message is transferred securely from the sender to the receiver with respect to the used protocols. Each message is sent as packets over a BLE connection either using BLE pairing or not. The Bluetooth pairing and DIDComm encrypted messages enforce security on different layers. Since encryption can be applied in either of these layers (the transport layer or application layer), several combinations of encryption are possible:

- Bluetooth pairing and non-encrypted DIDComm message
- Bluetooth pairing and encrypted DIDComm message
- No Bluetooth pairing and DIDComm non-encrypted message
- No Bluetooth pairing and DIDComm encrypted message

In analogy to how TLS may be used to provide additional security by applying transport layer encryption to DIDComm messages sent over HTTP, Bluetooth pairing can be used to encrypt the channel between two connected BLE devices and provide device authentication, while at the same time encryption of the DIDComm messages ensures security at the application level. Both BLE and DIDComm require a connection to first be established between two devices before a secure communication channel can be provided. In the proposed architecture, a secure Bluetooth connection is first established, followed by a exchange of peer DIDs over this BLE channel that relies on the security provided by this channel. Furthermore, replay attacks can be prevented by identifying each message using an id that can later be used to check if the message has previously been received.

Encrypted messages can be sent at the application layer after the DIDs have been shared. These messages follow the format as specified in the DIDComm messaging specifications and can use different encryption schemes depending on the packing mode. It also supports message authentication.

The JOSE (JSON Object Signing and Encryption) [62] framework consists of numerous standardized algorithms and formats for representing JSON data in encrypted or signed form. For encrypted content, JOSE follows the JWE specification, while signed content is formatted according to JWS, and the cryptographic algorithms is specified within JWA. DIDComm messages are based on JOSE and use a subset of the JSON Web Algorithms for encrypting the messages with standard methods, such as ECDH.

The supported encryption schemes are defined in the specifications and will not be covered in detail here. The proof-of-concept implementation has not been tested for vulnerabilities. Messages are encrypted using the "pack" function as provided by the Veramo framework. Upon receiving a message, the message payload is decrypted using the "unpack" function.

Interoperability. Interoperability is a critical feature of any SSI solution in order for it to be as widely used as possible, and it can exist at several levels. Here the focus will be on the technological interoperability and the various layers of the Trust over IP stack [2]. In addition, the high-level design given in Section 5 demonstrates how the system can be interconnected with other systems. The architecture allows for loose connectivity between components, allowing it to work with a variety of technologies. Other wireless technology, for instance, could be utilized to send a message from one device to another.

Many alternative DID approaches have been developed for various reasons [15]. However, for creating and resolving DIDs, the proof-of-concept application presently only supports the did:key method. This means the solution is incompatible with DIDs created with other methods, such as the did:peer method.

JSON Schema [63] provides a mechanism for validating the data structure and content of a verifiable credential in a standard way. A schema is responsible for defining the template used in creating verifiable credentials that are interoperable with credentials that utilize the same schema. In the proposed architecture for access control, schemas are also used as a template for issuing the access control verifiable credentials.

JSON-LD [64] is a lightweight serialization format for linked data based on JSON. It has been designed for web-based environments and compatible with systems that use JSON to assure compatibility. W3C creates the specifications, and the current version is 1.1. Linked data allows data to be linked in a way that machines can understand and generates a network of data that can be followed by links throughout the web. In general, the JSON-LD document's data model can be represented as a labeled directed network with nodes connected by arcs, where the node can either be a resource with properties or the data values of the property.

Using JSON-LD could make the proposed system more interoperable, however as JSON-LD context is usually retrieved from the Internet the system will not function if the Internet connectivity is unavailable. In contrast to a JWT certificate, JSON-LD relies on the contexts to verify the documents. Caching remote contexts in order to make them accessible for future use when the Internet is unavailable is one method to tackle this issue. This is also a recommended strategy for maintaining privacy and avoiding tracking. A cache of all the contexts used during access control is made available in the implementation, eliminating the requirement for remote fetching of the relevant contexts.

Different systems may use various contexts as specified by JSON-LD. Since verifiable credentials allow specifying the context of a VC, it is important that organizations define contexts that are interoperable across domains. In order to implement the access control verifiable credential, a new context was created specifically for it. However in other scenarios this context should be considered and be carefully defined according to each organization's needs in order to be widely usable.

A verifiable credential's data and proof formats can differ, and they can be represented in JSON or JSON-LD. The VCs and VPs are represented using JSON-LD in this implementation. For enforcing the data model, a VC can employ a variety of technologies. It may include context, schema, and type properties, each of which serves a different purpose. The type parameter is used to identify the VC and its collection of claims. Multiple types can be given in a single VC, but all credentials must have the type "VerifiableCredential". A verifier, for example, can ask for VC that are of a specific type. This type property could be used by a JSON-LD processor to enforce the VC's semantics.

The context attribute is used to express the meaning of the data in the VC in a way that a computer can understand. A VC can have many contexts, however the "https://www.w3.org/2018/credentials/v1" context is required for all VCs. The credential schema, on the other hand, serves as a template for issuing credentials by defining the credential structure and related data types for each property within the VC.

Both the context and credential schema properties can be used to define the structure, but when used together, they provide more assurance about the semantics and data types required [59]. In this current implementation we use a JSON-LD processor to enforce the semantics of the VCs and VPs. In addition, the JSON schemas are used in the implementation by verifier and holder to find candidate and validate credentials that meet the requirements of the presentation definition.

Because there is no agreed-upon standard, the developed application relies on the custom DIDComm BLE library for message transfer, which may cause compatibility issues with other solutions that use BLE for agent-to-agent communication. It contains several steps. First, there is the device discovery and identification process. Second, a common service and characteristics should be defined in order for devices to perform service discovery in an interoperable manner. Third, an agreement on the MTU and whether pairing is required needs to be determined. All of these elements have the potential to make the task more complicated. As a result, having a standard for employing BLE as the transport layer for DIDComm is essential. DIF has a page on GitHub that details a method for accomplishing DIDComm via Bluetooth [46], which relies on a method of discovering devices that support the DIDComm protocol using a shared UUID. In our implementation, however, we used a random self-generated BLE service UUID to identify the advertising device linked with a DID, which can also be stored in the service endpoint as an address (the BLE service UUID) for reaching this device. Because the Bluetooth Low Energy standard does not include official support for serial ports, multiple custom profiles [65] have been created to enable asynchronous serial communication between devices.

The DIDComm messaging protocol defines the core data format for the messages. It defines the formats and types of messages. The implemented application uses the DIDComm v2 reference implementation and should therefore be interoperable with other agents that use the same implementation on this level, although this is not tested. Moreover, because DIDComm relies on other protocols to achieve higher-level functionality it needs to implement these as well in order to achieve those specific functionality. For instance, the presentation exchange protocol.

Technology. Using Bluetooth Low Energy as the wireless technology for transferring the messages is advantageous because it is built into the majority of mobile devices. It is widely used in IoT and is energy efficient. It does not require other external infrastructure. Combining the DIDComm messaging protocol with Bluetooth Low Energy as the transport layer allows messages to be transmitted wirelessly while retaining the DIDComm benefits for secure and authenticated messages at the application-layer. Because the devices must be in close proximity to each other in order to communicate, only these devices may be able to use the access control system. However, if the devices are not within range, they will be unable to use the system.

Constrained devices introduce additional challenges when integrating SSI. Some of these have already been pointed out by Fedrecheski et al. [39]. These are related to asymmetric cryptography, communication overhead, DID resolution, traceability and the available frameworks for building the software.

NFC-tags were used in the approach to hold the OOB invitation used to bootstrap the connection process. It included using a NTAG215 with 492 bytes available for storage. After the base64 URL encoding the OOB message and writing it to the tag, 463 bytes of the total 492 bytes available was used. This shows that the invitation was able to fit the NFC-tag, however the OOB contains some overhead which could have reduced the necessary bytes. Also, since DIDComm relies on JSON for serialization, it causes some data overhead. Fedrecheski et al. [66] also proposed a method for reducing the data overhead of DID documents and DIDComm which could be utilized in the next iterations of PoC application.

Most authentication systems rely on both parties having an active Internet connection during the operation [3]. In our proof-of-concept implementation, instead of using a public-ledger to resolve DIDs, the other peer's DID documents are saved on their respective devices. This allows the system to function without connectivity to the Internet, which was one of the main objectives. However, without the Internet connectivity public peer discovery is not possible. To exchange DIDs, the system relies on two devices being in close proximity to one another. This is mostly used for one-to-one and one-to-many connections when Bluetooth protocol is used. In the absence of a Bluetooth connection, secure communication can still take place between the DIDs, because it may be contacted using other methods as specified in the service endpoint, for example NFC.

We concentrated on creating a server-less agent that operates without connection to the Internet. This was made possible by using a smartphone-based edge agent and the protocols outlined above. The proof-of-concept focuses on the Android platform, and it was created using Native Module for Android to provide the functionality required for BLE operations. The React Native framework consumes the API offered by this Native Module. Other Native Modules might be developed on other platforms to give the same BLE capabilities, which could then be used by React Native to make the implementation cross-platform.

Some of the technology used was experimental or in the early stages of development. However, we were able to evaluate the performance of messaging and access control over BLE. This research focused on employing a mobile application to enable access control between two devices in an offline environments. The same concepts might be applied to various categories of access control systems. However, more research into how the system functions in environments that should be able to handle multiple devices at the same time is still needed.

In addition, the system's access control was based on the RBAC model. As the number of roles grows within the RBAC, the complexity increases making it more difficult to manage. An Attribute-Based Access Control (ABAC) model could be used instead to have more flexibility [67]. Instead of simply assigning roles, the access control verifiable credential might be extended to allow an issuer to define any number of attributes within the VC. Based on these attributes, the verifier can decide whether authorization is given or denied.

10. Conclusion

The full potential of a system for achieving offline decentralized messaging and access control has yet to be explored in current research into SSI ecosystems. Our project investigated an offline messaging and access control case between two devices that used open SSI standards and Bluetooth Low Energy as the wireless communication technology. A high-level architecture was developed to show how the system supports loose coupling and interoperability with existing protocols. The offline mode was possible due to the decentralized nature of self-sovereign identity and DIDComm. Identity and access management procedures are carried out locally on peer devices rather than by a central authority or a third party, for which the Internet connectivity is necessary. Using a combination of emerging open SSI protocols together with Bluetooth Low Energy as the wireless technology, a system for secure and trusted peer-to-peer communication has been demonstrated. For achieving the secure transport of messages and the credentials exchanged between the devices, DIDComm messaging protocol was used. The request and verification of credentials was made using the presentation exchange protocol. We assessed the performance of the overall system. Other systems with equivalent capabilities were not available, therefore doing direct comparison of the performance was not possible. However, the performance findings show that common data flows between agents are feasible for practical applications, though there are still areas where improvements could be made, such as initial bootstrapping time, improved throughput, more formal analysis of privacy and security properties. Finally, we determined that the most difficult stage is bootstrapping the initial connection required to establish a secure communication channel, which includes ensuring that a device connects to the correct DID device.

11. Future work

We hope that this research will be useful to future developers who want to incorporate self-sovereign identity functionality into their applications. By building on the insights gained from this study, it may open up new opportunities in other domains that can extend the system to other use cases. Future IoT studies looking into decentralized secure messaging or identity and access management could benefit from this as well. Furthermore, while this work focuses on the performance of using BLE as the transport mechanism for agent-to-agent communication, the access control functionality given in this project is rather simplistic; more advanced models can be investigated. It would also be interesting to look into how this could be extended to environments with many devices, as the current project focuses on communication between two devices.

Future research could delve more into the mechanisms for bootstrapping the connection used for establishing a secure peer-to-peer DID connection with a device in an offline environment using Bluetooth Low Energy or other technologies, as well as looking into performance optimizations. It is also possible to apply the technique to other peer-to-peer wireless technologies such as NFC and Wi-Fi Direct.

CRediT authorship contribution statement

Alexander Enge: Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing – original draft, Writing – review & editing. **Abylay Satybaldy:** Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing – original draft, Writing – review & editing. **Mariusz Nowostawski:** Conception and design of study, Analysis and/or interpretation of data, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgment

All authors approved the version of the manuscript to be published.

References

[1] A. Barua, M.A. Al Alamin, M.S. Hossain, E. Hossain, Security and privacy threats for bluetooth low energy in IoT and wearable devices: A comprehensive survey, IEEE Open J. Commun. Soc. (2022).
[2] M. Davie, D. Gisolfi, D. Hardman, J. Jordan, D. O'Donnell, D. Reed, The trust over ip stack, IEEE Commun. Stand. Mag. 3 (4) (2019) 46–51.
[3] A. Abraham, S. More, C. Rabensteiner, F. Hörandner, Revocable and offline-verifiable self-sovereign identities, in: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), IEEE, pp. 1020–1027.
[4] G. Laatikainen, T. Kolehmainen, P. Abrahamsson, Self-sovereign identity ecosystems: Benefits and challenges, in: Scandinavian Conference on Information Systems, Association for Information Systems.
[5] G. Laatikainen, T. Kolehmainen, M. Li, M. Hautala, A. Kettunen, P. Abrahamsson, Towards a trustful digital world: exploring self-sovereign identity ecosystems, 2021, arXiv preprint arXiv:2105.15131.
[6] R. Soltani, U.T. Nguyen, A. An, A survey of self-sovereign identity ecosystem, Secur. Commun. Netw. 2021 (2021).
[7] A. Mühle, A. Grüner, T. Gayvoronskaya, C. Meinel, A survey on essential components of a self-sovereign identity, Comp. Sci. Rev. 30 (2018) 80–86.
[8] Statista, Share of internet users in Africa as of december 2020, 2020, Available: https://www.statista.com/statistics/1124283/internet-penetration-in-africa-by-country/, Accessed 22 March 2022.
[9] W3C Credential Community Group, Decentralized identifiers, 2022, Available at https://www.w3.org/TR/did-core/, Accessed 13 May 2022.
[10] W3C, Verifiable credentials data model 1.0, 2022, Available at https://www.w3.org/TR/vc-data-model/, Accessed 20 May 2022.
[11] DIF, DIDComm Messaging Specification, 2022, https://identity.foundation/didcomm-messaging/spec/, Accessed 28 March 2022.
[12] Decentralized Identity Foundation (DIF), DIF presentation exchange, 2022, https://identity.foundation/presentation-exchange/, Accessed 18 June 2022.
[13] A.H. Enge, A. Satybaldy, M. Nowostawski, An architectural framework for enabling secure decentralized P2P messaging using DIDComm and Bluetooth Low Energy, in: 2022 IEEE 46th Annual Computers, Software, and Applications Conference, COMPSAC, IEEE, 2022, pp. 1579–1586.
[14] G. Kellogg, P.-A. Champin, JSON-LD 1.1–A JSON-based serialization for linked data (W3C working draft), Proposed Standard (2019).
[15] W. Fdhila, N. Stifter, K. Kostal, C. Saglam, M. Sabadello, Methods for decentralized identities: Evaluation and insights, in: International Conference on Business Process Management, Springer, pp. 119–135.
[16] W3C, The did:key method v0.7, 2022, Available at https://w3c-ccg.github.io/did-method-key/, Accessed 10 May 2022.
[17] DIF, Peer DID method specification, 2022, https://identity.foundation/peer-did-method-spec/, Accessed 8 May 2022.
[18] DIF, Decentralized identity foundation, 2022, Available at https://identity.foundation, Accessed 10 March 2022.
[19] Internet Engineering Task Force (IETF), JSON web token, 2022, https://datatracker.ietf.org/doc/html/rfc7519, Accessed 4 June 2022.
[20] A. Preukschat, D. Reed, Self-Sovereign Identity, Manning Publications, 2021.
[21] Hyperledger, Aries RFC 0046: Mediators and relays, 2020, https://github.com/hyperledger/aries-rfcs/tree/main/concepts/0046-mediators-and-relays, Accessed 4 March 2022.
[22] Internet Engineering Task Force (IETF), JSON web message, 2022, https://tools.ietf.org/id/draft-looker-jwm-01.html, Accessed 4 June 2022.
[23] Internet Engineering Task Force (IETF), JSON web signature, 2022, https://datatracker.ietf.org/doc/html/rfc7515, Accessed 4 June 2022.
[24] IETF, JSON web encryption, 2022, https://datatracker.ietf.org/doc/html/rfc7516, Accessed 4 June 2022.
[25] IETF, JSON web algorithms, 2022, https://datatracker.ietf.org/doc/html/rfc7518, Accessed 4 April 2022.
[26] Internet Engineering Task Force (IETF), Public key authenticated encryption for JOSE: ECDH-1PU, 2022, https://datatracker.ietf.org/doc/html/draft-madden-jose-ecdh-1pu-04, Accessed 4 August 2022.

[27] Internet Engineering Task Force (IETF), Key agreement with elliptic curve diffie-hellman ephemeral static (ECDH-ES), 2022, https://datatracker.ietf.org/doc/html/rfc7518#section-4.6, Accessed 4 August 2022.

[28] N. Mohammadzadeh, S. Dorri Nogoorani, J.L. Muñoz-Tapia, Decentralized factoring for self-sovereign identities, Electronics 10 (12) (2021) 1467.

[29] H. Kasyap, S. Tripathy, Privacy-preserving decentralized learning framework for healthcare system, ACM Trans. Multimed. Comput. Commun. Appl. (TOMM) 17 (2s) (2021) 1–24.

[30] P. Papadopoulos, W. Abramson, A.J. Hall, N. Pitropakis, W.J. Buchanan, Privacy and trust redefined in federated machine learning, Mach. Learn. Knowl. Extr. 3 (2) (2021) 333–356.

[31] N. Prakash, D.G. Michelson, C. Feng, Cvin: Connected vehicle information network, in: 2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring), IEEE, pp. 1–6.

[32] Sovrin Foundation S.S.I. in IoT Task Force, SSI-and-IoT-whitepaper, 2020, https://sovrin.org/wp-content/uploads/SSI-and-IoT-whitepaper.pdf, Accessed 4 March 2022.

[33] Bluetooth Special Interest Group, Bluetooth® core specification, 2021, https://www.bluetooth.com/specifications/bluetooth-core-specification/, Accessed 8 March 2022.

[34] L. Nao, BLE pairing and bonding, 2018, https://www.kynetics.com/docs/2018/BLE_Pairing_and_bonding/, Accessed 4 March 2022.

[35] Z.A. Lux, D. Thatmann, S. Zickau, F. Beierle, Distributed-Ledger-based authentication with decentralized identifiers and verifiable credentials, in: 2020 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services, BRAINS, IEEE, pp. 71–78.

[36] N. Fotiou, V.A. Siris, G.C. Polyzos, Capability-based access control for multi-tenant systems using oauth 2.0 and verifiable credentials, 2021, arXiv preprint arXiv:2104.11515.

[37] D. Lagutin, Y. Kortesniemi, N. Fotiou, V.A. Siris, Enabling decentralised identifiers and verifiable credentials for constrained iot devices using oauth-based delegation, in: Workshop on Decentralized IoT Systems and Security, Internet Society.

[38] R. Belchior, B. Putz, G. Pernul, M. Correia, A. Vasconcelos, S. Guerreiro, Ssibac: self-sovereign identity based access control, in: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), IEEE, pp. 1935–1943.

[39] G. Fedrecheski, J.M. Rabaey, L.C. Costa, P.C.C. Ccori, W.T. Pereira, M.K. Zuffo, Self-sovereign identity for iot environments: a perspective, in: 2020 Global Internet of Things Summit (GIoTS), IEEE, pp. 1–6.

[40] P.C. Bartolomeu, E. Vieira, S.M. Hosseini, J. Ferreira, Self-sovereign identity: Use-cases, technologies, and challenges for industrial iot, in: 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), IEEE, pp. 1173–1180.

[41] M. Grabatin, W. Hommel, Self-sovereign identity management in wireless ad hoc mesh networks, in: 2021 IFIP/IEEE International Symposium on Integrated Network Management (IM), IEEE, pp. 480–486.

[42] M. Cäsar, T. Pawelke, J. Steffan, G. Terhorst, A survey on bluetooth low energy security and privacy, Comput. Netw. (2022) 108712.

[43] J. Tosi, F. Taffoni, M. Santacatterina, R. Sannino, D. Formica, Performance evaluation of bluetooth low energy: A systematic review, Sensors 17 (12) (2017) 2898.

[44] M.R. Albrecht, J. Blasco, R.B. Jensen, L. Mareková, Mesh messaging in large-scale protests: Breaking bridgefy, IACR Cryptol. ePrint Arch. 2021 (2021) 214.

[45] M.T. Schoolfield, Message transfer framework for mobile devices using bluetooth low energy (Ph.D. thesis), 2015.

[46] Decentralized Identity Foundation (DIF), DIDComm over Bluetooth, 2021, https://github.com/decentralized-identity/didcomm-bluetooth/blob/main/spec.md, Accessed 8 February 2022.

[47] A.M. Davis, Operational prototyping: A new development approach, IEEE Softw. 9 (5) (1992) 70–78.

[48] S. Cucko, M. Turkanovic, Decentralized and self-sovereign identity: Systematic mapping study, IEEE Access 9 (2021) 139009–139027.

[49] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, M. Smith, SoK: secure messaging, in: 2015 IEEE Symposium on Security and Privacy, IEEE, pp. 232–249.

[50] National Institute of Standards and Technology (NIST), Access control system - glossary | CSRC, 2022, https://csrc.nist.gov/glossary/term/access_control_system, Accessed 28 May 2022.

[51] S. Dramé-Maigné, M. Laurent, L. Castillo, H. Ganem, Centralized, distributed, and everything in between: Reviewing access control solutions for the iot, ACM Comput. Surv. 54 (7) (2021) 1–34.

[52] Hyperledger.org, Aries RFC 0454: Present proof protocol 2.0, 2021, https://github.com/hyperledger/aries-rfcs/tree/main/features/0454-present-proof-v2, Accessed 28 May 2022.

[53] D. Reed, Hyperledger aries: The next major step towards interoperable SSI, 2019, Available at https://www.evernym.com/blog/hyperledger-aries/, Accessed 18 April 2022.

[54] Spruce, 2022, Available at https://www.spruceid.com/, Accessed 18 May 2022.

[55] Trinsic, 2022, Available at https://trinsic.id/, Accessed 18 April 2022.

[56] DIF, DIDComm Messaging V2, 2022, https://github.com/decentralized-identity/didcomm-messaging, Accessed 20 March 2022.

[57] React Native, Native modules intro react native, 2022, https://reactnative.dev/docs/native-modules-intro, Accessed 28 May 2022.

[58] Android, Bluetooth low energy | android open source project, 2022, https://source.android.com/devices/bluetooth/ble, Accessed 4 March 2022.

[59] The World Wide Web Consortium (W3C), Verifiable credentials data model v1.1, 2022, https://www.w3.org/TR/vc-data-model/, Accessed 8 April 2022.

[60] M. Afaneh, Bluetooth 5 speed: How to achieve maximum throughput for your ble application, 2017, https://www.novelbits.io/bluetooth-5-speed-maximum-throughput/, Accessed 4 March 2022.

[61] B. Oniga, V. Dadarlat, A. Munteanu, Application-level authentication and encryption atop bluetooth stack for sensitive data communication, in: 2020 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR), IEEE, pp. 1–5.

[62] Internet Engineering Task Force (IETF), Javascript object signing and encryption (jose), 2022, https://datatracker.ietf.org/wg/jose/charter/, Accessed 13 October 2022.

[63] Internet Engineering Task Force (IETF), JSON schema: A media type for describing JSON documents, 2020, https://json-schema.org/draft/2020-12/json-schema-core.html, Accessed 28 May 2022.

[64] The World Wide Web Consortium (W3C), JSON-LD 1.1, 2020, https://www.w3.org/TR/json-ld11/, Accessed 28 May 2022.

[65] A. Letourneau, Bluetooth low energy serial: A valid design strategy?, 2019, https://punchthrough.com/serial-over-ble/, Accessed 8 March 2022.

[66] G. Fedrecheski, L.C. Costa, S. Afzal, J.M. Rabaey, R.D. Lopes, M.K. Zuffo, A low-overhead approach for self-sovereign identity in IoT, 2021, arXiv preprint arXiv:2107.10232.

[67] K. Andersson, I. You, F. Palmieri, Security and privacy for smart, connected, and mobile IoT devices and platforms, Secur. Commun. Netw. 2018 (2018) 5346596, http://dx.doi.org/10.1155/2018/5346596.

Alexander Enge recently graduated from the Norwegian University of Science and Technology with a M.Sc. degree in Applied Computer Science, where he specialized in decentralized technologies and self-sovereign identity systems. He has a B.Sc. degree in Computer Engineering. Alexander is currently working as a software developer and enjoys programming, always aiming to deliver high-quality software for the user.

Abylay Satybaldy is a Ph.D. candidate at Computer Science department of Norwegian University of Science and Technology. His M.Sc. studies were focused on computer networks. He is currently working on decentralized identity systems and privacy-enhancing technologies.

Mariusz Nowostawski is an Associate Professor at Norwegian University of Science and Technology. Previously, an academic lecturer at University of Otago, New Zealand. His M.Sc. studies were focused on AI and machine learning, and his Ph.D. on autonomous systems and computational modeling of the biological process of life. Passionate about self-organizing systems, adaptive and autonomous computation. Mariusz has worked on high-end networking applications on GPUs and multicore systems with Sun Microsystems and Oracle. He is currently involved in forensics research with Europol. Bitcoin anonymity. Cryptocurrencies.