International Journal of Information Security
https://doi.org/10.1007/s10207-024-00864-6

REGULAR CONTRIBUTION

Decentralised identity federations using blockchain

Mirza Kamrul Bashar Shuhan[1] · Syed Md. Hasnayeen[1] · Tanmoy Krishna Das[1] · Md. Nazmus Sakib[2] · Md Sadek Ferdous[3,4]

Abstract
Federated Identity Management offers numerous economic benefits and convenience to Service Providers and users alike. In such federations, the Identity Provider (IdP) is the solitary entity responsible for managing user credentials and generating assertions for the users, who are requesting access to a service provider's resource. This makes the IdP centralised and exhibits a single point of failure for the federation, making the federation prone to catastrophic damages. The paper presents our effort in designing and implementing a decentralised system in establishing an identity federation. In its attempt to decentralise the IdP in the federation, the proposed system relies on blockchain technology, thereby, mitigating the single point of failure shortcoming of existing identity federations and is designed using a set of requirements. In this article, we explore different aspects of designing and developing the system, present its protocol flow, analyse its performance, and evaluate its security using ProVerif, a state-of-the-art formal protocol verification tool.

Keywords Identify federation · Federated identities · SAML · Decentralised identity federation · Blockchain · ProVerif

1 Introduction

We live in a world that is rapidly undergoing a fundamental change: its different aspects are being digitally transformed. This is not just the Internet of Things (IoT) or mobile computing, digital transformations are happening to all societal systems- traffic, health, government, logistics, education, marketing, etc. Consequently, nowadays, more and more crucial service providers (SP) have opted to go for digital operations and require their users to register to their systems. This is required to ensure a personalised user experience for the user. To ensure security, users need to choose and later utilise a crucial attribute known as the *credential*, with the password being the most widely used credential in the world [39]. With the growing number of service providers, managing each user's credentials is becoming a challenging task [19].

A solution to this challenge has resulted in the creation of Identity Management Systems (IMS). Among many IMS, Federated Identity Management (FIM) is a popular IMS, particularly within Educational and Government settings. FIM facilitates the creation of Identity Federations, a trusted virtual boundary among different entities for sharing their resources [31]. Within an identity federation, there is a central entity called an Identity Provider (IdP). There could be multiple service providers (SPs) where each SP relies on the identity service provided by the IdP. Security Assertion Markup Language (SAML) is a major standard for creating and maintaining an identity federation. An identity federation provides several advantages, such as trusted service provisioning, Single-Sign-On (SSO) experience, reduced password management, improved role-based access, and other identity management activities [11], thus proving beneficial for both users and SPs. However, FIM suffers from a

✉ Md Sadek Ferdous
sadek.ferdous@bracu.ac.bd

Mirza Kamrul Bashar Shuhan
shuhan.mirza@gmail.com

Syed Md. Hasnayeen
rummanhasnayeen94@gmail.com

Tanmoy Krishna Das
tanmoykrishnadas@gmail.com

Md. Nazmus Sakib
mdnazmus.sakib@utdallas.edu

1 Shahjalal University of Science and Technology, Sylhet, Bangladesh

2 The University of Texas at Dallas, Dallas, USA

3 BRAC University, Dhaka, Bangladesh

4 Imperial College London, London, UK

Published online: 16 May 2024

crucial issue: the IdP being the central component introduces a single point of failure. If the IdP ceases to function, the SPs of the identity federation cannot function properly.

A blockchain is an immutable transaction ledger, maintained within a distributed network of peer nodes [12]. These nodes each maintain a copy of the ledger by applying transactions that have been validated by a consensus protocol, grouped into blocks that include a hash that binds each block to the preceding block. This increases the security and integrity of data. In addition, blockchain, due to its decentralised nature, has additional advantages such as distributed data sharing and data availability. In this article we explore a novel idea of integrating blockchain within the architecture of an identity federation as the functionalities of the IdP can be decentralised, thereby creating the notion of a decentralised identity federation.

Contributions. The main contributions of the article are:

- The elaboration of the idea of a blockchain-based decentralised identity federation.
- An architecture of the proposed system based on a rigorous threat model and requirement analysis.
- A Proof of Concept (PoC) implementation of the presented architecture using a state-of-the-art private blockchain network.
- Detailed performance analysis of the implemented PoC, showcasing its applicability.
- Rigorous security verification of the underlying protocol of the implemented PoC using ProfVerif, a state-of-the-art protocol verifier [7, 8].

Structure. We present a brief background on federated identity management, blockchain, and their different aspects in Sect. 2. We present our proposal of a decentralised identity federation along with a threat modelling and requirement analysis in Sect. 4. We discuss different components of the architecture of the system and its implementation details in Sect. 5. The protocol flow of the PoC is illustrated in Sect. 6. In Sect. 7, the performance of the developed PoC is evaluated against several blockchain network configurations and user loads. In Sect. 8, we discuss how the proposed system has satisfied different requirements, formally verify the protocol and discuss the advantages of the proposed system. Finally, we conclude in Sect. 9.

2 Background

In this section, we briefly discuss different aspects of Federated Identity Management (Sect. 2.1) and blockchain (Sect. 2.2).

2.1 Federated identity management

As per [22], an entity is a physical or logical object which can be uniquely identified within a certain context with its own *identity*. Federated Identity Management (FIM) can be considered a business and identity management model in which two or more trusted parties agree to an association through a technical contract to facilitate Identity Management. It also consists of functions and protocols to provide assurance regarding the identity of an entity (a user) for the purpose of authentication, authorisation, and service provisioning [20].

FIM enables a user to access restricted resources seamlessly and securely from different organisations in different Identity Domains. An identity domain is the virtual boundary, context, or environment in which a digital identifier is valid [30]. An identifier within an identity domain is an attribute whose value can be used to uniquely identify a user within that identity domain. Examples of identifiers are username and email as their values can uniquely identify a user.

FIM offers a good number of advantages to different stakeholders such as the separation of duties among different organisations, scalability, improved security and privacy, Single Sign On (SSO) for users, and so on [11]. For example, users can take advantage of SSO and thus authenticate themselves in one identity domain and receive personalised services across multiple domains without any further authentication. There are three major actors within an FIM System:

- *Identity Provider (IdP):* An entity that is responsible for managing the digital identities of users and providing identity-related services to different Service Providers. IdPs are also known as Asserting Parties (AP).
- *Service Provider (SP):* An entity that is responsible for providing online services to the users based on the identity information (identifiers and/or attributes) received from the IdP. SPs are also known as Relying Parties (RP).
- *Users:* An entity that receives services from an SP.

SAML-based FIM: One crucial component of an identity federation is how trust is established among different organisations. SAML (Security Assertion Markup Language) is the most widely used technology for establishing trust among organisations and deploying identity federations among themselves [9, 18]. SAML is an XML-based standard for exchanging authentication and authorisation information between trusted yet autonomous organisational domains. It is based on the request/response protocol in which one party (generally SPs) requests particular identity information about a user and the other party (IdPs) then responds with the information.

Metadata is a central component in a SAML-based identity federation and plays a crucial role in establishing trust

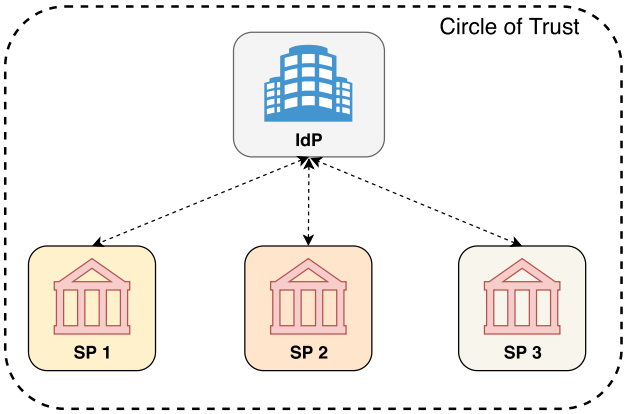


Fig. 1 SAML identity federation

Table 1 Identity, SAML & Cryptographic Notations

Notations	Description
$id_{sp\|idp}$	Entity Id of an of sp or idp
id_{req}	Identifier of an of a SAML request
$K_{sp\|idp}$	Public key of sp or idp
$K^{-1}_{sp\|idp}$	Private key of sp or IdP
$\{\}_K$	Encryption operation using K
$\{\}_{K^{-1}}$	Signature using a private key K^{-1}
AuthnReq	SAML Authentication Request
SAMLAssrtn	SAML Assertion
EncSAMLAssrtn	Encrypted SAML Assertion
SAMLResp	SAML Response

among organisations. It is an XML file in a specified format containing several pieces of information such as entity descriptor (identifier for each party), service endpoints (the locations of the appropriate endpoints for IdPs and SPs), certificate(s) to be used for encryption, the expiration time of metadata, contact information, and other information. To establish a federation, IdPs and SPs exchange their metadata with each other and store them at the appropriate repositories at their ends which helps each party to build up the so-called Trust Anchor List (TAL). IdPs only trust those SPs whose metadata can be found in their TALs and vice versa, thus creating the notion of a trusted relationship, the so-called Circle of Trust (CoT), within FIM. Therefore, the TAL of an IdP consists of the metadata of SPs federated with the IdP and vice versa. Figure 1 represents a SAML identity federation where the dotted lines represent the CoT for the entities within that federation.

A SAML protocol flow between a user (denoted as u), an IdP (denoted as idp) and an SP (denoted as sp) is presented below using the notations presented in Tables 1 and 2 [24].

While trying to access a service provided by sp, u is forwarded to a special service called the *Discovery Service* or

Table 2 SAML notations

$AuthnReq \triangleq \langle\, id_{req}, id_{sp} \,\rangle$

$SAMLAssrtn \triangleq \langle\, PROFILE^{u}_{idp} \,\rangle$

$EncSAMLAssrtn \triangleq \langle\, \{SAMLAssrtn\}_{K_{sp}} \,\rangle$

$Assrtn \triangleq \langle\, \{SAMLAssrtn\}_{K^{-1}_{idp}} | \{EncSAMLAssrtn\}_{K^{-1}_{idp}} \,\rangle$

$SAMLResp \triangleq \langle\, id_{req}, id_{sp}, id_{idp}, Assrtn \,\rangle$

Where Are You From (WAYF) Service. The WAYF shows a list of pre-configured trusted IdPs (IDP_{wayf}) to u. After choosing her preferred IdP, u is forwarded to idp with a SAML authentication request consisting of an identifier of the request and an identifier (called *entity ID* in SAML) of sp. A SAML request is denoted using *AuthnReq* and is modelled as presented in Table 2, where id_{req} representing the identifier in each SAML request and id_{sp} denoting the entity ID of sp. At idp, u is authenticated at first, and then idp prepares a SAML response with an embedded SAML assertion. The assertion contains the user profile (explained below) as released by idp. Mathematically, the assertion is denoted with *SAMLAssrtn* and is modelled as per Table 2.

Then, idp digitally signs the (encrypted or unencrypted) SAML assertion and then embeds it inside a SAML response. The response also contains the request identifier (id_{req}), the entity ID (id_{idp}) of idp and the entity ID (id_{sp}) of sp. A SAML response is denoted with *SAMLResp* and modelled as per Table 2.

Finally, the response is sent back to sp by the idp. Upon receiving the response, the (encrypted/unencrypted) SAML assertion is extracted. If the response consists of an encrypted assertion, sp decrypts the assertion at first with the private key of sp (K^{-1}_{sp}) and then validates the signature with the public key of idp (K_{idp}). In case of an unencrypted assertion, sp just validates its signature using the public key of idp. sp retrieves the embedded user attributes from the assertion, only if the signature is valid, otherwise, the assertion is discarded. To ensure privacy, idp does not release all the attributes to sp, instead, a subset of attributes are released. Such attributes are regarded as the profile of a user at idp, denoted with the notation $PROFILE^{u}_{idp}$. Consequently, a SAML assertion is modelled to consist of the profile in Table 2.

Issues in FIM: One crucial bottleneck within a federation is that it is centralised. There are two major implications of this issue. The first implication is that if the IdP within a SAML Identity federation malfunctions, the users of that federation will not be able to access federated services from that domain, thereby exhibiting a single point of failure. The second reason is that, if the IdP within a federation relies on a centralised database to store user credentials and attributes and such storage server does not work properly, the IdP will not be able to provide required identity services to the SPs within the

federation, thereby, causing disruptions towards federated services. In this work, we present a novel blockchain-based approach to tackle both these centralisation issues.

2.2 Blockchain

Because of the zero reliance on any central entity such as a central bank, Bitcoin is often considered as the first successful decentralised digital currency [36]. The technological innovation of Bitcoin is underpinned by a smartly-engineered solution known as *blockchain*. A blockchain is essentially a distributed ledger consisting of transactions which are grouped together using the concept of blocks and these blocks are chained consecutively following a strict set of rules [12]. Each transaction and block are consequently verified by many distributed Peer-to-Peer (P2P) nodes, thereby ensuring that the system can function even in the midst of attacking/corrupting P2P nodes which may not follow the protocol rules properly. Each transaction in the blockchain represents an instruction to transfer value or data from one entity to another. Blockchain offers a number of advantages such as data immutability, data provenance, distributed data sharing and so on. Evolving from Bitcoin, another generation of blockchain system has emerged which supports the deployment of smart contracts on top of the respective blockchain platform. A smart contract (SC) is a computer program which can be executed by a computing platform that is integrated with a blockchain system. Being rooted in blockchain, the code of an SC and the data it stores become an integrated part of an immutable ledger, thereby facilitating the notion of immutable logic, which is a sought-after property in many application domains [23]. A blockchain can be *public*, allowing everyone to participate, or *private*, where only authorised parties can participate. Bitcoin [36] and Ethereum [1] are examples of public blockchains, whereas Hyperledger platforms [28] and Quoram [40] are examples of private blockchain systems.

2.2.1 Smart contract

A smart contract is a computer program that not only defines the rules and penalties around an agreement in the same way that a traditional contract does, but it can also automatically enforce those obligations [45]. These contracts are stored in a decentralised ledger in the blockchain network. Blockchain is ideal for storing smart contracts because of the technology's security and immutability. Ethereum, a decentralised platform for developing, deploying and executing smart contracts, has been instrumental in popularising this concept, offering a programmable blockchain where developers can deploy their own smart contracts. Smart contracts have found applications in various fields, including finance, supply chain management, real estate and healthcare, promising to revolutionise traditional contractual agreements by introducing automation and trust in the respective application domain [1].

2.2.2 Consensus algorithm

Consensus mechanisms in blockchain networks play a critical role in maintaining the integrity and security of distributed ledger systems by ensuring agreement among network participants on the validity of transactions. Various consensus algorithms have been proposed and implemented, each with its own strengths and limitations. The most well-known consensus mechanism is Proof of Work (PoW), introduced by Satoshi Nakamoto in the Bitcoin whitepaper [36]. PoW relies on computational puzzle-solving to validate transactions and achieve consensus. Another prominent consensus algorithm is Proof of Stake (PoS), which selects validators based on the amount of cryptocurrency they hold and are willing to "stake" as collateral, as outlined in the seminal work by King and Nadal [32]. Other consensus mechanisms such as Practical Byzantine Fault Tolerance (PBFT) [10] and Delegated Proof of Stake (DPoS) have also been proposed to address scalability and energy efficiency concerns in blockchain networks [14]. These consensus algorithms continue to evolve as researchers explore novel approaches to achieve decentralised consensus while optimising factors such as security, scalability and environmental sustainability.

2.3 Hyperledger fabric

Hyperledger Fabric is a permissioned blockchain framework developed by the Linux Foundation's Hyperledger project [27]. It is designed to provide a modular and scalable platform for building enterprise-grade blockchain applications. Fabric distinguishes itself from other blockchain platforms by offering features such as support for smart contracts in various programming languages, flexible membership services, and a modular architecture that allows for plug-and-play components tailored to specific use-cases. Fabric employs a novel consensus mechanism called Practical Byzantine Fault Tolerance (PBFT) to ensure the integrity and consistency of transactions, as outlined in its whitepaper [3]. Moreover, Fabric's architecture allows for private transactions and confidential contracts, enabling businesses to maintain data privacy and confidentiality while still leveraging the benefits of blockchain technology. With its emphasis on modularity, scalability, and privacy, Hyperledger Fabric has gained significant traction in various industries, including finance, supply chain management, and healthcare, as a robust platform for developing and deploying enterprise blockchain solutions.

Hyperledger Fabric comprises several key components that collectively enable the development and deployment of

enterprise-grade blockchain applications. We present a brief discussion of these entities in the following.

- Membership Services Provider (MSP): MSP is responsible for managing identities and permissions within the network. It authenticates participants and controls access to resources based on predefined policies. Fabric's MSP model enables flexible membership management, allowing organisations to define their own membership rules and hierarchies.
- Ordering Service: The Ordering Service is responsible for ordering and packaging transactions into blocks, which are then distributed to the peers for validation and execution. In Fabric, the Ordering Service employs a pluggable consensus mechanism, with options including Kafka-based consensus and Solo consensus, providing flexibility and fault tolerance.
- Peer Nodes: Peer nodes maintain a copy of the ledger and execute transactions according to the smart contracts (known as chaincode in Fabric) deployed on the network. Fabric supports two types of peer nodes: endorsing peers, which simulate transaction execution and endorse transaction proposals, and committing peers, which validate endorsed transactions and update the ledger.
- Ledger: The Ledger consists of two main components: the World State and the Blockchain. The World State represents the current state of the ledger, providing efficient access to the latest data. The Blockchain stores the immutable history of transactions in a tamper-evident and append-only manner, ensuring data integrity.
- Chaincode: Chaincode defines the business logic governing transaction processing in Fabric. Written in programming languages such as Go, JavaScript, or Java, chaincode encapsulates the rules and operations specific to the application domain, enabling trustless and transparent execution of transactions.

Kafka-based consensus has emerged as a key component in enhancing the scalability and fault tolerance of Hyperledger Fabric. Originally introduced in version 2.0 of Fabric, Kafka-based consensus leverages Apache Kafka [4], a distributed streaming platform, to improve the ordering service architecture [3]. With Kafka-based consensus, the orderer nodes in Fabric no longer operate in a centralised manner, instead form a distributed ordering service that employs Kafka for message broadcasting and ordering. This distributed approach enhances the resilience of the ordering service, as it mitigates the risk of a single point of failure and enables horizontal scalability by allowing new orderer nodes to be added dynamically. Moreover, Kafka-based consensus improves transaction throughput and latency by parallelising the ordering process across multiple Kafka partitions, thereby distributing the workload more efficiently among orderer nodes.

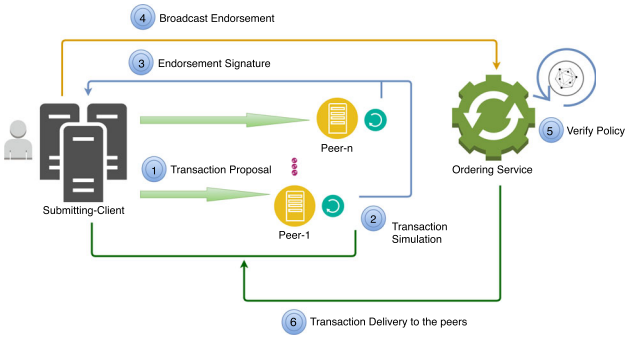


Fig. 2 Transaction flow in hyperledger fabric

The transaction flow in Hyperledger Fabric involves several steps (Fig. 2), ensuring the secure and efficient execution of transactions in a permissioned blockchain network. The following description outlines the typical transaction flow in Fabric [3].

- The client creates a transaction and sends it to endorsing peers of its choice (Step 1 in Fig. 2).
- The endorsing peer simulates a transaction and produces an endorsement signature (Step 2 in Fig. 2).
- The submitting client collects an endorsement for a transaction and broadcasts it through the ordering service (Steps 3 and 4 in Fig. 2).
- The ordering service verifies the transaction against its policy, combines them into a block and delivers the transaction within a block to the peers (Step 5 in Fig. 2).

3 Related work

Even though the concept of utilising blockchain within an identity federation is novel, there have been very few research works on this topic.

ElGayyar et al. presented an idea of a blockchain-based identity federation in [16]. Their idea is to create a completely new type of identity federation marketplace where different entities such as IdPs and SPs could be considered federated, that is within the same circle of trust if they are part of the same blockchain network. They implemented a prototype based on their proposal leveraging two different blockchain platforms: Ethereum private network and Hyperledger Fabric network. Conceptually the idea presented in this article is similar to the idea presented in [16]. However, there is one major difference: our concept is based on the SAML protocol where we modified the SAML components to integrate the blockchain network so as to create a decentralised federa-

tion. This is to ensure that the existing SAML-based SPs are minimally affected. On the other hand, the work presented in [16] presents a completely novel identity federation system which does not use SAML. Creating a new protocol and standard would require a significant security analysis based on a rigorous threat model which is completely missing in [16].

Another paper related to our concept was presented by Alom et al. in [2]. In their work, the authors leveraged the Hyperledger Fabric blockchain platform for creating and managing a SAML-based identity federation in a dynamic fashion. The authors did not explore how a SAML federation could be fully decentralised.

Liu et al. surveyed different blockchain-based identity management systems in [33]. However, they did not look at the possibility of decentralising an identity federation using blockchain and it can be considered loosely relevant.

Mell et al. in [34] presented an alternative approach to federated identity management to eliminate the need for Credential Service Providers or Identity Providers by keeping the user's data in a public blockchain with encrypted attributes. The authentication is done by validating if the user has the private key. The approach completely disrupts the architecture of FIM and cannot be integrated with existing FIMs. Furthermore, updating an attribute comes with a transaction fee which is difficult to integrate with the user experience.

Haddouti et al. in [15] introduced the *Fedidchain* framework which facilitated Attribute Aggregation mechanisms with blockchain technology to address key issues such as lack of interoperability and centralisation of attribute aggregation in the current Identity Federation solution. Through an initial implementation and execution using Shibboleth software and JavaScript, the authors demonstrated the practical feasibility of the Fedidchain framework. However, their motivation was not to decentralise the identity federation as our proposal. Also, it lacked a threat model analysis, security analysis and the security verification of the protocol. Furthermore, they developed their own private blockchain system for their framework. The security of this new blockchain is questionable as it has not been analysed at all.

4 Proposal, threat modelling and requirements analysis

In this section, we present our proposal (Sect. 4.1) and a threat model (Sect. 4.2) and then analyse many functional and security requirements (Sect. 4.3) for the proposed blockchain-based framework for a decentralised identity federation.

4.1 Proposal

To mitigate the identified issue of a single point of failure in an identity federation, we propose to devise a mechanism that will essentially decentralise the functionalities of an IdP. Towards that aim, we propose to disrupt the current setting of an identity federation by introducing a blockchain-based 'inner' federation of many IdPs within a single identity federation. These inner groups of federated IdPs within another single federation combinedly act like a single IdP to any SP within the federation. This novel proposal will require to make changes regarding how an identity federation is established and maintained, and its services are provisioned. Before explaining how we have achieved these goals, we present the threats and the requirements for such a system in the following sections.

4.2 Threat modelling

Threat modelling is a crucial step for designing and developing a secure framework for mitigating threats involving IT assets, and identity federations in the scope of this paper. To model threats, we have chosen a well-established threat model called STRIDE [41], which encapsulates six security threats. Next, we discuss five of these six threats modelled within the scope of the current work. The last threat (the 'E' threat in STRIDE which implies *Elevation of Privilege*) is excluded as it is related to an authorisation which is generally carried out by an SP once it receives an assertion from an IdP and thus, it is beyond the scope of the current work.

- *T1-Spoofing:* An entity (e.g. SP) can pretend to provide a federated service even though it is not part of any federation.
- *T2-Tampering:* An entity (e.g. SP or IdP) might alter the TAL of another entity to be in the CoT without exchanging the required metadata.
- *T3-Repudiation:* An IdP can repudiate that it has not released any assertion to an SP.
- *T4-Information Disclosure:* We consider two different types of T4 threat:
  - *T4-1:* User attributes are disclosed to an unauthorised attacker.
  - *T4-2:* User attributes from an IdP are disclosed to an SP without the user's knowledge or consent.
- *T5-Denial of Service (DoS):* The federated services become unavailable because of an entity (an IdP) being unavailable due to a DoS attack.

Apart from STRIDE threats, we also consider the following additional threats.

– *T7- Replay Attack:* An attacker can capture and reuse any previous SAML packet (request, response, or assertion) for any malicious intent.

4.3 Requirement analysis

Accurate and well-defined requirement analysis is an essential part of any successful application or system development process. Before implementing our proposed approach, we formulated many functional and security (non-functional) requirements. The functional requirements represent the absolutely necessary features of the proposed approach whereas the security requirements are needed to ensure the security of the approach so as to mitigate the identified security threats. Next, we present the requirements.

4.4 Functional requirements

At first, the functional requirements are presented

– *F1*: There should be a mechanism to establish a trusted relationship among different IdPs within another identity federation.
– *F2*: Such IdPs should have the provision to share the attributes of their users with other trusted IdPs.
– *F3*: In case an IdP ceases to function, a mechanism should be established for a user to select one of the other federated IdPs.
– *F4*: The functionalities should be integrated in such a way that it has a minimal impact on any existing components of the SAML-based system.

4.5 Non-functional (Security) requirements

Next, we present the security requirements of the system.

– *S1*: The usual trust requirements of any SAML federation are ensured. This trust requirement ensures that only federated entities can request and avail identity and federated services within a federation, hereby mitigating T1 and only trusted entities can exchange their metadata and store them in their respective TAL, thereby mitigating T2.
– *S2*: Every released assertion must be digitally signed so that the IdP cannot be repudiated. This mitigates T3.
– *S3*: An IdP should always respond with an encrypted assertion (*EncSAMLAssrtn*) for the requesting SP. In general, if all data transmission is carried out in an encrypted channel (HTTPS), then the encrypted assertion requirement can be relaxed. Any of these will mitigate T4-1.
– *S4*: Every user attribute should be released only after the respective user has provided their explicit consent to release such attribute. This will deter threat T4-2.

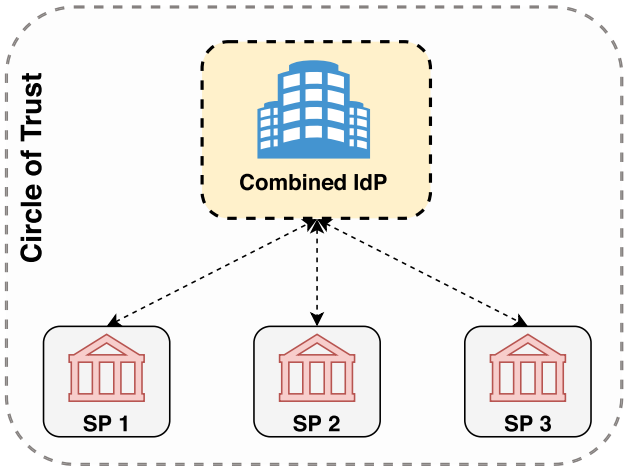


Fig. 3 Decentralised identity federation architecture

– *S5*: An SP within a federation should be able to avail the identity service of an IdP even if it becomes unavailable due to an attack such as DoS. This will mitigate threat T5.

5 Architecture and implementation

In this section, we present the architecture of the proposed system and discuss how the architecture has been implemented.

The architecture of the proposed system is presented in Figs. 3 and 4. Next, we explain the functionalities of different components of this architecture.

5.1 Combined IdP

Our proposal evolves around the idea that a number of IdPs are integrated in such a way that they act like a single IdP to all other SPs within the federation. This is illustrated in Fig. 3 where the integrated IdPs are denoted as the *Combined IdP*. These IdPs are inter-connected with each other using a blockchain platform (Fig. 4). The dotted circle around the Combined IdP indicates that these integrated IdPs essentially form an implicit and combined Circle of Trust (Fig. 4) even though they are not federated with each other in a traditional way (e.g. metadata exchange). Even though we have shown three IdPs in Figs. 3 and 4, we can add a few more IdPs to offer better availability of IdP services.

Each IdP has two sub-components (Fig. 3): SSPHP (SimpleSAMLPHP) and DApp (Decentralised Application). There are several implementations of the SAML standard, such as Shibboleth [13], SimpleSAMLphp [47] and ZXID [51]. Among them, we have used SimpleSAMLPHP (*SSPHP* in short), a SAML implementation developed in PHP, for our

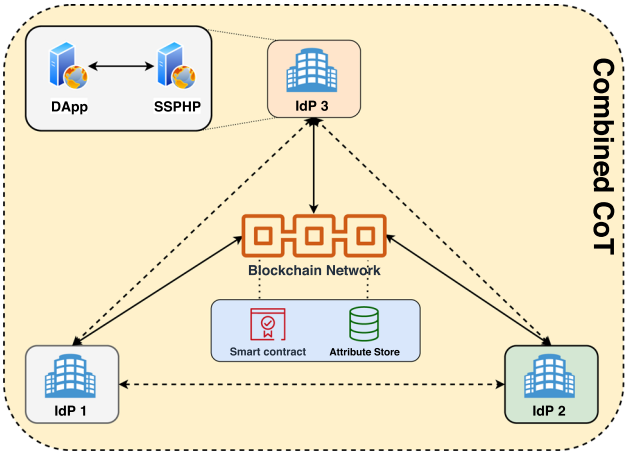


Fig. 4 Individual components inside combined CoT

implementation as it is light-weight, easy to deploy, has additional extensibility using the concept of modules and is fully open-sourced. We have modified the code base of the SSPHP to meet our requirements (explained subsequently). Each of the combined IdPs is attached to this modified SSPHP instance which takes care of most of the SAML functionalities.

A DApp is a web server that acts as the middleware between our SAML IdPs and a blockchain. Since a SAML-based IdP does not have any provision for interacting with the blockchain, we have utilised a DApp, each for one IdP, to interact with the blockchain platform. Each of these DApps exposes APIs to the respective SSPHP instance of the IdP and is also connected to a peer of the blockchain platform. The SAML interface of each IdP uses these APIs to submit some requests (e.g. for some specific purposes explained later) to the DApp. These requests are translated into blockchain transactions by the DApp for submission to the blockchain platform using the blockchain API via a peer.

The DApp in the proposed framework has been developed using Node.js with Express [25, 46]. Node.js is a server-side JavaScript platform that is widely used for creating DApps in the blockchain domain. Express is a web application framework for Node.js, which is used for developing web applications.

5.2 Blockchain platform

The blockchain platform plays a crucial role in achieving the desired functionalities of the proposed system. The blockchain platform serves the following three purposes within the proposed framework:

- Being rooted on top of a decentralised blockchain platform facilitates the provision to establish the notion of trust among the integrated IdPs.
- The distributed data-sharing nature of blockchain provides the underlying mechanism to share the user credentials and other attributes of among the IdPs in a timely and synchronised way. This implies that the blockchain will be used as an attribute store as well.
- The smart contract (SC) in the blockchain is used to encode immutable logic for managing the trust relationship between the IdPs and storing attributes and credentials.

An SC-supported public blockchain (e.g., Ethereum) is relatively slow, open to all and expensive to process and store data [12]. On the other hand, private blockchain systems are private and fast with no issue of energy consumption and provide a reasonable amount of security. Furthermore, a federation is essentially a closed network of trusted entities that highlights the requirement for a private blockchain platform. For these reasons, we have utilised a private blockchain platform.

Among many private blockchain platforms, Hyperledger Fabric is currently the most stable and popular private blockchain platform [6]. It is also equipped with a unique concept of *channel* which allows different fabric blockchains to be maintained within the same network, thus creating a privacy layer among different organisations. That is why Hyperledger Fabric was selected as our preferred blockchain platform during the deployment phase.

The fabric utilises many network entities such as peers, endorsers, and orderers. A smart contract is called a *chaincode* in Fabric terminology which can be invoked using transactions. Transactions are submitted via peers, endorsers are responsible for validating transactions and orderers create blocks.

The chaincode for the proposed framework has been written in Go. The Fabric network contains two organisations, each representing an IdP and an SP. Each organisation consists of two peers/endorsers. We have used Docker containers to deploy the platform with these network configurations. An additional entity called MSP (Membership Service Provider), including a CA (Certificate Authority), is also deployed with another container. All nodes are connected via a channel. We have utilised the Kafka (a distributed event streaming platform [4]) consensus algorithm with two additional orderer nodes for block creation and dissemination.

Service Provider (SP): The Service Providers (SPs) in the proposed architecture function mostly similarly to any traditional SAML SP. However, instead of being federated with a single IdP in the federation, each IdP is federated with all of the combined IdPs. This is used to bootstrap the trust relationship between the combined IdPs and other federated SPs. How it is carried out and the protocol flow involving an SP and the combined IdPs will be explained in Sect. 6.

Table 3 Cryptographic notations for protocol flow

Notations	Description
sp_i	A service provider in the federation
idp_i	An identity provider in the federation
$id_{sp_i \mid idp_i}$	Entity Id of an sp_i or idp_i
CID	Common Entity ID for the combined CoT
A	Admin of the federation
K	A symmetric encryption key
K_A	Public key of A
K_A^{-1}	Private key of A
K_D	Public key of DApp
K_D^{-1}	Private key of DApp
N_i	A fresh nonce
$H(.)$	A hash function
$[]_{https}$	Communication over an HTTPS channel
IDPList	List of currently active IdPs
AttList	List of user attributes and their values
a_i	A attribute name
v_{a_i}	Corresponding value for attribute a_i
msg	A textual message
B	Fabric Blockchain Platform
CC	Fabric Chaincode

To accommodate this protocol flow, we have modified the SP-side code-based of SimpleSAMLPHP as well.

6 Protocol flow and use-case

In this section, we present the protocol flow between different components of the proposed system. Before we illustrate the protocol flow, we introduce the mathematical notations in Table 3 and the data model in Table 4.

6.1 Data model

We start with the request (denoted with req in Table 4), which is submitted to the blockchain platform. req consists of type and data. Here, TYPE denotes the set of different data types within a request and $type \in TYPE$, whereas, DATA represent the set of corresponding data and $data \in DATA$. Both TYPE and DATA are defined as presented in Table 4.

There are two types of registration requests in the system: IdP registration (*idpReg*) and User Registration (*userReg*). *idpReg* signifies the registration of IdPs within the combined IdP set and contains the entity ID of the IdP (id_{idp_i}) and a digital signature as defined with *idpRegData* in Table 4.

On the other hand, the *userReg* signifies that the corresponding request will be a user registration request to an IdP consisting of the data set denoted with *userRegData* in Table 4. In *userRegData*, $h = H(Password)$ denotes the hash of the provided password, *userName* denotes the username (identifier) of the user and *AttList* contains the additional attribute and their values as required during the registration. This implies that a registration request must contain a username, the hash of the password, and additional attributes of the user. *loginData* also has a similar semantic in the sense that a login request must consist of the username and the hash of the provided password.

The *idpQuery* type denotes a special type of request to retrieve a specific IdP from the combined set of IdPs. The corresponding data for this type is *idpQData* which consists of the common entity ID (denoted with *CID*) for the combined set of IdPs. The motivation for utilising such a common entity ID for the combined IdPs is as follows. As mentioned earlier, each IdP within SAML will have a separate entity ID (id_{idp_1}, id_{idp_2}, id_{idp_3} and so on). Utilising all these IdPs might confuse the user. To improve the user experience, the combined IdPs will be externally denoted with *CID* even though internally they will be identified with their respective entity ID. The *idpReg* will be used to bind different IdPs within a common entity ID and the *idpQuery* is used to retrieve the binding entity IDs under a common entity ID.

Additionally, *authn* represents an authentication request whose data is essentially a SAML authentication request (denoted with $AuthnReq$ in Table 2).

A response with respect to a particular request type is denoted with *resp* where a response contains a textual message (msg), a SAML response (*SAMLResp*, as defined in Table 2) and a signature of the concatenated msg and *SAMLResp* by the IdP.

6.2 Protocol flow

In this section, we present the protocol flows involving different entities of the architecture. The protocol flows are divided into three phases: i) IdP Registration & setup phase, ii) User registration phase, and ii) Service provisioning phase.

6.2.1 Registration and setup phase

This is the first step towards creating a decentralised federation where three IdPs are merged as a combined IdP (CoT). An admin of the organisation is assumed to take responsibility for creating the federation. At first, the admin uses the SimpleSAMLPHP framework to set up three IdPs within the same organisation. These IdPs have three different entity IDs and are deployed in three different hosts. Then, these IdPs are connected to the same blockchain network so that they can share the same blockchain. To facilitate this, a DApp is deployed at each IdP. The SimpleSAMLPHP code for each of these IdPs has been modified so that it can interact with their corresponding DApp and via the DApp with the

Table 4 Data Model

$req \triangleq \langle type, data \rangle$
$TYPE \triangleq \langle idpReg, idpQuery, userReg, authn, login, cid \rangle$
$DATA \triangleq \langle idpRegData, idpQData, userRegData, AuthnReq, loginData \rangle$
$IDPList \triangleq \langle id_{idp_1}, id_{idp_2}, ..., id_{idp_n} \rangle$
$idpRegData \triangleq \langle id_{idp_i}, \{id_{idp_i}\}_{K_A^{-1}} \rangle$
$idpQData \triangleq \langle CID \rangle$
$userRegData \triangleq \langle userName, h, \{AttList\}_K \rangle$
$AttList \triangleq \langle \{(a_1, v_1^a), (a_2, v_2^a), ..., (a_n, v_n^a)\}_{K_{idp_i}} \rangle$
$loginData \triangleq \langle userName, h \rangle$
$resp \triangleq \langle msg, SAMLResp, \{msg \| SAMLResp\}_{K_{idp}^{-1}} \rangle$

Fabric blockchain component. This ensures that the SimpleSAMLPHP can be utilised for this as well as for the protocol flows for the other two phases.

To set up the Combined IdP, the smart contract (chaincode) of the blockchain component is utilised. The algorithm for the chaincode is presented in Algorithm 1 in Appendix A. The entry point for the chaincode is the *invoke* function. Every request transmitted via the DApp is transformed into a transaction and is intercepted by the invoke function. Depending on the type of the request, the invoke functions call different functions and the response from the calling function is sent back to the DApp (line 4 to 11 in Algorithm 1). For example, if the type of the request is *idpReg*, which signifies registering an IdP to the combined CoT, then *regIDP* function is called (line 7-8 in Algorithm 1). Similarly, to check if an IdP is part of the combined CoT, the request will contain an *idpQuery* type. In this case, the invoke function will call thee *idpQuery* function.

Next, the admin engages in the following protocol flow to create the notion of the combined CoT out of these three IdPs. The flow is illustrated in Fig. 5 and its corresponding protocol is presented in Table 5.

i Each IdP has a restricted service for registering the IdPs. The service can be accessed only by the admin of the IdP. The admin logs in to that restricted service of one of the IdPs (let us assume that it is denoted with idp_1). This is represented in $M1$ and $M2$ steps in Table 5.
ii The admin provides the entity ID of the logged in IdP and clicks the *Register* button ($M3$ in Table 5).
iii This information is submitted to the corresponding DApp of the IdP ($M4$ in Table 5).
iv The DApp creates and signs a transaction consisting of an IdP registration request. This request contains the type as *idpReg* and data as *idpRegData* as defined in Table 4. This request embedded within the transaction is then submitted to the blockchain to initiate the Fabric flow.
v Once the transaction is approved by the endorsers of the blockchain platform, it is forwarded to the invoke function of the chaincode ($M5$ in Table 5).
vi The invoke function checks the type of the request and as it is an *idpReg* request, it is forwarded to the *regIDP* function (line 6 to 9 in Algorithm 1).
vii The *regIDP* function firstly retrieves the entity ID of the IdP from the request (line 23 in Algorithm 1).
viii Then, the *regIDP* function retrieves the list of registered IdPs against the entity ID of the Combined IdP from the blockchain (line 24 in Algorithm 1). The entity ID of the combined IdP is denoted with CID in the algorithm and is generated when the chaincode is initiated for the first time (line 4 in Algorithm 1).
ix It is checked if the entity ID of the current IdP is already registered against *CID* (the entity ID of the Combined IdP) (line 25 in Algorithm 1). If not, the entity ID is added to the list of the registered IdPs and then the list is saved in the blockchain. In addition, a *TRUE* response is sent back to the DApp (line 26 to 28 and 9 in Algorithm 1). If the current IdP is already registered, a *FALSE* response is sent back to the DApp (line 30 and 9 in Algorithm 1, $M6$ in Table 5).
x The DApp then informs the admin with a meaningful message ($M7$ and $M8$ in Table 5).

The admin goes through the same protocol flow to register other IdPs as well.

6.2.2 User registration phase

In this phase, users are registered to one of the IdPs by the admin of the IdP. As mentioned earlier, the blockchain is used as the attribute store. When a user is registered to an IdP, their credentials and attributes are stored in the blockchain. Since all other IdPs share the same blockchain, all IdPs have access to the user credentials and attributes. It is to be noted that different IdPs may use different protocol flows for this phase.

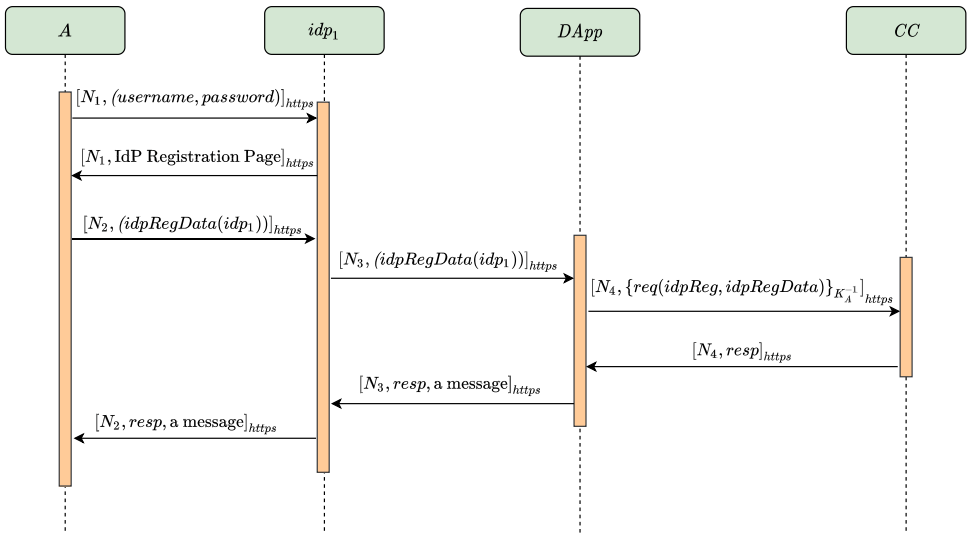


Fig. 5 Protocol Flow for IdP registration and setup

Table 5 Registration & setup protocol

$M1$	$A \rightarrow idp_1$:	$[N_1, username, password]_{https}$
$M2$	$idp_1 \rightarrow A$:	$[N_1, \text{IdP Registration Page}]_{https}$
$M3$	$A \rightarrow idp_1$:	$[N_2, idpRegData \text{ with } id_{idp_1}]_{https}$
$M4$	$idp_1 \rightarrow DApp$:	$[N_3, idpRegData \text{ with } id_{idp_1}]_{https}$
$M5$	$DApp \rightarrow CC$:	$[N_4, \{req(idpReg, idpRegData)\}_{K_A^{-1}}]_{https}$
$M6$	$CC \rightarrow DApp$:	$[N_4, resp]_{https}$
$M7$	$DApp \rightarrow idp_1$:	$[N_3, resp \text{ with a meaningful message}]_{https}$
$M8$	$idp_1 \rightarrow A$:	$[N_2, resp \text{ a meaningful message}]_{https}$

An envisioned protocol flow for this phase is illustrated in Fig. 6 and its corresponding protocol is presented in Table 6. We discuss the envisioned protocol flow next.

i The admin logs in to a restricted page of for the admin to idp_1 after providing their credential ($M1$ and $M2$ steps in Table 6).
ii After a successful login, the restricted page allows the admin to create user credentials such as a username and password for a user along with different attributes. Once completed, the admin clicks a *Register* button ($M3$ in Table 6).
iii When the *Register* button is clicked, the password is hashed, and all attributes are encrypted and submitted along with other information to the corresponding DApp of the IdP ($M4$ in Table 6).
iv The DApp creates and signs a transaction consisting of a user registration request. This request contains the type as *userReg* and data as *userRegData* as defined in Table 4. This request embedded within the transaction is then submitted to the blockchain to initiate the Fabric flow.
v Once the transaction is approved by the endorsers of the blockchain platform, it is forwarded to the invoke function of the chaincode ($M5$ in Table 6).
vi The invoke function checks the type of the request and as it is an *userReg* request, it is forwarded to the *regUser* function (line 6, 7, 12 and 13 in Algorithm 1).
vii The *regUser* function retrieves the user name, hashed password, and the list of encrypted attributes from the data field (line 33 to 35 in Algorithm 1). Then, the retrieved information is stored in the blockchain and a TRUE response is returned to the DApp (line 36 and 37 in Algorithm 1, $M6$ in Table 6).
viii The DApp then informs the admin with a meaningful message ($M7$ and $M8$ in Table 6).

Since all the IdPs are connected to the same blockchain platform, once the user credentials and attributes are stored, all other IdPs have access to every user's data. Furthermore, all IdPs share the same symmetric encryption key so that the data can be decrypted when returning to the user. In this way, the blockchain platform is used as an attribute store. This is in contrast to the traditional SimpleSAMLPHP setup where generally a database is used.

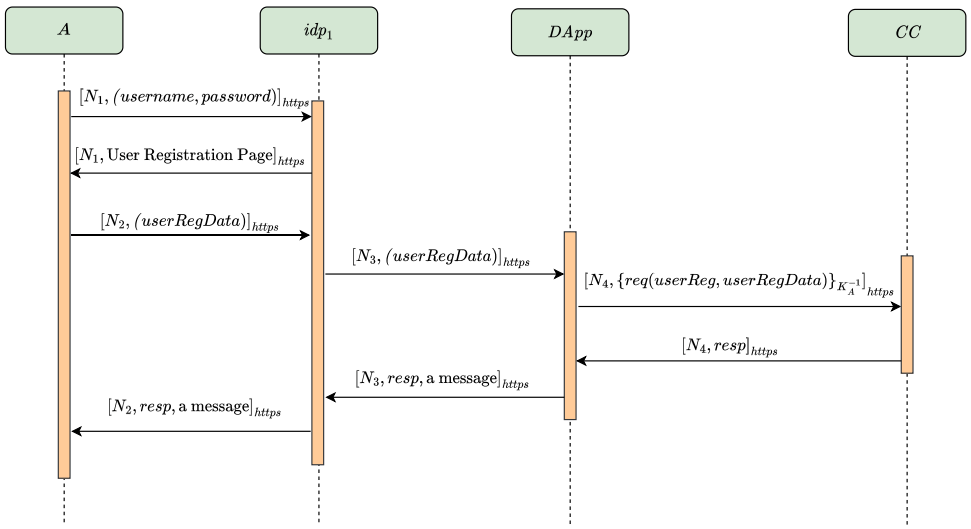


Fig. 6 Protocol flow for user registration

Table 6 User registration protocol

$M1$	$A \rightarrow idp_1$:	$[N_1, username, password]_{https}$
$M2$	$idp_1 \rightarrow A$:	$[N_1, \text{User Registration Page}]_{https}$
$M3$	$A \rightarrow idp_1$:	$[N_2, userRegData]_{https}$
$M4$	$idp_1 \rightarrow DApp$:	$[N_3, userRegData]_{https}$
$M5$	$DApp \rightarrow CC$:	$[N_4, \{req(userReg, userRegData)\}_{K_A^{-1}}]_{https}$
$M6$	$CC \rightarrow DApp$:	$[N_4, resp]_{https}$
$M7$	$DApp \rightarrow idp_1$:	$[N_3, resp \text{ with a meaningful message}]_{https}$
$M8$	$idp_1 \rightarrow A$:	$[N_2, resp \text{ a meaningful message}]_{https}$

6.2.3 Service provisioning phase

In this phase, we discuss how the combined IdP is utilised for a SAML service provisioning activity and to simulate the notion of a decentralised identity federation. To realise this flow, we have modified */-the SP codebase of SimpleSAMLPHP so that it can interact with its corresponding DApp. Algorithm 2 in Appendix A presents a partial algorithm snippet for the DApp utilised in this phase.

Before we discuss the protocol flow, it is assumed that a user would like to access a service by a service provider called *SP*. The SP is federated with each of the three combined IdPs following the traditional method of metadata exchange. This ensures that these IdPs trust the SP and vice versa. For simplicity, we have divided this flow into two parts: i) IdP resolving in which an IdP from the combined IdPs is selected and ii) Authentication in which the user is authenticated using the selected IdP to continue with the SAML authentication flow.

At first, we present the IdP resolving flow. The flow is illustrated in Fig. 7 and its corresponding protocol is presented in Table 7.

i A user visits the service page of the SP for the first time ($M1$ in Table 7).
ii The user is forwarded to the SAML WAYF (Where Are You From) page of the SP where the user can select an IdP (($M2$ in Table 7)).
iii The WAYF pages show the entity IDs of the three IdPs separately. The user selects one of the IdPs ($M3$ in Table 7).
iv Internally, the entity ID of selected IdP is sent to the DApp ($M4$ in Table 7).
v The DApp creates and signs a transaction consisting of a *cid* request which does not have any data field. This request embedded within the transaction is then submitted to the blockchain to initiate the Fabric flow.
vi Once the transaction is approved by the endorsers of the blockchain platform, it is forwarded to the invoke function of the chaincode ($M5$ in Table 7).
vii The invoke function checks the type of the request and as it is a *cid* request, the *CID* value is returned as a response. (line 16 to 18 in Algorithm 1, $M6$ in Table 7).
viii Then, the DApp calls the *idpResolver* function with the response *resp* as a parameter (Line 4 in Algorithm 2).
ix Within the function, the CID is retrieved from the response *resp* (Line 6 in Algorithm 2).
x The DApp creates and signs another transaction consisting of a *idpQuery* request which contains idqQData as data (as modelled in Table 4). This request embedded

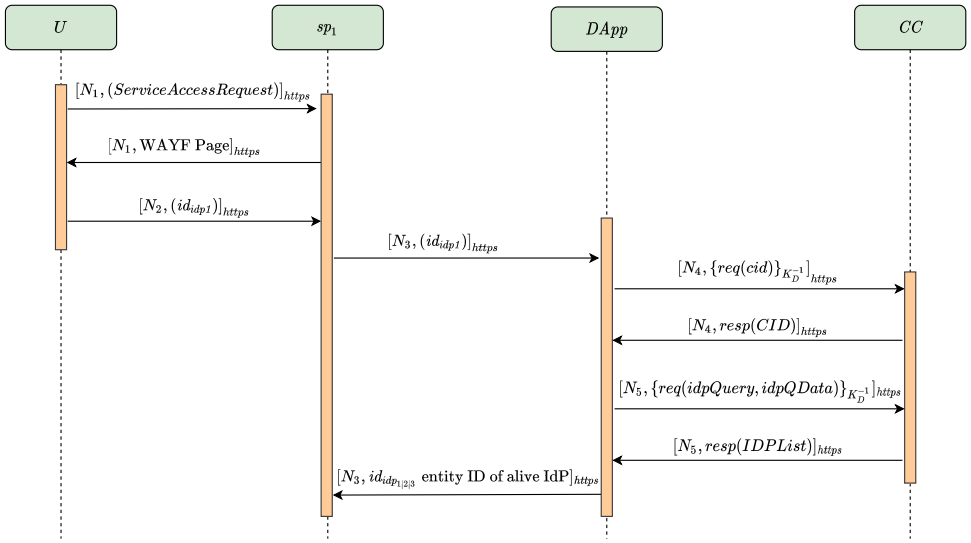


Fig. 7 Protocol flow for IdP resolving

within the transaction is then submitted to the blockchain to initiate the Fabric flow.

xi Once the transaction is approved by the endorsers of the blockchain platform, it is forwarded to the invoke function of the chaincode ($M7$ in Table 7).

xii The invoke function checks the type of the request and as it is an *idpQuery* request, it is forwarded to the *queryIDP* function (line 6, 7, 10 and 11 in Algorithm 1).

xiii Within the *queryIDP* function, CID is used to retrieve the list of the entity IDs of the three combined IdPs and return the list to the DApp (line 20, 21 and 11 in Algorithm 1).

xiv Using the CID, the list of the entity IDs of the combined IdPs is returned using the *getIDPList* function of the chaincode (Line 7 in Algorithm 2, $M8$ in Table 7).

xv Then, the DApp loops through the list of entity IDs to check if any of the IdPs is alive. For this, the DApp just checks if it can retrieve the metadata of the IdP from the entity ID URL. The entity ID of the first IdP that is found to be alive is returned to the WAYF Page (Line 8 to 12 in Algorithm 2, $M9$ in Table 7).

Next, we present the SAML authentication flow. The flow is illustrated in Fig. 8 and its corresponding protocol is presented in Table 8.

i The SAML SP codebase forwards the user to the selected IdP (assumed idp_2 to be alive in the protocol flow in the $M10$ step in Table 8) with a SAML Authentication request (a request with its type as *authn* and data as *AuthnReq* as presented in Table 4) where the user needs to authenticate.

ii For authentication, the IdP shows the user a login page where the user submits their username and password and clicks the *Submit* button ($M11$ in Table 8).

iii The IdP hashes the password and submits the username and hashed passport to the DApp ($M12$ in Table 8).

iv DApp creates a login request consisting of *login* as the type and *logInData* as the data as per Table 4. Then, the DApp submits a transaction consisting of the request to the blockchain to initiate the Fabric flow.

v Once the transaction is approved by the endorsers of the blockchain platform, it is forwarded to the invoke function of the chaincode ($M13$ in Table 8).

Table 7 IdP resolving protocol

$M1$	$U \rightarrow sp_1$:	$[N_1, \text{Service access request}]_{https}$
$M2$	$sp_1 \rightarrow U$:	$[N_1, \text{WAYF Page}]_{https}$
$M3$	$U \rightarrow sp_1$:	$[N_2, id_{idp_1}]_{https}$
$M4$	$sp_1 \rightarrow DApp$:	$[N_3, id_{idp_1}]_{https}$
$M5$	$DApp \rightarrow CC$:	$[N_4, \{req(cid)\}_{K_A^{-1}}]_{https}$
$M6$	$CC \rightarrow DApp$:	$[N_4, resp(CID)]_{https}$
$M7$	$DApp \rightarrow CC$:	$[N_5, \{req(idpQuery, idpQData)\}_{K_A^{-1}}]_{https}$
$M8$	$CC \rightarrow DApp$:	$[N_5, resp(IDPList)]_{https}$
$M9$	$DApp \rightarrow sp_1$:	$[N_3, id_{idp_{1\|2\|3}} \text{ entity ID of alive IdP}]_{https}$

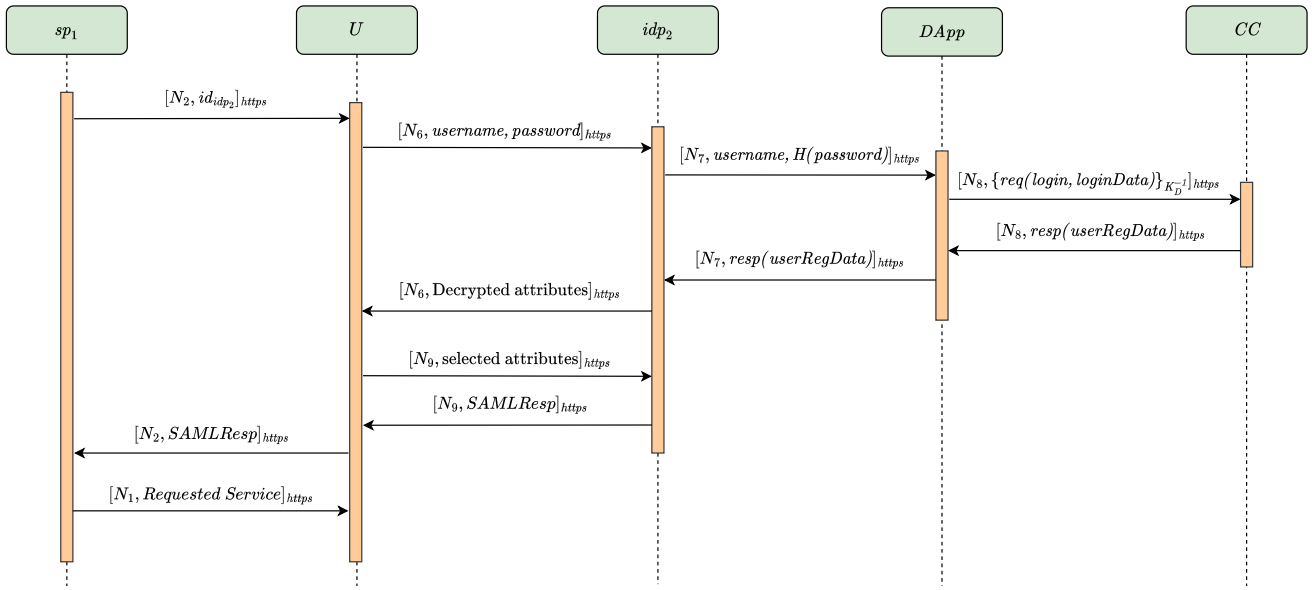


Fig. 8 Protocol flow for login

vi The invoke function checks the type of the request and as it is a *login* request, it is forwarded to the *loginUser* function (line 6, 14 and 15 in Algorithm 1).

vii Within the login function, the username and password hash are retrieved from the request (line 39 in Algorithm 1) and then uses the username to retrieve stored hashes from the blockchain (line 40 & 41 in Algorithm 1).

viii Then two hashes are matched. If they match, stored attributes of the users are returned to the DApp (line 44 & 16 in Algorithm 1, $M14$ in Table 8). If they do not match, a FALSE response is returned to the DApp (line 46 & 16 in Algorithm 1)

ix The response is returned to the IdP ($M15$ in Table 8). Depending on the response, either user attributes are decrypted and are shown to the user ($M16$ in Table 8) or a meaningful message (e.g. *username/password do not match*) is shown to the user.

x Assuming a positive response, the user can choose the attributes that they would like to release to the SP and once selected, the user clicks the *Submit* button ($M17$ in Table 8).

xi The IdP creates a SAML assertion, signs the assertion and embeds it into a SAML response. The SAML response (resp as modelled in Table 4) is returned to the SP via the user ($M18$ and $M19$ steps in Table 8).

xii The SP gets the assertion from the response, validates its signature and retrieves the user attributes. Based on the attributes, the SP takes an authorisation decision to allow (or reject) the user to access the service ($M20$ in Table 8).

The cornerstone of this approach is that the SP will be able to offer federated services as long as there is at least one IdP is available when IdPs become unavailable (e.g. due to technical reasons or an attack). This has been possible because of the utilisation of blockchain to create the notion of a combined IdP which shares the same credential and attribute data storage facility within the blockchain. In this way, we have been able to decentralise an identity federation and remove the bottleneck of a single point of failure because of its reliance on a single IdP. The best part is that the user does not need to worry at all about this process, the intricate complex protocol flows are tackled under the hood and the user experience will be exactly similar to the existing flow.

Table 8 Login protocol

$M10$	$sp_1 \rightarrow U$:	$[N_2, id_{idp_2}]_{https}$
$M11$	$U \rightarrow idp_2$:	$[N_6, username, password]_{https}$
$M12$	$idp_2 \rightarrow DApp$:	$[N_7, username, H(password)]_{https}$
$M13$	$DApp \rightarrow CC$:	$[N_8, \{req(login, loginData)\}_{K_A^{-1}}]_{https}$
$M14$	$CC \rightarrow DApp$:	$[N_8, resp(\text{encrypted attributes})]_{https}$
$M15$	$DApp \rightarrow idp_2$:	$[N_7, resp(\text{encrypted attributes})]_{https}$
$M16$	$idp_2 \rightarrow U$:	$[N_6, \text{Decrypted attributes}]_{https}$
$M17$	$U \rightarrow idp_2$:	$[N_9, \text{selected attributes}]_{https}$
$M18$	$idp_2 \rightarrow U$:	$[N_9, SAMLResp]_{https}$
$M19$	$U \rightarrow sp_1$:	$[N_1, SAMLResp]_{https}$
$M20$	$sp_1 \rightarrow U$:	$[N_1, \text{Requested service}]_{https}$

7 Evaluation

In this section, we assess and analyse the performance of a traditional identity federation using SimpleSAMLphp and

compare it to the performance of the Decentralised Identity Federation (DIF) that we proposed and developed. In this experiment, we used throughput, latency, and resource consumption as performance metrics. With these metrics, we conducted several load-testing experiments for multiple scenarios, modelling many use-cases and configurations. The use-cases involved regular SAML interactions between users, IdPs, and SPs as explained next.

- *Registering Users:* In this use-case, we create a group of user accounts and store their credentials within the Identity Provider (IdP) to be used for authentication later.
- *Requesting Service:* A user first submits a request to an SP for accessing a specific service. In our test scenarios, 2 SPs offer 2 separate services. To utilise a service from any SP, the user must first be authenticated by the federation's IdP.
- *Authenticating Users:* The user is subsequently redirected to the selected IdP's authentication page via a SAML authentication request. When the IdP receives the request, it checks to see if the SP is a federation member. Then the user authenticates at the IdP and receives a SAML response containing a SAML assertion which is forwarded back to the SP.
- *Responding to Service Request:* The assertion is retrieved from the SAML response when the SP receives it. The assertion's signature is then verified, and the user's attributes are extracted from the assertion. The user is then allowed to access the resource.

We used two separate test plans to investigate the use-cases during our study. We addressed the user registration process in the first test plan and simulated a typical user interaction consisting of service request, authentication, and response in the second test plan.

7.1 Experimental setup

For our experimental evaluation, we had 3 IdPs and 2 SPs. When using the Decentralised Identity Federation, each IdP participated in the blockchain network as a single organisation. Our system was deployed in Amazon Web Services (AWS) EC2 instances. The whole network was built using 5 Ubuntu instances, each with 4 VCPUs and 16 GB of RAM. The network configuration is shown in Fig. 9.

Here, 3 instances (Beta, Gamma, Delta) were used to represent the 3 participating IdP organisations (ORG1, ORG2, ORG3) and they hosted the necessary docker containers. The Orderer nodes and Certificate Authorities were hosted on the 4th EC2 instance (Alpha), while the SPs and IdPs were hosted on the 5th (Epsilon). Each organisation has 3 peers (peer0, peer1, peer2) and a dApp running to facilitate interaction with the network. The certificate authority of each organisation was hosted separately in the Alpha instance, and the ordering service consisted of 2 or 3 orderers (based on an experimental setup) with 3 ZooKeeper and 4 Kafka brokers.

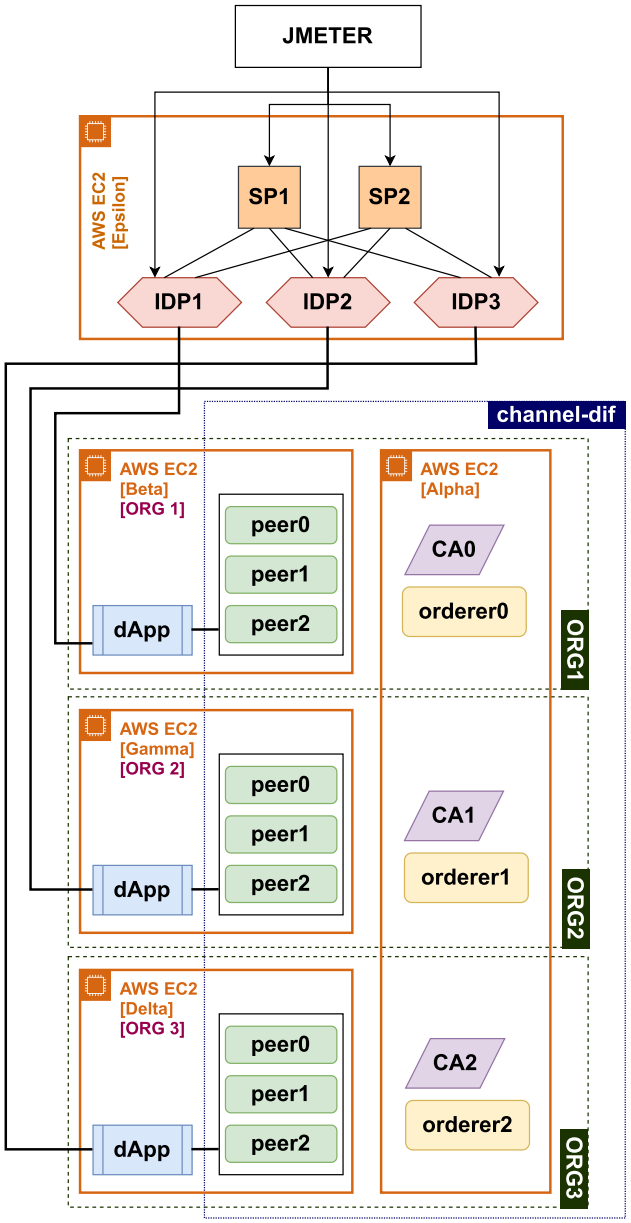


Fig. 9 Network configurations for experiments

To execute the test plans, we used Apache JMeter [5], an open-source performance measurement and load testing software. We carried out the load tests by gradually increasing the traffic. We started with 10 simultaneous users, and increased the number of users after each iteration of the test plan, going up to 150 simultaneous users. Each test plan was executed against 3 different experimental setups to investigate their impact on performance, 1) Conventional SAML setup without DIF (no-blockchain), 2) SAML with DIF using

Hyperledger Fabric (2 orderers), 3) SAML with DIF using Hyperledger Fabric (3 orderers).

In the conventional SAML setup (i.e. no-blockchain setup), we used the MySql [35] database for storing the credentials. However, in the other two blockchain-based network setups we took advantage of levelDB [29] for storing the world state of the blockchain.

The CPU and RAM consumption in all of the test situations stays within a fair range and never approaches the system limit. Similarly, increasing the number of orderers does not have much effect on the CPU usage, but RAM usage increases slightly when we go from 2 orderers to 3 orderers.

7.2 Performance analysis

We recorded the throughput, latency and resource consumption (memory and CPU usage) after each test cycle for comparison with the subsequent cycles. After completing the test, data from different setups and varying loads were aggregated to investigate the performance under different circumstances. Figures 10, 11a, b, 12, 13a and b show the comparative evaluation results of the two test plans (registration and login usage) in different situations.

7.2.1 Test plan

- *Registration latency and throughput*: Fig. 10 depicts that, the no-blockchain setup of the network has much better throughput and lower latency than the other two blockchain network configurations. We can also see that the throughput in the No-Blockchain SAML setup increases with the load, which indicates that it has not reached the maximum capacity yet. However, in the block-chain setups, the throughput doesn't change much because complex communication within the network limits the maximum throughput quite early.
- *Registration resource consumption*: From Fig. 11, the resource consumption, in terms of both CPU and RAM usages, for all load plans are considerably low. However, there are two trends: i) resource consumption for both resources tends to increase as more user loads are added and ii) resource consumption is higher for 3 orderers than 2 orderers. These trends are expected as more users and more orderers would imply more computations and storage are required.
- *Login latency and throughput*: From Fig. 12, we can see that, the latency of the system increases linearly as more loads are added. Similarly, the throughput also increases with the load. However, the change in throughput is not as steep as the latency. This is because once the throughput reaches its max capacity, after a certain amount of load, the throughput does not change as much even if it has to handle more loads. Using LevelDB helps our system execute queries faster than a relational database (MySQL) in a no-blockchain setup.
  An important observation from Figs. 10 12 is that the performance of the system varies significantly between the test plans. In the Registration test plan (Fig. 10), typical SAML implementation (no blockchain) outperforms our Decentralised Identity Federation. However, during the login process (Fig. 12), our system has slightly better performance than the typical SAML. This outcome is expected because, during the Registration phase, all the organisations must reach a consensus for creating the block that updates the world state, whereas, during the login process, the peers make queries on the local ledger which does not have the consensus reaching overhead. In a practical scenario, a user will go through the Registration process only once, but the login phase will be executed many more times whenever the user needs to authenticate to the system. Consequently, implementation of the proposed system will not hinder performance in most usage scenarios.
- *Login resource consumption*: It can be observed from Fig. 13 that the CPU and RAM usages for all load plans in the login process are considerably low. Also, like before, both CPU and RAM usage tends to increase as more user loads are added and resource consumption is higher for 3 orderers than 2 orderers.

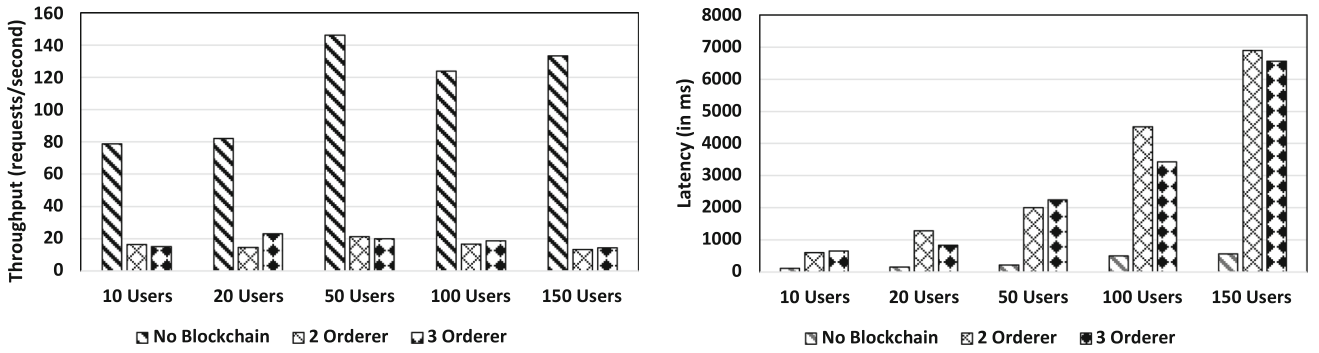
7.3 Protocol verification

We have formally analysed the security of our approach using a state-of-the-art Protocol Verification tool called ProVerif [7, 8]. Using ProVerif, we will analyse if the protocol of the developed system satisfies the secrecy as well as authentication goals. The secrecy goals check if a secret value truly remains secret while being transmitted between two entities [21]. On the other hand, the authentication goals check the authenticity of two entities while data are being transmitted between them. In the following, we discuss how we have formalised our protocol using ProVerif to evaluate the secrecy and authentication goals of the protocol.

7.3.1 Secrecy goals

In our ProVerif script, we declare some private variables. These variable represent sensitive information that we are trying to communicate between different entities. A brief introduction of these variables are described below.

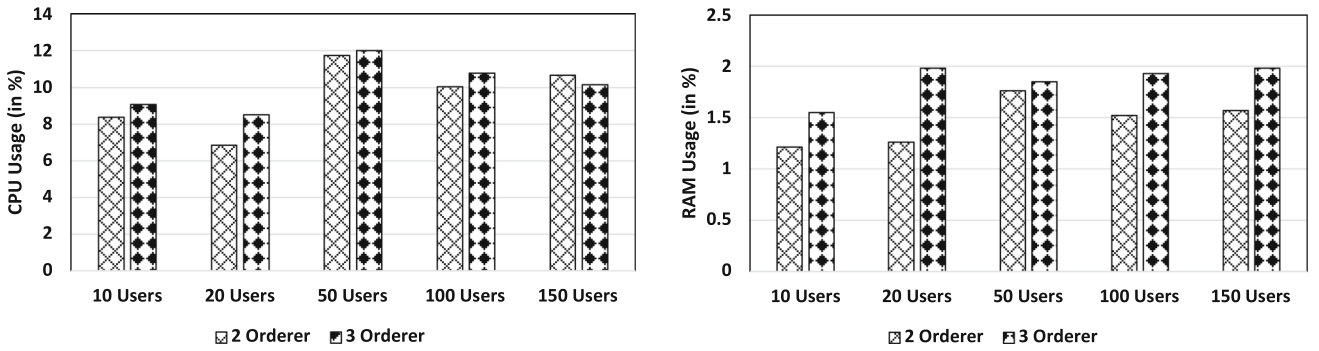
- *username, password*: The username and password of the admin/user.
- *IdpRegPage, IdpRegData*: The idp registration page and data.



(a) Throughput vs Load

(b) Latency vs Load

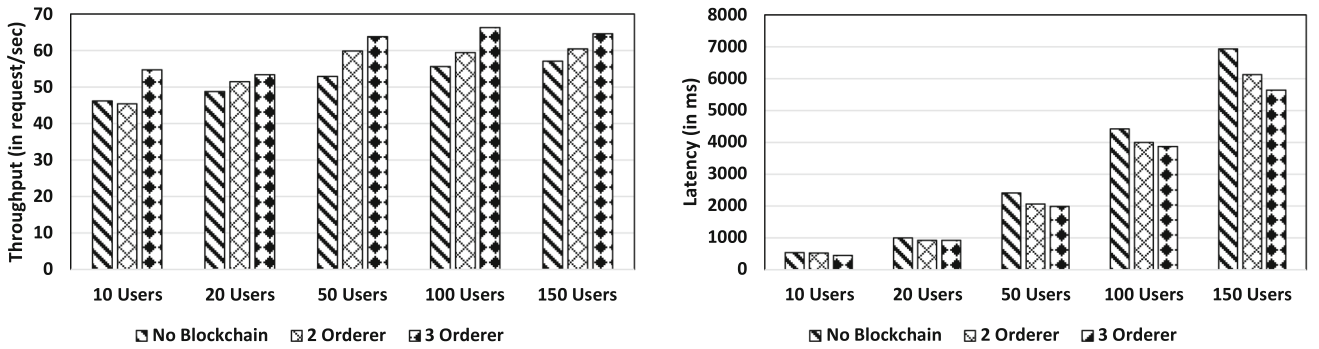
Fig. 10 Registration performance evaluation (Throughput and Latency)



(a) CPU Usage vs Load

(b) RAM Usage vs Load

Fig. 11 Registration resource consumption (CPU and RAM)



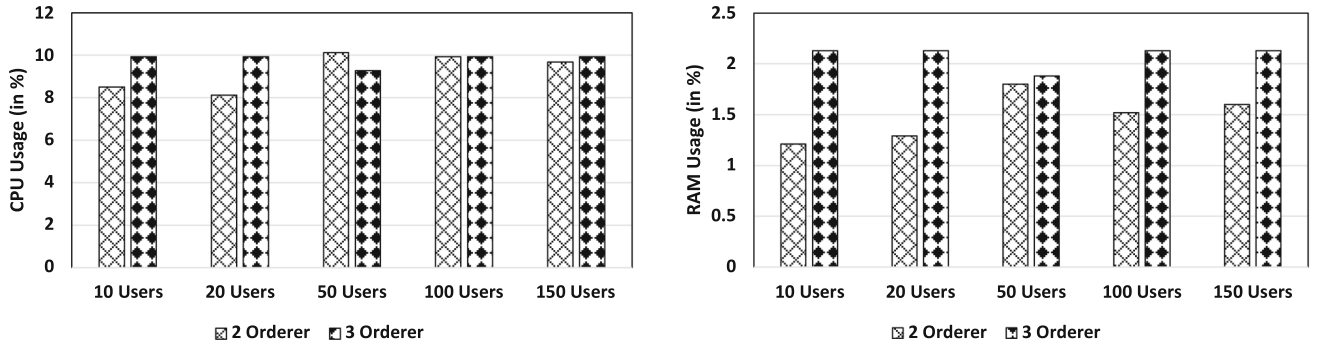
(a) Throughput vs Load

(b) Latency vs Load

Fig. 12 Login performance evaluation (Throughput and Latency)

- *UserRegPage, UserRegData*: The $user$ registration page and data.
- *IdpAlive*: The identity of an idp that is alive.
- *samlResp*: The $SAML$ response data.
- *decAtt*: The decrypted attributes.
- *CID*: Common Entity ID for the combined CoT

To analyse the reachability of these variables, we write the following lines of code to encode our secrecy queries for the protocols described in Tables 5, 6, 7 and 8.



(a) CPU Usage vs Load

(b) RAM Usage vs Load

Fig. 13 Login resource consumption (CPU and RAM)

```
(* User registration protocol *)
query attacker(username);
      attacker(password);
      attacker(UserRegData);
      attacker(UserRegPage).

(* Idp registration protocol *)
query attacker(username);
      attacker(password);
      attacker(IdpRegPage);
      attacker(IdpRegData).

(* Idp registration and Login protocol *)
query attacker(IdpAlive);
      attacker(password);
      attacker(username);
      attacker(samlResp);
      attacker(decAtt);
      attacker(CID).
```

Listing 1 Secrecy Queries

7.3.2 Authenticity goals

Correspondence assertions are used in ProVerif to analyse the link between occurrences. The protocol's validity is tested by this behaviour [49]. We highlight key stages in our protocols described in Tables 5, 6, 7 and 8 with the following events:

- *event beginLoginReq* and *endLoginReq* mark the starting and termination points of the login request event.
- *event beginServReqVal* (*endServReqVal*) marks the start (resp. the end) of the validation that happens at the *idp* end to determine which service the requesting entity has access to.
- *event beginRegPageReq* and *endRegPageReq* are triggered when an admin/user requests the registration page or the registration is served respectively.
- *event beginIdpReg* (*endIdpReg*) marks the beginning (resp. the end) of registering an entity in the identity provider, *idp*.
- *event beginSndIdpReg* (*endSndIdpReg*) takes place when the *DApp* sends (resp. receives) *idp* registration data to (resp. from) *CC*.
- *event beginUserReg* (*endUserReg*) triggers the beginning (resp. ending) of registering a user.
- *event beginSndUsrReg* (*endSndUsrReg*) takes place when the *DApp* sends (resp. receives) admin registration data to (from) *CC*.

After defining the events that make up the core of our protocol, we can now create correspondence queries for these events to verify our authenticity objectives. The below list contains the correspondences.

```
(* Idp registration protocol *)
query admin:host, idp1:host, nonce:nonce;
   event(endRegPageReq(admin,idp1,nonce))
   ==>
   event(beginRegPageReq(admin,idp1,
   nonce)).

query admin:host, idp1:host, nonce:nonce;
   event(endIdpReg(admin,idp1,nonce))
      ==>
   event(beginIdpReg(admin,idp1,nonce)).

query dapp:host, cc:host, nonce:nonce;
   event(endSndIdpReg(dapp,cc,nonce))
      ==>
   event(beginSndIdpReg(dapp,cc,nonce)).
```

Listing 2 Idp registration protocol

Fig. 14 Results of idp registration protocol

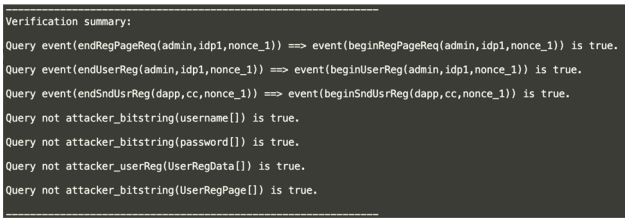


Fig. 15 Results of user registration protocol

```
(* User registration protocol *)
query admin:host, idp1:host, nonce:nonce;
    event(endRegPageReq(admin,idp1,nonce))
    ==>
    event(beginRegPageReq(admin,idp1,nonce)).

query admin:host, idp1:host, nonce:nonce;
    event(endUserReg(admin,idp1,nonce))
        ==>
    event(beginUserReg(admin,idp1,nonce)).

query dapp:host, cc:host, nonce:nonce;
    event(endSndUsrReg(dapp,cc,nonce))
        ==>
    event(beginSndUsrReg(dapp,cc,nonce)).
```

Listing 3 User registration protocol

```
(* Idp resolve and login protocol *)
query user:host, sp1:host, nonce:nonce;
    event(endLoginReq(user,sp1,nonce))
    ==>
    event(beginLoginReq(user,sp1,nonce)).
query user:host, sp1:host, nonce:nonce;
    event(endServReqVal(user,sp1,nonce))
    ==>
    event(beginServReqVal(user,sp1,nonce)).
```

Listing 4 Idp resolve and login protocol

These are the queries that we will ask with our ProVerif scripts to validate our protocols for secure communication.

Fig. 16 Results of idp resolve and login protocol

7.3.3 ProVerif query results

We will now discuss the findings from the aforementioned queries with the screenshots of the execution of the scripts with the ProVerif executable. These screenshots will confirm that the secrecy and authenticity objectives outlined before have been fulfilled. These screenshots are presented in Figs. 14, 15 and 16.

Figure 15 simulates the validation result for protocols described in Table 6. We have also demonstrated the protocol verification results for the protocol described in Table 5 in Fig. 14 and Protocols described in Tables 7 and 8 in Fig. 16. ProVerif tries to prove the negation of our secrecy reachability goal. According to the results, the secrets are not reachable by an attacker, thereby validating our secrecy goals. ProVerif also corroborates the authenticity of our protocol with injective correspondence, the results are also shown in the corresponding figures.

8 Discussion

Throughout this article, we have presented different aspects of our proposal of decentralising the identity provider entity of any existing Identity Federation through the use of blockchain. We have explained the motivation, outlined the proposal, presented the architecture and the implementation details along with protocol flows, and finally, analysed its performance and security. In this section, we explain how the architecture and its implementation have satisfied different functional and security requirements (Sect. 8.1). Furthermore, we also discuss its advantages, limitations, and potential future work in Sect. 8.2. Finally, we present a comparative analysis of our work with the reviewed research works in Sect. 8.3.

8.1 Analysing requirements

Through designing and implementing our system, we were able to meet all of the requirements for the system, both functional and non-functional.

In the following, we discussed how we achieved functional requirements.

- The combined IdPs are part of the same private blockchain network which binds their trust with each other. This is implicitly analogous to the trust establishment procedure within federated entities by exchanging metadata and hence, it satisfies requirement *F1*.
- These combined IdPs utilise the blockchain as the metadata store, thereby sharing the user attributes with each other to satisfy requirement *F2*.
- The protocol ensures that even when one of the IdPs from the combined IdPs is offline, SPs can leverage the service of other IdPs in a seamless way. The blockchain-based attribute storage facility ensures that any of the IdPs have access to the user attributes and carry out the SAML authentication functionality. This satisfies requirement *F3*.
- Finally, the protocol is designed in such a way that SAML SPs are minimally affected and hence, it satisfies requirement *F4*.

Subsequently, we explored how we attained non-functional (security) requirements.

- Each SP is federated with each of the combined IdPs in the traditional approach by exchanging SAML metadata manually. This is usually carried out by the admins of the respective entity and hence, it satisfies requirement *S1*.
- Each assertion is digitally signed to satisfy requirement *S2*.
- An IdP releases only encrypted assertions and all data are transmitted via HTTPS channels, thereby satisfying requirement *S3*.
- The consent module in the SimpleSAMLPHP ensures that user attributes are released only after an explicit consent by a user, this satisfies *S4*.
- Finally, even if an IdP from the combined IdP becomes unavailable, the protocol of our system ensures the availability of IdP services within the federation. This satisfies *S5*.

Analysis of additional attacks: A large number of attack mitigating techniques for attacks such as man-in-the-middle, session hijacking, and DoS for SAML are extensively discussed in [37] and [38]. SimpleSAMLphp also maintains a list of different security advisories (including session issues and DoS) in [42] along with a discussion of how these attacks have been patched in SimpleSAMLphp. Since our proposed system is fully based on SimpleSAMLphp, it inherits the security properties of SAML and its SimpleSAMLphp implementation. Interestingly, our system provides an interesting option during a DoS attack towards the IdP. Since the notion of IdP has been decentralised using blockchain, the system can be functional even if a single IdP of the combined IdP (Figs. 3 and 4) is under a DoS attack and therefore is not available. Another important attack to consider is identity spoofing of users. This attack depends on how SimpleSAMLphp facilitates the authentication of users. SimpleSAMLphp uses an authentication module [43] for user authentication with the ability to extend its core capability by developing additional extensions. This capability has been used to develop advanced authentication mechanisms such as two-factor authentication [44]. Since identity spoofing is dependent on the deployment of a specific authentication mechanism, we have not considered this in our research.

Table 9 Comparison with related works: functional and security requirements

Schemes	F1	F2	F3	F4	S1	S2	S3	S4	S5
ElGayyar et al. [16]	●	●	○	○	●	○	○	○	○
Alom et al. [2]	●	⌀	⌀	●	●	⌀	⌀	⌀	⌀
Liu et al. [33]	⌀	⌀	⌀	⌀	⌀	⌀	⌀	⌀	⌀
Mell et al. [34]	●	●	○	○	●	○	○	○	○
Haddouti et al. [15]	●	●	○	●	●	●	●	○	○
DIF	●	●	●	●	●	●	●	●	●

8.2 Advantages, limitations and future work

The proposed system offers a number of advantages. Here, we highlight a few of these advantages.

- This is the first work to showcase how the blockchain can be leveraged to decentralise a SAML-based identity federation. This ensures that the SPs in this federation will be able to access IdP services even if an IdP ceases to function.
- The system is designed in such a way that the functionalities of SPs within the federation are minimally impacted. For the PoC implementation, we are only required to make the majority of changes in the IdP codebase. This is to ensure that the implemented solution can be deployed as easily as possible.
- The proposed solution utilises the blockchain platform as an attribute store, thereby providing the immutability of user attributes. In addition, other important information (e.g. CID) is also stored in the blockchain platform. This is an improvement over the traditional SAML implementation where mainly a centralised database is used to store user attributes.

This research has some limitations that we see as opportunities for future improvements.

Table 10 Comparison with related works: other parameters

Schemes	FD	CS	PC	PV	TM	BC	PoC	PA
ElGayyar et al. [16]	○	○	●	○	○	■ (Ethereum, Fabric)	●	●
Alom et al. [2]	○	●	●	○	●	■ (Fabric)	●	●
Liu et al. [33]	⌀	⌀	⌀	⌀	⌀	⌀	⌀	⌀
Mell et al. [34]	○	○	○	○	○	□	●	○
Haddouti et al. [15]	○	●	○	○	○	■ (Own)	●	●
DIF	●	●	●	●	●	■ (Fabric)	●	●

– *Scalability concerns:* In Sect. 7.2, we saw that the system without blockchain had better throughput and lower latency, particularly during the registration phase. As every operation is going through smart contracts by participating nodes, the required cryptographic functions and network overhead add up to additional latency and decrease the overall throughput in this particular use-case. This is an issue that is attributed to the inherent nature of any blockchain system. However, it should be noted that blockchain scalability is an active research area. Different approaches, such as Sharding [50] and Layer 2 solutions [26], are emerging with the promise of better scalability than any current blockchain system. Once this happens, the performance bottleneck in the registration phase could be effectively removed.
– *Complexity of Implementations:* Decentralising a SAML-based identity federation involves distributing IdPs and SPs across multiple domains. This decentralised architecture increases the complexity of implementation, requiring careful coordination, configuration, and management of trust relationships between entities.
– *Regulatory Compliance:* Such a decentralised identity federation can pose regulatory compliance challenges related to data protection, privacy regulations, and cross-border data transfer restrictions. Ensuring compliance with regulations such as GDPR [17] or HIPAA [48]. Therefore, it might require careful consideration of data residency, data sovereignty, and data protection requirements.

The current version of the implementation only uses smart contracts (Fabric chaincode) for storing/retrieving information to/from the Fabric blockchain. There are potential advanced use-cases of smart contracts (e.g. smart contract-based access control within the SAML federations) which have not been explored. In the future, we would like to explore how such an access control mechanism could be integrated into the solution.

8.3 Comparative analysis

We provide a comparison of our proposed Decentralised Identity Federation (DIF) with the reviewed research works mentioned in Sect. 3. At first, we compare the reviewed works with our work using the formulated security and privacy requirements. The comparison is presented in Table 9.

Next, we compare existing research with our work against a number of other properties focusing mainly on different aspects of implementation. The considered properties are Full Decentralisation of SAML (FD), Compatibility with existing systems such as SAML (CS), Protocol (PC), Security Verification of Protocols (PV), Threat Modelling (TM), Type of Blockchain (BC), Proof of Concept (PoC) and Performance Analysis (PA). We present the result of this comparative analysis in Table 10.

FD will imply that if the system is fully decentralised for any SAML-based identity federation. *CS* denotes if the scheme can be integrated with the existing system without breaking the status quo. It is necessary to ensure the entry barrier for adoption. *PC* implies if the authors have defined a detailed protocol of the system whereas *PV* denotes if the authors have provided a security analysis by protocol verification. TM indicates whether the respective investigation is based on a threat model. BC specifies the blockchain type and the respective blockchain used in the implementation (if any). PoC denotes if the authors have developed a proof of concept of the scheme. PA is used to indicate if the paper has any performance analysis.

In both Tables 9 and 10, '●' implies if a research work satisfies the property and '○' denotes the opposite. '□' in the BC column denotes the public blockchain and '■' denotes the use of a private/enterprise blockchain. Finally, '⌀' means "Not Applicable".

From Table 9, only the work of Haddouti et al. [15] closely match with our work in terms of functional and security requirements. However, the motivations of both works are quite different and there are still requirements not fulfilled

by the work of Haddouti et al. As seen in Table 10, none of the works except our work provides full decentralisation for an SAML-based identity federation. In addition, most of the reviewed works lack any threat modelling and all reviewed works have no protocol verification. Taking into account the comparative analysis presented in Tables 9 and 10, it is clearly evident that our work stands out from any of the existing research work.

9 Conclusion

In this article, we have presented a proposal for a novel decentralised identity federation system. The existing SAML-based identity federation systems suffer from a serious centralisation issue because they rely on a single IdP within a federation. This leads to a single point of failure. The proposed blockchain-based decentralised system can effectively tackle this important issue. We have presented its architecture which is based on a rigorous threat model and requirement analysis. We have also designed the protocol flow and developed a Proof-of-Concept (PoC) of the proposed system and showcased several use-cases to show the applicability of the developed PoC. We have analysed its performance using several configurations with different user loads. The result of performance testing has yielded satisfactory results, illustrating the practicality of the system. In addition, we have successfully analysed the security of the underlying protocol of the system using a state-of-the-art protocol verified called ProVerif.

The concept of decentralisation in identity management is still an active research topic and is likely to be so for years to come. We believe that a decentralised federated identity system could offer many advantages over the traditional federated system. We hope that our implemented system will be a stepping stone for a future revolutionary change in the Identity Management domain.

Appendix

A Algorithm

Algorithm 1: Chaincode

```
1  Input: req → the request from the user
2  Output: resp → the chaincode generated response
3  Start
4      CID ← Generate CID ;
5      function invoke(req)
6          data ← req.data;
7          type ← req.type;
8          if type = idpReg then
9              resp ← regIDP(data);
10         else if type = idpQuery then
11             resp ← queryIDP(data);
12         else if type = userReg then
13             resp ← regUser(data);
14         else if type = login then
15             resp ← loginUser(data);
16         else if type = cid then
17             resp ← CID;
18         send resp back to DApp;
19     function queryIDP(data)
20         idps ← getState(CID);
21         return idps;
22     function regIDP(data)
23         idp ← data.id_idp_i ;
24         idps ← getState(CID);
25         if idp ∉ idps then
26             IDPList ← IDPList ∪ idp ;
27             putState(CID,IDPList);
28             return TRUE;
29         else
30             return FALSE;
31         end
32     function regUser(data)
33         userName ← data.userName;
34         hash ← data.h;
35         AttList ← data.AttList;
36         putState(userName,(hash,AttList));
37         return TRUE;
38     function loginUser(data)
39         userName ← data.userName;
40         hash ← data.h;
41         ledgerData ← getState(userName);
42         h ← ledgerData[0];
43         if h == hash then
44             return ledgerData[1];
45         else
46             return FALSE;
47         end
```

Data availability The experimental data that support the experimental evaluation presented in this article are available on GitHub with the URL: https://github.com/shuhanmirza/Decentralised-Identity-Federations-using-Blockchain

Declarations

Conflict of interest The authors have no Conflict of interest to declare that are directly or indirectly related to the content of this article.

Author Contributions Mirza Kamrul Bashar Shuhan: Conceptualisation, Methodology, Investigation, Software, Resources, Writing - Original Draft, Visualisation Syed Md. Hasnayeen: Conceptualisation, Methodology, Investigation, Software, Resources, Writing - Original Draft Tanmoy Krishna Das: Investigation, Visualisation, Writing - Original Draft Md. Nazmus Sakib: Formal Analysis, Writing - Original Draft Dr. Md Sadek Ferdous: Conceptualisation, Methodology, Resources, Writing - Original Draft, Writing - Review & Editing, Supervision, Project administration

Algorithm 2: DApp Code

```
Start
    ...
    send a cid request to chaiancode and get response resp;
    function idpResolver(resp)
        CID ← resp.CID;
        send a idpQuery request to chaiancode and get response
          idps;
        idp ← NULL;
        for idp ∈ idps do
            if idp is alive then
                break;
        end
        send idp back to user;
    ...
```

References

1. (2022) Ethereum: a next-generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper, accessed: 2024-03-11
2. Alom, I., Eshita, R.M., Harun, A.I., et al.: Dynamic management of identity federations using blockchain. In: 2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), IEEE, pp. 1–9 (2021)
3. Androulaki, E., Barger, A., Bortnikov, V., et al.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the thirteenth EuroSys conference, pp 1–15 (2018)
4. Apache Apache Kafka. https://kafka.apache.org/ Accessed: May 13, 2024 (2022)
5. Apache JMeter. https://jmeter.apache.org/, Accessed: 01-10-2022(2022)
6. Bhuiyan, M.S.I., Razzak, A., Ferdous, M.S., et al.: Bonik: a blockchain empowered chatbot for financial transactions. In: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 1079–1088 (2020)
7. Blanchet, B.: Automatic verification of security protocols in the symbolic model: the verifier ProVerif. In: Aldini, A., Lopez, J., Martinelli, F. (eds.) Foundations of Security Analysis and Design VII, FOSAD Tutorial Lectures. Lecture Notes in Computer Science, vol. 8604, pp. 54–87. Springer, Berlin (2014)
8. Blanchet, B., Smyth, B.: Automated reasoning for equivalences in the applied pi calculus with barriers. J. Comput. Secur. **26**(3), 367–422 (2018)
9. Cantor, S., Moreh, J., Philpott, R., et al.: Metadata for the OASIS security assertion markup language (SAML) V2. 0". http://docs.oasis-open.org/security/saml/v2.0/saml-metadata-2.0-os.pdf, Accessed: 01-09-2020 (2018)
10. Castro, M.: Practical byzantine fault tolerance. PhD thesis, laboratory for computer science, https://www.microsoft.com/en-us/research/publication/practical-byzantine-fault-tolerance-2/, george M. Sprowls Award (2001)
11. Chadwick, D.W.: Federated identity management. In: Foundations of Security Analysis and Design V, pp. 96–120. Springer, Berlin (2009)
12. Chowdhury, M.J.M., Ferdous, M.S., Biswas, K., et al.: A comparative analysis of distributed ledger technology platforms. IEEE Access **7**(1), 167930–167943 (2019)
13. Consortium, S.: Shibboleth. https://www.shibboleth.net/, Accessed: 01-09-2020 (2022)
14. Daniel, L.: Delegated proof of stake. https://bitshares.org/delegated-proof-of-stake-consensus/, [Accessed 16-03-2024] (2024)
15. El Haddouti, S., Ouaguid, A., Ech-Cherif El Kettani, M.D.: Fedidchain: an innovative blockchain-enabled framework for cross-border interoperability and trust management in identity federation systems. J. Netw. Syst. Manage. **31**(2), 42 (2023)
16. ElGayyar, M.M., ElYamany, H.F., Grolinger, K., et al.: Blockchain-based federated identity and auditing. International Journal of Blockchains and Cryptocurrencies. **1**(2), 179–205 (2020) https://www.inderscienceonline.com/doi/pdf/10.1504/IJBC.2020.109004
17. European Union. Gdpr–general data protection regulation. https://gdpr-info.eu/, accessed: 2023-03-22 (2023)
18. Ferdous, M.S., Poet, R.: Dynamic identity federation using security assertion markup language (saml). In: IFIP Working Conference on Policies and Research in Identity Management, pp 131–146 (2013)
19. Ferdous, M.S., Poet, R.: Portable personal identity provider in mobile phones. In: 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (IEEE TrustCom-13), IEEE, pp 736–745 (2013)
20. Ferdous, M.S., Poet, R.: Managing dynamic identity federations using security assertion markup language. J. Theor. Appl. Electron. Commer. Res. **10**(2), 53–76 (2015)
21. Ferdous, M.S., Poet, R.: Formalising identity management protocols. In: 14th Annual Conference on Privacy, Security and Trust (PST-16), IEEE, pp 137–146 (2016)
22. Ferdous, M.S., Norman, G., Poet, R.: Mathematical modelling of identity, identity management and other related topics. In: 7th International Conference on Security of Information and Networks, pp. 9–9. Glasgow, UK; ACM (2014)
23. Ferdous, M.S., Chowdhury, F., Alassafi, M.O.: In search of self-sovereign identity leveraging blockchain technology. IEEE Access **7**, 103059–103079 (2019)
24. Ferdous, M.S., Chowdhury, F., Alassafi, M.O., et al.: Social anchor: privacy-friendly attribute aggregation from social networks. IEEE Access **8**, 61844–61871 (2020)
25. Foundation, O.: Node.js. https://nodejs.org/en/, Accessed: 10-07-2022 (2022)
26. Gudgeon, L., Moreno-Sanchez, P., Roos, S., et al.: Sok: Layer-two blockchain protocols. In: Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers 24, Springer, pp 201–226 (2020)
27. Hyperledger Fabric: hyperledger fabric documentation. https://hyperledger-fabric.readthedocs.io/en/release-1.4/, accessed: 2023-03-22 (2023)
28. Hyperledger foundation: hyperledger project. https://www.hyperledger.org/, Accessed: 10-07-2022 (2022)
29. Jeffrey, Dean, et al.: Leveldb. https://github.com/google/leveldb, Accessed: 10-07-2022 (2022)
30. Josang, A., AlZomai, M., Suriadi, S.: Usability and privacy in identity management architectures. In: ACSW Frontiers 2007: Proceedings of 5th Australasian Symposium on Grid Computing and e-Research, Australian Computer Society, pp 143–152 (2007)
31. Khattak, Z.A., Sulaiman, S., Manan, J.L.A.: A study on threat model for federated identities in federated identity management system. In: 2010 International Symposium on Information Technology, pp 618–623, https://doi.org/10.1109/ITSIM.2010.5561611 (2010)
32. King, S., Nadal, S.: Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. https://peercoin.net/assets/paper/peercoin-paper.pdf, [Accessed 16-03-2024] (2024)
33. Liu, Y., He, D., Obaidat, M.S., et al.: Blockchain-based identity management systems: a review. J. Netw. Comput. Appl. **166**, 102731 (2020)

34. Mell, P., Dray, J., Shook, J.: Smart contract federated identity management without third party authentication services. arXiv preprint arXiv:1906.11057 (2019)
35. MySQL, A.B.: Mysql. https://www.mysql.com/, Accessed: 01-10-2022 (2024)
36. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Tech. rep, Manubot (2019)
37. OASIS Standard: Security and privacy considerations for the oasis security assertion markup language (saml) v2.0. https://docs.oasis-open.org/security/saml/v2.0/saml-sec-consider-2.0-os.pdf, accessed: 2023-03-22 (2005)
38. Open Web Application Security Project (OWASP) Saml security cheat sheet. https://cheatsheetseries.owasp.org/cheatsheets/SAML_Security_Cheat_Sheet.html, accessed: 2023-03-22 (2023)
39. Papathanasaki, M., Maglaras, L., Ayres, N.: Modern authentication methods: a comprehensive survey. AI Comput. Sci. Robot. Technol. (2022). https://doi.org/10.5772/acrt.08
40. Quorum: Quorum blockchain. https://www.goquorum.com/, Accessed: 10-07-2022(2022)
41. Shostack, A.: Threat Modeling: Designing for Security. Wiley, Hoboken (2014)
42. SimpleSAMLphp: security advisories - simplesamlphp. https://simplesamlphp.org/security/, Accessed: May 13, 2024 (2023)
43. SimpleSAMLphp: simplesamlphp modules. https://simplesamlphp.org/docs/stable/simplesamlphp-modules.html, accessed: 2024-03-22 (2023)
44. SimpleSAMLphp: simplesamlphp third-party modules. https://simplesamlphp.org/modules/, accessed: 2024-03-22 (2023)
45. Szabo, N.: Smart contracts: building blocks for digital markets. EXTROPY: J. Transhumanist Thought (16) (1996)
46. Holowaychuk, T.J., et al.: Express JS. https://expressjs.com/, Accessed: 10-07-2022 (2022)
47. UNINETT (2022) SimpleSAMLphp. https://simplesamlphp.org/, Accessed: 01-06-2022
48. U.S. Department of Health & Human Services. Health insurance portability and accountability act (hipaa). https://www.hhs.gov/hipaa/index.html, accessed: 2023-03-22 (2023)
49. Woo, T.Y., Lam, S.S.: A semantic model for authentication protocols. In: Proceedings 1993 IEEE Computer Society Symposium on Research in Security and Privacy, IEEE, pp 178–194 (1993) https://www.cs.utexas.edu/users/lam/Vita/IEEE/WooLam93a.pdf
50. Yu, G., Wang, X., Yu, K., et al.: Survey: sharding in blockchains. IEEE Access **8**, 14155–14181 (2020)
51. ZXID. ZXID http://www.zxid.org/, Accessed: 01-09-2020 (2020)