Accepted and presented in the Proceedings of the 3rd IEEE International Conference on Blockchain and Cryptocurrency (ICBC), 2021.

Dynamic Management of Identity Federations using Blockchain

Ifteher Alom*, Romana Mahjabin Eshita†, Anam Ibna Harun‡, Md Sadek Ferdous§†, Mirza Kamrul Bashar Shuhan¶, Mohammad Jabed M Chowdhury‖, Mohammad Shahidur Rahman†

*Samsung R&D Institute, Dhaka, Bangladesh
†Shahjalal University of Science and Technology, Sylhet, Bangladesh
‡Kona Software Lab, Dhaka, Bangladesh
§Imperial College London, London, United Kingdom
¶Progoti Systems Limited, Dhaka, Bangladesh
‖La Trobe University, Melbourne, Victoria, Australia

Email: ifteher.alom@gmail.com, eshita1697@gmail.com, anamibnaharun@gmail.com, s.ferdous@imperial.ac.uk, shuhan.mirza@gmail.com, m.chowdhury@latrobe.edu.au, rahmanms@sust.edu

***Abstract*—Federated Identity Management (FIM) is a model of identity management in which different trusted organizations can provide secure online services to their uses. Security Assertion Markup Language (SAML) is one of the widely-used technologies for FIM. However, a SAML-based FIM has two significant issues: the metadata (a crucial component in SAML) has security issues, and federation management is hard to scale. The concept of dynamic identity federation has been introduced, enabling previously unknown entities to join in a new federation facilitating inter-organization service provisioning to address federation management's scalability issue. However, the existing dynamic federation approaches have security issues concerning confidentiality, integrity, authenticity, and transparency. In this paper, we present the idea of facilitating dynamic identity federations utilizing blockchain technology to improve the existing approaches' security issues. We demonstrate its architecture based on a rigorous threat model and requirement analysis. We also discuss its implementation details, current protocol flows and analyze its performance to underline its applicability.**

***Index Terms*—Federated Identity Management, Identity Federation, Dynamic Identity Federation, Blockchain, Fabric.**

I. Introduction

With the rapid expansion and popularity of online services, more and more user accounts are being created online. The management of such user accounts poses a significant challenge both from the perspective of the users and organizations providing such services. The concept of Identity Management (IdM) was initiated by the industry to ensure seamless management of user identities. The system which is utilized for the management of identities is known as Identity Management Systems (IMS). These IMS leverage several IdM models such as SILO Model, Common Identity Domain Model, Federated Identity Model, and so on [1].

Among these models, Federated Identity Management (FIM, in short), based on the notion of Federated Identity Model (also known as Federation of Identities), has gained considerable attention [2], particularly in the Educations and Governmental settings. In essence, FIM is a service model in which different organizations form a virtual trust boundary so that users from one of the organizations can access the data and services of other trusted organizations. SAML (Security Assertion Markup Language) [3], an XML-based standard, is one of the most widely used FIM solutions that facilitates the exchange of authentication and authorization information between the trusted organizations and a trusted access of services across their security boundaries [4]. There are several implementations of the SAML standard, such as Shibboleth [5], SimpleSAMLphp [6] and ZXID [7].

A crucial step in creating an identity federation between different organizations is the exchange of metadata, an XML file that stores important information for each organization within the federation. Therefore, such metadata files must be securely stored by these organizations as their unavailability would seriously hinder the federation. Another important issue of any SAML-based federation is that their management is static, which is hard to scale when the number of entities within the federation is large. The existing proposals ([3], [8]–[12]) to mitigate this issue have two major disadvantages: i) the core SAML standard needs to be significantly changed and ii) several security considerations such as confidentiality, integrity, availability, authenticity and transparency have been mostly overlooked.

In recent years, blockchain technology has emerged as one of the fundamental technologies with the potential to disrupt several application domains. This is because blockchain exhibits some unique properties such as immutability and irreversibility of ledger state, distributed consensus, data provenance, data persistence, accountability, and transparency [13]. In this paper, we present a novel blockchain-based metadata management framework for SAML identity federations that effectively tackles the limitations in the current implementation of SAML identity federations as identified above.

Contributions: The major contributions of this paper are presented below:

- A novel blockchain-based metadata management framework, derived from rigorous threat modelling and requirement analysis, which facilitates secure and decentralized

storage of metadata to provide secure dynamic identity management.
- A detailed architecture of the proposed framework along with Proof-of-Concept (PoC) implementation and its associated protocol flow.
- PoC's performance analysis to highlight its advantages and limitations.

Structure: In Section II, we present a brief background on dynamic identity federations and blockchain. We introduce a threat model for the framework along with a requirement analysis in Section III. Then, in Section IV, we outline the architecture of the proposed framework, discuss how the architecture has been developed and illustrate the protocol flow in Section V. We evaluate the performance of the developed architecture in Section VI. In Section VII, we discuss how the developed proof of concept satisfies different requirements, analyze its advantages and limitations, and possible future work. Finally, we conclude in Section VIII.

II. Background

In this section, some of the basic terminologies and topics have been discussed briefly in light of the proposal in this paper. In the following, we discuss about different aspects of FIM (Section II-A) and blockchain (Section II-B).

A. Federated Identity Management (FIM)

Formally, Identity Management can be defined as a collection of technologies and policies which can be used for representing and recognizing users using digital identifiers within a specific application domain and for managing such identifiers [1]. Federated Identity Management (FIM) can be considered as 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.

FIM enables a user to access restricted resources seamlessly and securely from different organizations in different Identity Domains. An identity domain is the virtual boundary, context or environment in which a digital identifier is valid [1]. 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 organizations, scalability, improved security and privacy, Single Sign On (SSO) for users and so on [2]. For example, users can take advantage of SSO and thus authenticate themselves in one identity domain and receive personalized services across multiple domains without any further authentication. There are three major actors within a FIM System:

- **Identity Provider (IdP):** An entity that is responsible for managing digital identities of users and providing identity related services to different Service Providers.
- **Service Provider (SP):** An entity that is responsible for providing online services to the users.
- **Users:** An entity that receives services from an SP.

Metadata and Flow in FIM: Metadata is a crucial component in a SAML-based identity federation. 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 signing and encryption, 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). Metadata serves a major role in building trust within FIM. 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, it is imperative that such metadata file is securely stored by the corresponding IdPs and SPs. Usually, digital signatures are employed to ensure authenticity. However, there is still an issue of availability. In general, such metadata file is stored in a file system. If this file system is unavailable, such metadata cannot be accessed, thereby hampering corresponding federated services.

A simple flow of interactions between different entities within a SAML federation is discussed next [14]. At first, a federation between an IdP and different SPs is created. When a user submits a request to access a service from an SP, the user is forwarded to the respective IdP of the federation with a SAML authentication request. The IdP authenticates the user and then the user is redirected back to the SP with a SAML response containing a SAML assertion, a signed data containing user attributes. The SP retrieves the assertion, validates its signature and extracts the attributes. Based on the attribute values, the SP then takes an authorization decision.

Dynamic Federation: Another important issue of any SAML-based federation is the mechanism by which federations are created: by exchanging the corresponding metadata. This process is static in the sense that such metadata are exchanged in an out-of-bound fashion by the administrators of the respective organizations [12]. Also, this must be done before the entities can access any federated services. Exchanging and maintaining the repositories of metadata and thus managing trust becomes extremely difficult as the number of the IdPs and SPs grows within a federation and is a well-known problem of SAML [12], [15]. To mitigate this issue, the notion of dynamic identity federations has been introduced [8]. A dynamic federation is a mechanism in which a federation among the respective IdPs and SPs is created in a dynamic fashion. There have been several works that have explored how dynamic federations could be managed [3], [8]–[12].

However, these works require significant changes in the core SAML standard and they did not consider the issues of trust while creating dynamic federations. To counteract this issue, the authors in [3], [12] proposed mechanisms which do not require any significant change in the SAML standard. Instead, their proposal can be accommodated by modifying the respective implementation of SAML. Furthermore, they analyzed detailed trust issues. Because of these, we have used the works presented in [3], [12] as the basis of our current proposal. Next, we present a brief description of their approaches. Their approach allows any user of an IdP to dynamically federate an SP with the IdP. For this, the user generates a code in the IdP which is stored in the IdP database for a certain period. Then the user visits a specific page on the SP and provides the Entity ID of the IdP and the code to initiate the dynamic federation process. The SP interacts with the IdP by providing the code and both entities dynamically exchange their metadata which are then stored in their respective TAL, thus creating the federation. However, one common weakness in all existing proposals is the lack of any detailed security considerations. For example, they did not consider any rigorous threat model and other security properties such as confidentiality, integrity, availability and transparency have been mostly overlooked.

B. Blockchain

Bitcoin is regarded as the first widely-used decentralized digital currency that does not rely on a central entity, such as a central bank, for its creation and circulation [16]. Its main technological breakthrough is due to its underlying mechanism called *blockchain*, which is a smartly engineered distributed system featuring an immutable ledger of transactions shared and validated by a number of distributed Peer-to-Peer (P2P) nodes [13]. The ledger is an ordered data structure consisting of many blocks chained together by cryptographic mechanisms. Each block contains some transactions where each transaction enables a user to transact a certain amount of currency/data to another user/users. Each block refers to its previous block using a cryptographic hash, which refers to its previous block and so on, hence forming a chain and colloquially known as *blockchain*. *Smart-contract (SC)* are computer programs deployed on top of the respective blockchain [17]. Being part of the ledger makes SC and their executions immutable and irreversible, a sought-after property having a wide range of applications in different domains. A blockchain can be *public*, allowing everyone to participate or *private*, where only authorized parties can participate. Bitcoin [18] and Ethereum (Main-net) [19] are examples of public blockchains, whereas Hyperledger platforms [20] and Quoram [21] are examples of private blockchain systems.

In this paper, we present a novel application of blockchain to address the issues concerning confidentiality, integrity, availability, authenticity and transparency, in the setting of dynamic identity federations, which has not been explored before.

III. Threat Modelling & Requirement Analysis

In this section, we present a threat model (Section III-A) and analyze a number of functional and security requirements (Section III-B) for the proposed blockchain framework for metadata management and dynamic federations.

A. Threat Modelling

Threat modeling is a crucial step for designing and developing a secure framework that can be used for identifying and mitigating threats involving IT assets, metadata management, and dynamic federations in the scope of this paper. To model threats, we have chosen a well-established threat model called STRIDE [22], which encapsulates different security threats. Next, we discuss these threats modelled within the scope of the current work.

- **T1-Spoofing Identity:** An attacker pretends to be a user of the system for malicious purposes (e.g., initiating a federation process).
- **T2-Tampering with Data:** An attacker might alter the metadata of an entity (or other entities) stored in the file system in such a way that the usual federation functionality is no longer possible.
- **T3-Repudiation:** An attacker can repudiate after changing metadata or initiate a dynamic federation.
- **T4-Information Disclosure:** Sensitive data in the system is leaked to an attacker unintentionally.
- **T5-Denial of Service (DoS):** The data (e.g., metadata) or system (to be used for dynamic federations) becomes unavailable to a DoS attack.
- **T6-Elevation of privilege:** An attacker might elevate his privilege, without the knowledge of a responsible user (e.g., an admin), by exploiting other attack vectors such as malicious software in such a way that invades the system's authorization capabilities.

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

- **T7- Replay Attack:** An attacker can capture and reuse any previous message to dynamically add a previously removed entity (removed due to malicious activities, e.g., not satisfying the federation contract).

B. Requirement Analysis

In this section, we present a number of functional and security requirements to ensure that the system functions as intended and to mitigate the identified threats respectively.

Functional Requirements (FR):

F1. The system should provide a distributed metadata and TAL storage facility.

F2. The system should ensure that different entities can be federated in a dynamic fashion.

F3. The system should minimally affect the existing SAML functionalities such as authentication and SSO functions.

Security Requirements (SR):

S1. The system should ensure that only the authenticated and authorized users can access the corresponding metadata and initiate the federation process. This mitigates *T1*.

S2. Any data involving the management of metadata and dynamic federations must be communicated securely in the network so as to ensure the confidentiality, authenticity, and integrity of the corresponding request and response. This will mitigate *T2* and *T4*.

S3. The system should log every request and response in an immutable fashion. *S2* and *S3* can combinedly mitigate *T3* and *T6*.

S4. The system should take protective measures against any DoS attack so that it can deter *T5*

S5. The system should take the necessary steps so that an attacker cannot replay any previous request and response. This can mitigate *T7*.

IV. Architecture & Implementation

In this section, we present the architecture of the proposed framework, which is illustrated in Figure 1. The architecture consists of a single identity federation with an IdP and several SPs with each entity connected to a blockchain platform. At a higher level, there are three major components within the architecture: the blockchain platform, DApp (Decentralized Application), and federated entities. Next, we discuss the functionalities of each of these components (Section IV-A, Section IV-B and Section IV-C respectively) along with their implementation details and inter-component interactions.

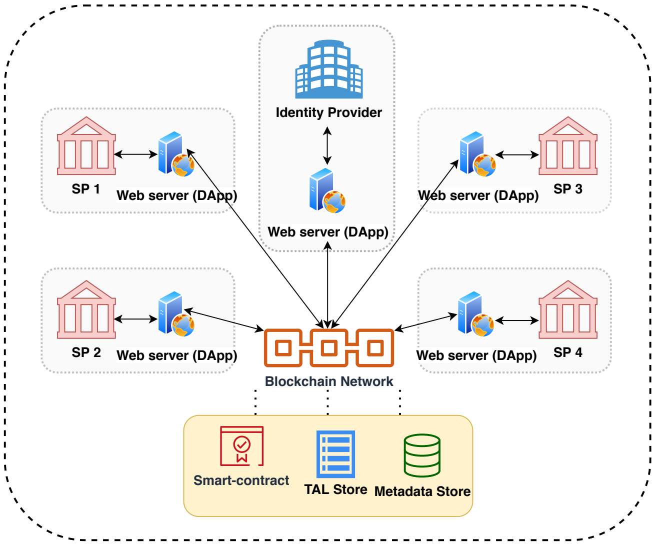


Fig. 1: High-level Architecture

A. Blockchain Platform

The blockchain platform is the central component in the architecture. Such a platform can be public or private. Public blockchain systems are more secure, however, such systems are still extremely slow, consume significant energy, open to all and there is a significant cost involved to process and store data in a SC supported public blockchain (e.g., Ethereum (Mainnet)) [13]. 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 itself remits for a private blockchain platform. For these reasons, we have utilized a private blockchain platform.

The blockchain platform serves the following three purposes within the proposed framework:

- It acts as the metadata store for each federated entity to store their respective metadata using a secure protocol.
- It acts as the TAL store for each respective entity.
- SC in the blockchain is used to store metadata, maintain TAL, and execute immutable logic for a federation.

There are many private blockchain platforms such as Hyperledger Fabric [23], Hyperledger Sawtooth [24], Quorum, and so on. However, Hyperledger Fabric is currently the most stable and popular private blockchain platform [25]. It also has a unique concept of *channel* by which different blockchains can be maintained within the same network, thus facilitating a privacy layer among different organizations. That is why Hyperledger Fabric is selected as our preferred blockchain system during the deployment phase.

The Fabric utilizes many network entities such as peers, endorsers, and orderers [25]. A SC is called a *chaincode* in Fabric terminology, which can be invoked using transactions. A user utilizes a peer for submitting a transaction which is then validated by Fabric entities. Once validated, a block is created by the Orderer and returned to other entities who add them to the blockchain.

The chaincode for the proposed framework has been written in Go, where the blockchain platform consists of three types of organizations (representing the blockchain authority, the IdP and the SP) and each organization consists of two endorsers and peers. The platform is deployed using Docker containers, where each container assumes the role of one of these entities. Also, there is an additional container that plays the role of the MSP (Membership Service Provider), including the CA (Certificate Authority). These entities are connected via a channel into which the chaincode is deployed. The consensus is based on Kafka (a distributed event streaming platform [26]) with two additional orderer nodes for block creation and dissemination as described above.

B. DApp

A DApp is a web server that acts as the middleware between any web application and blockchain. Since a SAML-based identity federation does not have any provision for interacting with the blockchain, a DApp is required for different entities to interact with the blockchain. We have DApps for the IdP and each SP where each DApp exposes APIs to the respective entity. The SAML interface of each entity uses these APIs to submit some requests (e.g., for storing and retrieving metadata) to the DApp. These requests are translated into blockchain transactions by the DApp and are submitted to the blockchain via a node. Each of these transactions essentially invokes a SC logic, and then the usual flows takes place.

The DApp in the proposed framework has been developed using Node.js with Express [27], [28]. 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. Fabric provides the required APIs to interact with any Node.js application. The DAapp has been integrated with each federated entity in such a way so that these entities can interact with the blockchain platform using the protocol discussed below (Section V).

C. Federated Entities

The federated entities consist of the IdP and several SPs. These entities are connected to the blockchain network via their respective DApp hosted within their own peer node. We have used SimpleSAMLphp [29] to provide SAML identity federations services (e.g., authentication and SSO services) for each entity. We have modified the SimpleSAMLphp codebase as per our requirements to support dynamic metadata exchange procedures and blockchain-based federation controls. The IdP within our setup stores the user identity credentials and other attributes in its local database. Each one of the SP and IdP is assigned a URL that acts as its *Entity ID*, a unique identifier for the entity. It is to be noted that, initially, the SPs and IdP are not members of an identity federation. Once they follow the protocol flow, described next, using our framework, they become part of an identity federation.

V. Protocol Flow

In this section, we present the protocol flow between different components of the proposed system. The mathematical notations and data model used in the protocol are introduced in Table I and Table II respectively.

As per the flow, the administrator (*admin*, in short) of the SP (denoted with A_{SP}) initiates the process by logging in to a dedicated page at the SP. On that page, the admin generates a random code, and submits an approval request for joining the federation to the target IdP. The administrator of the IdP (denoted with A_{IDP}) receives the request, generates another random code and submits a response back to the SP. At this moment, both the SP and IdP have received randomly generated codes from each other. The exchanged codes are verified, and if successful, the Entity ID of the SP is added in the TAL store of the IdP in the blockchain and vice versa. This completes the federation joining process.

Data Model: The proposed protocol is a request-response protocol. The request data model (denoted with *req* in Table II) consists of *type* and *data*, where *type* denotes the data type and *data* represents the data in the request. The response data model (denoted with *resp* in Table II) returns a *message* to denote if the operation has succeeded or failed.

There are three types of data: "*approval*" used for adding an SP to the IdP, "*addTal*" for adding the entity id of an entity to the TAL of the respective entity and "*removeTal*" for removing entity id from the TAL. The corresponding data model for the *approval* type is called "*approvalData*" and is defined in Table II where src_{eID} and $dest_{eID}$ denote the Entity ID of the requester and requestee respectively, C_{sp} and C_{idp} denote the randomly generated codes by the SP and IdP respectively and finally, $check_{sp}$ and $check_{idp}$ represent boolean values indicating whether the corresponding code is modified or not.

Similarly, the corresponding data model for both "*addTal*" and "*removeTal*" is denoted with "*talData*" and defined in Table II where $owner_{eID}$ represents the entity that controls the TAL and $service_{eID}$ represents the entity that is either to be added or removed from the previously mentioned TAL.

TABLE I: Cryptographic Notations

Notations	Description
eID	Entity ID of an entity
SP	A service provider willing to join a federation
IDP	An identity provider in a federation
K_{SP}	Public key of the service provider
K_{IDP}	Public key of the identity provider
K_{SP}^{-1}	Private key of the service provider
K_{IDP}^{-1}	Private key of the identity provider
N_i	A fresh nonce
$\{\}_K$	Encryption operation using a public key K
$\{\}_{K^{-1}}$	Signature using a private key K^{-1}
$[]_{https}$	Communication over HTTPS channel K

TABLE II: Data Model

$req \triangleq \langle type, data\rangle$
$resp \triangleq \langle message\rangle$
$TYPE \triangleq \langle approval, addTal, removeTal\rangle$
$DATA \triangleq \langle approvalData, talData\rangle$
$approvalData \triangleq \langle src_{eID}, dest_{eID}, C_{sp}, C_{idp}, check_{sp}, check_{idp}\rangle$
$talData \triangleq \langle owner_{eID}, service_{eID}\rangle$

Algorithm: The algorithm for the chaincode for our system is presented in Algorithm 1. The important functionalities of our system are to submit an approval, add (or remove) an Entity ID to (from) the TAL of the respective entity. The *invoke* function is the starting point in every chaincode. Depending on the type of the request, different functions are called in chaincode (Line 4 to Line 13 in Algorithm 1), where each function represents the three core functionalities of our system. The algorithm to process approval is handled by the *submitApproval* function (Line 15 to Line 29 in Algorithm 1). Similarly, the add and removal of TAL are handled by the *addTal* function (Line 34 to Line 38 in Algorithm 1) and by the *removeTal* function (Line 41 to Line 45 in Algorithm 1) respectively.

Protocol Flow: Next, we illustrate the protocol flow of creating and updating a dynamic federation using two use-cases respectively: adding an SP and removing an SP to/from the federation. The protocol flow for the first use-case is presented in Table III and illustrated in Figure 2. In this protocol, after authentication (skipped in the protocol flow for brevity), an admin of the SP generates a random code (C_{sp}). Then the admin provides the source Entity ID (its Entity ID, denoted with src_{eID}), the destination Entity ID (IdP's Entity ID, denoted with $dest_{eID}$) and C_{sp} to create an *approval* request and submits the request to the IdP for adding the SP to the IdP's TAL (*M1* in Table III). The SP also adds this request in the blockchain using the DApp with the *submitApproval* function in the chaincode (*M2* and *M3* messages in Table III).

After getting the approval request, IdP also generates a random code (C_{idp}) and creates another approval request using its Entity ID as src_{eID}, the SP's entity ID as $dest_{eID}$, C_{sp} and C_{idp}. The request is then sent back to the SP (*M4* in Table III). Like before, the IdP also stores this request in the blockchain via the DApp with the *submitApproval* function in the chaincode (*M5* and *M6* messages in Table III.

Once the SP receives the approval request from the IdP, it checks its sent C_{sp} with IdPs' C_{sp} and if both match, SP

Algorithm 1: SCC: `// ▷ System Chaincode`

```
Input: $req \rightarrow$ the request from the user
Output: $resp \rightarrow$ the chaincode generated response
Start
    function invoke($req$)
        $data := req.data$;
        $type := req.type$;
        if $req.type$ == $approval$ then
            $resp$ = submitApproval($data$);
        else if $req.type$ == $tal$ then
            $resp$ = addTal($data$);
        else
            $resp$ = removeTal($data$);
        end
        send $resp$ back to user;
    function submitApproval($data$)
        $newApproval$ := data.approval;
        $Owner$ := data.$src_{eID}$;
        $approvalArray$ :=
          fetchSubmittedApproval($data$);
        if $newApproval$ exist in $approvalArray$ then
            $existingApproval$ :=
              approvalArray[newApproval];
            $owner := existingApproval.src_{eID}$;
            if $owner$ == $idp$ then
                $existingApproval.C_{idp}$ :=
                  randomNumber;
            else if $owner$ == $sp$ then
                $existingApproval.C_{sp}$ :=
                  randomNumber;
        else
            $putState(approvalStore, newApproval)$;
        end
        return $TRUE$;
    function fetchSubmittedApproval($data$)
        $Owner$ := data.$src_{eID}$;
        $approval = getState(approvalStore, Owner)$;
        return $approval$;
    function addTal($data$)
        $Owner_{eID} := data.owner_{eID}$;
        $Service_{eID} := data.service_{eID}$;
        $Approval$ :=
          $getState(approvalStore, Owner_{eID})$;
        if $Approval.Check_{sp} == true$ And
          $Approval.Check_{idp} == true$ then
            $putState(talStore, Owner_{eID}, Service_{eID})$;
        return $approval$;
    function removeTal($data$)
        $Owner_{eID} := data.owner_{eID}$;
        $Service_{eID} := data.service_{eID}$;
        $removeState(talStore, Owner_{eID}, Service_{eID})$;
        return $TRUE$;
```

TABLE III: Protocol for adding an entity to a TAL

$M1$	$SP \rightarrow IDP:$	$[N_1$,approval, $\{approval\}_{K_{sp}^{-1}}]$https
$M2$	$SP \rightarrow DAPP:$	$[N_2, approval]_{https}$
$M3$	$DAPP \rightarrow BC:$	$approval$
$M4$	$IDP \rightarrow SP:$	$[N_3$,approval, $\{approval\}_{K_{idp}^{-1}}]$https
$M5$	$IDP \rightarrow DAPP:$	$[N_4, approval]_{https}$
$M6$	$DAPP \rightarrow BC:$	$approval$
$M7$	$SP \rightarrow IDP:$	$[N_5$,approval, $\{approval\}_{K_{sp}^{-1}}]$https
$M8$	$IDP \rightarrow SP:$	$[N_6$,approval, $\{approval\}_{K_{idp}^{-1}}]$https

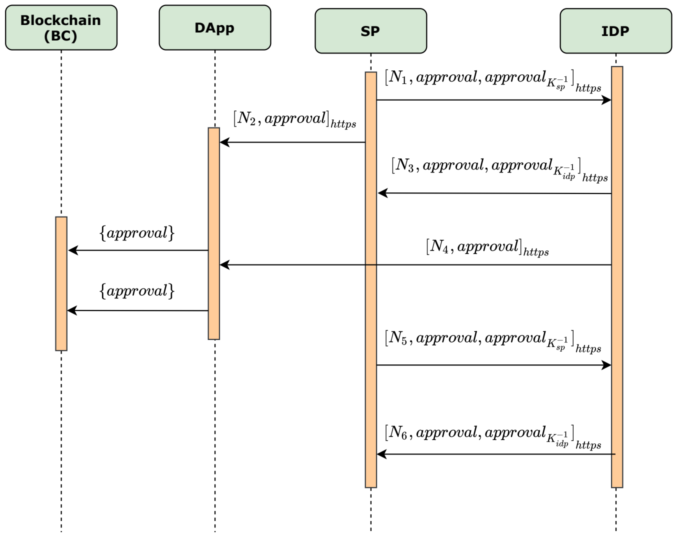

Blockchain (BC)
DApp
SP
IDP
$[N_1, approval, approval_{K_{sp}^{-1}}]_{https}$
$[N_2, approval]_{https}$
$[N_3, approval, approval_{K_{idp}^{-1}}]_{https}$
$\{approval\}$
$[N_4, approval]_{https}$
$\{approval\}$
$[N_5, approval, approval_{K_{sp}^{-1}}]_{https}$
$[N_6, approval, approval_{K_{idp}^{-1}}]_{https}$

Fig. 2: Protocol flow of adding an entity

fills the $Check_{sp}$ field with *TRUE*, otherwise, it is tagged as *FALSE*. Next, SP sends the updated approval again to the IdP (*M7* in Table III). The IdP then retrieves the C_{idp} from the last approval and checks whether it matches with his generated C_{idp}. If they match and $Check_{sp}$ is true, IdP adds SP to his TAL in the blockchain using the *addTal* function of the chaincode. In the last step, IdP then sends the approval back to the SP by modifying the $check_{idp}$ field with *TRUE*, assuming all checks match (*M8* in Table III). After receiving the last approval from the IdP, the SP checks whether both the $check_{sp}$ and $check_{idp}$ are true or not. Assuming both are true (indicating mutual checks are valid), the SP adds the IdP to its TAL using the *addTal* function of the chaincode. When an IdP or SP would like to remove a particular entity from its TAL, they initiate the protocol flow presented in Table IV. From the table, the requesting entity just submits a removal request to the blockchain via the DApp (*M1* message in Table IV which invokes the *removeTal* function in the chaincode and the requested entity is removed from its corresponding TAL.

It is to be noted that all requests and responses are transmitted over an HTTPS channel as indicated in the protocol in Table III and Table IV. Also, all requests are digitally signed to ensure authenticity.

TABLE IV: Protocol for removing an entity from a TAL

$M1$	$SP \rightarrow DAPP:$	$[N_1, E_{sp}, E_{idp}]_{https}$
$M2$	$DAPP \rightarrow BC:$	E_{sp}, E_{idp}

VI. Evaluation

In this section, we evaluate and compare the performance, concerning throughput and latency, between a typical SAML based identity federation using SimpleSAMLphp and a dynamically federated identity federation using our framework. Towards this aim, we have conducted load testing experiments by simulating several use-cases for each of these two scenarios. The use-cases involve typical SAML interactions between a user, an IdP, and an SP as discussed next.

- **User Creation:** A group of user accounts has been created and stored in a local database under the Identity Provider (IdP) for authenticating users and accessing various services from different SPs.
- **Service Request:** At first, a user submits a request to access a particular service from an SP. In our experimental settings, two SPs provide four different services. To access a service from any SP, the user must be authenticated by the corresponding IdP of the federation.
- **User Authentication:** The user is then redirected to the authentication page of the designated IdP via a SAML authentication request. Upon receiving the request, the IdP verifies if the SP is a member of the federation and then the user authenticated. Afterwards, the user is redirected back to the SP with a SAML response.
- **Service Response:** The SP receives the SAML response, retrieves the assertion, verifies the signature in the assertion and finally, retrieves the attributes of the user from the assertion. The user is then granted access.

That is, a single cycle of the use-case consists of service request, user authentication, and service response. For different scenarios, different simultaneous cycles, each cycle representing a single user, have been simulated using Apache JMeter [30], a load testing software. For the first scenario (the typical SAML deployment), load testing has been conducted by increasing the traffic steadily, starting from 10 users (10 cycles) up to 100 users (100 cycles). For the second scenario (dynamic identity federation using our proposed approach), there are three variations in each cycle: Fabric blockchain with 1 orderer, 2 orderers, and 3 orderers. Thus, each variation represents a different blockchain network configuration with a varying number of orderers and peers to investigate their impact on the performance. For both scenarios, our system has been deployed in Amazon Web Services (AWS) EC2 instances powered by 4 VCPUs and 16 Gigabytes of RAM. The different blockchain network configurations have been initiated with various orderer nodes using docker in AWS EC2 instances.

Performance evaluation: At the end of each test cycle, the data are collected and recorded for comparison with the next cycle. Latency and throughput are two main matrices that have been selected for the evaluation of the performance of our framework as they are more representative to evaluate performances in comparison to other matrices such as CPU usage, data transfer, and so on. The comparative evaluation result of different experiments involving two scenarios are presented in Figure 3.

An important observation from Figure 3a is that the latency of the system has increased significantly in a linear fashion with the increase in the traffic load. This is mainly because an instance of user authentication in an identity federation has to go through several request-response protocols and redirects before an SP can either grant or reject access. This complex flow of messages between various entities introduces a certain amount of delay in the system.

The throughput of the system under varying workloads is depicted in Figure 3b. It can be observed that the system reaches a maximum value and then continues to decrease. This trend can be observed in both the typical SAML federation as well as in the proposed blockchain-based solution. From our experiments with the varying number of orderers in the Hyperledger Fabric platform, it is also evident that our proposed mechanism has not hindered the performance of the system. In fact, it has outperformed the typical SAML federation architecture in most of the cases with higher throughput and lower latency.

One might wonder about the impact of the significantly high latency and low throughput as the number of users tends to grow. However, this is true for both the typical scenario and the scenarios involving blockchain, even though our proposed system, as mentioned earlier, performed slightly better. Therefore, we have achieved the level of security that a blockchain platform provides (addressing the aforementioned security issues in II-B) without compromising any performance.

VII. Discussion

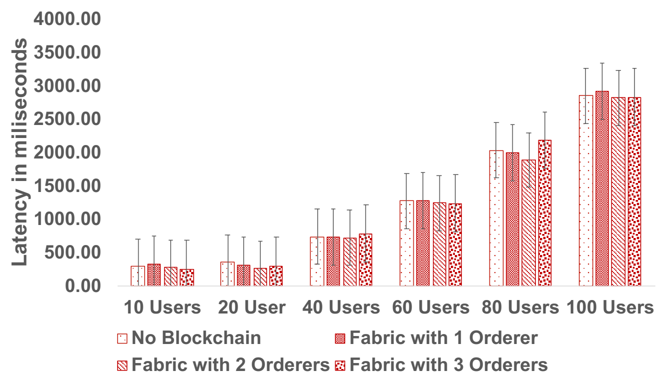
In this section, we explore how our proposed system has satisfied different requirements (Section VII-A) and discuss its advantages, disadvantages (Section VII-B) and future work (Section VII-C).

A. Analysing requirements

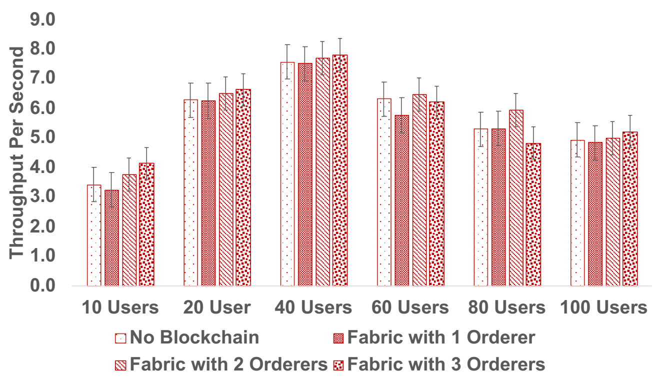
The functional and security requirements are analyzed first.

Functional requirements: The proposed system leverages blockchain as metadata and TAL store. The distributed nature of blockchain thus satisfies *F1*. The protocols for the proposed system can be used to federate any new entity dynamically, thereby satisfying *F2*. Finally, the framework has been integrated with a SimpleSAMLphp implementation which facilitates all functionalities (e.g., SSO and authentication) of a SAML identity federation without any hindrance and hence, satisfies *F3*.

Security requirements: Only properly authenticated and authorized users (mostly admins) can initiate a dynamic federation process. This satisfies *S1*. Data between every single entity and the DApp are exchanged over secure HTTPS channels, thus ensuring confidentiality and integrity of transmitted data. Furthermore, metadata stored in the blockchain remains immutable and thus, proves integrity for data storage. Every single blockchain transaction submitted by any DApp is digitally signed by the respective private key of a node which ensures authenticity. All these mechanisms combinedly satisfy *S2*.



(a) Latency vs Load



(b) Throughput vs Load

Fig. 3: Performance evaluation

Fabric records every related transaction request and response in the blockchain and hence satisfies *S3*. Being a blockchain platform, Fabric with proper fault tolerance mechanisms has strong support to mitigate any DoS attack. This satisfies *S4*. To satisfy *S5*, nonces have been extensively used in every step of the protocol step, thus satisfying *S5*.

B. Advantages & Disadvantages

Our blockchain-based approach to managing dynamic identity federations offers several advantages:

- We have modified the process by which SimpleSAMLphp manages TAL within its file system. This feature has enabled TAL to be stored in a distributed storage, ensuring integrity and availability.
- We have used the chaincode (SC) in the Fabric blockchain to execute and validate the main business logic for the proposed system. This includes access control for the TAL of all entities, public-key encrypted storage for metadata for the hosts that do not require a dedicated metadata management feature, and a set of rules for handling the flow of dynamic federation creation. This ensures that the process is autonomous, distributed, and decentralized and hence, is resilient against any single point of failure.
- The blockchain-based system also provides a strong guarantee for immutability and transparency by storing every interaction in the blockchain.

However, the current deployment also has a few disadvantages:

- The database for storing user credentials in the IdP is centralized and hence can create a single point of failure, To mitigate this, blockchain can be used as a credential store as well. However, proper security measures must be ensured to facilitate this.
- The presence of only a single IdP can introduce a single point of failure within a federation. A blockchain-based IdP can mitigate this issue.

C. Future Work

The blockchain-based mechanism that we have proposed in this paper has been developed to allow unknown parties to join a federation dynamically. In addition to user authentication, a dynamic access control protocol mechanism is needed. We could also move the user attributes to a decentralized platform as a step further. Even we could disrupt the existing SAML identity federation mechanism by simulating the functionality of an IdP using SC in a blockchain. In addition, there is scope for improvement in the performance of such systems. It will be interesting to see how other blockchain platforms or consensus algorithms perform and provide different levels of fault tolerance when integrated with an identity federation system. In the future, we will also investigate how blockchain configurations can be optimized to reduce the latency and/or increase throughput.

VIII. Conclusion

This paper has presented a blockchain-based dynamic identity federation framework that aims to address crucial issues: metadata decentralization and dynamic identity federation. Initially, we have formulated a set of functional and security requirements based on a threat model. Our proposed framework has been designed to satisfy the developed requirements and mitigate the identified issues. Furthermore, we have evaluated its performance and analyzed its security features, advantages, and limitations. Using our dynamic identity management framework, enterprises can set up their authentication and SSO services. Apart from this, the facility of dynamic federations will allow inter-enterprise cooperation and service sharing. Being rooted in a state-of-the-art private blockchain platform, Hyperledger Fabric, the dynamic identity management framework, offers several security advantages over existing technologies. Thus, we firmly believe that our method will usher a new research area in the setting of federated identity management.

Acknowledgement

This work was in part supported by the Shahjalal University of Science and Technology Research Centre Grant (No. AS/2018/2/25) and AWS Cloud Credits for Research Grant.

References

[1] A. Josang, M. AlZomai, and S. Suriadi, “Usability and privacy in identity management architectures,” in ACSW Frontiers 2007: Proceedings of the Fifth Australasian Symposium on ACSW Frontiers, 2007, pp. 143–152.

[2] D. W. Chadwick, “Federated identity management,” in Foundations of security analysis and design V. Springer, 2009, pp. 96–120.

[3] M. S. Ferdous and R. Poet, “Dynamic identity federation using security assertion markup language (saml),” in IFIP Working Conference on Policies and Research in Identity Management, 2013, pp. 131–146.

[4] S. Cantor, J. Moreh, R. Philpott, and E. Maler. (2018) “Metadata for the OASIS security assertion markup language (SAML) V2. 0”. Accessed: 01-09-2020. [Online]. Available: http://docs.oasis-open.org/security/saml/v2.0/saml-metadata-2.0-os.pdf

[5] S. Consortium. “Shibboleth ”. Accessed: 01-09-2020. [Online]. Available: https://www.shibboleth.net/

[6] UNINETT. “SimpleSAMLphp”. Accessed: 01-09-2020. [Online]. Available: https://simplesamlphp.org/

[7] ZXID. “ZXID”. Accessed: 01-09-2020. [Online]. Available: http://www.zxid.org/

[8] P. Harding, L. Johansson, and N. Klingenstein, “Dynamic security assertion markup language: Simplifying single sign-on,” IEEE Security & Privacy, vol. 6, no. 2, pp. 83–85, 2008.

[9] Y. Xiang, J. A. Kennedy, M. Egger, and H. Richter, “Network and trust model for dynamic federation,” in Proceedings of the Fourth International Conference on Advanced Engineering Computing and Applications in Sciences, 2010, pp. 1–6.

[10] Y. Zuo, X. Luo, and F. Zeng, “Towards a dynamic federation framework based on saml and automated trust negotiation,” in International Conference on Web Information Systems and Mining, 2010, pp. 254–262.

[11] P. A. Cabarcos, F. A. Mendoza, A. Marín-López, and D. Díaz-Sánchez, “Enabling saml for dynamic identity federation management,” in Joint IFIP Wireless and Mobile Networking Conference, 2009, pp. 173–184.

[12] M. S. Ferdous and R. Poet, “Managing dynamic identity federations using security assertion markup language,” Journal of Theoretical and Applied Electronic Commerce Research, vol. 10, no. 2, pp. 53–76, 2015.

[13] M. J. M. Chowdhury, M. S. Ferdous, K. Biswas, N. Chowdhury, A. Kayes, M. Alazab, and P. Watters, “A comparative analysis of distributed ledger technology platforms,” IEEE Access, vol. 7, no. 1, pp. 167 930–167 943, 2019.

[14] M. S. Ferdous, F. Chowdhury, M. O. Alassafi, A. A. Alshdadi, and V. Chang, “Social anchor: Privacy-friendly attribute aggregation from social networks,” IEEE Access, vol. 8, pp. 61 844–61 871, 2020.

[15] R. Poortinga-van Wijnen, M. S. Bargh, B. Hulsebosch, and H. Zandbelt. (2010) Scalability of trust and metadata exchange across federations. Accessed: 2020-09-15. [Online]. Available: https://tnc2011.terena.org/getfile/693

[16] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” Manubot, Tech. Rep., 2019.

[17] M. S. Ferdous, F. Chowdhury, and M. O. Alassafi, “In search of self-sovereign identity leveraging blockchain technology,” IEEE Access, vol. 7, pp. 103 059–103 079, 2019.

[18] “Bitcoin”. Accessed: 2020-07-10. [Online]. Available: https://www.bitcoin.org/

[19] “Ethereum”. Accessed: 2020-07-10. [Online]. Available: https://www.ethereum.org/

[20] “Hyperledger”. Accessed: 2020-07-10. [Online]. Available: https://www.hyperledger.org/

[21] “Quorum Blockchain”. Accessed: 2020-07-10. [Online]. Available: https://www.goquorum.com/

[22] A. Shostack, Threat modeling: Designing for security. John Wiley & Sons, 2014.

[23] “Hyperledger Fabric”. Accessed: 2020-07-10. [Online]. Available: https://www.hyperledger.org/use/fabric

[24] “Hyperledger Sawtooth”. Accessed: 2020-07-10. [Online]. Available: https://www.hyperledger.org/use/sawtooth

[25] M. Bhuiyan, S. Islam, A. Razzak, M. S. Ferdous, M. J. M. Chowdhury, M. A. Hoque, and S. Tarkoma, “Bonik: A blockchain empowered chatbot for financial transactions,” in IEEE TrustCom 2020, 2020.

[26] “Apache Kafka”. Accessed: 2021-02-26. [Online]. Available: https://kafka.apache.org/

[27] “Node.js”. Accessed: 2020-07-10. [Online]. Available: https://nodejs.org/en/

[28] “Express JS”. Accessed: 2020-07-10. [Online]. Available: https://expressjs.com/

[29] SimpleSAMLphp. (2020, December 06) Simplesamlphp documentation. Accessed: 2020-12-06. [Online]. Available: https://simplesamlphp.org/docs/stable/

[30] A. JMeter. (2020, December 16) Apache jmeter. Accessed: 2020-12-16. [Online]. Available: https://jmeter.apache.org/