Trustless Blockchain-based Access Control in Dynamic Collaboration

Mouhamad Almakhour
Lebanese University, Faculty of Engineering-CRSI, Univ. Campus,
Hadath, Lebanon
mohammadmakhour@gmail.com

Layth Sliman
EFREI Engineering School-Paris
Villejuif, France
layth.sliman@efrei.fr

Abed Ellatif Samhat
Lebanese University, Faculty of Engineering-CRSI, Univ.Campus,
Hadath, Lebanon
samhat@ul.edu.lb

Walid Gaaloul
Telecom SudParis
Evry, France
walid.gaaloul@mines-telecom.fr

***Abstract*—AC (Access Control) is the process of ensuring that an authenticated user accesses only what he or she is authorized to do with respect to certain models and security policies. In business collaboration systems, services are designed to conduct actions requested by a customer, using service provider's infrastructure. In such context, the agreement on a conventional access management system is difficult because it will depend on different infrastructures and security policies implemented by each involved party. In this paper, we investigate the authorization process that manages permissions and rights of access to shared services in a federation of enterprises and we propose a solution based on the Ethereum Blockchain platform and the Attribute Based Access Control Model (ABAC) to define this authorization process.**

Keywords— Access Control, collaboration, Trust, BlockChain, ABAC

I. INTRODUCTION

Trust is a major factor in business collaboration. Managing trust involves complex and costly processes and brings its own risks to companies' information systems because relying on trust often involves the presence of a "trusted" third party and may compromise the security. However, the lack of trust may lead to major business opportunities lose, for instance in "on the fly" collaboration. In such scenarios companies collaborate by conducting actions on each other's' system in order to achieve the common objective. One of the most interesting technologies that may resolve this dilemma is Blockchain. The Blockchain provides trust without a trusted third party. Blockchain is a technology that provides a decentralized "database" on a network that is scalable, secure, tamper-proof, and accessible by each peer on the network. Thus, using Blockchain allows coping with both trust and data integrity issues. However, managing identity and access control in Blockchain based collaboration brings its own issues. Conventional methods to manage access control reach their limitations in such a context taking into consideration the constraints related to the Blockchain infrastructure on one side and the heterogeneous access control policies implemented by each involved party, on the other side.

In this paper, we consider the authorization process that manages permissions and rights of access to shared services in a federation of enterprises. To do so, we propose a solution based on the Ethereum Blockchain platform and the Attribute Based Access Control Model (ABAC) to define this authorization process via smart contracts. To validate our proposal, we implement cross-organizations authorization process as a smart contract, which is deployed on the Testnet BlockChain of Ethereum.

The rest of the paper is organized as follows: in section II, we present the related works, mainly the different identity management models as well as the existent model of access management. In section III, we discuss the proposed solution and the motivation of the adopted models. The implementation of the proposed solution is presented in section IV where different scenarios are given also. Section V concludes this paper.

II. RELATED WORKS

The concept of virtual team or virtual company was born in the 1990s to describe the new forms of management and digital exchanges between teams or between companies. The virtual organization is defined as "a temporary alliance of independent, connected, geographically dispersed organizations, institutions, industries, enterprises, etc., including a high level of trust, who collaborate and share their resources and skills in order to respond to customer requests "[2]. The difficulty lies in the differences in infrastructure and security policies implemented by each partner. Each of them must interconnect with others and share resources while maintaining the security of their own organization. All must provide a means of communication that ensures the integrity and confidentiality of the data. Similarly, they must have a way to verify the identity of the people and systems involved in the collaboration. Then Access management should guarantee to each and all people involved in the organization's projects, at all times, all the means necessary to carry out the mission entrusted to them, with respect to the permissions and security policies of the different involved organizations. That these means be at every moment limited to the just necessary. Below, we briefly describe the different identity management models as well as the different access management models.

A. *Identity Management Models*

1) Isolated identity

In this model, each service provider uses its own identity domain, that is, its own identity provider. A user must use a different ID and credential to authenticate with each domain. From the point of view of each identity provider, identity

management is simpler. In addition, in case of identity corruption in an identity domain, the other service providers are not impacted. This model also allows to define a different level of security for credentials (length of the password, number of credentials to be presented, etc. . .) Indeed, the latter must repeat the steps of authentication and identification with each of the identity domains attached to the service providers. Therefore, he must manage and remember as many identifiers and information useful for authentication as services he must access. This increases the risk of forgetting or losing this information, especially for services that are rarely accessed. In addition, this situation can be a source of weak adherence to the organization's security policy, which will be deemed too restrictive.

2) Federated identity

Identity federation is defined as a set of agreements, standards, and technologies that allow a group of service providers to recognize credentials from other service providers that belong to the federation [1][3]. The federation gives users the illusion of using only one unique identifier while continuing to present a different one to each service provider. In federated identity architecture, each service provider uses its own identity provider, but is able to accept identities from other providers. Access to a service provider can then be through an identity of an identity provider other than his own.

3) Centralized identity

In this model, only an identifier and a credential are used by the service providers. Three examples of implementation of this type of identity management [3]:

3.1 Common Identity

In this model, a single entity acts as the identity provider for all service providers. The mode of operation is halfway between the isolated identity model and the federated identity model. With this type of implementation the single identity provider is a central and sensitive point for all service providers. Indeed, in the event of a failure or modification at the level of the identity domain, all the dependent entities are impacted.

3.2 Meta identity

The implementation of a meta-identity domain allows service providers to share information about identities.

3.3 Single Sign-On (SSO)

The Single Sign-On approach is similar to an identity federation, but no identity match is required because there is only one identity provider. In this architecture, a user needs to authenticate only once (single sign-on) with a service provider. It is then authenticated de facto with other service providers. The Single Sign-On model can be associated with the identity federation model, allowing single-domain inter-domain authentication.

B. Access Management Models

For access management, the following model can be found in the literature and used in exciting systems.

1 Identity based access control (IBAC)

The IBAC model [1] is historically the first type of access control and still used by operating systems in the personal computer markets, with Microsoft Windows for example, and servers with UNIX and Linux systems. This model is based on a matrix composed of a finite set of entities, target resources and rules. It leads to the establishment of a comprehensive list of access rights i.e. Access Control List (ACL). This implies that any unauthorized access is prohibited. Thus, the rights are assigned directly to the user accounts (each right is assigned by name). One of the implementations of the IBAC model is Discretionary Access Control (DAC) [4], which is based on the concept of the owner of the resource. The latter has total control over the resource he or she has created and for which he or she is responsible. It determines which entity has permission to conduct what type of action on its resource. The complexity of ACLs increases according to the number of identities and the number of resources since it is necessary to list the authorizations for each Identity-Resource combination. In fact, when a new resource is available or when a new user arrives, the list of authorizations must be updated.

2 Mandatory Access Control (MAC)

In the case where the owner of an information system should not be responsible for managing the underlying security, MAC-type models can limit access based on the sensitivity of the data. For this purpose, the target entities are hierarchized in different levels of security called labels. Bell and Level [1] developed a model where a minimum level of security is required to access the resource. This level defines the level of privilege of the user. Similarly, a security level is assigned to the resource. This level determines the level of classification of the resource. The user then has access to the resource only if his level of authorization is greater than or equal to the classification level of the resource. In addition, for an application, an execution level, called the current level, is also defined. The current level of an application is always less than or equal to the privilege level of the user responsible for running the application. The "no read up" condition implies that an application can read access to information only if the current level of the application is greater than or equal to the classification level of the resource that manages the data. Similarly, the "no write down" condition assumes that an application can transmit information to a resource only if its current level is less than or equal to the classification level of the target resource. This access control model is also called Rule Based Access Control (RuBAC) because access is governed by rules.

3 Role Based Access Control (RBAC)

Unlike the IBAC model where entitlements are granted directly to the user, in the RBAC model developed by the National Institute of Standards and Technology (NIST), permissions are assigned to roles. The management of authorizations is then simplified. In addition, NIST proposes several variants of the RBAC [5], for instance, using the notion of inheritance between roles. In RBAC, the difficulty lies in the completeness and the granularity of the roles. In fact, too broad roles admit too many rights and too small roles will increase the difficulty of administration.

4 Attribute Based Access Control (ABAC)

The model ABAC, defined in [6], proposes to have the access rights according to the characteristics of the identities. Like the IBAC model, the access rights policy can be materialized by a matrix, but not based on identities. As a result, access rights to a resource or service are defined for

one or more attributes that identities may have. This paradigm therefore offers more flexibility. In addition, by defining an attribute closer to the notion of role, ABAC makes it possible to simulate the behavior of an RBAC model, but generalizes it by not limiting the access rights to the only users present in the organization. It allows in particular to determine access rights with a finer granularity. In addition, defining a role as a set of attributes makes it easier to handle conflicts. The management of access rights is facilitated because it does not require additional information. However, access security is then based on the values assigned to the attributes and thus on the quality and integrity of the information related to the identities.

5 Organization Based Access Control (OrBAC)

With the OrBAC model [4], the organization is perceived from an abstract perspective as a set of activities that roles have permission, prohibition or obligation to achieve through views. Just like in RBAC, it is possible to use the notion of inheritance for roles. Concretely, authorizations are granted to subjects for actions on objects through three-dimensional matrices.

III. PROPOSED SOLUTION

After reviewing the main identity and access management model, we focus on the requirements of our authorization process that manages permissions and rights of access to shared services in a federation of enterprises. The requirements include:

- Providing trust between different members of the federation who do not know each other.
- Denying any illegal access from outside the Federation, unauthorized access or no access agreement.
- Decentralization of authorization process that prevents the risk of loss of access control that allows services to be stopped.

Today, one of the most important technologies used in the security field is the Blockchain. The Blockchain ensures trust without a trusted third party. Confidence is obtained by:

• Validation of transactions added to blocks

• For a transaction added to BlockChain, you cannot edit or delete.

Based on such technology, smart contracts are agreements between parties that are written in executable code on the Blockchain instead of being written in natural language,. The execution is then managed automatically by the BlockChain according to the conditions described in the contract.

We propose a solution based on the smart management system contracts. The idea behind this solution is to define an authorization process that manages permissions and access rights to shared services in a business federation. This process satisfies the requirements by providing trust between different members using Blockchain. It also denies any illegal access from outside the federation. With Blockchain, the decentralization of authorization process can be also done. All steps are done automatically from Smart Contracts, so no needs for a manager on the process.

Our solution is based on ABAC access management model explained earlier to define an authorization process that manages the permissions and rights of access to shared services in a business federation and the implementation is based on the Ethereum Blockchain platform.

A. Ethereum Blockchain technologie

Blockchain technology allows a distributed computing architecture where the transactions are publicly announced and the participants agree on a single history of these transactions (or some kind of ledger) [10] [12]. The transactions are grouped into blocks, given timestamps, and then published. The hash of each block includes the hash of the previous block to form a chain, making published blocks difficult to alter. As Bitcoin began attracting attention, developers have taken advantage of the features of BlockChain technology as an infrastructure to create their own platforms (aside from the main use of Blockchain in facilitating the transfer of digital currency in Bitcoin). On the one hand, some platforms use the Bitcoin network as infrastructure for notarization or proof of existence of digital files, crowd funding, dispute mediation, and spam control, among others. On the other hand, some platforms have emerged and took the form of “alt coins", which are alternative Blockchain-based crypto currencies that aim to improve the capabilities of Bitcoin (or lack thereof) by implementing their own features and capabilities. The “improvements" can come in the form of a different proof-of-work algorithm (to shorten the verification time of transactions) or different hashing algorithm. There are a lot of alt coins, but the biggest ones that have attracted a following and attention is Ethereum [7]. In our work and following the schema illustrated in figure 1, we use Ethereum as it is a public or unlicensed block chain. Anyone with a computer and open source software can participate by listening, trading, or exploring data, which means that all data included in a transaction or smart contract is public. We need a database to store users, attributes, permissions and logs. Each company shares its service in the federation for the benefit of users. The transactions can be public or private according to the agreement between the members of the federation and the governance should not be controlled by a member of the federation.

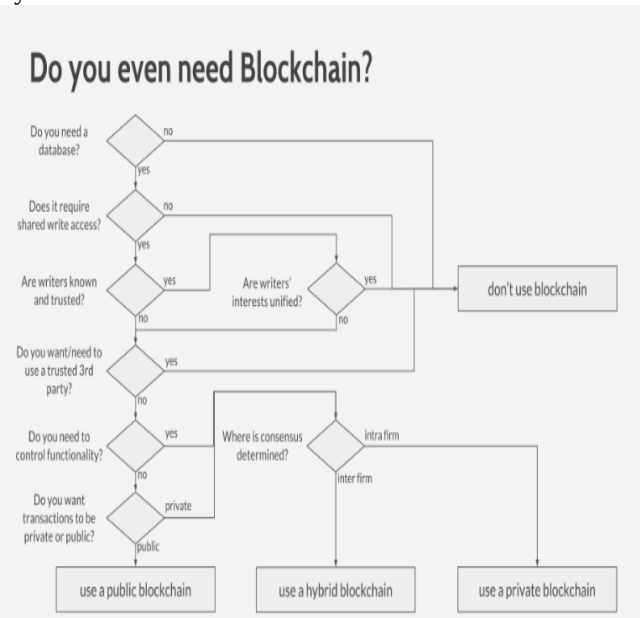

Figure 1: Types of Blockchain [10]

Regarding other well-known blockchain technology, we note that Bitcoin [2] is a crypto currency and uses the technique of the blockchain as a payment network. In addition, Bitcoin does not support the necessary smart contracts required in our process. As for Hyperledger Fabric [11], it is a private or licensed blockchain protocol designed for B2B business applications. Most managed blockchain protocols allow authentication, authorization and authorization of actions. This makes Hyperledger Fabric more suitable for companies in various industries (such as supply chain, healthcare, and banks) that want to use blockchain technology for internal or collaborative purposes without operating on public networks. As our need requires a public blockchain and the type of the Hyperledger fabric is private blockchain and which is for internal application without operating on public networks, we decided to use Ethereum blockchain technology. In the current implementation of Ethereum, consensus is reached by mining based on proof of work.

B. ABAC Access Management

In this part we discuss and compare the access management models mentioned in related work section.
In the IBAC model, access controls are based on an exhaustive list of entitlements for each authorized account. The complexity of ACLs increases according to the number of identities and the number of resources since it is necessary to exhaustively list the authorizations for each combination. The federation contains a large number of users so this model is not compatible with the federation of companies.

The RBAC model allows to reduce the list size of the authorizations. Access controls are performed on the roles assigned to the accounts. Application roles are granted based on the business profile. The difficulty lies in the completeness and the granularity of the roles. In fact, too broad roles admit too many rights and too small roles will increase the difficulty of administration. In the federation each user has a different right or permission so we did not use this template because of the granularity of the user roles.

In the OrBAC model, permissions or prohibitions are based on contextual expressions defined according to the organizational structure of the institution. In our work, the federation defines the users that they have the right of access and we cannot define them in advance.

The MAC model relies on flow control. Constraints are defined on data and resources. The level of entitlement of an account then determines whether or not he has the right to access the information. In our process, we do not have constraints to define because the permission is provided according to the user.

In the ABAC model, access controls check the presence and value of application attributes defined at the account level. It is then possible to simulate the RBAC behavior by mapping the attributes on the definition of the roles. This model is the appropriate one for our process because according to the attributes of each user, access rights are given.

IV. ETHEREUM BASED IMPLEMENTATION

A. ETHEREUM

In 2013, Ethereum was proposed by Vitalik Buterin to create a BlockChain-based distributed computing platform with the capability of building and running decentralized applications or smart contracts [7] [8]. As a BlockChain-based cryptocurrencies, it offers the same features as Bitcoin of easy mobile payments, reliability, full control of one's own money, high availability, fast international payments, zero or low fees, protected identity, and privacy. Ethereum, however, offers more than enabling online transfer of digital money; it enables its users to build and deploy smart contracts. Ethereum is composed of most of the protocols that other cryptocurrencies, like Bitcoin, also use. For example, Ethereum also includes a peer-to peer protocol for the Blockchain. And the Blockchain is managed and kept secure by nodes in the network. In addition to these protocols, the main modification and innovation of Ethereum is being a programmable Blockchain, i.e., it allows its users to create, deploy, and run decentralized applications on the Blockchain.

B. ETHEREUM VIRTUAL MACHINE

At the center of Ethereum is the Ethereum Virtual Machine (EVM), which can execute codes of arbitrary algorithmic complexity. Therefore, applications that are created using known programming languages, such as JavaScript, can be run on the EVM. To facilitate the execution of codes in the blockchain and to maintain consensus, the nodes of the network run the EVM and execute the same instructions. Computations in the EVM are payed in ether (ETH), which is the currency used in Ethereum.

C. ETHEREUM ACCOUNTS

Ethereum’s basic unit is the account. Ethereum uses two types of accounts: Externally Owned Account (EOA) and Contract Account. An EOA is controlled by a corresponding private key, has an ether balance, can send transactions (transfer ether to another account or trigger a contract code), and does not have an associated code. Similar to a Bitcoin address, an EOA is in the form of random numbers and letters, and therefore looks anonymous and can be shared publicly. A contract account (or simply called contract) has an ether balance and has an associated code. All actions in the blockchain are set in motion by the transactions created by EOAs. This means that the code in a contract is executed when it receives a transaction from an EOA, where the input parameters for the code execution are included in the transaction. Therefore, contracts can be considered as autonomous agents inside the EVM that execute a specific piece of code when poked by a transaction. Code execution in a contract can also be triggered by messages from other contracts (see the next subsection for detailed explanation on transactions and messages). In contrast to Bitcoins script, a contract performs Turing-complete computations and is typically written using some high-level language, such as Solidity, Serpent, and Lisp like Language. A contract's behavior is fully dependent on its code and on the transactions sent to it and therefore offers the possibility for creating decentralized and trusted systems.

D. TRANSACTIONS AND MESSAGES

An Ethereum transaction is a signed data package that stores a message from an EOA to another account on the blockchain. A transaction contains the Ethereum address of the recipient, a signature that identifies the sender, the amount of ether being transferred, an optional data field, and startGas and gasPrice values. The startGas limits the maximum amount of gas the code execution triggered by the message can incur. And the gasPrice is the amount in ether to be paid for one unit of gas consumed (see the next subsection for detailed explanation on gas). When users send transactions, they pay a small transaction fee in ether to the network. This fee protects the blockchain from malicious computational tasks, such as distributed denial-of-service (DDoS) attacks and infinite loops [9] [12]. A message is a virtual object that can only be sent by a contract to another contract. A message contains the identity of the sender, the identity of the recipient, the amount of ether being transferred, input data, and a startGas value. Similar to a transaction, a message leads to the recipient account running its code. Therefore, contracts can have relationships with other contracts in exactly the same way an EOA can.

E. ETHER AND GAS

Ether (ETH) is Ethereum's native value token and is the currency of the network. The sender of a transaction needs to pay for the code it wants to execute, including computation and data storage. When a code in a contract is executed as a result of being triggered by a message or transaction, every node in the network executes this code. The cost of this execution is expressed in gas. Gas is purchased for ether from the miners that execute the code (miners are the nodes in the Ethereum network that receive, propagate, verify, and execute transactions). Gas and ether are decoupled because gas is supposed to be constant cost of network utilization, whereas ether, and currencies in general, is volatile. Therefore, even if the price of ether increases, the gas price in terms of ether of executing a function in a contract remains constant. Every computational step that is executed in a contract or transaction requires gas, and each transaction includes a gas limit and a fee that it is willing to pay per gas. The price of the gas is decided by the miners, and miners have the choice of including the transaction and collecting the fee or not (similar to the transaction fee in Bitcoin, wherein miners can decide to get the fee or not). Ethereum clients automatically purchase gas for the ether specified by the sender as maximum expenditure for a transaction, and the excess gas not used by the transaction execution is returned to the sender in ether. Therefore, overspending on the gas is not an issue because the user will only be charged for the gas consumed by a transaction. Readers can refer to [7] [12] to read more about Gas.

F. MINING AND PROOF-OF-WORK

Transactions are grouped together in blocks, which are then added to the blockchain through the process called mining. The mining process uses a proof-of-work (PoW) system wherein miners all around the world use special software to solve mathematical problems. Blocks are connected and linked together to form a blockchain, where a new block is added to the block that came before it. Every block contains the hash of the previous block, and thus, creating a chain that connects the first block (genesis block) to the current block. The miner who solves a block is rewarded with ether (currently at 5 ETH). The cost of the gas used in the transactions that are mined, and an extra reward of 1/32 per uncle. Uncles are stale blocks with parents that are ancestors of the including block. Valid uncles are rewarded to increase the security of the network by neutralizing the effect of network lag on the dispersion of mining rewards. The PoW algorithm used in Ethereum is called Ethash (a modified version of the Dagger-Hashimoto algorithm) and requires a brute force solution i.e. miners scan and test for a nonce to find a solution that is below a certain difficulty threshold. The difficulty is adjusted accordingly so that it takes approximately 15 seconds to find a valid nonce. The Ethash PoW is a memory hard computational problem, that is, it is application-specific integrated circuit (ASIC) resistant and allows a more decentralized distribution of security (as compared to specialized hardware used by many mining pools that dominate the mining in Bitcoin). The security of the blockchain relies on this PoW system, which inherently means that a block cannot be modified without redoing the work spent on it, including the work spent on blocks chained after it. Therefore, an attacker will be outpaced by honest miners as long as majority of the overall computation power participating in the Ethereum network are controlled by honest miners. In this case, a block recorded in the blockchain is almost impossible to modify.

G. Scenarios

At the beginning, a necessary agreement by the entire federation member defines the rules for access to services and the requirement (attributes) by a user to obtain service access, this agreement is called Primary contract which is signed by all businesses.

First the attributes indicated in the agreement (main contract) are pre-loaded in the smart contracts, then each director (Local manager) records the users and their attributes. An access request is in the form of a Token request is created automatically after a comparison between the users attributes to the pre-loaded one. Token represents the permission and the right of access. The token contains the user's ID and address, date to expire, Token status, and delegation option fields. Depending on the attributes, the Token is created with different values. Different scenarios are given below.

- *Scenario 1: Request a Service Access Token*

A user requests a Token to access a service. The Smart contract verifies the identity of the user from his ID and his address. The smart contract examines the attributes to give a request for access to the specific service from the pre-loaded conditions. If the attributes satisfy the conditions, a Token is created and returned to the user, otherwise the request is rejected. After having a Token, each access to the services is checked (verification of the expiration date and status of the Token).

- *Scenario 2: Revoke an Access or User*

This is used to revoke an access of a user left the company or who changed status that does not allow preserving the old

permissions. The local manager sends a revocation request containing "ID user" and Token to Smart contracts which in turn add the user and their Tokens to the revocation list.

- *Scenario 3: Request service access*

To obtain a service provided in the Federation, an access request will be made by the user by presenting their "TOKEN ID" in the smart contract Verification which in turn verifies this Token (@user, ID user, expiry date and status of the Token) and returns access to the user. If the answer is negative the request is rejected.

- *Scenario 4: Service Delegation*

If a user gives his Token to another user, when a service access request is made the Smart Contracts Verification Service takes the Token and the delegated user address and checks whether it is Token with delegation. If it is the case, the access to service is given.

H. Design and implementation

According to the scenarios presented above, we propose the following design:

- **Component Diagram**

In our solution for this process, 5 components are proposed. Figure 2 illustrates the component diagram and relationships between components:

• Service/Local User Manager: Its function is to add and identify all users of the company (ID, @, Attributes) and to manage add-on and revocations.

• Owner Manager: Several people are authorized by the company to define the attributes for the creation of a Token and for the revocation conditions of the Token.

• Smart contracts Add User: This Smart contract is pre-loaded by the data of all Users (ID, Attributes, address User). This Contract represents the core of the project because it includes the main database which allows any user to obtain access to the services.

• Smart contracts Token: The intelligent contract allows the creation of Tokens by taking as input the Users ID. This Smart contract is pre-loaded with attributes (defined by Owner service).

• Smart contracts Authentication/Service: it takes as input Token ID, then it checks if this Token exists in the BlockChain and if it is valid. If yes it provides the service otherwise the request is rejected.

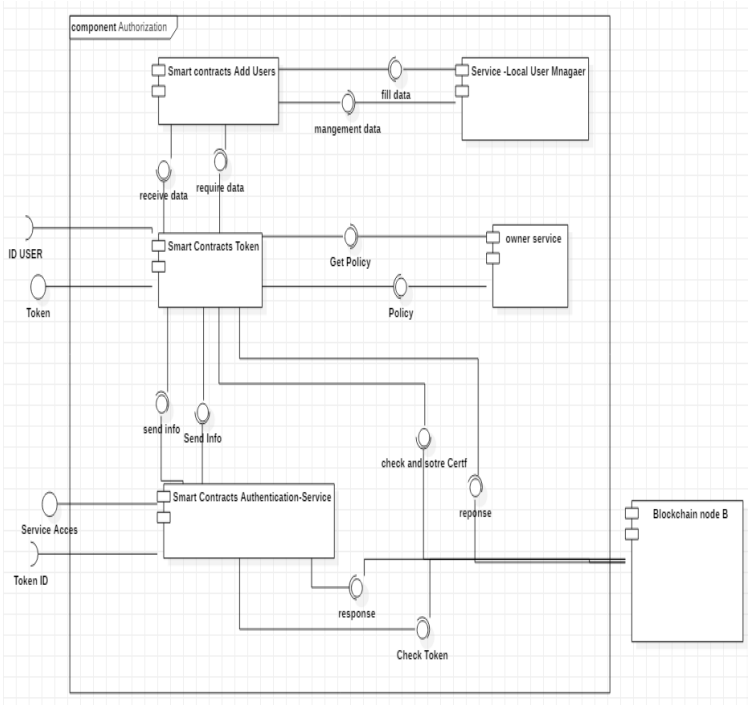

Figure 2: Component diagram

- **Sequence diagram**

We show in Figure 3, the sequence diagram to present how the implementation functions and the different phases of our process are working. The process consists of 5 phase:

1. Loading policies phase to set the attributes in order to have an access service.
2. The second phase is to create User in order to add users of companies.
3. The third phase is create Token, which is responsible for giving users access to benefit from the services in the federation.
4. Access Service phase is the fourth to verify the token before giving the access.
5. Finally the revocations phase in order to revoke a user or expired token.

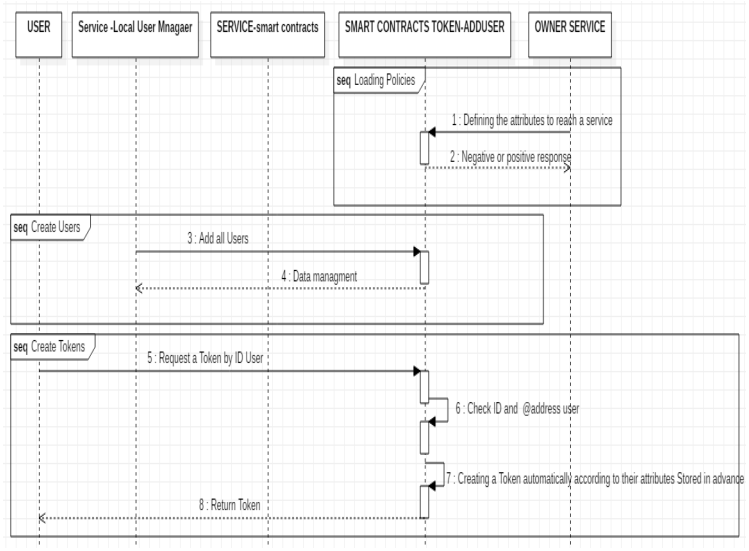

Figure 3: Sequence Diagram (1/2)

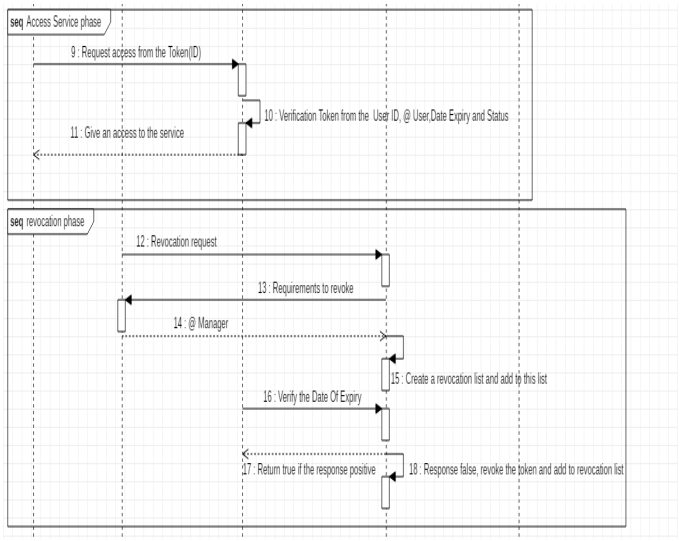

Figure 4: Sequence Diagram (2/2)

V. CONCLUSION

In this paper, we presented a new smart contract-based authorization management system. It manages permissions and rights of access to shared services in a federation of enterprises. We proposed a blockchain-based management system by adopting the ABAC access model. The blockchain is a machine used to ensure that the execution of transactions is carried out in strict accordance with previously established rules. So it's a kind of digital trust. According to the effectiveness of the platforms, we choose Ethereum as a platform for our solution. In the ABAC model, access control checks the presence and value of application attributes defined at the account level. In this solution, the confidence is obtained by:

• Validation of transactions before being added to blocks

• A transaction added to the Blockchain cannot be modified or deleted.

As future work, we will work to raise the level of security. More precisely, define other types of tokens and will come out of the general Token for all the service in a federation of companies in the specific Token at the same service level and develop this process to become more efficient.

REFERENCES

[1] Guillaume HARRY, “IAM - Gestion des identités et des accès : concepts et états de l’art ,” CC-BY-NC-ND, 2017.www.crns.fr

[2] Satoshi Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” October 2008 , www.bitcoin.org/bitcoin.pdf

[3] A. Jøsang, J. Fabre, B. Hay, J. Dalziel, S. Pope. “Trust Requirements in Identity Management. Australasian Information Security Workshop,” 2005 volume 44, pages 99-108, 2005.

[4] F. Cuppens, N. Cuppens-Boulahia. “ Les modèles de sécurité. Dans Sécurité des systèmes d'information,” (Traité IC2, série Réseaux et télécoms). Hermès, pages 13-48, 2006.

[5] Guillaume HARRY, “Failles de sécurité des applications Web,” contenu sous licence Creative Commons CC-BY-NC-ND CNRS,38pages,2012.www.resu.dsi.cnrs.fr/IMG/pdf/failles_de_securite_v1-3.pdf

[6] L. Wang, D. Wijesekera, S. Jajodia. “A logic-based framework for attribute based access control,” *ACM* workshop on Formal methods in security engineering, pages 45-55, 2004.

[7] Ethereum. “Blockchain App Platform,” Accessed: Nov. 28, 2017. www.ethereum.org/

[8] G. Wood. “Ethereum: A Secure Decentralised Generalised TransactionLedger,” Yellow Paper. Accessed: Nov. 28, 2015.www.ethereum.github.io/yellowpaper/paper.pdf

[9] Jason Paul Cruz, “ RBAC-SC: Role-Based Access Control Using Smart Contract,” Graduate School of Information Science and Technology, Osaka University, Suita 565-0871. Japan ,March 7, 2018

[10] V. Gatteschi, F. Lamberti, C. Demartini, C. Pranteda and V. Santamaría, "To Blockchain or Not to Blockchain: That Is the Question,"in IT Professional, vol. 20, no 2, pp.62-74, Mar./Apr. 2018.

[11] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos, Christidis Angelo,“Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains,” arXiv:1801.10228v2 [cs.DC] ,IBM, 17 Apr 2018.

[12] Imran Bashir, “Mastering Blockchain Distributed ledgers decentralization and smart contracts explained“ First edition book 2017, www.packtpub.com.

[13] Damiano Di Francesco Maesa, Paolo Mori, and Laura Ricci, ”Blockchain Based Access Control”, University of Pisa, Department of Computer Science, Pisa, Italy.

[14] Jan Mendling et al., Blockchains for Business Process Management-Challenges and Opportunities, in ACM Transactions on Management Information Systems, Volume 9 Issue 1, February 2018