Privacy-preserving Resource Sharing using Permissioned Blockchains

(the Case of Smart Neighbourhood)

No Author Given

No Institute Given

Abstract. In a resource sharing system, users offer goods and service with specified conditions of access that if met, the access will be granted. Traditional resource sharing systems use a trusted intermediary that mediates users' interactions. Our work is motivated by a decentralized resource sharing platform (proposed in WTSC'19) that uses a permissioned blockchain to allow users to share their resources with their specified attributed-based access policies that are enforced through a set of smart contracts, and removes the need for a trusted intermediary. The system, however, is privacy-invasive and allows users' accesses to be traced, and offers limited availability as access to a resource requires its owner to be on-line. We design a decentralized attribute-based access control system that achieves the same functionality while preserving user's access privacy, and automating access which in consequence leads to the availability of resources. We use two cryptographic primitives, Ciphertext Policy Attribute-Based Encryption (CP-ABE), and ring signatures, and design smart contracts that allow specification and cryptographic enforcement of the users' specified policies for their resources. We analyze the security and privacy of the system, provide the description of smart contracts and implement a proof of concept implementation of the smart contracts and cryptographic algorithms that are used in the system. Our design and implementation are general and can be used for privacy-preserving resource sharing with fine-grained access control in other settings including data and information sharing among collaborating parties.

Keywords: Privacy-preserving resource sharing, blockchain, smart contract, attribute-based access control, attribute-based encryption, anonymous authentication, smart neighborhood

1 Introduction

Resource sharing platforms enable peers to acquire, provide, or share goods and services through intermediary service providers. Sharing platforms such as Airbnb [1] and Uber [2] are centralized online platforms that connect resource owners with resource providers, and manage their interactions and payments. In centralized systems the platform provider is a single Trusted Authority (TA) that mediates all interactions. Centralized designs have the drawback of having a single point of failure and a single TA with full access to users' data. This is not only highly privacy-invasive, but in the case of compromise of TA, reveals all users information and history of their interactions.

In WTSC 2019, a decentralized resource sharing platform for smart neighbourhood was proposed [3] that addresses some of the shortcomings of centralized systems. Authors motivated their work by considering a smart neighbourhood application where an initial trust among parties (and so willingness to share) exists because of the geographic proximity, but there is a need for user control of their sharing so that they are confident about the outcome. Authors proposed a decentralized system that uses a permissioned blockchain that is maintained by a set of Consensus nodes (or C-nodes for short), and enables users to specify conditions of access to their resource and be confident that they will be enforced. Sharing will be among "registered" users of the system that are approved by the C-nodes that use *proof of authority* consensus among themselves to control access to the blockchain. C-nodes verify transactions to ensure they are generated by registered users and publish them on the blockchain. The system uses *attribute-based access control* to allow users to express fine-grained access conditions of access to resources by specifying associated *attribute-based access policies*. Resources have certificates for their properties (attributes) that can be verified by users. A resource requester provides their (certified) attributes in their request, which will be verified by C-nodes and resource providers to match the stated policy. If the request is accepted, the resource owner issues an access token that allows the requester to access the resource. Authors designed a set of smart contracts that automates and implements the system and enforces the stated access policies, effectively removing the need for a single TA. The system however has major drawbacks: it offers no privacy for users and allows their attributes and accesses to be visible by the C-nodes and resource providers; It has limited availability: for accessing a resource, the resource provider must be online to issue the access token. Our goal is to redesign the system such that it is a *privacy-preserving* decentralized attribute-based access control system that automates users' accesses and removes the need for the resource owner to mediate each access.

1.1 Our contributions.

We use two cryptographic building blocks, *Ciphertext Policy Attribute-based Encryption (CP-ABE)* [4] to cryptographically enforce access control, and anonymous authentication and access using *ring signature* [5]. Using CP-ABE protects the need to send the requesters' attributes in plaintext to the blockchain, and ring signature enables anonymous authentication and access request validation, preventing traceability of the requests. Effective use of these primitives needs an overhaul of smart contracts in [3] and design of a new set of contracts. We provide a detailed security and analysis of our design, and provide a proof of concept implementation to demonstrate feasibility of using advanced cryptographic primitives in real-life applications.

Although our work is motivated by sharing in smart neighbourhood, our design and implementation are general and will have wider applications for privacy-preserving resource sharing with fine-grained access control systems, for example, trusted information sharing among collaborating organizations.

Proposed system. In the following, we provide an overview of the system and highlight its new components. The system consists of a number of entities that interact through the blockchain and smart contracts to publish and access resources according

to the stated policies.
Entities of the proposed system. In addition to users, we consider three types of authorities: (i) *Blockchain Authorities (BA)* that maintain the blockchain, membership and authentication of users, (ii) *Certificate Authorities (CA)* that are trusted for issuing certificates for shareable objects and attributes of users that can be verified by BA as well as users, and (iii) *CP-ABE Attribute Authority (CP-ABE AA)* that is responsible for generating CP-ABE private and public keys for registered users (registration by BA) according to their attributes.

Smart contracts for controlling access. We use smart contracts to advertise resources with their associated access policies, and the CP-ABE encrypted metadata. These smart contracts provide the required information to requesters to allow them to find items of interest and using associated policy determine if their attributes satisfy the policy. We consider the following smart contracts:

i. User directory contract (`uDir`): this contract is deployed by the blockchain authority (BA). It holds a table containing the pseudonym and public key of each registered user together with its certificate (corresponding to the user's public key). This contract allows users to choose ring members (public keys of a group of users) to generate their ring signature.
ii. Object directory contract (`oDir`): this contract is deployed by the blockchain authority (BA). It holds a table containing the pseudonym and public key of the resource owner, object Id, description of the objects, the link to the the `objPropRep` and `objACC` contracts.
iii. Object property repository contract (`objPropRep`): this contract is deployed by the resource owner. It holds the Id and properties of the shareable object.
iv. Object access control contract (`objACC`): this contract is deployed by the resource owner. It holds the object Id along with the access policies and CP-ABE encrypted metadata of the resource. This contract differs from ACC contract of [3] since we use CP-ABE for enforcing the access control, and evaluation of policies is done off-chain.
v. Adjudicator contract (`Adj`): this contract is deployed by BA, and allows requesters to report misbehaviors. The system can choose a verifier to validate the reports and penalize the misbehaving owners.

The system uses a permissioned blockchain that is accessible to the system registered users.

User authentication. BA runs an authentication service that uses Authentication Servers *AuthServ*, that register users and provide them with certified public keys (using public key certificate), and verify users' accesses to the blockchain. User accesses are transactions that are signed by the users and verified by the BAs. For privacy, the user identity is verified through an anonymous authentication protocol, and the signature on the transaction will be based on an anonymous signature. We note that using only an anonymous signature on the transaction will make the system vulnerable to replay attacks and so the transaction anonymity and user authentication both must be ensured.

We use a challenge and response anonymous authentication system, where the response is the ring signature [5] of the prover on the challenge sent by the verifier. The challenge in our system is obtained from a randomness beacon service (provided by the BA) that is broadcasted at regular intervals. The random string has length ℓ that is determined by the system security level (we use $\ell = 512$ bits in our implementation). A user request (issued transaction) will include the current value of the challenge, and will be considered valid (after verification of the signature) if it has been generated within a defined interval. This effectively combines user authentication and transaction validation (transaction is generated by a registered user within a close time interval). The transaction that is sent to the BA will be encrypted using the BA's public key.

Processing a request. There are two types of requests, (i) non-anonymous requests which are used to browse blockchain and obtain information about existing resources and users, and (ii) anonymous requests, which are used for a specific resource. The sequence of steps in making a request to access an item, and its processing are as follows: (i) User B_r authenticates itself to an *AuthServ* using a traditional digital signature, and browses `oDir` and `uDir` contracts on the blockchain; (ii) B_r makes an anonymous request (using ring signature) to the `objPropRep` contract for an object O that is listed in `oDir`; *AuthServ* authenticates the (anonymous) request of B_r and passes it to `objPropRep` contract; (iii) B_r receives the properties of the object O and the corresponding certificates, and if verified, makes an anonymous access request for the object O to the `objACC` contract; (v) *AuthServ* authenticates the (anonymous) request of B_r and passes it to the `objACC` contract; (vi) B_r receives the access policies and the CP-ABE encrypted metadata, and decrypts it using its CP-ABE private key. If the private key matches the access policy of CP-ABE encrypted metadata, B_r will obtain the link to the object O.

Security and Privacy. We use a CP-ABE scheme to ensure a requester whose attributes satisfy the item's stated policy is able to decrypt it (security), and use ring signature based authentication protocol to ensure that accesses are by authorized users (security), and the BA and the resource owner cannot link the access requests to the resource requester. Outsiders will only see encrypted communications. A detailed security and privacy analysis is given in Section 4.

Proof of concept implementation. We use Ganache [6] to setup the required private (permissioned) Ethereum blockchain infrastructure. Smart contracts are written in *Solidity* language using Remix IDE [7], and *Truffle* is used for contract deployment and run of the experiments. We evaluate computation cost of using cryptographic primitives namely, (CP-ABE) [4], ring signature [5] and AES symmetric key encryption scheme [8], as well as the execution cost of the smart contract functions in Ethereum blockchain. Our results are presented in Section 5 and show viability of our system for real-world applications.

Organization. Section 2 reviews preliminaries, Section 3 describes the details of our system, assumptions and security goals and user's interaction with the system. Section 4 gives the security and privacy analysis of the proposed system. Section 5 gives the details of the implementation and evaluation result, and finally section 6 concludes the paper.

1.2 Related work

Anonymous access and participation in system has been used in a variety of applications including electronic voting [9], e-mail [10], and social networking [11]. Access control models have evolved over years and new models that more efficiently capture the requirements have been proposed [12,13]. Attribute-Based Access Control (ABAC) is a more recent model that allows a fine-grained approach to expressing access control requirements [14,15]. In cryptographic systems, attribute-based encryption (ABE), first proposed by Sahai and Waters [16], are proposed to allow access control based on attributes of decryptor. Goyal et al. [17] introduced ciphertext policy attribute-based encryption (CP-ABE), where each ciphertext is associated with an access structure. A user receives a secret key associated with their set of attributes and is able to decrypt a ciphertext if and only if their attributes satisfy the access structure of the ciphertext. Bethencourt et al. [4] proposed the first CP-ABE construction using the generic bilinear group model [18,19]. CP-ABE is then used in many privacy-preserving resource sharing infrastructures including [20,21,22,23,24].

Using blockchain to store and enforce access control policies are given in [25,26,27]. In [28], smart contracts are used for enforcing role-based access control policies. The work of [29] uses smart contracts to establish an access control for IoT systems. In [30], a blockchain-enabled decentralized capability-based access control is proposed for propagation and revocation of the access authorization. A multi-authority attribute-based access control scheme is proposed in [31]. The work of [31] uses Ethereum smart contracts to develop an attribute-based access control system. Smart contracts for access control have also been considered in [32,33,34].

2 Preliminaries

Table 8 in Appendix A summarises the notations used in this paper.

Ciphertext-Policy Attribute-Based Encryption (CP-ABE). A CP-ABE scheme $\mathsf{CP\text{-}ABE}$ consists of four algorithms: (i) the setup algorithm $\mathsf{CP\text{-}ABE.Setup}(1^\lambda)$ that for a security parameter λ, generates the master secret key msk and the system's public key pk_{abe}; (ii) the key generation algorithm $\mathsf{CP\text{-}ABE.KGen}(1^\lambda)$ that generates the CP-ABE private key sk_{abe} for a given attribute set $attr$; (iii) the encryption algorithm $\mathsf{CP\text{-}ABE.Enc}$ that takes a massage m, and a policy P and produces the ciphertext $c^{CP\text{-}ABE}$; and (iv) the decryption algorithm $\mathsf{CP\text{-}ABE.Dec}$ that takes a ciphertext $c^{CP\text{-}ABE}$ that is obtained as above and a private key sk_{abe} and outputs m if $attr$, the associated attribute set of sk_{abe}, satisfies the policy P, else returns $\perp$. In our implementation we use Bethencourt et al. CP-ABE scheme [4].

Ring Signature: Ring signatures [5] is an anonymous signature scheme that hides a user's public key among a set of N public keys, making the signer of the message the holder of one of the N keys. A ring signature scheme RingS consists of three algorithms: (i) A key generation algorithm $\mathsf{RingS.KGen}(1^\lambda)$ that takes the security parameter λ and produces a set R of N public and private key pairs $R=\{(pk_i,sk_i),i\in\{1,2\cdots N\}\}$; (ii) Signature generation algorithm $\mathsf{RingS.Sig}$ signs a message m using the signer's private key sk_i, and the public keys in R, and outputs a ring signature σ_R; and (iii)

RingS.Vf that takes public keys in R, message m, and the signature σ_R as input and verifies the signature: if valid, returns 1, else 0. A secure ring signature ensures anonymity, unlinkability, and existential unforgeability.

Blockchain and smart contracts. A blockchain is a distributed ledger technology which stores data (transactions) in a growing chain of ordered blocks that are securely and irreversibly linked to each other through a cryptographic hash function. All peer nodes who run the system use a consensus algorithm to agree on the validity and the order of blocks [35]. We consider *permissioned blockchains* where the consensus algorithm is run by a set of privileged computing nodes, referred to as *Blockchain Authorities* (BA in this paper)[1], that verify transactions of users, and if verification succeeds, publish the result on the chain based on an agreed consensus algorithm. The consensus algorithm is a *Proof of Authority algorithm* that will be defined and agreed upon by the BAs at the time of system setup. A permissioned blockchain is only accessible to registered members of the system. In non-permissioned blockchain such as Ethereum [36] anyone can join the blockchain and participate in the consensus algorithm.
Smart contract is a computer program that runs in a blockchain consensus computing network [37]. Each program instruction is agreed upon through the consensus algorithm and so the execution of the program will be trusted.

Randomness Beacon. A randomness beacon is a service that periodically generates and publishes (broadcasts) a random string. Randomness beacons must be *unpredictable, unbiased* and *available* [38]. An additional desireable property of randomness beacon is *public verifiability* that ensure the claimed randomness of the beacon. An example of a randomness beacon is the NIST randomness beacon [39] that uses hardware-generated randomness.

3 System design

We consider three types of authorities: (i) Blockchain authorities, BA, that register users and issue certificates for users' public keys (for a digital signature algorithm), authenticate users (cf. section 3.2 for more details), and manage interaction of users with the blockchain. BA also runs a randomness beacon service that broadcasts random numbers at regular intervals. BA publishes two public keys, one public key is for verifying the signed certificates, and the second public key is for an encryption algorithm that is employed by users to communicate with blockchain. (ii) Certificate authorities, CA, verify users attributes and objects' properties, and issue the corresponding certificates for them. CA is also responsible for checking the legal restrictions of the objects and only issues certificate for valid objects. (iii) CP-ABE attribute authority, $CP\text{-}ABE\ AA$, generates a master secret key msk and a system public key pk_{abe} for the CP-ABE cryptosystem, and publishes pk_{abe} on the blockchain. It also generates private keys for users (corresponding to their certified attributes).
Let A_o be a user who owns object O, and B_r denote a user who makes a request r. The resource owner A_o can deploy contracts for their objects on the blockchain that will be visible to all registered users. A contract can hold the information about multiple objects, and can be updated by A_o.

[1] These nodes were referred to as C-nodes in [3]

Owners will be represented by pseudonyms that can be mapped to their identities by the BA and so are traceable by BA when they use their public keys. User authentication to BA is through a challenge-response authentication protocol between the user and the BA, where BA issues a random challenge, and the user response is the signed challenge, possibly together with other transaction data. A user can use their secret key to sign the challenge, in which case their identity will be known by the BA, or use a ring signature for anonymous authentication. The challenge in both cases is obtained by the latest broadcasted randomness. More details on this protocol in 3.2.

We consider five contracts in the system: User directory contract (`uDir`), Object directory contract (`oDir`), object properties repository contract (`objPropRep`), and object access control contract (`objACC`) and Adjudicator contract (`Adj`). `uDir`, `oDir`, and `Adj` are deployed on the blockchain by BA when the system is set up and they hold the information related to all users. `objACC` and `objPropRep` are deployed by resource owners, upon sharing a resource. A user B_r who makes a request may receive an object that does not match its advertised properties. In such cases, the requester can report misbehavior to BA that is implemented by a Adjudicator contract. The details of contracts are given in Section 3.3.

3.1 Security goals

We consider the following entities:(i) users, (ii) BA, (iii) CA and CP-ABE AA, and (iv) outsiders.

Trust assumptions. We assume BA is semi-honest: they follow the protocol but want to infer information about users and their accesses. For example, BA may link requesters and owners, or link requests of users. CA and CP-ABE AA are fully trusted: they follow the protocol and manage the keys as required. Users can deviate arbitrarily from the protocol: an owner may be unregistered, claim properties that the object does not have, or provide invalid link to the object. Users may be curious about others.

Security. The goal is to ensure that access to an object is only granted if (i) the requester is registered, and (ii) the user's specified access policy is satisfied.

Privacy. We consider two privacy requirements.

i. Registered users, BA, and outsiders cannot link the requester of an object O to the object owner;
ii. Registered users, BA, and outsiders should not be able to link access requests of users.

In our privacy analysis we consider two cases:

i. *Pre-interaction* case, where the system is ready to function (user registration and advertising resources are complete) but no browsing or access requests has been made to the BA;
ii. *Post-interaction* case, where some successful interactions to view or access objects have been made.

We define *information leakage (privacy breach) of the system* as the difference between the views of an entity in the above two cases and what can be inferred from it.

3.2 Authentication scheme

Let $\mathsf{Beacon}(t)$ denote the Beacon's generated random value at time t. We use $\mathsf{Beacon}(t)$ as the challenge value for user authentication across the system for the time period $[t, t{+}1]$. A user response will be generated using the digital signature scheme $\mathsf{DS} = (\mathsf{DS.KGen}, \mathsf{DS.Sig}, \mathsf{DS.Vf})$ and using the user's private key, on a message that includes $\mathsf{Beacon}(t)$. For anonymous authentication, a ring signature based on this signature scheme will be used. Steps of response generation are as follows.

(i) Get auxiliary information that are required for sending message to blockchain (e.g. list of registered users for choosing the public key set set_{pk}, the pubic key of $AuthServ$ for encryption).

(ii) Use $r_t = \mathsf{Beacon}(t)$ where current time is in $[t, t{+}1]$.

(iii) Sign $(pk_{TYP}||m||r_t)$ where m is the request, r_t is as defined above, and pk_{TYP} is the public key of signature type $TYP \in \{S, RS\}$, where S and RS denote traditional digital signature and ring signature, respectively. Let σ denote the signature value. The response is (m, r, σ, pk_{TYP}).

We consider two types of requests, (i) non-anonymous requests, which are used for browsing the blockchain , and (ii) anonymous requests that target a specific resource. Non-anonymous authentication is used by a user, a resource requester, for browsing the blockchain [2]. For such requests, pk_{TYP} is the public key of the user[3]. Anonymous authentication is used by the resource requester to access a specific resource that must remain unlinkable to the requester. In this type of request pk_{TYP} is equal to a set set_{pk} of public keys of the system that includes the public key of the user, and will be used in the ring signature.

We require transmitted messages to the blockchain be encrypted with the BA's (who is also $AuthServ$) public key Pk_{AS}. Users will use pk_{AS} to encrypt their message to BA $C = \mathsf{Enc}(pk_{AS}, (m, r, \sigma, pk_{TYP}))$. Then, $AuthServ$ decrypts C, and verifies the requester and request as follows.

(i) Verify that $r = \mathsf{Beacon}(t)$ for the time interval $[t, t{+}1]$ that user is sending the message.

(ii) Verify that pk_{TYP} has been already recorded in `uDir` contract.

(iii) Verify σ based on the signature type specified by Pk_{TYP}, i.e., $result = \mathsf{DS.Vf}(\sigma, pk_{TYP}||m||r, pk_{TYP})$. If $result{=}1$ send (m, pk_{TYP}) to blockchain, otherwise send $reject$ to the user.

Choosing ring members. We assume set_{pk} comprises of two subsets $set_{pk} = set_1 \cup set_2$; set_1 contains public keys that are randomly chosen from the list of registered users, and set_2 is randomly selected public keys from the pk_{set} of last request issued to BA. We assume that the size of ring is determined based on the total number of malicious users in the system such that it can guarantee in each randomly chosen ring at least two parties are honest (the signer and one of the ringer members).

[2] We highlight that the transactions of resource owner to blockchain are also non-anonymous.

[3] We can use the ring signature with ring size equal to 1 as a regular signature

Inheriting from the ring signature, our authentication system provides anonymity, unlinkability, existential unforgeability, and real-time authentication (please see Appendix B for more details).

3.3 Contracts

The details of contracts are given below (see Appendix G.1 for algorithms). Note that all the contracts have a *self-destruct* method (to make the contracts inaccessible) which can be called only by the contract owner.

User directory contract (`uDir`): This contract holds a table containing the pseudonym, public key and certificate of registered users (cf. Table 1). These information can be browsed by requesters to form the set of public keys set_{pk} for the ring signature. This contract has the following interfaces:

Table 1. `uDir` table

PId	pk	cert
Alice	pk_{A_o}	$Cert_{BA}^{pk^A}$
Bob	pk_{B_r}	$Cert_{BA}^{pk^B}$

– `registerUser():` is used by *AuthServ* to set the information of registered users.
– `deleteUser():` is used by `Adj` contract (if a user misbehaves) or *AuthServ* to delete the information about the user.

Object directory contract (`oDir`): This contract holds a table of object Ids, pseudonym and public key of the resource owner, description of the object, address of the `objACC` contract, ABI of `objACC` contract, address of the `objPropRep` contract, ABI of the `objPropRep` contract, and the state of the object (cf. Table 2). This contract has the following interfaces:

Table 2. `oDir` contract

Oid	pId_o	pk_o	ODesc	`objACC` address	`objACC` ABI	`objPropRep` address	`objPropRep` ABI
M_i	Alice	0x456ab7..	Cartoon, 90 m,...	0xfd45322..	$[setAccessInfo(M_i),...]$	0xab49871..	$[setPropertyInfo(M_i),...)]$

–`registerResource():` is used by each resource owner to register their objects and provide information for accessing the objects.
–`updateResource():` is used by resource owners to update their resources. Only the user whose public key has been stored in the table is able to update the object.
–`deleteResource():` is used by resource owners and `Adj` contract to delete the information of the inaccessible objects.
– `getContractInfo():` is used by requesters to retrieve the address and ABI of the `objACC` and `objPropRep` for all objects.
– `getAdvertiseInfo():` is used by `objACC` to get the advertisement info for a specific object including the owners information and object description.

Object property repository contract (`objPropRep`): This contract is deployed by each resource owner and stores the list of objects, their properties, and certificates (cf. Figure 3). `objPropRep` has the following interface:

Table 3. `objPropRep` table

Oid	properties	cert
M_i	$prop_1$,...,$prop_\nu$	$cert_{CA}^{M_i}$

– `setPropertyInfo():` is used only by the resource owner to add a new object and its properties and certificates.
– `getPropertyInfo():` is used by requesters and `objACC` contract to retrieve the information of a specific resource. `objACC` only receives the properties of the object, whereas requesters will receive both the properties and certificates of the requested object.
– `updatePropertyInfo():` is used only by the resource owner to update the information of a specific object.
–`deletePropertyInfo():` is used by the resource owner to remove the information of the resources which are not longer accessible.
Access control contract (`objACC`): This contract is deployed by each resource owner to upload the ciphertexts (cf. Table 4) and policies that are required for decrypting the CP-ABE ciphertext.

Table 4. `objACC` table

Oid	Ciphertext	access policies
M_i	$c^{CP-ABE}(M(c_O))$	$Age > 6$, $Preference = local \vee international$, $clubmembership = Club2$

–`addAccessInfo():` is used only by the resource owner to add a CP-ABE encrypted metadata and its access policies.
–`updateAccessInfo():` is used by the resource owner to update CP-ABE encrypted metadata and policy for an object.
– `deleteAccessInfo():` is used by the resource owner only to remove the information of objects that are not accessible anymore.
– `getAccessInfo():` is used by requesters to get the CP-ABE encrypted metadata and the access policies for decrypting CP-ABE metadata.
– `setContactAddress():` is used by the owner to set the address of the `objPropRep` and `oDir` contracts.
– `getRequestHistory():` is used by the resource owner only to retrieve the history of the requests that has been made to the contract. Each `objACC` contract stores the requests, i.e. the authentication information provided by the requester, the Oid of the accessed object, and the time of the request.
Adjudicator contract (`Adj`): This contract is used to record misbehaviors. It keeps a table containing the *Oid* of the resource, the public key of the resource owner, the misbehavior, time of report, and state of the complain. Note that since we are ensuring anonymity for requesters `Adj` cannot be used to record the misbehavior of requesters (unlike [3]). However, if a resource owner provides invalid link, a requester can anonymously complain about it for further checks by a trusted entity (verifier) that is determined by BA. (cf. 5)

Table 5. `Adj` table

Oid	pk_o	Misbehavior	Time	state
M_i	0x4598abc678...	Incorrect link	10:05 12/9/2020	Unchecked

–`registerVerifier():` is used by BA to set a verifier(s) who can check the complains.
– `reportMisbehavior():` is used by requesters to complain about a resource owner

and object information.

– `setMisbehaviorState():` is used by the specified verifier to set the result of checking the misbehavior. `Adj` contract calls `deleteResource`(M_i) of `oDir` contract to delete the information about the resources that are proved to be incorrect and calls `deleteUser`(pk_o) of `uDir` contract to delete the resource owners who are misbehaving.

–`getLatestMisbehavior():` is used by users to get the information of the latest misbehavior for a specific resource owner.

3.4 Interactions of users with system

There are three stages in our scheme (cf. Appendix C for the algorithm).

1. *Registration:*
   (a) Each user with $Id_{\mathcal{A}_o}$ (or similarly $Id_{\mathcal{B}_r}$ for requester) chooses their pseudonym $pId_{\mathcal{A}_o}$ and present it to the blockchain authority (BA) at the time of registration and gets the public-private key pair $(Pk_{\mathcal{A}_o}, Sk_{\mathcal{A}_o})$ and a certificate $cert^{BA}_{Pk_{A_o}}$ (corresponding to user's public key). Once the registration is complete, the BA publishes the list $(pId_{\mathcal{A}_o}, Pk_{\mathcal{A}_o}, cert^{BA}_{Pk_{A_o}})$ to blockchain.
   (b) User $\mathcal{A}_o$ contacts the certification authorities and gets certificates for his set of attributes and the properties of the objects.
   (c) The user contacts CP-ABE AA and uses his certificate of BA to authenticate himself, and obtain attribute private keys attached to his $pId_{\mathcal{A}_o}$ (or $Pk_{\mathcal{A}_o}$).
2. *Advertising resource* (see Figure 2 in Appendix D for the flow of the protocol):
   (a) User $\mathcal{A}_o$ who is the resource owner wants to advertise a digital object $\mathcal{O}$. $\mathcal{A}_o$ encrypts the object $\mathcal{O}$ using a symmetric key, $c_{\mathcal{O}} = \mathsf{Enc}(k, \mathcal{O})$ and uploads $c_{\mathcal{O}}$ to cloud.
   (b) Next, $\mathcal{A}_o$ creates CP-ABE metadata $(M(c_{\mathcal{O}}))$. $M(c_{\mathcal{O}})$ consists of (i) additional resource content description, (ii) symmetric key k, and (iii) download link to resource content (link to encrypted file). A_o then encrypts the $M(c_{\mathcal{O}})$ with policy $P_{\mathcal{O}}$ using CP-ABE encryption scheme and gets the encrypted metadata $c^{CP\text{-}ABE}(M(c_{\mathcal{O}}))$.
   (c) $\mathcal{A}_o$ deploys two contracts to share his objects: (i)`objACC`, and (ii)`objPropRep`.
   (d) The blockchain authority (BA) deploys three contracts. (i) `uDir`, (ii) `oDir`, and (iii) `Adj`.
   (e) When $\mathcal{A}_o$ adds an object, or updates an object in `objACC` contract, `objACC` retrieves the information of the objects, including objects descriptions and properties, as well as $\mathcal{A}_o$'s information, including its pseudonym and public key from the `oDir` and `objPropRep` contracts and issues an event to all registered users containing the advertisement information.
3. *Requesting an access* (see Figure 1 for the flow of requesting an access):
   (a) User $\mathcal{B}_r$ browses `uDir` to obtain the list of registered users and chooses the ring members according to Section 3.2. Then, he browses `oDir` to find an object $\mathcal{O}$ that he wants to get access to that. User $\mathcal{B}_r$ forms a request (either for getting data from `objPropRep` contract or `objACC` contract) that includes (Oid), signs it using a ring signature and sends it to BA.

(b) BA runs an authentication service (*AuthServ*). *AuthServ* verifies the signed request, and if valid the respective contract ($\texttt{objPropRep}_O$ or $\texttt{objACC}_O$) returns requested data to requester $\mathcal{B}_r$, $\mathcal{O}$ is the object. $\mathcal{B}_r$ decrypts the CP-ABE metadata using his own CP-ABE private key and gets access to the object.

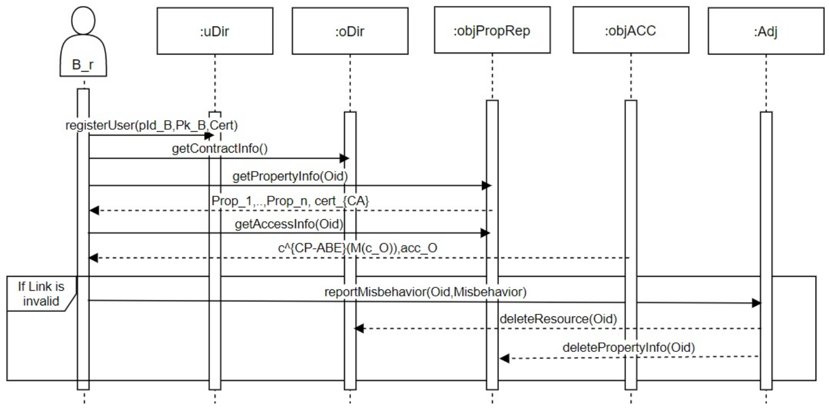

Fig. 1. Sequence diagram of requesting access from a resource requester.

4 Security and privacy analysis

For security we need to show that *only registered users whose attributes match the specified access policies of their requested object can obtain access.* For privacy we only consider post-interaction phase, and show that *(i) no registered user or BA is able to link a request to an existing resource owner, and (ii) no registered user or BA is able to link the access requests of a resource requester to their previous requests. (iii) no outsider can link the requests.* For pre-interaction phase, published contracts can be used to develop a profile of a resource owner's resources. Protection of this profile is not a design goal of the system.

Security. We consider three cases, (i) outsiders are not able to access the object, (ii) requesters with attributes different from the access policies cannot get access to the object, (iii) requesters who were a registered user in some point of time but they have been removed from the system (for e.g. because of misbehaving) are not able to get access to any object that matches their attributes later, and (iv) honest requesters with attributes specified in access policies of an object can get access to the object.

Case (i) is true since any entity who issues a request proves to BA that they are a registered user, through the ring signature-based anonymous authentication scheme. Due to the unforgeability of this scheme, an outsider adversary cannot generate a valid ring signature for a new request. If the outsider adversary can successfully capture one of honest user's requests in time t_m (by eavesdropping the communication channel), it cannot use it in a later time $t_r > t_m$. The reason is that the beacon value at time of request t_r is different from the beacon value at time t_m and BA will reject the requests with stale random value.

Case (ii) is ensured because of the security of CP-ABE scheme. All registered users

can retrieve the CP-ABE encrypted metadata, but only the users with valid CP-ABE private keys can decrypt it and get access to the object. The CP-ABE private keys are issued by CP-ABE AA who is trusted, and generates private keys after verifying the attribute certificates (which have been issued by trusted CA).
Case (iii) is guaranteed because of the use of beacons in our authentication scheme. Note that we assume the private/public keys never expire, and a CP-ABE encrypted metadata can be decrypted with any valid CP-ABE private key. If we do not use beacon in our authentication scheme, any user who has been removed from the system can capture other users' messages (by eavesdropping on their communication channel to BA) and send it later in time. As the authentication passes, attacker can get access to CP-ABE encrypted metadata (which it can decrypt). However, we protect against this attack by using fresh randomness. The copied requests will not pass the authentication, because we assume the time duration for pulsating a new random value is less than the time required for the attacker to copy a request, and hence the request contains a stale random value, which will be rejected by BA.
Case (iv) is ensured because of the correctness property of the anonymous authentication and CP-ABE scheme. The registered users can pass the authentication and retrieve the CP-ABE encrypted metadata. Users who hold a valid CP-ABE private key can decrypt the CP-ABE encrypted metadata correctly and obtain the link to the object (if the resource owner is malicious and the link is not correct, the requesters can report misbehavior to `Adj` contract which can be checked by verifiers of the system).

Privacy. We use the users' and the BAs' views to determine the privacy breach. For *pre-interaction privacy analysis*, a registered user can see `uDir`, `oDir`, and owners contracts `objPropRep` and `objACC` and can develop a profile of the resource owner only by analyzing the advertisement that is published by the resource owner. As stated earlier we do not seek a solution for owners. We show the privacy only for the post-interaction regime; (i) no registered user and BA are able to link a request to an existing resource owner, (ii) no registered user and BA are able to link the access requests of a resource requester to their previous requests, and (iii) no outsider can link the requests.
Case (i) is guaranteed because resource requesters use anonymous authentication to prove that they are registered users. Due to the anonymity of our authentication scheme (described in section 3.2), the public key of the requester will not be revealed to BA. Additionally, BA only publishes the request of the requester and its ring information to the blockchain. Therefore, other users cannot determine who is the requester. Even if other users know the real identity of a set of public keys in a given ring, since at least two honest parties exist in each ring, for an honest requester the level of anonymity is equal to $\frac{1}{2}$ in the worst case. Malicious users are not able to choose invalid private/public keys in order to break anonymity of the scheme, since we have a registration stage that validates the generation of public keys.
Case (ii) is guaranteed because (a) the attributes of requester are not revealed to BA (requester decrypts the CP-ABE encrypted metadata off-chain), and (b) the unlinkability property of our anonymous authentication scheme (described in section 3.2). BA only sees a set of public keys (a ring) which consists of a subset of public keys chosen from previous requests (rings have intersection) that provides unlinkability for consecutive requests made by a requester.

Case (iii) is ensured since outsiders cannot see the contents of messages sent to BA (the messages are encrypted using the public key provided by BA). Although, an outsider can see the messages coming from BA to a user and they can observe the ring used for authentication if the resource owner makes a request to see the access request history, they cannot determine the member who has issued the request.

5 Implementation

In this section, we give details of our implementation for resource sharing in a permissioned Ethereum network. The goal of our proof-of-concept implementation is to analyze the practicality of our proposed model by measuring the cost (in *time*) of the cryptographic operations in our system and the cost (in *gas*) for the blockchain operations.

5.1 System Setup

Actors: We consider users (e.g., resource owner and resource requester), a blockchain authority, a certificate authority and a CP-ABE attribute authority.
Blockchain setup: For proof of concept implementation, we use a private (permissioned) Ethereum blockchain to set up the required blockchain infrastructure that will be maintained by the blockchain authorities (BA). BA deploys the smart contracts (`uDir`, `oDir` and `Adj`) and A_o deploys the smart contracts (`objPropRep` and `objACC`), in the blockchain.
Crypto tools: We use OpenSSL library [40] for creating certificates, object identifier (*Oid*) (using SHA256), and generating public-private key pairs. In addition, for the symmetric key encryption scheme, we use OpenSSL library supported AES-256-CBC encryption scheme. For the ring signature we use implementation of the original algorithm by Rivest et. el [5] and for CP-ABE we use CP-ABE toolkit supported by advanced crypto software collection service [41].
Device specification: We evaluate the performance of our system on Windows 10 with a 3.60 GHz Intel Core i7 CPU and 8 GB RAM.
Smart contracts in the system: We consider the five smart contracts proposed in Section 3.3. All the smart contracts are written in *Solidity* language and developed using the *remix* IDE. Algorithms of these contracts are given in the Appendix G[4].

5.2 Evaluation

To show the practicality of our proposed scheme, we measured the user's cost in different phases: *(i)* registration, *(ii)* resource advertising and *(iii)* resource request. We implemented an example scenario of a movie sharing for our evaluation (see Appendix F for the example details). We used *Ganache* [6], as a private (permissioned) Ethereum blockchain, Remix IDE [7] for writing the smart contracts in *Solidity* and *Truffle* to deploy and run experiments.
For *registration*, we measure the cost in terms of *time*. Table 6 shows the average time required for each user to generate the RSA keys along with the (fixed) size of the keys.
For *resource advertising*, we used AES-256-CBC encryption scheme on the object

[4] The codes for our smart contracts can be found in [42].

'Tom's Trip to Moon" of size 16MB and used CP-ABE encryption scheme to get the CP-ABE metadata. Note that the size of the CP-ABE encrypted metadata depends only on the size of the access policies and it is independent of the object size. The reason is that the CP-ABE encrypted metadata contains the symmetric encryption key (e.g., AES 256-bit key) and the link to the object.The encryption and key generation times are shown in Table 6 with size of the outputs for each of these operations. Using the CP-ABE toolkit, it takes 0.023 seconds to generate master and CP-ABE public key. The size of master key is 156 bytes and the size of public key is 888 byte (in total 1044 byte). Moreover, each Ethereum operation is associated with an explicit cost which is expressed in *gas* [37]. We measure the cost (in terms of *gas*) of program execution in Ethereum blockchain by the resource owner for deploying two smart contracts and for executing different functions (e.g., add, update or delete) in the smart contracts. Table 7 shows the gas required for each of the mentioned tasks.

Table 6. Cost of cryptographic operations

Algorithm	**Time (ms.)**	**Size (bytes)**	**Actor**
Public Key Generation (RSA)	0.055	451	A_o,B_r
AES Encryption	10.704	6361840	A_o
AES Decryption	5.266	6361816	B_r
CP-ABE Private Key Generation	1.526	71353	A_o
CP-ABE Private Key Generation	2.526	119227	B_r
CP-ABE Encryption	75.177	4024	A_o
CP-ABE Decryption	0.107	496	B_r
Ring Signature Generation	2.834	4981	B_r
Ring Signature Verification	1.513	-	BA
Certificate Generation	25.23	1009	BA

Table 7. Cost of blockchain operations (by resource owner)

Tasks	`objACC` **(gas)**	`objProRep` **(gas)**	`oDir` **(gas)**
deployment	1070938	1064347	NA
add	1307190	1243438	NA
update	437233	418354	NA
delete	159076	151414	NA
registerResource	NA	NA	614273

In case of a *resource request*, the costs includes the CP-ABE key generation, ring signature generation and CP-ABE decryption which are shown in Table 6. During this phase, for authentication, the BA sends a challenge (i.e., a random string) of size 512 bits to the requester, and the requester responds with a transaction (signed using ring signature with ring size 10) that includes this challenge value. The requester also needs to make calls to smart contract functions in order to get the required object information (e.g., CP-ABE encrypted metadata, object access policy etc.). Since these function calls does not alter the state of the blockchain (i.e., does not change any variable's value in the contract), there is no cost associated to these calls. In summary, our proof of concept implementation results in Table 6 shows that the overhead for the cryptographic operations are not high. In addition, it should be noted that, although we measured the gas cost of the smart contract function execution in Table 7, for our model, this cost is not vital as we consider a permissioned blockchain setting. The goal is to estimate the complexity of operations performed on blockchain, provide a benchmark for possible future comparisons, and show the concrete cost in case the smart contracts are deployed on a public blockchain. Overall, the values in the tables indicate that our proposed model is feasible for developing real-world applications.

6 Concluding remarks

We designed and provided a proof of concept implementation of a blockchain-based privacy-preserving resource sharing platform that enforces user defined attribute-based access policies. Our design uses cryptographic algorithms to provide privacy and direct enforcement of access policies. By leveraging these primitives one needs to balance security and efficiency. For example increasing the level of anonymity will decrease the efficiency of system, since generating and verifying ring signatures will increase with the size of the anonymity set.

There are limitations in our design that can be addressed in future work. Firstly, using CP-ABE for enforcing policies requires users to obtain their private keys from *CP-ABE AA* which results in a single point of trust for the system (allowing *CP-ABE AA* to be able to access all the resources). Distributing this role among multiple authorities using multi-authority ABE schemes [43,44] can significantly improve this limitation. A second challenge in using CP-ABE is the change in the user attributes that will affect their private keys. Efficient updating of users' private keys to reflect their current attribute sets, will be an interesting direction for future research also.

Our work can be extended in a number of ways including, providing anonymity for the resource provider, developing the platform into a marketplace by linking it to a cryptocurrency, providing an effective support for adjudication and handling of complaints, and formal analysis of the system's security and privacy.

References

1. I. Airbnb, "Vacation rental company." `https://www.airbnb.com`, 2020.
2. U. T. Inc, "Transport company." `https://www.uber.com`, 2020.
3. K. M. Venkateswarlu, S. Avizheh, and R. Safavi-Naini, "A blockchain based approach to resource sharing in smart neighbourhoods," in *International Conference on Financial Cryptography and Data Security*, pp. 550–567, Springer, 2020.
4. J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *2007 IEEE symposium on security and privacy (SP'07)*, pp. 321–334, IEEE, 2007.
5. R. L. Rivest, A. Shamir, and Y. Tauman, "How to leak a secret," in *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 552–565, Springer, 2001.
6. Ganache, "Ganache one click blockchain." `https://www.trufflesuite.com/ganache`, 2019.
7. remix, "Remix-solidity ide." `https://remix.ethereum.org`, 2019.
8. N.-F. Standard, "Announcing the advanced encryption standard (aes)," *Federal Information Processing Standards Publication*, vol. 197, no. 1-51, pp. 3–3, 2001.
9. F. Baiardi, A. Falleni, R. Granchi, F. Martinelli, M. Petrocchi, and A. Vaccarelli, "Seas, a secure e-voting protocol: design and implementation," *Computers & Security*, vol. 24, no. 8, pp. 642–652, 2005.
10. D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, no. 2, pp. 84–90, 1981.
11. Q. Xie and U. Hengartner, "Privacy-preserving matchmaking for mobile social networking secure against malicious users," in *2011 Ninth Annual International Conference on Privacy, Security and Trust*, pp. 252–259, IEEE, 2011.
12. R. S. Sandhu and P. Samarati, "Access control: principle and practice," *IEEE communications magazine*, vol. 32, no. 9, pp. 40–48, 1994.
13. R. S. Sandhu, "Role-based access control," in *Advances in computers*, vol. 46, pp. 237–286, Elsevier, 1998.

14. V. C. Hu, D. Ferraiolo, R. Kuhn, A. R. Friedman, A. J. Lang, M. M. Cogdell, A. Schnitzer, K. Sandlin, R. Miller, K. Scarfone, *et al.*, "Guide to attribute based access control (abac) definition and considerations (draft)," *NIST special publication*, vol. 800, no. 162, 2013.
15. T. Ahmed, R. Sandhu, and J. Park, "Classifying and comparing attribute-based and relationship-based access control," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pp. 59–70, 2017.
16. A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 457–473, Springer, 2005.
17. V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM conference on Computer and communications security*, pp. 89–98, 2006.
18. D. Boneh, X. Boyen, and E.-J. Goh, "Hierarchical identity based encryption with constant size ciphertext," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 440–456, Springer, 2005.
19. V. Shoup, "Lower bounds for discrete logarithms and related problems," in *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 256–266, Springer, 1997.
20. S. Narayan, M. Gagné, and R. Safavi-Naini, "Privacy preserving ehr system using attribute-based infrastructure," in *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*, pp. 47–52, 2010.
21. Z. Wan, R. H. Deng, *et al.*, "Hasbe: a hierarchical attribute-based solution for flexible and scalable access control in cloud computing," *IEEE transactions on information forensics and security*, vol. 7, no. 2, pp. 743–754, 2011.
22. T. Jung, X.-Y. Li, Z. Wan, and M. Wan, "Privacy preserving cloud data access with multi-authorities," in *2013 Proceedings IEEE INFOCOM*, pp. 2625–2633, IEEE, 2013.
23. S. Belguith, N. Kaaniche, A. Jemai, M. Laurent, and R. Attia, "Pabac: a privacy preserving attribute based framework for fine grained access control in clouds," in *SECRYPT 2016: 13th International Conference on Security and Cryptography*, vol. 4, pp. 133–146, SciTePress, 2016.
24. B. Waters, "Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization," in *International Workshop on Public Key Cryptography*, pp. 53–70, Springer, 2011.
25. G. Zyskind, O. Nathan, *et al.*, "Decentralizing privacy: Using blockchain to protect personal data," in *2015 IEEE Security and Privacy Workshops*, pp. 180–184, IEEE, 2015.
26. D. D. F. Maesa, P. Mori, and L. Ricci, "Blockchain based access control," in *IFIP international conference on distributed applications and interoperable systems*, pp. 206–220, Springer, 2017.
27. A. Ouaddah, A. Abou Elkalam, and A. Ait Ouahman, "Fairaccess: a new blockchain-based access control framework for the internet of things," *Security and communication networks*, vol. 9, no. 18, pp. 5943–5964, 2016.
28. Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, "Smart contract-based access control for the internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1594–1605, 2018.
29. J. P. Cruz, Y. Kaji, and N. Yanai, "Rbac-sc: Role-based access control using smart contract," *Ieee Access*, vol. 6, pp. 12240–12251, 2018.
30. R. Xu, Y. Chen, E. Blasch, and G. Chen, "Blendcac: A blockchain-enabled decentralized capability-based access control for iots," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 1027–1034, IEEE, 2018.

31. H. Guo, E. Meamari, and C.-C. Shen, "Multi-authority attribute-based access control with smart contract," in *Proceedings of the 2019 International Conference on Blockchain Technology*, pp. 6–11, 2019.
32. D. D. F. Maesa, P. Mori, and L. Ricci, "Blockchain based access control services," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 1379–1386, IEEE, 2018.
33. D. D. F. Maesa, P. Mori, and L. Ricci, "A blockchain based approach for the definition of auditable access control systems," *Computers & Security*, vol. 84, pp. 93–119, 2019.
34. N. Tapas, F. Longo, G. Merlino, and A. Puliafito, "Experimenting with smart contracts for access control and delegation in iot," *Future Generation Computer Systems*, 2020.
35. M. Raikwar, D. Gligoroski, and K. Kralevska, "Sok of used cryptography in blockchain," *IEEE Access*, vol. 7, pp. 148550–148575, 2019.
36. V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *white paper*, 2014.
37. G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.
38. J. Bonneau, J. Clark, and S. Goldfeder, "On bitcoin as a public randomness source.," *IACR Cryptol. ePrint Arch.*, vol. 2015, p. 1015, 2015.
39. J. Kelsey, L. T. Brandão, R. Peralta, and H. Booth, "A reference for randomness beacons: Format and protocol version 2," tech. rep., National Institute of Standards and Technology, 2019.
40. The OpenSSL Project, "OpenSSL: The open source toolkit for SSL/TLS." `www.openssl.org`, April 2003.
41. J. Bethencourt, "Advanced Crypto Software Collection." `http://acsc.cs.utexas.edu/cpabe/`, December 2006.
42. Code, "Privacy-preserving resource sharing." `https://anonymous.4open.science/r/e2d9b071-52d8-4dff-80ea-9fd78d4aaf55/`, 2021.
43. M. Chase, "Multi-authority attribute based encryption," in *Theory of cryptography conference*, pp. 515–534, Springer, 2007.
44. M. Chase and S. S. Chow, "Improving privacy and security in multi-authority attribute-based encryption," in *Proceedings of the 16th ACM conference on Computer and communications security*, pp. 121–130, 2009.
45. A. Bender, J. Katz, and R. Morselli, "Ring signatures: Stronger definitions, and constructions without random oracles," in *Theory of Cryptography Conference*, pp. 60–79, Springer, 2006.

A Notations

Table 8 shows the notations that are used throughout the paper.

Table 8. Table of notations

Notation	Description
`uDir`	user directory contract
`oDir`	object directory contract
`objACC`	object access control contract
`objPropRep`	object property repository contract
`Adj`	Adjudicator contract
BA	Blockchain authority
AuthServ	authentication service
CA	certificate authority
CP-ABE AA	CP-ABE attribute authority
$Prop_O$	Property of object O
acc_O	access policy for object O
Oid	object identifier
$ODesc$	object description
pId_A	pseudonym of user A
Id_A	identity of user A
pk_A	public key of user A
$Cert_j^i$	certificate issued by j for i
$M(c_o)$	metadata for the encrypted object
$c^{CP-ABE}(M(c_O))$	CP-ABE metadata
A_o	resource owner
B_r	resource requester

B On the security of anyonymous authentication

We consider a ring signature that provides basic anonymity, unlinkability, and existential unforgeability (for the formal definitions please see [45]). Because we have a registration authority which checks the validity of generated keys, and we assume that there are at least two honest members in the ring, the ring signature is not vulnerable to adversarially-chosen key attacks and basic anonymity is sufficient for our system. Choosing part of the ring randomly prevents the BAs from finding a pattern for the ring used by a particular requester in multiple requests and ensures that anonymity is preserved even in multiple executions of the protocol. Additionally, We assume that the random challenge that is concatenated with the message is unpredictable and unbiasable, and it can provide real-time authentication for users. However, in our authentication scheme, all users that are making request in the same time interval use the same random challenge, this is different from existing point to point authentication schemes. The question is whether the multicast of the challenge random value can give an opportunity to the attacker to break the security of the scheme, specifically, to an outsider to replay the message of an honest user. However, we assume that the time interval for generating a beacon is less than the time that is needed for the attacker to capture and resend the message to BA. So, if attacker sends the copied message the authentication fails. For unlinkability, we consider that the chosen rings by different (or even same) requesters has intersection with each other and provides some level of mixing. This prevents the BAs to link the consecutive requests from a requester.

C Interactions in different phases

The interactions between different entities of our system are shown in the following Algorithm 1 where *the highlighted lines represent the interactions that involve blockchain.*

Algorithm 1 Interactions in proposed resource sharing system.

```
    (Registration) */
 1: User $\mathcal{A}$ chooses pseudonym $pid_{\mathcal{A}}$ and $(pk_A, sk_A)$
 2: $\mathcal{A} \to BA$: $pid_A$, $(PK_A, SK_A)$
 3: $BA \to \mathcal{A}$: $cert_{BA}^{pk_A}$
 4: $BA$: deploy contract uDir
 5: $BA \to$ uDir: $registerUser\ (pid_A, pk_A, cert_{BA}^{pk_A})$
 6: $\mathcal{A} \to CA$: request certificates
 7: $CA \to \mathcal{A}$: certificate for user attribute and $prop_{\mathcal{O}}$        ▷ $prop_{\mathcal{O}}$: properties of object $\mathcal{O}$
 8: $\mathcal{A} \to CP-ABEAA$: $register(cert_{pk_A}^{BA})$
 9: $CP\text{-}ABEAA \to \mathcal{A}$: $(pk_{CP-ABE}^{A}, sk_{CP-ABE}^{A})$

    (Advertising resource) */
10: $\mathcal{A}$: get $c_{\mathcal{O}} =$ Enc$(k, \mathcal{O})$ and upload $c_{\mathcal{O}}$ to cloud
11: $\mathcal{A}$: create CP-ABE metadata $(M(c_{\mathcal{O}}))$, and encrypt it to $c^{CP-ABE}(M(c_{\mathcal{O}}))$
12: $\mathcal{A}$: deploy objPropRep contract with $(\mathcal{O}, prop_{\mathcal{O}}, cert_{\mathcal{O}}^{CA})$
13: $\mathcal{A}$: deploy objACC contract with $(\mathcal{O}, acc_{\mathcal{O}}, c^{CP-ABE}(M(c_{\mathcal{O}})))$
14: $BA$: deploy user directory contract (uDir) and object directory contract (oDir)
15: $\mathcal{A} \to$ oDir: $registerResource(Oid, pid, pk_{\mathcal{A}}, ODesc,$ address and ABI of objACC and objPropRep$)$

    Requesting an access */
16: $\mathcal{A} \to$ oDir: $Oid$
17: oDir $\to \mathcal{A}$: $(object\ description,$ address of objACC/objPropRep$)$
18: $\mathcal{A} \to$ objACC/objPropRep: $Oid$
19:     objPropRep $\to \mathcal{A}$: $(properties, certificate)$
20:     objACC $\to \mathcal{A}$: encrypted metadata $(c^{CP-ABE}(M(c_{\mathcal{O}})))$, policies $(acc_{\mathcal{O}})$
21: $\mathcal{A}$: decrypt $c^{CP-ABE}(M(c_{\mathcal{O}}))$ using its secret key $sk_{CP-ABE}^{A}$
```

D Advertising an object

Please see Figure 2 for the sequence diagram of advertising an object by a resource owner.

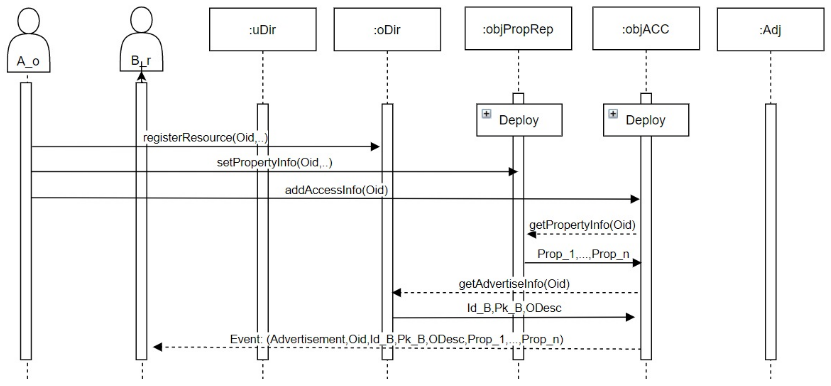

Fig. 2. Sequence diagram of advertising an object by resource owner

E Requesting an access.

Resource requester B_r searches its local database that contain the list of advertised resources and finds the identifier of the resource they want to get access to. Then user A retrieves the address and ABI of `objACC` and `objPropRep` contracts. If resource is available user A retrieves the properties and certificates of the resource from `objPropRep` contract. Then, they send a request to `objACC` contract to gets policies and CP-ABE metadata. BA perform the authentication and replies to the requests of user accordingly. When user gets CP-ABE metadata, he decrypts it using his private key. If the decrypted link does not provide the resource that user wants, he make a complain to `Adj` contract for further checks.

$B_r \rightarrow$ `oDir`: *Oid*
`oDir`$\rightarrow B_r$: pId_{A_o},pk_{A_o}, *ODesc*, *`objPropRep` address*, *`objPropRep` ABI*, *`objACC` address*, *`objACC` ABI*
$B_r \rightarrow$ `objPropRep`: *Oid*
`objPropRep`$\rightarrow B_r$: $Prop_o$, $Cert^o_{CA}$
$B_r \rightarrow$ `objACC`: *Oid*
`objACC`$\rightarrow B_r$: $c^{CP-ABE}(M(c_O))$, acc_o

F A case study.

In this section, we present an example of a digital object sharing using CP-ABE to describe the application of our blockchain based resource sharing scheme. Suppose, A_o owns a movie *"Tom's Trip to Moon"* that he wants to share. We assume that the attribute universe of each user is $\{$*Age*, *Preference*, *Club membership*$\}$, where $Age \in [5,80]$ is an integer value, $Preference \in \{local, international\}$, and *club membership* is realized by a (possibly empty) subset of three clubs, $club1, club2$, and $club3$ corresponding to *Fan-club of cartoon movies*, *Fan-club of adventure movies* and *Fan-club of horror movies*, respectively. Objects (shareable data) are associated with a *title* and a set of

properties from the $\{Type,Quality,Size\}$ universe, where $Type \in \{Movie,e\text{-}book,image\}$, $quality \in \{SD,HD,UHD\}$[5], and $1MB \leq size \leq 10GB$.
Based on this description, we have the following setup for our use case scenario: *(i)* attributes of user A_o, $attr_A$ = {*31, local, club2*} and user B_r, $attr_B$ = {*20, international*, {*club1, club2*}}, *(ii)* movie properties, $prop_O$ = {*Movie, HD, 16 MB*}, *(iii)* metatdata, M_{c_O} = "*Tom's Trip to Moon is a story of a child who dreams to travel to moon someday.*", *256-bit symmetric key*, "*https://onedrive.com*", and *(iv)* access policy, acc_O ={"Age" >6 $\wedge$ "Preference" = (*local* $\vee$ *international*) $\wedge$ "club membership" = *club*2}.

G Smart contracts in our system

We have five smart contracts in our system. The abstract of these contracts are given below (cf. Algorithms 2 to 6) and details of their implementation are given in the next section G.1.

Algorithm 2 Abstract `uDir` smart contract.

```
contract uDir {
constructor (address Adjudicator);
modifier (onlyAuthServ, onlyAdjAuth, onlyBA);
function registerUser (string pseudonym, string pk, string certificate) onlyAuthServ public
function deleteUser(string pk) onlyAdjAuth public
function selfDestruct() onlyBA public
}
```

Algorithm 3 Abstract `oDir` smart contract.

```
contract oDir {
    constructor (address Adjudicator);
    modifier (onlyObjectOwner, onlyOwnerAdj, onlyBA);
    function registerResource (bytes32 Oid, string pid, string pk, string Desc, address ACC_addr,
string ACC_abi, address PR_addr, string PR_abi) public
    function updateResource(bytes32 Oid, string desc) onlyObjectOwner public
    function deleteResource(bytes32 Oid) onlyOwnerAdj public
    function getContractInfo(bytes32 Oid) public
    function getAdvertiseInfo(bytes32 Oid) public
    function selfDestruct() onlyBA public
}
```

Algorithm 4 Abstract `objPropRep` smart contract.

```
contract objPropRep {
constructor (address objACC);
modifier (onlyOwner);
function setPropertyInfo (bytes32 Oid, string properties, string certificate) onlyOwner public
function updatePropertyInfo(bytes32 Oid, string properties, string certificate) onlyOwner public
function deletePropertyInfo(bytes32 Oid) onlyOwner public
function getPropertyInfo(bytes32 Oid) public
function selfDestruct() onlyOwner public
}
```

[5] corresponding to standard definition(SD), high definition (HD) and ultra high definition (UHD) qualities

Algorithm 5 Abstract `objACC` smart contract.

```
contract objACC {
    constructor ();
    modifier (onlyOwner);
    function addAccessInfo (bytes32 Oid, string CM, string[] policy) onlyOwner public
    function updateAccessInfo(bytes32 Oid, string CM, string[] policy) onlyOwner public
    function deleteAccessInfo(bytes32 Oid) onlyOwner public
    function getAccessInfo(bytes32 Oid) public
    function setContractAddress(address oPropRep, address oDir) onlyOwner public
    function getRequestHistory(bytes32 Oid) onlyOwner public
    function selfDestruct() onlyOwner public
}
```

Algorithm 6 Abstract `Adj` smart contract.

```
contract Adj {
    constructor ();
    modifier (onlyBA, onlyVerifier);
    function registerVerifier (address verifier) onlyBA public
    function reportMisbehavior(bytes32 Oid, string pk, string misbehaviour, uint time) public
    function setMisbehaviorState(string state, bytes32 Oid, string pk) onlyVerifier public
    function getLatestMisbehavior(string pk) public
    function selfDestruct() onlyBA public
}
```

G.1 Details functionality of the contracts

Algorithm 7 User directory contract functions.

```
1:  function CONSTRUCTOR($address\_Adjudicator$)
2:      set $AuthServ = msg.sender$; $index = 0$; Adj = $address\_Adjudicator$;
3:  set modifier $ownerOnly$    {if ($msg.sender == AuthServ$) _};
4:  function REGISTERUSER($pseudonym$, $public\_key$, $certificate$) $ownerOnly$ public
5:      set $pid[index] = pseudonym$; $pk[index] = public\_key$; $cert\_pk[index] = certificate$;
6:      update $index = index + 1$;
7:  function DELETEUSER($pk$) public    ▷ can be called by Adj contract (if a user misbehaves) or $AuthServ$ to delete user information
8:      $require$ ($msg.sender ==$ Adj || $msg.sender == AuthServ$);
9:      find $index$ of user $pk$
10:     delete $pid[index]$, $pk[index]$, $cert\_pk[index]$;
```

Algorithm 8 Object directory contract functions.

```
1:  function CONSTRUCTOR($address\_Adjudicator$)
2:      set $BA = msg.sender$; Adj = $address\_Adjudicator$;
3:  set modifier $ownerOnly$    {if ($msg.sender == BA$) _};
4:  function REGISTERRESOURCE($objId$, $pseudonym$, $pub\_key$, $desc$, $ACC\_addr$, $ACC\_abi$, $PR\_addr$, $PR\_abi$) public
5:      set $Oid[objId] = objId$; $pid[objId] = pseudonym$; $pk[objId] = pub\_key$; $oDesc[objId] = desc$;
6:      set $objACC\_address[objId] = ACC\_addr$; $objACC\_abi[objId] = ACC\_abi$;
7:      set $PropRep\_address[objId] = PR\_addr$; $PropRep\_abi[objId] = PR\_abi$;
8:  function UPDATERESOURCE($Oid$, $desc$) public
9:      Check $msg.sender$ is in the list    ▷ user whose $pk$ is stored in the table
10:     update $oDesc[Oid] = desc$
11: function DELETERESOURCE($Oid$) public    ▷ can be called by Adj contract (if a user misbehaves) or resource owner (RO) to delete resource
12:     $require$ ($msg.sender ==$ Adj || $msg.sender == RO$);
13:     delete ($Oid[Oid]$, $pid[Oid]$, $pk[Oid]$, $oDesc[Oid]$, $objACC\_address[Oid]$, $objACC\_abi[Oid]$, $PropRep\_address[Oid]$, $PropRep\_abi[Oid]$);
14: function GETCONTRACTINFO($Oid$) public
15:     returns ($objACC\_address[Oid]$, $objACC\_abi[Oid]$, $PropRep\_address[Oid]$, $PropRep\_abi[Oid]$);
16: function GETADVERTISEINFO($Oid$) public
17:     returns ($pid[Oid]$, $pk[Oid]$, $oDesc[Oid]$);
```

Algorithm 9 Object access control contract functions.

```
1: function CONSTRUCTOR($oACC\_addr$)
2:     set $RO = msg.sender$;
3: set modifier $ownerOnly$   {if ($msg.sender == RO$) _};
4: function ADDACCESSINFO($Oid$, $CM$, $policy$) $ownerOnly$ public
5:     set $CP{-}ABE\_metadata[Oid] = CM$; $access\_policy[Oid] = policy$;
6: function UPDATEACCESSINFO($Oid$, $CM$, $policy$) $ownerOnly$ public
7:     update $CP{-}ABE\_metadata[Oid] = CM$; $access\_policy[Oid] = policy$;
8: function DELETEACCESSINFO($Oid$) $ownerOnly$ public
9:     delete $CP{-}ABE\_metadata[Oid]$, $access\_policy[Oid]$;
10: function GETACCESSINFO($Oid$) public
11:     require ($msg.sender == valid\_requester$)
12:     return ($CP{-}ABE\_metadata[Oid]$, $access\_policy[Oid]$);
13: function SETCONTRACTADDRESS($oPropRep\_addr$, oDir_$addr$) $ownerOnly$ public
14:     set objPropRep = $oPropRep\_addr$; oDir = oDir_$addr$;
```

Algorithm 10 Object Property Repository contract functions.

```
1: function CONSTRUCTOR($oACC\_addr$)
2:     set $RO = msg.sender$; objACC_$address = oACC\_addr$
3: set modifier $ownerOnly$   {if ($msg.sender == RO$) _};
4: function SETPROPERTYINFO($objId$, $properties$, $certificate$) $ownerOnly$ public
5:     set $Oid[objId] = objId$; $prop[objId] = properties$; $cert\_obj[objId] = certificate$;
6: function UPDATEPROPERTYINFO($objId$, $properties$, $certificate$) $ownerOnly$ public
7:     update $prop[objId] = properties$, $cert\_obj[objId] = certificate$;
8: function DELETEPROPERTYINFO($objId$) $ownerOnly$ public
9:     delete $Oid[objId]$, $prop[objId]$, $cert\_obj[objId]$;
10: function GETPROPERTYINFO($objId$) public
11:     require ($msg.sender == valid\_requester$ || $msg.sender ==$ objACC_$address$)
12:     return ($prop[objId]$, $cert\_obj[objId]$);
```

Algorithm 11 Adjudicator contract functions.

```
1: function CONSTRUCTOR
2:     set $BA = msg.sender$; $index = 0$;
3: set modifier $onlyBA$   {if ($msg.sender == BA$) _};
4: set modifier $onlyVerifier$   {if ($msg.sender == verifier$) _};
5: function REGISTERVERIFIER($verifier\_address$) $onlyBA$ public
6:     set $verifier[index] = verifier_address$;
7:     update $index = index + 1$;
8: function REPORTMISBEHAVIOR($Oid$, $pk$, $misbehaviour$, $time$) public   ▷ called by $requester$ to report a misbehaviour
9:     set $mis\_object[pk] = Oid$; $mis\_type[pk] = misbehaviour$;
10:     set $mis\_time[pk] = time$; $mis\_state[pk] =$ "$Unchecked$"
11: function SETMISBEHAVIORSTATE($state$, $Oid$, $pk$) $onlyVerifier$ public
12:     require ($msg.sender == verifier$)
13:     set $mis\_state[pk] =$ "$Checked$"
14:     if $mis\_status = true$ then   ▷ misbehaviour detected
15:         call oDir.$deleteResource$ ($Oid$);
16:         call uDir.$deleteUser$ ($pk$);
17: function GETLATESTMISBEHAVIOR($pk$) $onlyBA$ public
18:     return ($mis\_object[pk]$, $mis\_type[pk]$, $mis\_time[pk]$, $mis\_state[pk]$)
```