Blockchain Based Access Control

Damiano Di Francesco Maesa[1(✉)], Paolo Mori[2], and Laura Ricci[1]

[1] Department of Computer Science, University of Pisa, Pisa, Italy
damiano.difrancescomaesa@for.unipi.it, laura.ricci@unipi.it
[2] Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche, Pisa, Italy
paolo.mori@iit.cnr.it

Abstract. Access Control systems are used in computer security to regulate the access to critical or valuable resources. The rights of subjects to access such resources are typically expressed through access control policies, which are evaluated at access request time against the current access context. This paper proposes a new approach based on blockchain technology to publish the policies expressing the right to access a resource and to allow the distributed transfer of such right among users. In our proposed protocol the policies and the rights exchanges are publicly visible on the blockchain, consequently any user can know at any time the policy paired with a resource and the subjects who currently have the rights to access the resource. This solution allows distributed auditability, preventing a party from fraudulently denying the rights granted by an enforceable policy. We also show a possible working implementation based on XACML policies, deployed on the Bitcoin blockchain.

Keywords: Bitcoin · Blockchain · Access control · XACML

1 Introduction

Access Control systems are used in computer security to regulate the access to critical or valuable resources such as data, services, computational systems, storage space, and so on. The rights of subjects to access resources are typically expressed through access control policies, which are evaluated at access request time against the current access context. In Attribute-based Access Control (ABAC) [1], policies consist of a set of conditions over the attributes which describe the features of the subjects, resources, environment, etc., involved in the access request. Among the subject attributes there could be, for instance, his ID, the ID of the company he works for, his role in this company, the name of the projects assigned to him, his physical position, the number of resources he is currently using, and so on.

Some scenarios require that access rights can be transferred from a subject to another for some reasons. For instance, a user could sell its access right to another user. Another example is the one where an employee of a company who was supposed to perform a given computation on a Virtual Machine delegates

L.Y. Chen and H.P. Reiser (Eds.): DAIS 2017, LNCS 10320, pp. 206–220, 2017.
DOI: 10.1007/978-3-319-59665-5_15

the execution of this task to another employee, who needs to access that same Virtual Machine.

Moreover, the evaluation of the access control policy in order to decide whether the requested access to a resource can be executed is performed by a party which is trusted by (the owner of) that resource, but it could be not trusted for the subject of the request who, instead, would like to be guaranteed against unduly denial of access. For example, the Access Control system can run directly on a server of the owner of the resource. In fact, the party which actually evaluates the policy and enforces the result on the resource could maliciously force the system to deny the access to a subject although the policy would have granted it. Hence, in this scenario there is the need for the subjects to have a mean for verifying which policy has been enforced when they performed an access request which has been denied.

This paper proposes an approach based on blockchain technology to represent the right to access a resource and to allow the transfer of such right among users. The proposed approach is validated by a preliminary implementation exploiting the Bitcoin framework.

The paper is structured as follows: Sect. 2 presents a background on blockchain technology and Bitcoin as well as a survey of related works on the subject at hand, while Sect. 3 gives a brief overview of our proposed novel approach. In Sect. 4 we describe the architecture of the access control scheme proposed and Sect. 5 presents our real world implementation example. Finally, Sect. 6 discusses the conclusions and presents our future work.

2 Background and Related Work

A blockchain is a distributed, always available, irreversible, tamper resistant, replicated public repository of data. It allows trustless users to agree on an immutable and auditable piece of data without third party interaction. In other words, blockchain technology allows to build an append only secure database relying on a distributed consensus protocol to decide what valid new data to add in a distributed manner.

Historically blockchain technology was first introduced to support cryptocurrencies and, up to date, cryptocurrencies are still its main field of real practical application, even if several proposals in other fields are being studied. The first blockchain was used by the Bitcoin cryptocurrency protocol [2] and today Bitcoin is still the most popular and widespread example of blockchain technology adoption. This is why we have decided to provide an implementation of this paper proposed approach on this particular protocol.

Bitcoin, as other cryptocurrencies, exploits the blockchain as a public ledger to store value exchanges called 'transactions'. This ledger is divided in blocks where each single block is a collection of non conflicting transactions. The linking between blocks is achieved by saving the hash of the header of the previous block in the header of the next block of the chain. To make each block header (and so its hash) dependent from all transactions contained in that block, the root of the

(implicit) Merkle tree [3], built from the block transactions hashes is included in the block header. Deciding which block to add to the ledger at each step is resolved by a distributed consensus algorithm called 'Nakamoto consensus' that relies on HashCash Proof-of-Works [4].

From a data point of view, the Bitcoin blockchain can be seen simply as a list of transactions. Transactions are created to exchange funds between users, represented by their addresses. An address is a double hash (firstly SHA-256 [5] is applied and then Ripemd-160 [6]) of a public key derived from a ECDSA key pair [7]. Addresses (and hence public keys) are used by users to send and receive payments, while the corresponding private keys are used to provide proofs of ownership (through digital signatures). Creating new addresses is as cheap as creating new ECDSA key pairs, so each user can create and use multiple addresses. Moreover, users are incentivized to use different addresses since the pseudonymity given by addresses is the only (weak) anonymity protection in Bitcoin.

Since the entire state of the system is only defined by the list of transactions saved in the blockchain, transactions are the only mean to manage funds. Funds can be divided or aggregated only by being spent. Transactions are multi input and multi output, hence a transaction may withdraw funds from more than one address and can transfer funds to more than one output address. Furthermore each input is signed by the owner with the private key corresponding to the address spending the funds. A transaction can also specify a voluntary fee to cover the expenses of the validation process. This fee is meant as an incentive for users to take part in the consensus protocol mentioned previously. In a transaction, each output can be seen as a couple (amount, receiver address). Each input specifies, instead, where to withdraw the funds, i.e., the previous transaction (through its hash) where the funds were created. The Bitcoin protocol uses a not Turing complete stack based scripting language, and scripts are (mostly) used in transactions to specify conditions needed to redeem the funds of that transaction. It is beyond the scope of this paper to analyze in detail Bitcoin scripting language, we will only mention its features relevant to this work in Sect. 5. Finally we note that new transactions are created by any user and notified to the community with a gossip style broadcast message on the P2P Bitcoin network.

According to [8] even if blockchain technology is mostly well known for applications in cryptocurrencies such as Bitcoin, it can be used outside of the monetary domain as well, for instance to trace the origin and transformation in a supply chain. [9] shows how blockchain can be exploited to create decentralized, shared economy applications that allow people to monetize, securely, their things to create more wealth. [10] observes that the ability to have a globally available, verifiable and untamperable source of data provides anyone wishing to provide trusted third party services the ability to do so cheaply and robustly.

3 Proposed Approach

In this paper we propose to use blockchain technology to represent the rights to access resources and to transfer them from one user to another. In particular,

we propose to store the representation of the right to access a resource in a blockchain, allowing the management of such right through blockchain "transactions"[1].

The main advantages of the proposed approach are:

- the right to access a resource can be easily transferred from a user to another through a blockchain transaction created by the last right owner, without the intervention of the resource owner;
- the right is initially defined by the resource owner through a transaction, and all the other transactions representing the right transfers are published on the blockchain. Hence, any user can inspect them at any time in order to check who currently holds the rights to perform a given action on a given resource. Consequently, a user who had its access request denied, can check whether the entity in charge of verifying the existence of the required right actually made the right decision.

A common way of expressing access control rights is through Attribute-Based Access Control (ABAC) policies. Roughly speaking, an attribute-based access control policy combines a set of rules expressing conditions over a set of attributes paired to the subject, to the resource or to the environment. The rules are conjunctively or disjunctively combined and they must be satisfied accordingly in order for the access right to be granted. A well-know policy language allowing to express ABAC policies is the eXtensible Access Control Markup Language (XACML), defined by the OASIS consortium [11].

The actors of our reference scenario are the resource owner, say P, (unique for each resource) and a number of subjects, S_i. The resource owner is the entity who has the control of the policy for each of its resources, say R_j, and it creates, updates and revokes such policies. Note that we consider for simplicity that the policy issuer is also the corresponding resource owner. The subjects hold the rights to perform actions on resources, as specified by the respective policies. The subjects can transfer the action rights specified by policies, even by refining or splitting them (as explained in Sect. 3.1).

Hence, our approach requires that P and S_i perform distinct actions, independently one from the others. The policy issuer takes no part in the policy rights exchange, and, similarly, the subject currently owning a right takes no part and needs not to be online when the policy issuer modifies the policy (even if this action might of course affect the subject right).

[1] In the following, we refer to cryptocurrency style blockchains, because they are the main application of blockchain technology currently implemented. Consequently, we assume to have transactions, which are typical of cryptocurrencies. However, cryptocurrencies are just one of the possible applications of blockchain technology. In those cases, we have to define proper transactions to implement the approach we propose.

3.1 Policy Creation, Update, and Revoke

The policy which defines the access rights on the resource R is defined by the resource owner P, and it is stored in the blockchain through a new transaction called Policy Creation Transaction (PCT). After its creation, a policy can be updated by P any number of times and, at the end, it can be revoked, i.e., canceled.

In our approach, the policy consists of:

- the condition which defines the ID of the subject to whom the policy grants the access right;
- the conditions which define the sets of values allowed for the attributes of the subject, resource and environment for the access to be granted.

In other words, the resource owner decides the subject to whom it wants to initially grant the access right and a set of conditions that must hold to grant the access. In our scheme we allow these conditions to be properly modified by the right holders when they transfer these rights to other users. By *properly modify* we mean that the current right holder is allowed to:

- add new conditions in AND with the conditions already defined in the policy;
- split the set of values allowed for an attribute by an existing condition C of the policy in two (or more) sets by defining proper disjunct conditions, C_i and C_j. i.e. the set of attribute values which satisfy C_i OR C_j is the same set of values which satisfy C, and there is no value of the condition attribute which satisfies both C_i and C_j.

We note that adding conditions (done by a right holder) is not the same as executing policy updates (doable only by the policy issuer). Since the conditions added to the policy by right holders are combined with the existing ones through an AND operator, the resulting overall policy can only be more restrictive than the original one. This means that the original policy conditions cannot be violated. During a policy update step, instead, the meaning of the policy can be completely changed. This is correct since the policy issuer is the only one that can update a policy. We also remark as the conditions added by a right holder are added incrementally for each exchange of rights, so they cannot be modified by the new right owners. This is correct since a right owner should be allowed only to restrict the rights it wants to transfer, not to expand them.

For the sake of simplicity, we suppose that each policy concerns one subject ID only. This is not a limitation, because when P wants to grant the access to a resource to several subjects, it can simply produce a distinct policy for each one of these subjects. Moreover, in our approach we suppose that each policy includes one rule only, and this rule includes all the conditions of the policy, properly combined with AND and OR logic operators.

We remember that a blockchain can be seen as a distributed append-only database replicated among all the users. This means that every piece of data added to the blockchain cannot be subsequently removed and it will constitute a

permanent burden on the entire network. This is the reason why, when defining a new protocol, we should try to minimize the amount of data saved on the blockchain, storing essential information only. The problem with our approach is that the standard policy language XACML is a very verbose formalism and policies can be relatively big. Storing policies in XACML format directly on a blockchain will result in a serious space occupation problem.

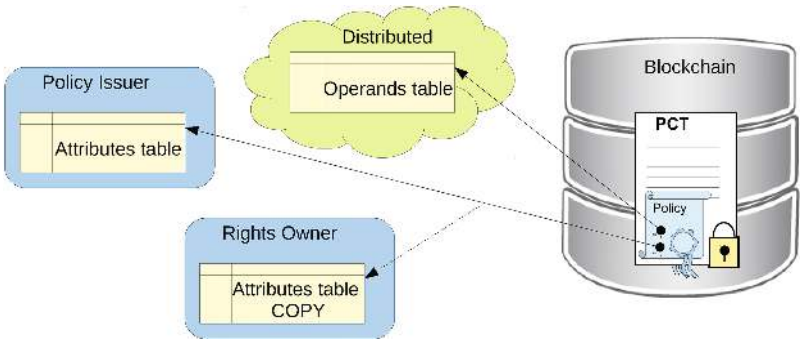

Fig. 1. Proposed hybrid policy storage approach.

The easiest solution would be to store in the blockchain only a link to an external source containing the policy, coupled with a cryptographic hash of the policy itself to make it tamper proof. For example, the blockchain could save only a tinyurl or a torrent descriptor pointing to an external source hosting the actual policy (written in a standard format as, for example, XACML) [12]. The advantage of this solution is obviously to minimize the quantity of information to be stored on the blockchain, since the space occupation of the policy is constant independently of the policy size. The main disadvantage is that policies themselves are stored outside of the blockchain, thus not benefiting of blockchain technology advantages (i.e. availability, security, etc.).

Our approach (shown in Fig. 1) adopts an hybrid solution between saving in the blockchain the entire policy or just a link to it. We chose to store policies directly in the blockchain but coded in a custom built efficient format that favors compression and avoids information repetitions.

First we rewrite a policy expressed directly in ABAC format as a list of basic conditions over attributes. Each condition can be written as three pieces of information:

- the right attribute name;
- the operand connecting right and left term;
- the left term that can be either an attribute name or a constant value (possibly a set of constant values).

Conditions are combined through the logic operator AND/OR to form a unique condition.

If we want the policy storage to be scalable in the size of the policy we would want each of the above listed informations of a condition to be represented with constant size. The logic connector of the policy is of course easy to codify with one bit (0 for OR and 1 for AND). To try to compress the rest of the condition as much as possible we want to compress both attribute names and operands in a fixed size field (for example one byte). To compress operands we can define a protocol defined table of symbols representing the mapping between every possible operand usable in a policy and a numerical code. This map would be maintained at protocol level (open source) and updated with new usable symbols during future protocol versions. We can then follow a similar approach to map attribute names to a short numerical value. The difference is that attribute names are different between users and so the mapping have to be defined by the policy issuer. The attribute mapping is a publicly available mapping of attribute names (identifier in the verbose XACML format) with one unique code of fixed size (for example one byte). The list to be validly published (and accepted by other users to be used in policies) must be signed by the issuer. This public key/identity should be the same used to create new policies using such mapping in the blockchain. A cryptographic hash of this list is then inserted in every policy using the attributes of the list. Such hash is necessary to know what mapping is being used and it prevents the policy issuer from creating a new mapping, potentially changing the meaning of an already existing policy. The policy issuer could still delete the mapping at a future point (since it is stored locally and not on the blockchain), so it is recommended for the user buying the rights derived from a policy to locally save the corresponding mapping. In case of future dispute the right owner can prove that the mapping is correct because the hash matches and the policy issuer cannot deny to be the mapping creator because of the signature attached to the mapping.

We note that this solution allows to save verbose informations about an attribute off the chain, so without space constraints. For example we can save the attribute values type, making the type of operand non ambiguous (i.e. for example differentiating an equality over integers from an equality over strings).

If we adopt this solution, the left term of the condition is the only one of potentially variable length. If it is a parameter name it can be represented as a reference to an entry of the issuer attribute table as for the right term, but if it is a constant value we need to represent it directly, eventually in a compressed format. Furthermore, since we know the type of the attribute (expressed in the verbose attributes table) we can save the values in a suitable format. For example we would save a number or a date in a numerical representation rather than in its string representation.

3.2 Right Transfer

A relevant feature of our approach is that the right to access a resource R can be transferred from the subject who is the current right holder, say S_i, to another subject, say S_j, through a custom data structure stored in the blockchain, called

Right Transfer Transaction (RTT). Each RTT must contain a (direct or indirect) link to the policy whose rights are being exchanged. It is worth noting that the only parties involved in a RTT are S_i and S_j, the RTT is created by S_i, and so the intervention of the owner of the resource is not required during any rights transfer.

When transferring its right through a RTT, S_i can modify the mutable conditions regulating its right only by restricting them. For instance, supposing that a changeable condition defined by the resource owner (or by the previous right owner) states the access can be performed from 9.00 AM to 5.00 PM, S_i could transfer this right to S_j by restricting the access time from 9.00 AM to 1.00 PM. S_i can also split its right in two (or more) parts, and transfer a part of it to a subject, and the other part to another subject. With reference to the previous example, S_i could transfer the access right from 1.00 PM to 5.00 PM to a third subject S_h.

We note that the subjects are only owners of rights to perform actions, in general they have no other right neither on the policy nor on the resource. We also remark that the subjects are able to freely exchange action rights between themselves without any interaction with the policy issuer. That implies that the policy issuer (in general corresponding to the resource owner) has no knowledge in advance of which subjects will be the policy right beneficiaries (even if it can of course model a subject prototype by specifying the correct attributes conditions to be satisfied inside the policy).

We also note that policy updates from a resource owner can potentially change the meaning of a policy. This means that subjects can gain rights on a certain resource that can be later changed by the policy issuer, but, since the blockchain never forgets and timestamps both the right transfer and the policy updates, those changes are manifest and traceable.

4 Architecture of the Proposed Framework

The architecture of the framework we propose for the enforcement of blockchain based access control, shown in Fig. 2, is based on the XACML reference architecture [11], which has been integrated with blockchain technology. Specifically, in order to allow the enforcement of blockchain based access control policies, we customized the Policy Enforcement Point (PEP) and the Policy Administration Point (PAP). The resulting workflow is hence an extension of the standard one.

When requesting to perform an action on the resource, beside the IDs of the subject, of the resource, and of the action, the PEP must also retrieve an additional information to unequivocally link the subject S_i with a RTT in the blockchain. As an example S_i might be required to sign a challenge nonce with the private key corresponding to the identity it used to get the access rights in the RTT. This is no different from a classical authentication scheme in a classical access control scenario. All those informations are properly included in the request which is passed to the Context Handler (CH). The CH is in charge of managing the workflow of the decision process, interacting with all the other components of the authorization system.

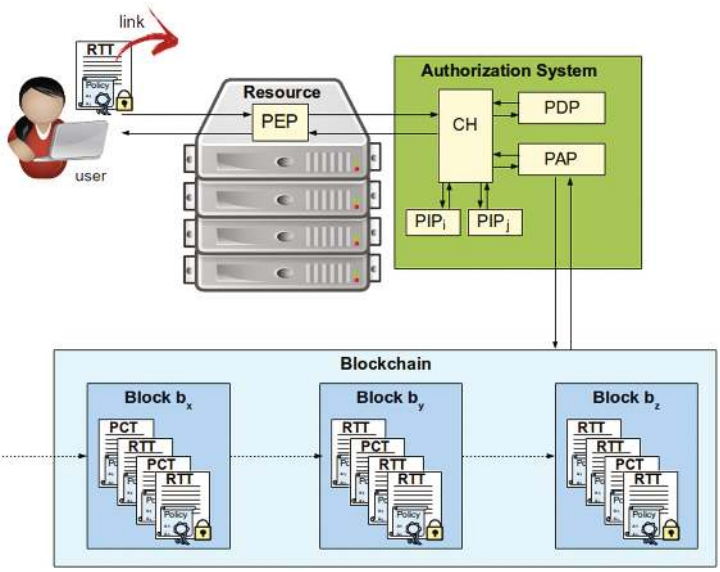

Fig. 2. Architecture of the Blockchain based access control framework.

First of all, the CH sends the request to the PAP. The PAP extracts the RTT link from the request, and retrieves from the blockchain this RTT and all the other RTT related to this policy, as well as the initial policy and the related policy updates issued by the resource owner. The PAP combines the retrieved data to produce a standard XACML policy, and sends this policy back to the CH.

Once the security policy has been reconstructed from the blockchain and verified, its evaluation against the access request follows the process defined by the XACML standard and described in [11]. Briefly, the CH asks the Policy Information Points to retrieve the relevant attributes, it embeds these attributes in the original request, and it passes the policy and the new request to the Policy Decision Point (PDP), which evaluates it and returns to the CH the decision: permit or deny. The CH then forwards the decision to the PEP, which enforces it on the resource by executing the request or not.

5 Bitcoin-Based Implementation

This section describes an example of how the proposed model is deployable in a blockchain technology model. In particular, we developed a proof of concept implementation scheme based on the Bitcoin blockchain. Aim of this section is also to show how our protocol can be immediately deployed on top of an already existent real world blockchain, as the Bitcoin blockchain is, without any modification to the underlying blockchain implementation required. As an example we report in Fig. 3 a real PCT Bitcoin transaction we broadcasted in the Bitcoin network as publicly visible from the site https://blockexplorer.com.

In our scenario firstly a resource owner creates a new policy. Then, an arbitrary number of policy updates and right transfers can be executed, where each of the two actions can be performed independently of the other one. Finally the resource owner can revoke the policy. In our implementation each step (policy creation, policy update, policy revoke or right transfer) is performed atomically by a single Bitcoin transaction.

5.1 Storing Data

As described in Sect. 2, the Bitcoin blockchain was designed to be used as a distributed ledger to manage a very specific kind of data: transactions. In other words, the Bitcoin blockchain was not designed to store arbitrary data. To overcome this limitation, we employ two commonly used methods based on Bitcoin transactions scripting language to store arbitrary data on the blockchain: the OP_RETURN script op code and the MULTISIG transactions (either through a MULTISIG output script or a multisignature P2SH output) [13]. Without going in further details we only note that our implementation automatically chooses the method to be exploited without the need of user intervention. Whatever storage method we use the policies and conditions data is encoded in a compressed custom format that follows the hybrid approach showed in Sect. 3.1.

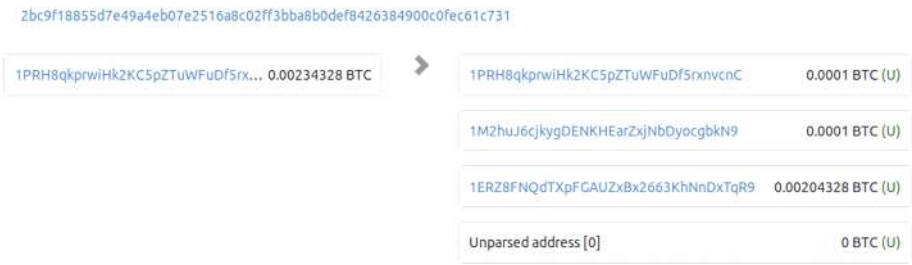

Fig. 3. A real example of PCT in our Bitcoin-based proof of concept implementation.

Since each step is performed exploiting a Bitcoin transaction, each step has a price, i.e., the price of the underlying transaction, defined as the transaction fee paid by the transaction, which is dependent on the transaction size [14]. So we can evaluate the cost of a step as the size of the underlying transaction necessary to perform it. Every Bitcoin full node also keeps in its main memory a data structure to keep track of all unspent transactions outputs (UTXO), so if we include a big output in a transaction (e.g. by including a big multisignature output) this will also encumber precious main memory space of all the users. Finally, we point out that during each step the transaction price is payed by the beneficiary of the action. For policy creation, revoke and update transactions the price is payed by the policy owner (that is the one benefiting from such

operations), while for a rights exchange the price is payed by the buyer (since the buyer is the one who will benefit from the rights).

To embed data in a transaction we first need to create a Bitcoin transaction and so we need value to be exchanged. To build transactions we will use fixed amounts of BTC to represent *tokens*, using an approach similar to the Colored-Coins proposal [15]. We call them tokens because the value they represent will be used in transactions to carry data through the connected scripts, so we are not interested in the monetary value they represent but rather on the information they carry (visible only to those who take part in our protocol). The actual trade value of such tokens is completely independent from their nominal value (i.e. the number of BTC they represent). The fixed amount chosen for a token should be low enough so that it is easy to be owned by any user (otherwise only rich users could take part in the protocol) and its economical value is not relevant compared to its protocol specific value, but also high enough so that it can be transacted freely between users (above the dust limit [16]). In our current implementation we chose 0.0001BTC that corresponds to few euro cents at the exchange rate at the time of writing. We will indicate this value as *CommonAmount* in the rest of this paper.

5.2 Policies Management

Policy Creation. A new policy is issued by the resource owner by creating a new Bitcoin transaction with one or more inputs and two or three outputs. Each of the first two outputs will create a new token, so it is paying out the value of CommonAmount. The only purpose of the inputs is to provide enough funds to create these two tokens and so should include any number of resource owner funds so that $\sum(input\ values) \geq 2 * CommonAmount + fee$. The first two outputs are mandatory, and their structure is defined by the protocol, while the third output is optional, and it represents the change address for the resource owner to keep the unspent input. The order of the first two outputs is important (it can not be changed):

- the first output creates the token that will be subsequently used to perform rights exchanges among subjects. It is credited either to an address that will be used by the policy issuer to sell the action rights to the first subject, or to the first subject directly.
- the second output creates a token containing as data the policy encoded in our custom format. This token is credited to an address controlled by the resource owner and it will be used by the policy issuer to update/revoke this policy in the future.

When the resource owner creates this transaction, the network is notified and, eventually, this PCT will be inserted in the blockchain. If the policy is too large to be included in the second output data field, the policy issuer creates a normal PCT and then creates a chain of policy update transactions (as explained later) to include all the information required. We note that the policy creator

does not have to wait for the PCT to be included in a block before starting to create policy update transactions, since he is the owner of all input and output addresses in both policy creation and update transactions and, consequently, there is no risk of double spending attempts. In the end, this means that a very long policy will generate several transactions and, consequently, it will be simply more expensive for the owner (due to more fees to pay).

Policy Update/Revoke. At any time the policy issuer can update or revoke a policy it created before. To do so, it creates a new transaction spending the second output of the creation policy transaction if the policy was never modified before, or spending the output of the last update policy transaction if the policy was already updated at least once. Obviously, only the policy issuer can create those transactions because it is the only one that can spend the corresponding output.

- *Update:* the update transaction has two (or more inputs). The first input corresponds to the previous update or PCT output and the additional inputs are meant to provide the value necessary to be spent as fees to pay for this transaction. The transaction has one or two outputs, the first one carries on the token of the previous policy update or creation step, while the second one is only used as change address to collect the money left after paying the transaction fees. The update token contained in the first output is used to store the data containing the policy update informations.
- *Revoke:* to revoke a policy, the policy issuer must spend the related token (even to himself), i.e., it must use it as value instead of using the embedded information. To this aim, it just creates a transaction spending the input corresponding to the previous update or PCT. This effectively destroys the token, thus canceling the policy.

5.3 Rights Exchange

To allow the exchange of access rights between two (or more) subjects we assume the existence of some kind of marketplace (or any way of exchanging messages between users) where subjects interested in selling or buying action rights take part. We also note that, since each policy and its updates are publicly visible in the blockchain, each subject can first check a policy to verify the actual rights it is buying. The right exchange between two subjects is achieved through the participation of the subjects in a message exchange protocol to allow them to jointly build and sign the RTT. Main goal of the message exchange is to guarantee that both subjects sign the RTT only after checking that it fulfills the exchange agreement. The RTT is basically a transaction where the token representing the access right is passed from the current subject to the new one and, in exchange, the new subject accredits some money (expressed in BTC) to the current owner. Furthermore the token can be enriched by the old owner with new data to refine the policy conditions and it can be divided in different tokens (as explained in Sect. 3.1). We have seen in Sect. 5.2 that the right transfer token

is created initially by the resource owner in the policy creation transaction, this means that the resource owner is the first one to sell the rights to a subject.

Note that the fact that rights are represented by a token, coupled with the fact that every output can be spent only once, guarantees that the same rights can be transferred only once. Note also that the subject that has currently the policy action rights can also decide to destroy those rights. To do so it only needs to spend the corresponding token as if it was just normal value (using the same process explained previously used by the policy owner to revoke a policy). This is semantically correct since the current owner has payed for the rights and so it can do with them whatever it wants. It could as well decide to never sell the rights again, which is the same for the other users as if it had destroyed them. The advantage is that the resource owner can see from the blockchain when a subject rights token has been destroyed, and so it could choose to revoke the old policy and issue a new one. We also note that revoking a policy or destroying subject rights actively removes the policy data heavy outputs from the UTXO (see Subsect. 5.1) of all the users, so any policy stops encumbering the network once it is not active anymore.

5.4 Policy Evaluation

Let us suppose that a policy granting access rights to the resource R has been created and updated m times, and that this right has been transferred among subjects n times. This means that the blockchain includes a PCT, say pt, defined as in Sect. 5.2 with a chain (actually a tree in case of rights splits) of n RTT defined as in Sect. 5.3 originating from the first input of pt and a single chain of m policy update transactions originated from the second output of pt.

When the PEP receives a request, it only receives a link (for example a cryptographic hash) to the last RTT, say rt, and this is the only information it needs to pass forward in a request to the CH (see Sect. 4). Given a request the PAP can access the blockchain and navigate backward the chain of n RTT from rt all the way back to pt, collecting at each step the additional conditions added by right owners. Once the PAP has reached pt it can read the policy from the blockchain. Then it traverses forward the chain of all m policy update transactions, updating the policy accordingly with the data read at each update step. Once it has the fully updated policy it can add the restricting conditions inserted by right owners and read during the RTT chain traversal. At the end of this process the PAP has derived the completely updated policy in a standard format ready for evaluation by the PDP.

Note that the above policy reconstruction can be done by anyone, given a RTT, since all the informations are publicly visible in the blockchain. This is particularly important for the interested subjects that can retrieve the same way the updated policy from the blockchain and then decide whether to buy the rights for themselves or not.

6 Conclusions

This paper defines an approach to create, manage and enforce access control policies exploiting blockchain technology. The main advantages of this approach are that the policy is published on the blockchain, thus being visible to the subjects of the scenario, and that the access rights can be transferred from one user to another simply through a blockchain transaction. The approach has been validated through a reference implementation based on Bitcoin.

We plan to extend our work to study how to better embed an access control system in blockchain technology. In particular we are studying the possibility of using smart contracts to obtain self enforcing policies. We are exploring how to formulate the classical access control scheme (see Sect. 4) as a smart contract that can be stored and executed in the blockchain to automatically evaluate and enforce policies. Moreover we plan to improve our approach in order to also manage multi-rule XACML policies and policy sets. We are also currently studying the privacy implications of our approach and how to mitigate them.

References

1. Hu, V.C., David, F., Rick, K., Adam, S., Sandlin, K., Robert, M., Karen, S.: Guide to attribute based access control (abac) definition and considerations (2014)
2. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
3. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 369–378. Springer, Heidelberg (1988). doi:10.1007/3-540-48184-2_32
4. Dwork, C., Naor, M.: Pricing via processing or combatting junk mail. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 139–147. Springer, Heidelberg (1993). doi:10.1007/3-540-48071-4_10
5. NIST, U.: Descriptions of sha-256, sha-384 and sha-512 (2001)
6. Preneel, B., Bosselaers, A., Dobbertin, H.: The cryptographic hash function ripemd-160 (1997)
7. Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ECDSA). Int. J. Inf. Secur. **1**(1), 36–63 (2001)
8. Pilkington, M.: Blockchain technology: principles and applications. In: Xavier Olleros, F., Zhegu, M. (eds.) (2015)
9. Huckle, S., Bhattacharya, R., White, M., Beloff, N.: Internet of things, blockchain and shared economy applications. In: International Workshop on Data Mining and IoT Systems (DaMIS 2016), pp. 461–466. (2016)
10. Mainelli, M., Smith, M.: Sharing ledgers for sharing economies: an exploration of mutual distributed ledgers (aka blockchain technology). J. Finantial Perspect. **3**, 38–69 (2015)
11. OASIS: eXtensible Access Control Markup Language (XACML) version 3.0, January 2013
12. Zyskind, G., Nathan, O., et al.: Decentralizing privacy: using blockchain to protect personal data. In: 2015 IEEE Security and Privacy Workshops (SPW), pp. 180–184. IEEE (2015)

13. Hidden surprises in the Bitcoin blockchain. http://www.righto.com/2014/02/ascii-bernanke-wikileaks-photographs.html. Accessed 24 Feb 2017
14. Bitcoin Wiki. https://en.bitcoin.it/wiki/transaction_fees. Accessed 24 Feb 2017
15. Bitcoin Wiki. https://en.bitcoin.it/wiki/colored_coins. Accessed 24 Feb 2017
16. Current Standard for Dust Limit. https://github.com/bitcoin/bitcoin/blob/v0.10.0rc3/src/primitives/transaction.h#l137. Accessed 24 Feb 2017