A Distributed Access Control System for Cloud Federations

Shorouq Alansari, Federica Paci, Vladimiro Sassone
University of Southampton
{sa3n13; f.m.paci;vsassone}@soton.ac.uk

Abstract—**Cloud federations are a new collaboration paradigm where organizations share data across their private cloud infrastructures. However, the adoption of cloud federations is hindered by federated organizations' concerns on potential risks of data leakage and data misuse. For cloud federations to be viable, federated organizations' privacy concerns should be alleviated by providing mechanisms that allow organizations to control which users from other federated organizations can access which data. We propose a novel identity and access management system for cloud federations. The system allows federated organizations to enforce attribute-based access control policies on their data in a privacy-preserving fashion. Users are granted access to federated data when their identity attributes match the policies, but without revealing their attributes to the federated organization owning data. The system also guarantees the integrity of the policy evaluation process by using blockchain technology and Intel SGX trusted hardware. It uses blockchain to ensure that users identity attributes and access control policies cannot be modified by a malicious user, while Intel SGX protects the integrity and confidentiality of the policy enforcement process. We present the access control protocol, the system architecture and discuss future extensions.**

Index Terms—**Blockchain, Access control, Anonymous identities, Cloud federation.**

I. Introduction

Cloud computing has enabled cross-organizational collaborations called *cloud federations*. In cloud federations partner organizations share data hosted on their private cloud infrastructures in order to achieve a common business goal. Thereby, the data stored in one cloud infrastructure are available not only to the users of the organization owning the data, but also to users of other organizations that are part of the federation.

However, the realization of federations of private clouds is hindered by partner organizations' privacy concerns sharing its data across private cloud infrastructures.

For cloud federation to become a reality, researchers need to develop an identity and access management framework that allows organizations to authenticate users from other federate organizations and to determine which authenticated users can access which data in federated organizations' cloud infrastructures.

These systems should satisfy several security requirements. First, they should allow federated organizations to specify *fine-grained* access control policies based on data consumers' properties often called *identity attributes* rather than consumers' role. Second, As identity attributes encode consumer's sensitive information and revealing them to other federated organizations may not be desirable [1], systems should be able to preserve the privacy of these attributes. Third, the framework should guarantee the *integrity of the policy evaluation* process. Enforcing access control policies in a federated cloud requires an architecture where distributed components participate to the evaluation and enforcement of access control policies. Due to its distributed nature such an architecture is vulnerable to attacks that can compromise the policy evaluation process. A malicious user or software could take control of the host where the policy evaluation engine is running or the access control policies are stored. For example, it could modify the evaluation process to return always the same access control decision e.g permit or modify the access control policies.

Few access control frameworks to support secure data sharing in cloud federations have been recently proposed [2], [3], [1]. However the proposed frameworks do no support identity attributes privacy and do not guarantee the integrity of the policy evaluation process.

Contribution. In this paper we present an identity and access management system for secure data sharing in cloud federations. The framework allows federated organizations to specify fine-grained access control policies in terms of users identity attributes. The enforcement of the policies preserves users' identity attributes privacy in that a user does not reveal his identity attributes in clear to the federated organization owning the data nor the organization learns which policy is satisfied by the user. Policies are enforced by means of a cryptographic approach that supports efficient key management: data is encrypted with a symmetric key and user is able to reconstruct the key only if he satisfies the access control policy of the federated organization owning the data. The process extends a previously proposed cryptographic approach to privacy-preserving document broadcast [4].

In order to guarantee the integrity of the policy evaluation process the framework adopts two novel technologies: blockchain [5] and Intel's SGX [6] trusted hardware. Blockchain is a distributed ledger of data and computations on these data cannot be modified. The integrity of the data and computations stored on the blockchain is guaranteed by a group of nodes called *miners* that run a *consensus protocol*. Therefore, we use the blockchain to store the identity attributes of the users of federated organizations and the access control policies that protect the access to the federated data. However, we cannot use blockchain technology to ensure the integrity of the cryptographic approach to policy enforcement for two main reasons: the approach would be public and it would be too costly to deploy and execute on the blockchain the cryptographic operations on which the approach is based. For this reason, to guarantee the integrity of the policy enforcement process we adopt Intel's SGX, which is a trusted execution environment that preserves the integrity and confidentiality of sensitive code and data.

Paper Structure. The next section introduces basic technical background. Section III introduces the identity and access management protocol. Section IV presents the architecture's components. Section V discusses the related work, and Section VI concludes the paper outlining future extensions.

II. Background

In this section we provide basic background on the main technologies and cryptographic protocols used by the proposed access control system.

A. Blockchain and Smart Contracts

The blockchain was the novel technique behind Bitcoin [5]. In general, the blockchain is a public ledger of transactions managed by all the nodes within the cryptocurrency network. The blockchain keeps a log of all the transactions that have ever occurred in the network. Unlike traditionally centralized banking system, the blockchain does not relay on any trusted central authority. Instead, every node within the network is trusted to follow the protocol and maintain the blockchain.

Smart contracts is the key element in the second generation of blockchains, which enables a generally programmable infrastructure (i.e. Ethereum) [7]. Smart contracts are deployed and executed on the blockchain network and can be used to reach agreements and solve common problems with minimal trust.

B. Trusted Hardware

Trusted hardware platforms encompass the ability to run security-sensitive programs in secure and isolated software execution environment to protect the integrity and confidentiality of the computations against several attacks; for instance, Intel's newly released Software Guard Extensions (SGX) [6]. SGX is a set of extensions to Intel architecture that allows running trusted computations on a remote system. To achieve this, SGX relies on software attestation. Attestations provide users with a proof that a piece of software is running in a secure container, called *enclave*, hosted by the trusted hardware [8]. The enclave contains only the private data and the code that operates on it. Once instantiated, the the enclave is given a credential, also known as *report*. The report is digitally signed using a hardware-protected key to produce a proof, also called *quote*, which can be verified by a remote system [8].

C. Cryptographic Building Blocks

a) Pedersen commitment: Pedersen commitment scheme [9] is a two-phase protocol. The first phase, *commit*, enables one party Sender *S* to commit a value to another party Receiver *R*, in such a way that *R* does not know which value has been committed, and *S* cannot change the committed value. The second phase, *reveal*, includes revealing the original value, whereby *R* can verify that this is indeed the value to which *S* has committed.

Pedersen Commitment scheme is based on the discrete logarithm problem and works as follows:

Setup. *R* chooses large primes p and q such that q divides $p-1$. Let g be a generator of G_q, the order$-q$ subgroup of $\mathbb{Z}_p^*$. *R* picks a random secret a from $\mathbb{Z}_q$ and computes $h = g^a mod\ p$. *R* publishes p, q, g, h as system's parameters, while keeping a a secret.

Commit. *S* commits to $x \in \mathbb{Z}_q$ by choosing random $r \in \mathbb{Z}_q$ and sends $c = g^x h^r mod\ p$ to *R*.

Reveal. *S* shows the values x and r to open the commitment, *R* verifies that $c = g^x h^r mod\ p$.

b) OCBE protocols: Oblivious Commitment-Based Envelope (OCBE) protocol [10] enables users to deliver information in an oblivious way. The protocol involves three actors: a Sender *S*, a Receiver *R*, and a trusted Central Authority *CA*. *S* allows *R* to decrypt a sent information only if its identity attributes committed values satisfy an attribute-based access control policy and without *S* learning *R*'s attribute values.

The cryptographic building blocks of OCBE protocols are:

1) The Pedersen commitment scheme.
2) Symmetric key encryption algorithm $\mathcal{E}$ with key length k-bits. We use $\mathcal{E}_{key}[M]$ to denote the encrypted plaintext M with encryption key under the encryption algorithm $\mathcal{E}$.
3) Cryptographic hash function $H : G \rightarrow \{0,1\}^k$, where G is a finite cyclic group of large primes.

The protocol consists of the following phases:

Setup. *CA* takes a security parameter t and outputs the public parameters $Params$ for *commit*, a set $\mathcal{V}$ of possible values, and a set $\mathcal{P}$ of predicates. Each predicate in $\mathcal{P}$ maps an element in $\mathcal{V}$ to either *true* or *false*.

Commit. *R* chooses a value $a \in \mathcal{V}$ (*R*'s attribute value) and sends it to *CA*. *CA* picks a random number r and computes the commitment $c = \text{commit}_{Params}(a, r)$. *CA* gives c and r to *R*, and c to *S*.

Initialization. *S* chooses a message $M \in \{0,1\}^*$ and a predicate $Pred \in \mathcal{P}$ and then reveals $Pred$ to *R*. By the end of this phase, *S* has $Pred, c$ and M, while *R* has $Pred, c, a,$ and r.

Interaction. *S* sends an envelope containing an encryption of M to *R* via an interactive protocol.

Open. *R* extracts the massage M if $Pred(a)$ is *true*; otherwise, *R* does nothing.

III. The Proposed Protocol

The framework builds and extends a previous privacy-preserving scheme for content dissemination [4] in order to be executed on top of blockchain and Intel SGX. It involves four main entities: *O*, the federated organization owning the shared data, *R*, the user requesting access to the shared data, *IP*, the identity provider who manages the identity of the users in the cloud federation, *ACM*, the access control manager responsible for encrypting the data and generating the parameters needed to retrieve the decryption key. The main phases of the protocol are the followings:

a) Policy Specification: O defines a set of access control policies *ACP*s that specify which data D_i *R*s are authorized to access based on *R*s' identity attributes. Access control policies are formally defined as a conjunction of attribute conditions $cond_1 \wedge \ldots \wedge cond_n$. Each attribute condition $cond$ is in the form of $\langle name_a, op, v\rangle$, where $name_a$ is the name of an identity attribute a, op is a comparison operator such as $=, \neq, \leq, <, \geq, >$, and v is the value of attribute a. *O* sends the data[1] and the predefined policies *ACP*s to *ACM*.

b) Data Encryption: To encrypt the data, *ACM* goes through the following steps:

- *ACM* combines the different policies *ACP* that apply to the same data element D_i into a *Policy configuration* Pc_i. There can be multiple data elements $D_1, \ldots, D_i \in D$ which have the same policy configuration.
- *ACM* chooses an $\ell\prime$-bit prime number q, a cryptographic hash function $H(\cdot)$ whose output bit length is no shorter than $\ell\prime$, Key space $KS = \mathbb{F}_q$, where $\mathbb{F}_q$ is a finite field with q elements, and a semantically secure symmetric-key encryption algorithm with key length $\ell\prime$ bits. These public parameters are published and stored on-chain.
- *ACM* selects a key $K \in KS$ for a symmetric key encryption algorithm for each policy configuration Pc. *ACM* uses *K* to encrypt all data elements under the same Pc. So when the same policy configuration applies to multiple data elements, all data elements $D_1, \ldots, D_i \in D$ are encrypted with the same key *K*. The encrypted data is forwarded to an off-chain storage. While only references to the encrypted data Ref($\mathcal{E}_K[D]$) and the access control policies *ACP*s are kept on chain.
- *ACM* generates $r_{i,j} \in \mathbb{F}_q$ a conditional subscription secret (CSS) for each requester with pseudonym nym_i[2] and attribute condition $cond_j$. CSSs are kept hidden in *ACM*[3] until delivered to nym_i on the next phase. *ACM* creates a table *T* to maintain the delivered CSSs with respect to each nym_i and $cond_j$ in *ACP*.
- *ACM* picks N random bit strings $(z_1, \ldots, z_N)$ to create the access matrix A as in [4]. *ACM* then solves for a nonzero (N + 1)-dimensional column $\mathbb{F}_q$ -vector Y such that $AY = 0$. *Y* is called an access control vector. *ACM* sets the vector

  $$X = (K, 0, 0, \ldots, 0)^T + Y,$$

  Where v^T denotes the transpose of vector v, and K is the pre-chosen symmetric key. *ACM* publishes $\langle X, (\mathrm{z}_1, \ldots, z_N)\ \rangle$ on-chain along with the reference to data Ref($\mathcal{E}_K[D]$).

[1]To avoid sending data in clear or using a secure channel between O and *ACM*, the data itself is encrypted under the $\mathcal{W}_{ACM}$ public key pk_{ACM}. The decryption is done in $Encl_{ACM}$.

[2]later in the registration phase, *ACM* gives each requester a random unique pseudonym *nym*, as such all identity tokens for the same *R* have the same *nym*.

[3]Particularly, in $Encl_{ACM}$ as we will see later.

c) Identity Token Issuance Phase: *R* provides its identity attributes to *IP* and for each attribute, *IP* issues an identity token *IT*. *IT* is a tuple

$$IT = (nym, id - tag, c),$$

Where nym is a pseudonym for uniquely identifying the *IP* in the system, $id - tag$ is the name of the identity attribute, c is a Pedersen commitment for the attribute value x. *IP* digitally signs *IT*, then sends it to *R* to sign the identity token again. Finally, the identity tokens are stored on the blockchain. Without the signature of *R* on the identity token, any user could use the identity token to gain access to data shared within the federation because everything stored on the blockchain is public.

d) Identity Token Registration Phase: *R* registers its identity tokens with *ACM* to receive the CSSs required to derive the decryption key K for the shared data. During the registration, *R* receives a set of CSSs, based on the identity attribute names corresponding to the attribute names in the identity tokens. *ACM* retrieves the signed *IT*s from the blockchain. Then, it checks if $id - tag$ matches the name of the identity attribute in the access control policies, and verifies both the *IP*'s signature and *R*'s signature. If the signatures are valid, *ACM* publishes reference to the encrypted data Ref($\mathcal{E}_K[D_i]$) to *R*. *ACM* starts an OCBE session to send the CSS to *R*. It is worth mentioning that CSSs are the only information that demands a secure channel to be sent. *ACM* updates table *T* by adding the delivered CSSs along with the associated *R* pseudonym nym_i and the policy condition $cond_i$.

e) Data Access: *R* uses Ref($\mathcal{E}_K[D_i]$) to retrieve the encrypted data from off-chain storage. Then, *R* uses the public parameters published on the blockhain, and the CSSs to reconstruct the key *K* as in [4]. Once reconstructed the key *K*, *R* decrypts the data.

IV. System Architecture

The system architecture consists of several components distributed across the cloud infrastructures of the organizations that are part of the federation (see Fig.1). In what follows we will describe the behaviour of the main components of the architecture based on a formal abstraction of Intel SGX proposed in [11].

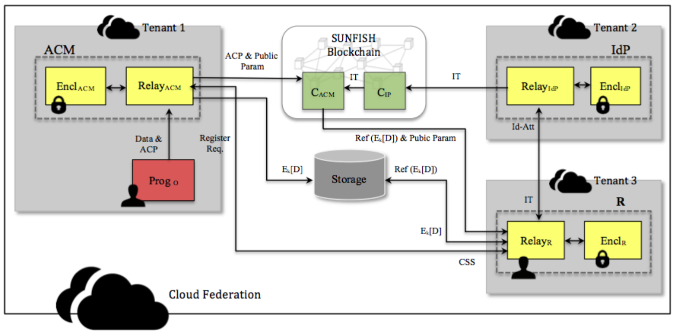

Fig. 1. System architecture

a) Data Owner O: a web application that allows data owner to specify access control policies and provide data to be shared to *ACM*.

b) Data Requester R: an SGX-enabled application which includes two main sub-components:

- $Relay_R$ provides in and out network traffic for $Encl_R$ (the program for $Relay_R$ is shown in Algo.1), as all SGX enclaves lack networking capabilities. It provides an interface to the users of the federation to requesting the necessary identity tokens *IT*s from *IP*, and then registering them with *ACM* to obtain CSSs.
- $Encl_R$ is the the part of code running on SGX enclave (the program for $Encl_R$ is shown in Algo.2) which securely uses the CSS to derive the key *K* and decrypt the data. $Encl_R$ has a key pair (pk_R, sk_R), which is used for message authentication.

Algorithm 1: Relay Program $Relay_R$ for data Requester

```
Initialize
    Send initialize to $\mathcal{F}_{sgx}[Encl_R, Relay_R]$
    On receive $(pk_R, \sigma_{att})$ from $\mathcal{F}_{sgx}[Encl_R, Relay_R]$:
        Publish $(pk_R, \sigma_{att})$
Relay
    Send CreateToken $(params)$ to $DApp_{IP}$
    On receive $(requestID, IT)_\alpha$ from $DApp_{IP}$:
        Send SignToken $(requestID, IT)_\alpha$ to $Encl_{IP}$
    On receive $(requestID, IT)_{\alpha\beta}$ from $Encl_R$:
        Send $(requestID, IT)_{\alpha\beta}$ to $DApp_{IP}$
        Send RegisterToken (IT) to $DApp_{ACM}$
    On receive (CSS, Ref($\mathcal{E}_K[D_i]$) from $DApp_{ACM}$:
        Fetch $\mathcal{E}_K[D_i]$ from Off-chain storage using
      Ref($\mathcal{E}_K[D_i]$)
        Retrieve public parameters from Ledger L
        Send Decrypt ($\mathcal{E}_K[D_i]$, CSS) and public
      parameters to $Encl_R$
```

Algorithm 2: Enclave Program $Encl_R$ for data Requester

```
Initialize
    On receive $initialize$ from $Relay_R$:
        return $pk_R, \sigma_{att}$
SignToken (IT)
    On receive $(requestID, IT)_\alpha$ from $Relay_R$:
        $\beta$:= $\sum.Sign(sk_R, (requestID, IT)_\alpha)$
    return $(requestID, IT)_{\alpha\beta}$
Decrypt ()
    On receive ($\mathcal{E}_K[D_i]$, CSS) and public parameters
      from $Relay_R$:
        Reconstruct the key K
        Decrypt $\mathcal{E}_K[D_i]$ using key K
```

c) Identity Provider IP: an application that combines a smart contract running on the blockchain with a program running on a secure SGX enclave off the chain. The purpose behind such a complex combination is ensuring the privacy and integrity of policy evaluation. On chain computations are trusted for integrity, yet computations in smart contracts are slow, expensive and publicly visible. While SGX computations are executed privately, correctly and efficiently. To guarantee the integrity of the data stored on chain and the code running on SGX enclave, *IP* is implemented using the following components:

- a smart contract C_{IP} that stores the identity tokens.
- an SGX-enabled application. The application consists of the following:
  - $Relay_{IP}$: provides network communication to $Encl_{IP}$ (the program for $Relay_{IP}$ is shown in Algo.3). Since sending information to the blockchain can only be done via transaction, $Relay_{IP}$ also offers an interface, known as ÐApp$_{IP}$, to the blockchain. ÐApp$_{IP}$ incorporates an Ethereum client, geth that can be configured to run as a JSON RPC server and a wallet $\mathcal{W}_{IP}$ that allows to execute the contract C_{IP} to store the identity tokens.
  - $Encl_{IP}$: is the code running on SGX enclave (the program for $Encl_{IP}$ is shown in Algo.4), which is responsible for issuing the identity tokens to *R*. $Encl_{IP}$ has a key pair (pk_{IP}, sk_{IP}), which is used for message authentication. $Encl_{IP}$ creates $\mathcal{W}_{IP}$ with public key pk_{IP} and this key is hardcooded into C_{IP}. This allows the blockchain to verify the signature on each massage with no additional cost.

Algorithm 3: Relay Program $Relay_{IP}$ for Identity Provider

```
Initialization
    Send initialize to $\mathcal{F}_{sgx}[Encl_{IP}, Relay_{IP}]$
    On receive $(pk_{IP}, \sigma_{att})$ from
      $\mathcal{F}_{sgx}[Encl_{IP}, Relay_{IP}]$:
        Publish $(pk_{IP}, \sigma_{att})$
Relay
    On receive $(params)$ from $Prog_R$:
        Send CreateToken $(params)$ to $Encl_{IP}$
    On receive $(requestID, IT)_\alpha$ from $Encl_{IP}$:
        Send SignToken($(requestID, IT)_\alpha$) to $Prog_R$
    On receive $(requestID, IT)_{\alpha\beta}$ from $Prog_R$:
        AuthSend $(requestID, IT)_{\alpha\beta}$ to $\mathcal{C}_{IP}$ as $\mathcal{W}_{IP}$
```

d) Access Control Manager ACM: Similar to *IP*, includes off-chain and on-chain components, as follows:

- a smart contract $\mathcal{C}_{ACM}$ which stores access control policies ACPs when received from *O*, all the public parameters required to reconstruct the key, and references to the encrypted data Ref($\mathcal{E}_K[D_i]$) (see Algo.7).
- an SGX-enabled application. The application consists of the $Relay_{ACM}$ and $Encl_{ACM}$. $Relay_{ACM}$ provides bidirectional network access for $Encl_{ACM}$ (the program for $Relay_{ACM}$ is shown in Algo.5). It also offers an interface, commonly refer to as ÐApp$_{ACM}$, to the blockchain which incorporates an Ethereum client, geth that can be configured to run as a JSON RPC server and a wallet

Algorithm 4: Enclave Program $Encl_{IP}$ for Identity Provider

```
Initialize
    On receive $(initialize)$ from $Relay_{IP}$:
        Set Counter:= 0
    return $pk_{IP}$, $\sigma_{att}$
CreateToken (params)
    On receive (params) from $Prog_R$:
        requestID := Counter; Counter:= Counter+1
        Parse $params$ as (att-name, att-value,
      requesterID)
        nym:= requesterID
        c:= Compute pedersen commitment
      $(att - value)$
        id-tag:= att-name
        IT:= (nym,id-tag,c)
        $\alpha$:= $\sum .Sign(sk_{IP}, (requestID, IT))$
    return $(requestID, IT)_\alpha$
```

$\mathcal{W}_{ACM}$ that allows to execute the contract C_{ACM} to match and verify the identity tokens. $Encl_{ACM}$ is the the part of code running on SGX enclave (the program for $Encl_{ACM}$ is shown in Algo.6). $Encl_{ACM}$ holds all CSSs for the encrypted data and maintains a table T of the delivered CSS after registration. $Encl_{ACM}$ has a key pair (pk_{ACM}, sk_{ACM}), which is used for message authentication. $Encl_{ACM}$ creates $\mathcal{W}_{ACM}$ with public key pk_{ACM} and this key is hardcooded into C_{ACM}.

e) Off-chain Storage: As the blockchain was not designed to store massive amounts of data, we adopt *Distributed Hash Tables (DHTs)* to store the encrypted data.

Algorithm 5: Relay Program $Relay_{ACM}$ for Access Control Manager

```
Initialize
    Send initialize to $\mathcal{F}_{sgx}[Encl_{ACM}, Relay_{ACM}]$
    On receive $(pk_O, \sigma_{att})$ from
      $\mathcal{F}_{sgx}[Encl_{ACM}, Relay_{ACM}]$:
        Publish $(pk_{ACM}, \sigma_{att})$
Relay
    On receive $ACP(D_i)$ from $Prog_O$:
        Send $ACP(D_i)$ to $Encl_{ACM}$
        AuthSend $ACP(D_i)$ to $\mathcal{C}_{ACM}$ via $\mathcal{W}_{ACM}$
    On receive $D_i$ from $Prog_O$:
        Send Encrypt $(D_i)$ to $DApp_{ACM}$
    On receive $\mathcal{E}_K[D_i]$, Ref($\mathcal{E}_K[D_i]$), and public
      parameters from $Encl_{ACM}$:
        AuthSend Ref($\mathcal{E}_K[D_i]$), public parameters to
      $\mathcal{C}_{ACM}$ via $\mathcal{W}_{ACM}$
        Send $\mathcal{E}_K[D_i]$ to Off-chain storage
    On receive DeliverCSS from $\mathcal{C}_{ACM}$
        Send DeliverCSS to $Encl_{ACM}$
```

Algorithm 6: Enclave Program $Encl_{ACM}$ for Access Control Manager

```
Initialize
    On receive $(initialize)$ from $Realy_{ACM}$:
        Create table T
    return $pk_{ACM}, \sigma_{att}$
Encrypt
    On receive $ACP(D_i)$ from $Relay_{ACM}$:
        Generate policy configuration $Pc$ for $D_i$
        Generate public parameters
        Select K $\in \mathcal{F}_q$
    Encrypt $D_i$ using K $\rightarrow \mathcal{E}_K[D_i]$
    Generate CSS      Create Access Matrix $A$
        Calculate Access Control Vector Y as $AY = 0$
        Set $X = (K, 0, 0, ....0) + Y$
        Choose $z_1, z_2, ..z_i$ randomly
    Send $\mathcal{E}_K[D_i]$ to Storage S
    return Ref($\mathcal{E}_K[D_i]$) and public parameters
DeliverCSS
    On receive DeliverCSS from $Relay_{ACM}$:
        Deliver CSS to $Prog_R$ via OCBE
        Update table T with $nym_i$ and delivered $CSS_{i,j}$
```

Algorithm 7: Blockchain Contract $\mathcal{C}_{ACM}$ for Access Control Manager

```
StorePermissions
    On receive $ACP(D_i)$,Ref($\mathcal{E}_K[D_i]$), and public
      parameters from $DApp_{ACM}$:
        Store $ACP(D_i)$, Ref($\mathcal{E}_K[D_i]$), and public
      parameters $\langle X, (z_1, ..., z_n) \rangle$ on Ledger L
RegisterToken
    On request $(RegistrationRequest)$ from Data
      Requester:
        Send Retrieve $IT$ to $\mathcal{C}_{IP}$
        Match (id-tag, name[$ACP(D_i)$])
        Verify $(pk_{IP}$,$\alpha)$
        Verify $(pk_R$,$\beta)$
        Send DeliverCSS to $DApp_{ACM}$
    return Ref($\mathcal{E}_K[D_i]$)
```

V. Related work

In this section, we review related work which leveraged blockchain technology to enforce access control and works on access control in the cloud.

a) Blockchain-based Access Control: The work of Zyskind et al. [12] offers a decentralized privacy solution for personal data that is collected and controlled by a third-party. This model depends on a blockchain, which acts as an access control manager to guarantee transparency over data and retains only pointers (hash) to the data and an off-blockchain Distributed Hashtable (DHT), which can be accessed through the blockchain and stores the encrypted data. When a user

signs up to use the service, a new compound identity (user, service) is created and shared. The compound identity consists of signing key-pairs for the user and service, and a symmetric key used to encrypt and decrypt the data. The blockchain verifies the signature for either the user or the service and checks whether the service is granted permission to access the data, then provides the hash to retrieve the data from the off-chain storage.
This work is extended by Enigma [13], a decentralized computational platform based on Multi-party Computation (MPC). Enigma deploys a public blockchain to ensure data correctness and an off-chain distributed hash-table (DHT) to assure data privacy. Unlike [12], only references to the data is stored in the DHT, while the actual data is divided over several nodes across the network. Thereby, nodes can compute functions together without leaking information to other nodes. The main difference between these approaches and ours is that the process of generating users' identities in our approach is privacy preserving, hence the identity of user requesting or granting access to data is not exposed. Moreover, our access control solution is fine-grained based on users's identity attributes.

b) Access Control in Multi-Clouds: Many solutions to access control in cloud environments have been proposed. The work of [2] proposed distributed access control architecture for cloud computing. This architecture relies on RBAC model and supports both federated and loosely coupled collaboration models. Suzic et al. [3] provides an XACML-based approach for federated cloud environments. [1] propose a proxy multi-cloud computing framework supporting dynamic and runtime collaborations between cloud-based services. The role of proxies is to act as mediators between applications not only to offer dynamic collaboration but also to provide resource sharing. In our solution, we propose an access control framework for cloud federations based on users' identity attributes, while using a cryptographic approach to enforce the access control policies.

VI. Discussion and Conclusion

In this paper, we presented a novel identity and access control framework for federated clouds. The framework satisfies several security properties:

- *User Privacy*. The framework preserves users privacy in that the federated organization owning data learns nothing about the users' identity attributes. The privacy is guaranteed by the use of the OCBE protocol to deliver the CSSs to the user requesting access to the data.
- *Fine-Grained Access Control*. CSSs necessary to reconstruct the decryption key can only be delivered to a user if he has identity attributes that satisfy the attribute conditions in the access control policy.
- *Integrity*. The framework preserves the integrity of the identity tokens issued to the users in the cloud federation and the access control policies protecting access to the federated data. The integrity is ensured by storing them on the blockchain via a smart contract. Similarly, the integrity of the cryptographic policy enforcement protocol is guaranteed by running the protocol within the Intel SGX's enclave.

We are planning to extend the cryptography-based access control enforcement protocol with an approach that considers the risk associated with a particular user accessing a particular piece of information owned by a particular federated organization within the cloud federation. The approach will rely upon the cryptocurrency system of the blockchain to price information access by users where the price will quantify the risk for the data provider to disclose the data with a particular users. The price will be used to reward federated organizations for sharing their personal data and to penalize those users from other organizations that misuse it.

We will also develop a proof-of-concept prototype using Ethereum blockchain and Intel SGX trusted execution environment and conduct an extensive evaluation of the protocol with respect to security, performance, and cost of execution.

Acknowledgment

This work has been supported by the EU H2020 Programme under the SUNFISH project, grant agreement N. 644666.

References

[1] M. Singhal, S. Chandrasekhar, T. Ge, R. Sandhu, R. Krishnan, G. J. Ahn, and E. Bertino, "Collaboration in multicloud computing environments: Framework and security issues," *Computer*, vol. 46, no. 2, pp. 76–84, 2013.

[2] A. Almutairi, M. Sarfraz, S. Basalamah, W. Aref, and A. Ghafoor, "A distributed access control architecture for cloud computing," *IEEE Softw.*, vol. 29, no. 2, pp. 36–44, Mar. 2012.

[3] B. Suzic, B. Prünster, D. Ziegler, A. Marsalek, and A. Reiter, "Balancing utility and security: Securing cloud federations of public entities," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 2016, pp. 943–961.

[4] N. Shang, M. Nabeel, F. Paci, and E. Bertino, "A privacy-preserving approach to policy-based content dissemination," in *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*. IEEE, 2010, pp. 944–955.

[5] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[6] I. CORP., "Intel(r) software guard extensions (intel(r) sgx) sdk," https://software.intel.com/en-us/sgx-sdk, 2015.

[7] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger, 2014," *Ethereum Project Yellow Paper*, 2014.

[8] IntelCorp., "Intel (r) software guard extensions enclave writer's guide," Tech. Rep., 2015.

[9] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Annual International Cryptology Conference*. Springer, 1991, pp. 129–140.

[10] J. Li and N. Li, "OACerts: Oblivious Attribute Certificates," *Dependable and Secure Computing, IEEE Transactions on*, vol. 3, no. 4, pp. 340–352, 2006.

[11] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," Cryptology ePrint Archive, Report 2016/168, 2016, http://eprint.iacr.org/2016/168.

[12] G. Zyskind, O. Nathan, and A. Pentland, "Decentralizing privacy: Using blockchain to protect personal data," in *Security and Privacy Workshops (SPW), 2015 IEEE*, May 2015, pp. 180–184.

[13] ——, "Enigma: Decentralized computation platform with guaranteed privacy," *CoRR*, vol. abs/1506.03471, 2015.