A blockchain-based access control scheme

Maryline Laurent, Nesrine Kaaniche, Christian Le, Mathieu Vander Plaetse

An Access Control Scheme based on Blockchain Technology

M. Laurent*, N. Kaaniche*, C. Le, M. Vander Plaetse
SAMOVAR, CNRS, Telecom SudParis, University Paris-Saclay,
** Member of the Chair Values and Policies of Personal Information, Paris, France*
first_name.last_name@telecom-sudparis.eu

Keywords: access control, data secrecy, blockchain, smart contract, Ethereum

Abstract: Recent years have witnessed the trend of increasingly relying on remote and distributed infrastructures. This increased the number of reported incidents of security and privacy breaches, mainly due to the loss of data control. Towards these challenges, we propose a new access control scheme based on emerging blockchain infrastructures. Our approach relies on the use of *smart auditable contracts* deployed in blockchain infrastructures. Thus, it offers transparent and controlled access to outsourced data, such that malicious entities cannot process data without data owners' authorization. In fact, the effectiveness of the authentication relies on the blockchain intrinsic properties. Moreover, an implementation of the proposed solution based on Ethereum Blockchain is presented to show the applicability of our scheme in real-world scenarios.

1 INTRODUCTION

Data security and privacy are major challenges in the adoption of remote data storage applications, mainly due to the loss of data control (Kaaniche and Laurent, 2017b), and thus the possibility for third-party data storage providers to get privacy-sensitive information about users and potentially leak confidential information about the content. Relying on cryptographic mechanisms at the client side is a good alternative to mitigate data secrecy concerns. However, the use of conventional encryption approaches is not sufficient to support the enforcement of fine-grained access control policies and flexible data sharing among dynamic groups of users. Thus, the challenge is to define a comprehensive access control mechanism for outsourced data while both ensuring data confidentiality and protecting users' privacy.

This need for designing security mechanisms to ensure privacy-preserving access control schemes to outsourced data files is emphasized by the recent adoption of the new General Data Protection Regulation (GDPR), in 2016 (Regulation(EU), 2016) that will be enforced in every European member state in May 2018. The new key concepts behind the GDPR include the user consent that must be collected by a service provider before processing users' data, accountability which makes mandatory that any service providers prove data processing is compliant to former user's consent.

Recently, various accountable technical systems appeared, namely *Bitcoin* [1] which enables users to transfer cryptocurrencies (i.e. bitcoins) securely with no need for a centralized authority, using a publicly verifiable open ledger, referred to as *blockchain*. Consequently, blockchain technologies are widely adopted for data accounting and auditing features, thanks to their main intrinsic properties, namely, tamper-proof infrastructure and availability.

In this paper, we propose a new blockchain-based access control scheme, in a data-owner centric manner. That is, access privileges are defined by the data owner, via access control lists associated with auditable smart contracts, and managed by the blockchain infrastructure that supports the authentication of requesting users.

Paper Organization –Section 2 introduces the necessary background about the blockchain technology, reviews blockchain related works for data protection and highlights the security and privacy requirements for designing a secure access control scheme. Section 3 presents our proposed blockchain-based access control mechanisms and details the different procedures. Section 4 provides a security and privacy discussion of the proposed access control scheme. Section 5 discusses the implementation of the defined algorithms, based on Ethereum blockchain environment. Section 6 concludes the paper.

[1] https://bitcoin.org/en/

2 BACKGROUND AND DESIGN REQUIREMENTS

In this section, we first introduce the blockchain technology (subsection 2.1). Then, we detail related work for data protection (subsection 2.2) and highlight the design and security requirements (subsection 2.3).

2.1 Blockchain Technology

Bitcoin appeared as an innovative technology enabling users to directly transfer cryptocurrencies in between with no intermediaries. It is considered as the first decentralized cryptocurrency transfer system. It relies on cryptographic proofs of work, digital signatures, and peer-to-peer networking to provide a distributed ledger containing transactions, and referred to as a *blockchain* (Crosby et al., 2016), (Swan, 2015). Two approaches, known as permissionless blockchains, have emerged to implement decentralized services and applications. The first approach relies on the existing Bitcoin blockchain and builds a new framework on top of it. The main advantage of this approach is that the Bitcoin blockchain already exists and is adopted by many users, which makes it more secure, transparent and resilient. The disadvantage is that blocks are mined every 10 minutes, and the Bitcoin scripting language is not Turing-complete (Swan, 2015). The second approach is to build an alternative blockchain with all the desired features, which promises full decentralization, such as Ethereum[2]. Additionally to functions already supported by other public blockchain platforms such as bitcoin, e.g. mining of the digital currencies and transaction management, Ethereum also provides a contract functionality known as *smart contract*.
Transactions submitted to the Ethereum environment are organized into blocks and chained to each other based on a cryptographic hash function, initially relying on a pre-computed genesis block. Once a block is added to the blockchain, it cannot be modified or removed for two reasons: first, a block modification would lead to wrong verification of the chain of hash values, and second, the block modification would require intensive efforts to change every replicate of the blockchain supposed to be hosted on a large number of independent nodes. The verification and addition of new blocks to the blockchain is based on the *mining* process, which relies on the *proof of work* feature. Indeed, *miners* have to solve a cryptographic challenge and winners are rewarded. The main idea behind the cryptographic challenge is the regulation of the new block creation operation.

[2]https://www.ethereum.org/

2.2 Blockchain Related Work for Data Protection

The nature of the blockchain is particularly suitable for data accounting and auditing features. It has attracted interest of the research community due to its shared and fault-tolerance database. Indeed, several constructions have been introduced to ensure provenance tracking (Fu et al., 2017), (Ouaddah et al., 2016), (Zyskind et al., 2015), (Kaaniche and Laurent, 2017a).

In (Zyskind et al., 2015), Zyskind et al. presented a personal data management system that combines blockchain, considered as an access control moderator, and off-blockchain storage solution. Designed as unique owners of their personal data, clients are aware of data collected about them by service providers and how they are used. However, the (Zyskind et al., 2015) proposal permits to only define simple permit/deny access policies through a white/blacklisting. Afterwards, Ouaddah et al. proposed, in (Ouaddah et al., 2016), a blockchain based access control framework for IoT applications, referred to as FairAccess. Their proposal relies on the blockchain-based bitcoin technology as an access moderator that permits to distribute authorization tokens, where each authorization token represents the data owner signature of the granted access right. In (Fu et al., 2017), Anmin et al. introduced a blockchain-based auditing system for shared data in cloud applications. In order to mitigate the power abuse of single tracing authorities, (Fu et al., 2017) presents a threshold approach, where at least t entities have to collaborate to recover the identity of a malicious user, thus ensuring the non-frameability of users. Based on a blockchain architecture, the proposed construction enables group users to trace data changes and recover latest correct data blocks when current data are damaged.

Recently, Neisse et al. discussed design requirements of blockchain-based solutions for data provenance tracking (Neisse et al., 2017), namely client-centric, server-centric and data-centric approaches. The authors also presented an evaluation of their implementation results, in order to give a comprehensive overview of different defined approaches. Later, in (Kaaniche and Laurent, 2017a), Kaaniche and Laurent presented a blockchain-based platform for data usage auditing while preserving personal data secrecy and ensuring data availability, relying on the use of the hierarchical ID-based cryptographic technique.

2.3 Security and Privacy Requirements

Our blockchain-based access control solution has to consider a set of security and functional properties, defined as follows:

- Authenticated access control — the proposed scheme has to ensure an efficient access control to outsourced data, where requesting entities are authenticated.
- Management efficiency — the proposed scheme should offer efficient management processes.
- Privacy through pseudonymity and unlinkability measures — entities' privacy is preserved thanks to the pseudonymity supported by the blockchain and the inability to directly link some data to an entity, or an access session to a requesting user identity. Privacy is strengthened with untraceability in case of one-time blockchain accounts, with one account used per system interaction.
- Auditability — each data owner should have a transparent view over how data are collected, accessed and processed.

3 A NEW BLOCKCHAIN-BASED ACCESS CONTROL SCHEME

In this section, we first give an overview of our proposed blockchain-based access control scheme in subsection 3.1. Then, we detail the different procedures in subsection 3.2.

3.1 Overview

Our proposed access control scheme relies on four different entities defined as follows:

- *data storage provider (DSP)* — it has significant resources to govern distributed remote servers and to host application services. These services can be used by the data owner to manage his data stored in the remote servers. Note that the DSP is not an active client of the blockchain, thus having only read access and not write access to the blockchain.
- *data owner (DO)* — a data owner makes use of data storage provider's resources to store and share data with multiple entities. He is responsible for defining a whitelist of authorized entities with respected access rights for a specific file stored on DSP. The whitelist is stored in a per file smart contract (C) into the blockchain.
- *data retriever (DR)* — data retrievers are able to access the content stored in remote servers, depending on their access rights which are authorizations granted by the DO. As the DR is known as a blockchain client, the blockchain can participate in the authentication of DR with DSPs before granting access to outsourced data.
- *blockchain infrastructure (BC)* — the blockchain is considered as an access control mediator, as it permits to authenticate DRs, to keep traces of each DR access to data and to preserve the access rights history of each DO.

The notations used in this paper are listed in Table 1.

Table 1: Our notations

Notation	Description
BC	blockchain infrastructure
DSP	data storage provider
DO	data owner
DR	data retriever
C	smart contract

Our scheme aims at providing the DO with the capacity of defining the access rights for each of his data resource (file, directory, image...), and of dynamically deleting these rights when needed. Rights are expressed per DR and registered in a smart contract as a whitelist of authorized DRs with a detailed specific access control list. The interest is therefore multifold. First, no one can alter the list of authorized entities to access certain resources, as all blockchain-specific operations are considered as secure and non-corruptible, thus ensuring non-tamper proofs of data access activities. Second, our scheme relies on a data owner-centric model, as the data owner creates a contract for each outsourced data resource, including the access control list w.r.t. data usage. Third, the entire identification system and the robustness of the authentication process rely on BC properties. Indeed, DRs and DOs are known through their BC identifiers, which make them be uniquely identified. Fourth, any DR access is registered in BC, thus leading to later possible auditing activities to take place over the access control system. As such, the use of BC permits to publish a whitelist which remains under the control of the DO, to support efficient identification and authentication processes of DOs and DRs, and to enable auditing activities thanks to useful registered traces (whitelists, access requests).
In a nutshell, DO is responsible for creating a smart contract in the BC, adding or removing an address from the whitelist. A DR trying to access the resources outsourced on a DSP has to go through the

corresponding smart contract. He is authorized to access a resource based on the authentication protocol played between the DSP and DR with BC support, and the whitelist registered in the BC. Hence, interactions of entities with BC support the following rules:

- The DO is the only entity that might modify the whitelist, as he is the owner of the outsourced data. He must therefore be able to add or remove an authorized address, referring to a specific authorized DR, from this list. Note that the contract has to include the address of the DO, its creator, for assigning modification of right permissions.
- The DR is authorized to issue a transaction w.r.t. the contract to request access to resources. If the transaction is accepted by the BC, it means both that the DR is authenticated, and it is authorized to access the resources. Thus, a successful authentication leads to the smart contract registering into the blockchain the address of DR as a permitted entity. Note that our scheme assumes that the comparison process is performed by the hosting server. As a result, the contract must be passive when it receives users' transactions.
- Any entity including DSP, can read the content of the whitelist, for performing the addresses comparison, for every DR's access request.

Note that although the contract script becomes unchangeable after its deployment on BC, its state may vary. Indeed, variables are defined (i.e; DR' addresses in our case) so that they can be modified by the DO w.r.t. subsequent solicitations. That is, the whitelist can be modified, but the contract script cannot.

3.2 Procedures

3.2.1 Whitelist Creation Process

As stated above, each smart contract, created by a data owner, permits to list the addresses of authorized DRs to access a given data file, outsourced on a remote server. Hence, the question is about the definition of the relation between the smart contract (C) and the outsourced data file. For this purpose, we focus on the process of the smart contract creation by DO with the DSP. Note that our blockchain-based access control mechanism requires the deployment of a client interface, such that each DO can easily perform the whitelist creation process. The different steps, depicted by Figure 1, are as follows:

1. DO authenticates with DSP and shares the name of the data file, that he intends to protect, with DSP.
2. DO creates a smart contract. To do so, the DO sends a transaction to the null address. The address of the created contract is given randomly, using the client interface. The smart contract has also to record the whitelist in the BC, in order to keep the history of the whitelists, such that the DSP can retrieve the last valid one.

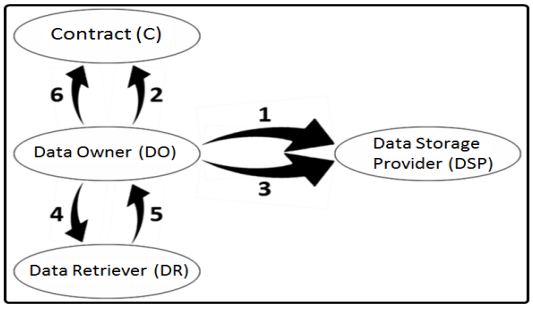

Figure 1: Whitelist Creation Process

3. DO sends the address of the newly created contract as well as the data file to the remote DSP.
4. DO sends the address of the created contract to the authorized DRs. In fact, we emphasize that each authorized DR has to communicate the address of the contract, when he wants to access to the corresponding outsourced data file.
5. DO recovers the address of each authorized DR, to be included in the whitelist.
6. DO sets up the whitelist, by introducing the addresses of authorized DRs, via the client interface.

Note that it is also possible to remove an address from the whitelist using the same process used to add a newly authorized address. In fact, the contract is linked to the outsourced file thanks to its address stored in the DSP. A main advantage of our proposed solution is that the whitelist can be modified without any need to interact with the DSP. Indeed, only a *hyperlink* to the whitelist is stored on the DSP.

3.2.2 Resource Access Process

When an authorized DR wants to access to an outsourced data file, he starts the resource access process with the remote hosting DSP (cf. Fig.2), as follows:

1. the DR sends an access request to the DSP, via an off-blockchain channel.
2. the DSP sends a randomly generated nonce to the requesting DR. Subsequently, the DSP starts listening to the blockchain by scanning the newly added transactions associated with the corresponding contract and analyzes transactions containing the given nonce.

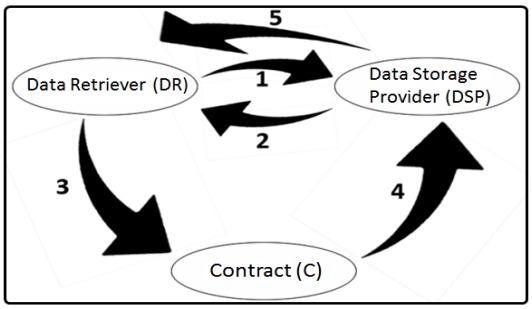

Figure 2: Access to Resources Process

3. the DR sends a transaction to the contract with the nonce as input data. Recall that the contract remains passive during this step, unlike for the solicitation issued by the DO. This transaction does not result in making the contract react, but instead it leaves a trace on the BC, and it proves the transaction is authentic.
4. Once the generated nonce has been identified by the DSP in the input data field of a transaction associated with the corresponding contract, the DSP selects the address of the entity that issued the transaction. Then, it compares the authorized addresses of the whitelist with the originating address of the transaction.
5. In the case where the originating address of the transaction corresponds to one of the authorized addresses, as defined in the whitelist, the DSP identifies the DR.

Finally, we have to emphasize the necessity of including a randomly generated nonce for each authentication session. That is, the nonce permits to prove the freshness of the transaction to the DSP. This latter makes the link between a received request and a successful authentication by the BC thanks to the generated nonce. Our access control application makes use of a public blockchain technology for decentralized authentication, ensuring data auditability and transparency requirements. In fact, although the list of authorized addresses is publicly verifiable, it is impossible to alter the whitelist, where the DO is the unique entity that can grant privileges to DRs.

4 SECURITY DISCUSSION

In this section, we first present our threat model. Then, we provide a security discussion of our proposed blockchain-based access control scheme, with respect to the security and privacy requirements detailed in section 2.3.

4.1 Threat Model

For designing a secure blokchain-based access control scheme, we consider that an attacker is able to read, send and drop a transaction addressed to the blockchain. The attacker targets data owners, data retrievers, data storage providers as well as the blockchain, as follows:

- based on previous data access requests sessions, as well as provided blockchain data, an attacker tries to impersonate a data owner to afford a honest data storage provider some rights to be logged into the blockchain without the legal data owner's granted privileges. This attack is considered with respect to the authenticated access control and auditability requirements.
- an attacker tries an attack against the privacy property w.r.t. both data owners, while trying to directly link a smart contract to a specific owner, and data retrievers while attempting to link an access session to a requesting entity.
- an attacker attempts to prevent the publication of a legitimate transaction in the blockchain. For example, an attacker may try a DoS attack against an access list modification activity or attempt a flooding attack on the blockchain with invalid information. This attack is considered against the auditability and the availability requirements.

4.2 Security And Privacy Analysis

AUTHENTICATED ACCESS CONTROL — our approach ensures an authenticated access control for several reasons here-below listed:

- blockchain-based authentication – our scheme relies on the blockchain infrastructure to enforce the authentication of the participating entities. That is, each entity has to include its blokchain-account address to either outsourcing (i.e; for DOs) or accessing (i.e; for DRs) requests.
- access lists' integrity – in our approach, access control lists are stored in the blockchain as well as on the remote hosting server. That is, as access lists are stored on all blockchain nodes, each entity has a copy of all smart contracts. Thus, in order to afford a honest data storage provider some rights to be logged into the blockchain without the legal data owner's granted privileges, the attacker has to corrupt all blockchain-hosting nodes to tamper access lists.

PRIVACY — The privacy property is ensured thanks to the following technical features:

- one smart contract per outsourced data resource – for each outsourced data resource to a remote server, the DO creates a new smart contract which points out the authorized DRs' addresses. Thus, it is impossible for an attacker to link a smart contract with its DO, mainly in case of one-time accounts, as emphasized in section 2.3.
- a per-access session nonce – for each access session, the remote server generates a random nonce that is used by the requesting DR in its access transaction w.r.t. the smart contract associated to the outsourced data resource, thus proving freshness, and pseudonymity ownership while not revealing the true identity.

AUDITABILITY — Our proposed access control scheme ensures the auditability requirement as:

- tamper-proof architecture – all blockchain-specific operations, such as transaction anchoring activities, are considered as secure and non-corruptible, thus ensuring non-tamper proofs of data access activities.
- transparent usage – our approach is based on a public blockchain infrastructure, that permits public access (i.e; read privilege) to the contract and its associated transactions, to anyone.

Remark 1. *As a highly decentralized infrastructure, the blockchain technology helps also in terms of availability. It becomes possible to provide liveness guarantees of data usage. To prevent DOS attacks, our access control mechanism requires that both DSP and DR use a per-session nonce, randomly generated by the remote server, such that the DR has to include the freshly derived nonce in his access request transaction w.r.t. to a given smart contract.*

5 IMPLEMENTATION

In this section, we detail the implementation of our proposed blockchain-based access control scheme, based on the Ethereum environment, w.r.t. the smart creation process (cf. Section 5.1), installation scripts of the blockchain (cf. Section 5.2) and the client interface development (cf. Section 5.3).

For the implementation of our access control scheme, we first point out three different softwares, namely the client software for the DO, the client software for the DR and the DSP software. Then, we define the structure of the smart contract that will contain the whitelist, and its instantiating BC script.

5.1 Smart Contract Creation

To implement our smart contract, we are solidity language [3] which is a specific Ethereum programming language. We note that there exist several programming languages for smart contract creation, but solidity is a high-level language that perfectly matches our design requirements. To be interpretable by the Ethereum blockchain, the smart contract has to be compiled into bytecodes. For this purpose, we used a compiler called browser-solidity. It is a compiler with a web interface.
Hereafter, we detail the code of each created smart contract. First, we define the attributes of our contract, namely `owner` and `whitelist`. Then, we define a set of functions, namely the whitelist and get_whitelist functions for creating the whitelist and the add(address a) and remove(address a) for adding and removing DR addresses respectively.

`owner` is an address that maps the address of the entity instantiating the contract and `whitelist` is an array that is used to store addresses of authorized DRs.

```
address public owner;
address[] whitelist;
```

The whitelist () function refers to the constructor. We assign the `owner` attribute to the `msg.sender` address, such that DO is the unique entity that can get use of the defined constructor.

```
function WhiteList() public
{owner=msg.sender;}
```

The get_whitelist () function returns the array of allowed addresses. This function permits to the check if the whitelist contains a given DR's address.

```
function get_whitelist() constant
returns (address[])
{return whitelist;}
```

The add (address a) function permits to add a new address to the whitelist. In the following, we detail the different execution of this function. First, a checking line code is added to verify if the entity (msg.sender) trying to modify the existing whitelist is the owner of the contract (`owner`).

```
if(msg.sender==owner){
```

Then, to save storage capacities and avoid redundancy and inconsistency, the add (address a) function verifies whether the new address is not already present in the list. This is particularly important for the deletion step, ensuring that a given address is presented only once at the same whitelist.

```
for(i=0; i<whitelist.length; i++){
 if(whitelist[i] == a){
 throw;}}
```

[3] https://solidity.readthedocs.io/en/develop/

Afterwards, the adding function checks if there exists zero entries in the whitelist table. The checking loop aborts if one of the addresses is zero or if it reaches the end of the table. In case of the existence of a zero entry, it is then replaced by the new address otherwise the new address is appended at the end of the table:

```
while(i<whitelist.length && whitelist[i] != 0)
{i++;}
if(i!=whitelist.length)
{whitelist[i] = a;}
else
{whitelist.push(a);}
```

Similarly, the remove (address a) function first checks whether the requesting entity is the allowed DO (`msg.sender`). Then, it points out the corresponding address to proceed for its deletion. For adding a new smart contract in the blockchain, several javascript commands from the `geth` console (i.e; blockchain interface) are defined. The browser-solidity compiler provides a script, generated with bytecode to allow easy integration of smart contracts such as:

```
var whitelistcontract =
web3.eth.contract([{"constant":false,"inputs":[
{"name":"a","type":"address"}],"name":"add",
"outputs":[],"payable":false,"type":"function"}
,{"constant":true,"inputs":[],"name":"owner",
"outputs":[{"name":"","type":"address"}],"payab
le":false,"type":"function"},{"constant":true,
"inputs":[],"name":"get_whitelist","outputs":
[{"name":"","type":"address[]"}],"payable":fals
e,"type":"function"},{"inputs":[],"payable":fal
se,"type":"constructor"}]);
```

Notice that the script creates a variable "whitelistcontract" needed to define the interface of the smart contract such that the web3.eth.contract (...) function takes as input the contract ABI (i.e; Application Binary Interface) corresponding to the signature of all the defined contract functions.
To generate a new contract, the variable "whitelistcontract" receives a contract instantiation, such that:

```
var whitelist = whitelistcontract.new({
```

The whitelistcontract.new (...) function takes as input the creator of the instance and the bytecode of contract. It also takes a `callback` function that displays the address of the contract. Indeed, it is this address that has to be provided to any entity for interacting with the newly created contract. In addition, we added to this script generated by the compiler a function, called `search(end)`, defined as follows:

```
function search(end){
bn=0; while(bn==0){
tx=eth.getTransactionFromBlock(eth.blockNumber);
 if(tx!=null && tx.to==null &&
 tx.from==eth.coinbase){
 bn=eth.blockNumber;}}
```

The `search(end)` function returns the address of the contract to the authorized requesting entity ("owner").

5.2 Installation scripts of the blockchain

Several scripts have been implemented to enable interaction between the blockchain and the created contract. The first script is designed for the DO that needs to protect his outsourced data resources. This script permits to create a whitelist associated with each data file, add and delete addresses as well as save the corresponding whitelist. These functions are accessible through a user-friendly interface, detailed in subsection 5.3, to ease the manipulation of our access control scheme in real-world applications. In addition to these scripts, we defined and implemented a set configuration scripts that allow to deploy and test the consistency of the proposed construction. First, a script is launched to enable connection with the BC:

```
#!/bin/bash
isLaunched=$(ps aux | grep datadir=$1 | head -1)
```

Consequently, the script checks whether an instance of `geth` was launched with the corresponding node number. If there is no instance, a new instance has to be created specifying the listening ports to communicate with other nodes. To communicate with the main node, its exact address, referred to as enode, is then required as follows:

```
if  [[ ! $isLaunched =~ geth ]] ;
then
geth --datadir="$1" -verbosity 6
--ipcdisable --port 303$1 --nodiscover
--rpcport 81$1 console 2>> $1.log
else geth --datadir="$1" --port 303$1
--nodiscover --rpcport 81$1 attach
ipc://$PWD/$1/geth.ipc
fi
```

Similarly to the connection process, the specified options for the creation of a new nodes include the specification of listening ports for our nodes. That is, the init command allows create a node, and "genesis.json" is a file that describes the first block in the chain. In particular, for our implementation, we defined the difficulty for mining, and put it relatively low to speed up operations on our blockchain.

```
geth --datadir="$1" -verbosity 6 --ipcdisable
--port 303$1 --nodiscover --rpcport 81$1
init genesis.json 2>> $1.log
```

Afterwards, we create an account, accessible by "eth.coinbase". This account permits to the created node to perform BC transactions. To do so, we define a default password "test", and send the command to geth using the "connect.sh" script. Subsequently, we recover the enode of the created node to be stored as a result in the "enodes.txt" file. Thanks this script, a new blockchain is created, a base-account is derived and the enode (i.e; identifier) is stored on a text file. Using the same genesis block, multiple nodes, able to cooperatively perform tasks on the same BC, can be created. Finally, a python [4] script is defined, to re-

[4] https://www.python.org/

cover the saved enodes in the "enodes.txt" file, and generate an addEnode.js javascript [5] file that permits to connect different nodes once they are launched.

Note that miner.sh script is also designed to launch a node in miner mode, as follows:

```
#!/bin/bash
geth --datadir="$1" --port 303$1 --nodiscover
--rpcport 81$1 --mine
```

5.3 User Interface

For our access control scheme, we designed one composite server-interface and two simple user interfaces. Indeed, user interfaces include the DO-interface that permits to the DO to define the authorized accessing entities' addresses, via the creation of a whitelist, as defined in subsection 5.1, and the DR-interface that enables a DR to request access to a given outsourced data file. The composite server-interface is defined w.r.t. two different scripts, where the first script maintains DOs' storing requests and the second script handles DRs' access requests.
For ease of presentation – and submission guidelines– , only the DO-interface is detailed, w.r.t. to BC commands. To do so, we first recover the name of the file that will be sent to the remote server. Once recovered, a DO-DSP connection is instantiated and the data file is sent to the remote server, as follows:

```
s = socket.socket()
host = socket.gethostname()
port = 1234
s.connect((host, port))
s.send((dataf+".dat").encode('ascii'))
```

The DO then creates a smart contract and retrieves its associated address, as detailed in subsection 5.1. The contract address is then sent to the hosting server such as:

```
s.send(json.dumps(data).encode('ascii'))
```

Afterwards, the DO updates the whitelist dictionary which corresponds to the new created whitelist as shown hereafter, where "whtext" is a variable that displays the whitelist in the GUI (Graphic User Interface) and "stringFormat" function allows to obtain a string of characters representing the whitelist.

```
whitelist["address"]=address
 whitelist["list"]=[]
 whtext.set(stringFormat(whitelist))
```

Finally, the DO-interface proposes two different functions, as explained in subsection 5.1, namely the `addWL` function which allows to add an address to the whitelist and the `removeWL` function that removes addresses from the access list. The definition of both functions is almost similar.

[5] https://www.javascript.com/

6 CONCLUSION

In this paper, we presented a new blockchain-based access control scheme that permits data owners to define access rights for each of their outsourced data resources on remote storage servers, and to dynamically delete these granted privileges when needed. Access rights are expressed per data retriever and registered in a smart contract as a whitelist of authorized users with a detailed specific access control list. Our proposed scheme provides an authenticated access control, conducted via the blockchain infrastructure, and ensures an adequate management process w.r.t. efficient whitelists definition. In addition, thanks to the pseudonymity supported by the blockchain, our access control mechanism enables to preserve entities' privacy. Stronger unlinkability properties can be provided in case of one-time blockchain accounts thus ensuring unlinkability between different access sessions and between different data resources belonging to the same owner.

REFERENCES

Crosby, M., Pattanayak, P., Verma, S., and Kalyanaraman, V. (2016). Blockchain technology: Beyond bitcoin. *Applied Innovation*, 2:6–10.

Fu, A., Yu, S., Zhang, Y., Wang, H., and Huang, C. (2017). Npp: A new privacy-aware public auditing scheme for cloud data sharing with group users. *IEEE Transactions on Big Data*.

Kaaniche, N. and Laurent, M. (2017a). A blockchain-based data usage auditing architecture with enhanced privacy and availability.

Kaaniche, N. and Laurent, M. (2017b). Data security and privacy preservation in cloud storage environments based on cryptographic mechanisms. *Computer Communications*, 111:120–141.

Neisse, R., Steri, G., and Nai-Fovino, I. (2017). A blockchain-based approach for data accountability and provenance tracking. *arXiv preprint arXiv:1706.04507*.

Ouaddah, A., Abou Elkalam, A., and Ait Ouahman, A. (2016). Fairaccess: a new blockchain-based access control framework for the internet of things. *Security and Communication Networks*, 9(18):5943–5964.

Regulation(EU) (2016). 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data, ojeu l 119/1 of 4.05.2016.

Swan, M. (2015). *Blockchain: Blueprint for a new economy*. O'Reilly Media, Inc.

Zyskind, G., Nathan, O., et al. (2015). Decentralizing privacy: Using blockchain to protect personal data. In *Security and Privacy Workshops (SPW), 2015 IEEE*, pages 180–184. IEEE.