Journal of Network and Computer Applications 212 (2023) 103577

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Self sovereign and blockchain based access control: Supporting attributes privacy with zero knowledge

Damiano Di Francesco Maesa [a], Andrea Lisi [a,b,*], Paolo Mori [b], Laura Ricci [a], Gianluca Boschi [c]

[a] Department of Computer Science, University of Pisa, Largo Bruno Pontecorvo, 3, 56127, Pisa, Italy
[b] Institute of Informatics and Telematics, National Research Council, Via G. Moruzzi, 1, 56124, Pisa, Italy
[c] Department of Engineering, University of Pisa, Via G. Caruso 16, 56122, Pisa, Italy

ARTICLE INFO

Keywords:
Access control
Blockchain
Privacy
Self Sovereign Identity
XACML
Zero-knowledge proofs

ABSTRACT

Recent years have witnessed, especially in Europe, a shift aimed at bringing users back at the center of digital systems. This has driven innovation towards the affirmation of decentralized systems, in line with the Self Sovereign Identity paradigm. User control over the consumption and disclosure of their data is a key topic of such drive. In this paper we show how it is possible to apply this increasingly popular concept to a traditionally centralized and opaque digital process: Access Control systems. To this aim we expand the XACML standard for Attribute Based Access Control systems with the novel concept of private attributes, i.e. attributes whose values should not be disclosed while still contributing to a policy evaluation result after user consent. Basing our proposal on blockchain systems, we show how to leverage smart contracts and zero knowledge proofs to allow for transparent policies evaluation without disclosing the value of such sensible attributes. Beside formalizing our goals, presenting the system architecture, and discussing its advantages and drawbacks with respect to the traditional model, we provide a reference example to show our proposal innovative capabilities and provide a prototype experimental evaluation to prove its feasibility.

1. Introduction

The rapid expansion of the digital sphere in our everyday lives has brought forth more and more pressing concerns over data collection, management, and monetization. This has fostered new technologies and initiatives to promote data decentralization and the bringing of users back in control of their data. As an example, the European Commission has moved in this direction with its newly (Spring 2022) agreed upon proposals of *Digital Services Act* and *Digital Markets Act* (European Commission, 2022). Key concepts of these proposals are services interoperability and user control over data collection and exploitation by big centralized services. This is in line with the trend towards users focus in data management formalized by the European General Data Protection Regulation (EurLex, 2016). Both proposals clearly follow the Self Sovereign Identity (SSI) paradigm (Preukschat and Reed, 2021), by shifting control from the platforms where data is generated to the owners of said data. With this respect, the European Commission is pushing towards this paradigm with the eIDAS framework, which allows the usage of national identity credentials all over the member states (European Union, 2014; Joinup, 2022). In SSI, credentials are signed and issued to subjects by issuers. Subjects, in turn, sign and show them to verifiers who accept them if, alongside satisfying the conditions, they are signed by trusted issuers. Subjects are responsible of storing their credentials on their premises, similarly to the covid-19 certificates used in EU. SSI is not a new concept (Loffreto, 2012), but only recently mature enough tools to support it have been made available. In fact, SSI has found suitable ground in the recent decentralized web (web3) (Web3, 2022) revolution, which, in turn, was largely made possible by the affirmation of blockchain technology.

Despite this, many traditional digital processes still remain highly centralized, risking to lag behind and not benefit from such revolution. An example is Access Control Systems (ACSs). Access Control Systems typically require information about users to operate, called attributes, e.g., an user role inside a company to access a given benefit, yet users do not partake at all in the management of such information. For example, consider the traditional Attribute Based Access Control (ABAC) model. In such a model, to protect their resources, resource owners write policies that depend on the attributes of the subjects attempting to access such resources, but those attribute values are managed by third party entities, named Attribute Managers, which provide them directly to resource owners as required. There is no subjects involvement in such a model, the data is passed directly from the Attribute Manager to the resource owner. Moreover, data need to

* Corresponding author at: Department of Computer Science, University of Pisa, Largo Bruno Pontecorvo, 3, 56127, Pisa, Italy.
E-mail address: andrea.lisi@phd.unipi.it (A. Lisi).

https://doi.org/10.1016/j.jnca.2022.103577
Received 3 July 2022; Received in revised form 30 November 2022; Accepted 28 December 2022
Available online 5 January 2023

be passed in clear for a policy to be evaluated on the resource owner premises. In other words, subjects data is managed and disclosed by intermediaries, the Attribute Managers, beyond the subjects control, clearly violating the SSI principles previously described.

Our work introduces a new architecture to solve this incompatibility, proposing the integration of an SSI model to improve the traditionally centralized model of ABAC systems. In our model, Attribute Managers still manage subject attributes, but the subjects alone are the ones in charge of communicating such values to resource owners for policy evaluation. First of all, this removes intermediaries and brings back real data control to users, thus allowing them to choose to deny disclosure of some of their attribute values to certain resource owners. Secondly, this model allows us to propose a protocol that enables users to keep attribute values private while still enabling for a transparent evaluation of a policy dependent on such values. This allows ACSs to reap all the benefits of the rapidly evolving and innovative user-centric movement of SSI, and it is technically made possible by the integration of traditional ACSs with blockchain and zero-knowledge proofs.

Since our proposal is meant to be general, we focus on ACSs implementing the widespread XACML standard (The OASIS Technical Committee, 2013). For this reason, our proposal is based on the *Smart Policy* model introduced in Di Francesco Maesa et al. (2018). This model proposes to encode XACML based Access Control policies as smart contracts containing the policy access logic, named *Smart Policies*. The blockchain properties guarantee that, once deployed, the Smart Policy will grant access as specified by its code without the possibility for the policy owner to influence its access decisions (self execution and immutability) while keeping such access decisions visible to all participants (transparency). However, traditional Smart Policies still rely on third party Attribute Managers to provide the required values for their evaluation, and require that those values are provided in clear to not hamper the decision transparency. To solve this issue we propose to improve the original proposal by adopting an SSI model based on zero-knowledge proofs for private attributes management. The zero-knowledge model used in our proposal belongs to the family of zkSNARK proofs (Bitansky et al., 2012), and the SSI roles are modeled as follows: we consider Attribute Mangers as *issuers* of *Verifiable Credentials* regarding subject attributes, subjects, instead, cover the role of *holders* and provide proofs derived from *Verifiable Credentials* to the Smart Policies that act as SSI *verifiers* on behalf of resource owners. In practice, Attribute Managers maintain a set of public verifiers that can be used by Smart Policies to verify if zero-knowledge proofs submitted by subjects on given attributes are correct. This model allows for Smart Policies to reach a transparent Access Decision without ever revealing the attribute values themselves. In other words, it allows for policies to be evaluated in a trustworthy, accurate, SSI compliant, and transparent way, without the need for the used attribute values to be disclosed.

The main contributions of this work are the following:

- we propose a protocol enabling SSI oriented Access Control systems. Our proposal is general enough to be adapted to any Access Control policy compliant with the dominant XACML standard;
- we introduce and formalize the new concept of private attributes and we describe the actors, operations, trust assumptions, and properties characterizing the proposed Access Control system supporting them;
- we provide a reference example to showcase the proposed system and a prototype implementing the reference example to prove and evaluate its feasibility;
- we critically analyze the benefits and drawbacks of our proposal both from a theoretical point of view and from the practical results of our prototype.

The paper is structured as follows: Section 2 describes the background on the main concepts used in this paper; Section 3 presents the related work on blockchain-based and privacy oriented Access Control systems presented in the literature and describes ZoKrates, an important tool used in this paper; Section 4 describes the original architecture of Di Francesco Maesa et al. (2018) that we build upon in this work; Section 5 describes our proposed system; Section 6 shows a reference example of our proposal; Section 7 presents the experimental evaluation we carried on to test the system; Section 8 presents a critical discussion of the findings of this paper; finally, Section 9 draws the conclusions and suggests future directions.

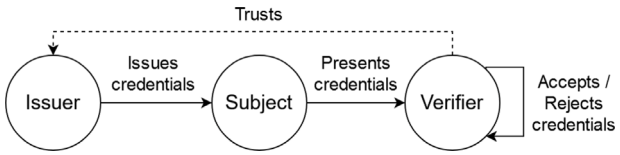


Fig. 1. Overview of SSI.

2. Background

This section describes the base concepts and technologies we leverage in this paper. More precisely blockchain technology, Self Sovereign Identity, XACML attribute-based Access Control, and zkSNARK zero-knowledge proofs.

2.1. Blockchain and smart contract technologies

A blockchain is a data structure that is replicated, maintained, and updated by the nodes of a Peer-to-Peer network following a distributed consensus protocol. Its first implementation came with Bitcoin (Nakamoto, 2008) to support the homonym decentralized currency. A smart contract is a software, composed by a list of state variables and functions, whose code and state is stored on the blockchain and whose execution is replicated by all the nodes of the network. As such, the Peer-to-Peer network forms a decentralized computational platform, with Ethereum (Wood, 2017) being the most famous example. Applications built on smart contracts inherit the underlying blockchain properties such as transparent execution, tamper-proof data storage, including the smart contract code itself, and decentralization.

However, decentralization comes with a cost. The execution of a transaction requires the sender to pay for a fee since all the nodes of the network must execute that transaction, thus consuming resources. When executing a smart contract's function, the fee is proportional to the complexity of the function and to the size of the input data, if any. For example, Ethereum defines *gas* (Wood, 2017) as unit of measurement to compute such fee and the transaction sender sets a parameter, called *gas price*, specifying the amount they are willing to pay to the network nodes for each unit of gas spent.

2.2. Self Sovereign Identity

Opposed to the model involving identity providers as trusted entities storing identity information about individuals, the Self Sovereign Identity (SSI) paradigm (Preukschat and Reed, 2021; Sedlmeir et al., 2021) has been proposed to give the users the responsibility of their own attributes, allowing them to choose which ones they want to share and to whom. As shown in Fig. 1, SSI is built over this triangle: Issuers issue attributes in the form of credentials to Subjects, who store such credentials and send them to Verifiers, who accept credentials only from Issuers they trust. To see examples, Sedlmeir et al. (2021) lists various SSI-based projects.

Two important components in SSI are *Decentralized Identifiers* (DIDs) (Sporny et al., 2022b) and *Verifiable Credentials* (VCs) (Sporny et al., 2022a). A DID is derived from a private/public key pair and uniquely identifies Subjects, who can own multiple DIDs each dedicated to a specific digital identity. A VC is a document digitally signed by the

Issuer declaring a set of attributes, also known as claims, assigned to a DID. The W3C defines a model of a VC represented as a JSON document that includes the credential metadata, such as the context, the issuance and expiration dates, the issuer's DID, the credential claims, i.e. all the attributes about the Subject, and the credential proof, which consists of the Issuer's digital signature embedded in a JSON Web Signature (JWS) and the relative data, such as the algorithm used. The Subject can build and sign, with the private key associated to their DID, a *Verifiable Presentation* that consists of a subset of their VCs to show to a Verifier, e.g., a service provider, who requested them. The structure of a presentation is identical to a credential with the difference of storing a list of VCs instead of a list of claims. The Verifier validates the digital signature of the presentation and of its credentials, and checks whether the credentials' attributes satisfy their requirements. The presentation of a VC to a Verifier can follow two approaches: showing the values of the attributes or providing zero-knowledge proofs proving that the attributes satisfy some conditions while hiding the values of the attributes.

The Self Sovereign Identity framework may also include a *verifiable data registry* used to store data about DIDs, for example additional public keys: this is useful, for example, for Issuers rotating their keys. A blockchain is a potential registry (Ferdous et al., 2019), for example uPort is built on Ethereum while Sovrin is built on a public and permissioned blockchain based on Hyperledger Indy (Hyperledger, 2018b). Another duty of the verifiable data registry is to support the revocation of VCs: upon receiving and evaluating the authenticity of a credential, a Verifier needs also to query the registry to check that credential has not been revoked. uPort proposes a simple revocation registry smart contract that marks as "revoked" the SHA3 of a VC (uPort project, 2019), while Indy detects if a credential has been revoked if it does not belong to the cryptographic accumulator of the set of valid credentials (Hyperledger, 2018a).

2.3. Access Control systems

Access control systems are meant to protect Resources by regulating the action that users, named Subjects, can execute on them according to a given policy. A widely used Access Control model is called Attribute-Based Access Control (ABAC) (Hu et al., 2014b,a), which defines *attributes* meant to describe relevant features of subjects, resources, and the access context, and define conditions about their values in the policies. For example, only a subject whose attribute *role* has value *Professor* and attribute *teacherOf* has value *Cryptography* is allowed to access to the students' grades database of the cryptography course.

In this work we focus on the ABAC model and its main implementation through the eXtensible Access Control Markup Language (XACML) (The OASIS Technical Committee, 2013) standard. XACML provides a standard to express policies, access requests and responses in a XML based formalism, as well as a reference architecture for the evaluation of such policies. The top-level element of an XACML policy is the *policy set*, which includes a set of policies. Each policy includes a *target* section and a set of *rules*. Each rule consists of three main components: the target, the condition, and the effect of the rule on the decision, which can be either Permit or Deny. The target of a policy/rule defines to whom access requests the policy/rule can be applied. Conditions, instead, are complex predicates that are used to compare attributes with constant values or other attributes, using logic and mathematical functions. If the target and the condition included in the rule are satisfied, a rule is applicable to an access request, and the effect declared for that rule determines whether the access request is permitted or denied. The policy specifies a *combining algorithm* which is used to decide the effect of the policy if there are more rules applicable to an access request with conflicting effects (i.e., some rules would permit the access while other rules would deny the access).

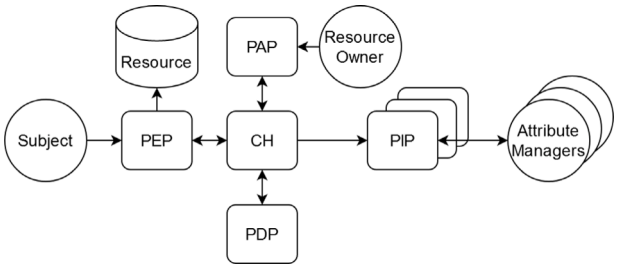


Fig. 2. XACML reference architecture (The OASIS Technical Committee, 2013).

The XACML standard defines a reference architecture as well, which is shown in Fig. 2. The Policy Enforcement Point (PEP) is the component paired with the resource to be protected and is responsible of intercepting and suspending access requests from the Subjects, requesting the access decision, and forwarding the result to the Subject. The Policy Administration Point (PAP) is the component in charge of managing Access Control policies, allowing to retrieve relevant ones for evaluating access requests. Attribute Managers are the components that actually manage the attributes of Subjects, resources, and the environment, providing their values when required for policy evaluation. Policy Information Points (PIP) provide the proper interfaces for interacting with each Attribute Manager, thus allowing to retrieve the latest values of all the attributes required for policy evaluation. The Policy Decision Point (PDP) is the evaluation engine that takes a policy, an access request, and the current attribute values as input, evaluates the policy and returns the related access decision (i.e., PERMIT, DENY, INDETERMINATE or NOTAPPLICABLE) to the PEP. Finally, the Context Handler (CH) is the component which coordinates the execution of the decision process, interacting with the previously described components of the architecture.

2.4. Zero-knowledge proofs and ZoKrates

Zero-knowledge proofs (Feige et al., 1988) allow a prover to demonstrate they know an information to a verifier without explicitly disclosing that information. One approach is using interactive schemes where the verifier challenges the prover multiple times until they are convinced the prover knows that secret information. Such scheme needs to satisfy the following properties (Goldwasser et al., 1989; Partala et al., 2020): (*i*) *Soundness*, if the prover knows the information, they can convince the verifier; (*ii*) *Completeness*, if the prover does not know the information, they cannot convince the verifier that they do; (*iii*) *Zero-knowledge*, the verifier does not learn anything besides that a statement involving the information is true. On the other hand, the *Non interactive zero-knowledge proof* (Blum et al., 1991) approach does not involve an active challenge-response pattern but rely on the computation of a proof demonstrating the zero-knowledge statement. zkSNARK (zero-knowledge Succint Non-interactive ARguments of Knowledge) proofs (Bitansky et al., 2012) are an example that have the following properties: proofs are short, and the prover can convince the verifier with one message; the proof can be generated also using data private to the prover, meaning that the verifier will not be able to know it during the verification; the verification cost is constant. zkSNARK verification schemes are defined on Quadratic Arithmetic Programs (QAPs), based on polynomials, whose a high-level abstraction is Rank-1-Constraint-Systems (R1CS), systems that encode a program as a set of conditions over its variables: finding a solution (witness) of the program means finding a correct assignment of the variables (Eberhardt and Tai, 2018).

ZoKrates, proposed and developed by Eberhardt and Tai (2018), is a toolbox to easily specify zero-knowledge challenges with a high-level language, create proofs, verify them, but most importantly export the verification program as Ethereum smart contracts known as *verifier*. As

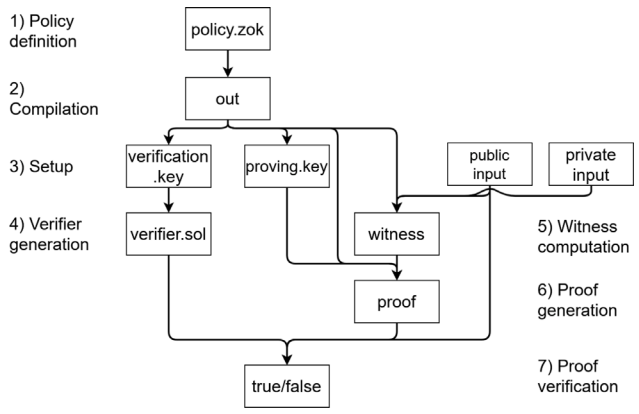


Fig. 3. ZoKrates workflow.

shown in Fig. 3, ZoKrates works as follows: (1) implement a *.zok* program that codifies a condition, or a challenge, defined on *private inputs*, which represent the data the prover must prove to know, and *public inputs*, which can be support variables; (2) compile the program into Rank-1-Constraint-Systems (R1CS), hence compatible with zkSNARK proof systems, which generates the binary file *out*; (3) execute the setup process that from the *out* file generates two public keys, a *proving key* and a *verification key*. This phase is known as *trusted setup* because it requires a random seed to generate the keys, known as toxic waste, which must be deleted otherwise the actor who performed the setup could forge false proofs; (4) the verification key is used to verify an input zkSNARK proof and also to generate the Solidity *verifier* smart contract; (5) the prover, who wants to prove to know the data, i.e. the combination of private and public inputs that solve the program, generates a *witness* and, (6) from that file plus the proving key, generates the zkSNARK *proof*; (7) finally, the prover submits the proof, together with the public inputs, to the verifier smart contract that verifies it and returns true or false. Due to the clash of names between the SSI and ZoKrates verifiers, we call the smart contract zkVerifier.

Besides ZoKrates, other blockchain-related projects integrate zero-knowledge proofs. ZCash (ZCash community, 2021) implements zk-SNARK to support an anonymous transfer of cryptocurrency. To deal with the trusted setup, ZCash implements a complex multiparty computation ceremony (ZCash community, 2021) that decentralizes the generation of the zkSNARK parameters generated by the toxic waste. As a result, no entity participating to the ceremony will know the full toxic waste, thus preventing anyone to forge false proofs (i.e. spending coins). Currently, ZoKrates does not ask the user for a random seed as it internally handles and deletes it (Extropy.IO on Medium, 2018), but the paper suggests to support, as a future improvement, multiparty computation to reduce the trust assumption during the setup (Eberhardt and Tai, 2018). Zero-Knowledge Scalable Transparent ARguments of Knowledge (zkSTARK) (Ben-Sasson et al., 2018) is a new type of proofs whose setup is transparent and not trusted as for zkSNARK, therefore an improvement. The main project adopting zkSTARK technology is StarkNet (2021), a layer-2 scaling solution implementing ZK-rollups for Ethereum. Differently, Monero (Monero community, 2022) aims to provide full anonymity by implementing ring signatures to hide the sender of a transaction, stealth addresses to hide the recipient, and Pedersen commitments to hide the amount. However, ZCash and Monero do not provide Turing-complete languages for smart contracts like Ethereum, and therefore they are not suitable tools for this paper.

3. Related work

Through the years there has been, and still is, a consistent effort into studying anonymization techniques applied to attributes and credentials issued to individuals, in particular employing zero-knowledge proofs outside of a blockchain system (Chase et al., 2016; Garrido et al., 2022). Backes et al. (2005) propose a zero-knowledge protocol ABAC to protect the privacy of the attributes of the requester and, at the same time, provide accountability in case of malicious access. Delignat-Lavaud et al. (2016), instead, apply zero-knowledge proofs to X.509 certificates and, to simplify their usage with zero-knowledge, propose a novel structure and a compiler that, given a policy, generates the code to verify it. In our paper we exploit the idea of translating an XACML policy into a SmartPolicy, a smart contract containing the evaluation code (Di Francesco Maesa et al., 2018) (see Section 4).

The application of blockchain technology to Access Control Systems has been widely studied in the past years. For example, Ding et al. (2019) focus on the Internet of Things (IoT) space and propose an ABAC system based on a consortium blockchain (Hyperledger Fabric) whose nodes are responsible of managing attributes and identities of IoT devices. In the same field, the authors of FairAccess (Ouaddah et al., 2016) consider the problem of the lost of ownership of user's personal data generated by IoT devices once they are shared with a third party. The authors propose to expand the Bitcoin protocol with new types of transactions specific to Access Control: grant, get, delegate, and revoke an access right. An attribute based model is employed by Zhu et al. (2018) as well. In their work the authors propose a platform for ABAC compliant digital asset management on blockchain defining Bitcoin-like transactions to store access management operations. Other approaches less close to our work are shown in Rouhani and Deters (2019), a comprehensive survey of blockchain-based Access Control systems.

Differently from the last two works just presented, and our own previous work (Di Francesco Maesa et al., 2017), that all extend Bitcoin style transactions, our current proposal leverages the novel capabilities of smart contracts to implement self enforcing and transparent Access Control Policies. This approach was presented in Di Francesco Maesa et al. (2019) and Di Francesco Maesa et al. (2019) and it is described in depth in Section 4. However, granting auditability to access request evaluations might also cause privacy issues, potentially making blockchain-based Access Control mechanisms limited to non sensitive attributes. There are different approaches to add privacy to blockchain operations, for example using off-chain communication channels such as in the Bitcoin Lightning Network (Poon and Dryja, 2016), employing cryptographic techniques as in Monero (Monero community, 2022), or outsourcing storage or computation off-chain (Eberhardt and Tai, 2017). We follow the approach of this last proposal by using zkSNARK proofs produced with ZoKrates. Since the proofs do not reveal the values of the attributes they have been computed, but only the outcome of a statement computed over the attributes, the usage of a transparent storage and computational platform is not an issue for privacy. Indeed, zk-creds and zkclaims (Rosenberg et al., 2022; Schanzenbach et al., 2019) are two systems based on zkSNARK applied to hide identity information in credentials, and both papers propose the proofs, being anonymous, can be stored in public networks such as Distributed Hash Table and blockchains. A more comprehensive survey of zero-knowledge techniques applied to blockchain technology can be found in Partala et al. (2020).

As our work focuses on maintaining users information confidentiality, the most relevant works are those attempting to maintain privacy in a blockchain based Access Control System. The closer related work we could find to such topic are Li and Xue (2020), Sharma et al. (2020), Song et al. (2021) and Yang and Li (2020).

Song et al. (2021) propose a system to exchange tokens containing zero-knowledge proofs of access rights to allow IoT devices to be granted access anonymously. Even if this work proposes to use zero-knowledge proofs to enhance a blockchain based Access Control system, it is radically different from our proposal in both scope and privacy focus. Firstly the work has a narrower scope as it focuses on IoT devices, while we provide a system general enough as to be compliant with the XACML standard. Secondly, the goal of their proposal is not to

maintain confidentiality on the attribute values, but rather on the identities of users that have been granted access. This means that attribute values are publicly visible on chain, which is highly undesirable from a privacy point of view, and is what we propose to avoid in this paper. Yang and Li (2020) propose a similar solution of issuing tokens to link private attributes to owner identities with zero-knowledge proofs. This allows to keep the linking confidential until the attributes need to be used. Not only the same consideration as above about the different scope of confidentiality holds for this work, but the application scenario is even further away from our work. In fact, the work focuses on digital Identity Management systems, which do share some similarities with Access Control systems, but lack the same generality in scope. Sharma et al. (2020) propose a system using zero-knowledge proofs to allow for confidentiality in health data management with blockchain. The work is specific to the use case considered, and is a very precise application of Access Control, as such lacks the generality and scenario independence of our proposal. Moreover, confidentiality is focused on the data to be protected, i.e. the resource in the Access Control terminology, not on the attributes that influence the access decision, as in our proposal. Finally, the Access Control protocol presented in Li and Xue (2020) can be seen as a subset of the work presented in this paper since it is limited to the access request phase, delegated to a smart contract verifying a zkSNARK proof. However, the paper does not consider other aspects, for example the problem of revocation and replay attacks.

The off-chain issuance of the attributes follows the Self Sovereign Identity paradigm, which has been extensively studied in the context of access control. Feulner et al. (2022) studies the paradigm for the issuance of event tickets, Fotiou et al. (2022) to access to IoT devices, and Enge et al. (2022) in peer-to-peer offline scenarios using Bluetooth Low Energy communication. An interesting work was carried out by Belchior et al. (2020) who design an Self Sovereign Identity-based access control following the XACML standard. However, the papers mentioned in this paragraph utilize the Verifiable Credentials inside the presentations without concealing the value of the attributes. Belchior et al. mention zero-knowledge proofs but they do not provide any details about their construction and usage. Partial concealment of attributes inside the same credentials could be achieved by implementing selective disclosure techniques (Chadwick et al., 2019; Sonnino et al., 2019), for example based on selective disclosure signatures (Mukta et al., 2020) or hashed values (De Salve et al., 2022), but they do reveal the disclosed attributes to the verifier.

Summarizing, in this paper we combine several components, blockchain-based access control, XACML, Self Sovereign Identity, and zero-knowledge proofs, into a single solution and we explain their interactions in detail. We believe it to be a novel contribution with respect to the related work.

4. A blockchain-based Access Control system

Since in this paper we are proposing a privacy preserving and blockchain based Access Control system implementing the XACML standard, we take the system proposed in Di Francesco Maesa et al. (2019) as a reference and we extend it to protect the privacy of users' attributes.

The underlying idea, initially proposed in Di Francesco Maesa et al. (2018), is to implement an ABAC system on top of a blockchain using smart contracts, named SmartPolicies, executing the logic of XACML policies in a decentralized and auditable way. An XACML policy is translated into a SmartPolicy by the Policy Translation Point (PTP). It is then deployed on the blockchain and it is later invoked to evaluate access requests providing transparent access decisions as intended by the original XACML policy.

The main advantage introduced by implementing an Access Control system on top of a blockchain protocol consists in the full transparency of the policy evaluation process. In particular, since SmartPolicies are deployed on the blockchain, any blockchain participant can verify in advance the authorization rules in a SmartPolicy. Moreover, since the SmartPolicy is a smart contract, the policy evaluation process consists in executing such smart contract on the blockchain network. This execution is considered trusted, and the result is visible to all the blockchain participants. This effectively makes the policy evaluation process not tamperable by any entity.

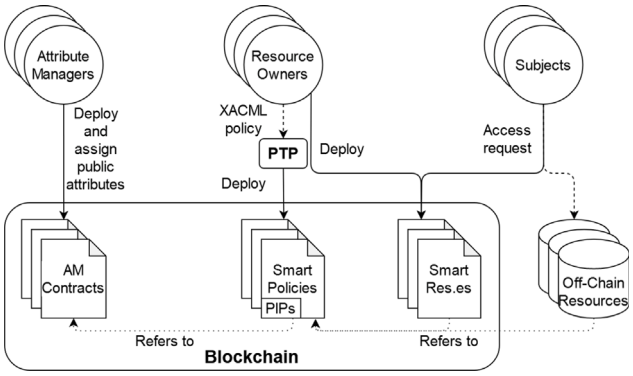


Fig. 4. Simplified reference architecture of the smart policy based proposal presented in Di Francesco Maesa et al. (2019).

In this paper we consider the architecture presented in Di Francesco Maesa et al. (2019) as reference architecture, whose simplified representation is provided in Fig. 4. In such architecture, the Policy Translation Point (PTP) is the component capable of encoding an XACML policy in a SmartPolicy, and it is typically executed on the Resource Owner premises. The resulting SmartPolicy is then deployed on the chosen blockchain by the Resource Owner. The SmartPolicy acts as the XACML Policy Decision Point, evaluating access requests in the current access context against the encoded logic. The core of the SmartPolicy is the *evaluate* function, which implements the PDP functionality by evaluating the policy logic using the current attribute values. To allow a SmartPolicy to reach a decision with a single transaction, the needed attribute values should be available at request time. Attribute values are managed by specific smart contracts named "Attribute Manager smart contracts" (AMContracts in the following), and they are retrieved by the SmartPolicy at decision time through its specific internal functions performing the role of Policy Information Points (PIPs). The proposal assumes the existence of a number of AMContracts created, deployed, maintained, and advertised by third parties.

The Resources protected by a SmartPolicy can be smart contracts (in the following, we call them SmartResources), or traditional off-chain resources, e.g., a Cloud service. The access decision produced by the SmartPolicy is enforced differently between SmartResources and off-chain Resources. SmartResources, being smart contracts, provide by design trusted enforcement of access decisions, since their code is public. Hence, any blockchain participant can check the code to verify that the SmartPolicy is evaluated before invoking the methods of the SmartResource, and that the methods are really executed only in case of positive access decisions. In case of an off-chain Resource, instead, the access decision enforcement is not trusted by design, because it is in charge of the off-chain Resource itself which, by assumption, is not trusted. For instance, a Subject S could have paid a subscription to a Cloud service allowing the execution of a service instance every day. However, when its resources are overloaded, a malicious Cloud service provider could unduly deny the access to S pretending that S has already executed its daily instance. An advantage of the blockchain based system is that unduly denial of accesses can always be detected by Subjects. They can check on the blockchain whether their access requests have been processed by the SmartPolicy the Resource Owners declared for that resource, as well as the related access decisions.

We remark that the transparency of the decision process comes at the cost of privacy, as all values are readable on the chain, thus being

visible to all the users of the blockchain. We remark how this problem is also partially present for traditional (centralized) Access Control systems, as Subjects's attributes must be disclosed to the Resource Owner or to the other entity which is in charge of evaluating the Access Control policy.

5. Privacy preserving decentralized policy evaluation

In this section we outline our main contribution, i.e., an SSI and blockchain based Access Control system that employs zero-knowledge proofs to perform the policy evaluation without disclosing selected users' attributes (typically the ones representing personal information), that we call private attributes, while maintaining the policy evaluation transparent.

In Section 5.1 we provide the problem definition, presenting the entities involved, our assumptions, and the goals of the proposed system, while in Section 5.2 we present the architecture we defined for achieving such goals under the provided assumptions.

5.1. Problem definition

To model the problem of preserving attribute privacy in the evaluation of SmartPolicies, we first identify the actors involved and how they interact each other. Later on, we define the structure of the attributes that are used to evaluate access requests, and the metadata required to evaluate and check the non-revocation of the attributes on the blockchain without disclosing their values. Finally, we describe the trust assumptions under which our proposal works.

5.1.1. Actors involved

As we propose an Access Control system implementing the XACML standard based on Di Francesco Maesa et al. (2019), we do inherit the actors presented in Section 4, although the roles of some of them have been extended to deal with private attributes. For instance, in addition to Di Francesco Maesa et al. (2019), the Attribute Managers in charge of managing private attributes do not store their values on the blockchain. Instead, they behave like the Issuers in the SSI paradigm, i.e., they assign the attributes to the Subjects in form of Verifiable Credentials. Therefore, the Subjects have full control and responsibility on the usage of their private attributes embedded in the Verifiable Credentials. In particular, when Subjects want to access a protected Resource, they inspect the related SmartPolicy to check which private attributes are involved in the conditions of the SmartPolicy, they choose which of them they want to submit for the SmartPolicy evaluation, and for such attributes they compute (on their premises) the related proofs to be embedded in the access requests for such Resource. On the other hand, the Resource Owner defines (and maintains) the Access Control policies that are translated in SmartPolicies and deployed on a blockchain, which are paired with the resources to be protected. Each SmartPolicy acts as a Verifier in the SSI paradigm, evaluating the proofs submitted by Subjects.

5.1.2. Private Attributes

As the goal of our proposal is to maintain the confidentiality of some users' attributes, the Private Attributes, we need to formally define what we mean by Private Attributes. We define first the following properties:

Property 5.1 (*Confidentiality*)**.** *The attribute value is known only to the* Subject *it refers to and to the* Attribute Manager *who issued it. No information about such value is disclosed to any other entity during and after policy evaluation, beside the one provided by the following Zero-Knowledge property.*

Property 5.2 (*Zero-Knowledge*)**.** *During and after any policy evaluation, the only information about the value of a private attribute that can be disclosed is the result of the policy conditions the attribute is part of.*

Property 5.3 (*Correctness*)**.** *The policy evaluation should achieve the same access decision as if the attribute value was not private. Do note that this should hold for any policy, including the one with just a single condition on that single attribute.*

Given the aforementioned properties we can formally define a Private Attribute:

Definition 5.1 (Private Attribute)**.** A Private Attribute is an attribute holding the *Confidentiality*, *Zero-Knowledge*, and *Correctness* properties, i.e., an attribute whose value should not be disclosed to any blockchain user (including the Resource Owner) when used for the evaluation of a SmartPolicy (*Confidentiality*), but rather its evaluation result (*Zero-Knowledge*), while providing consistency to the access decision process (*Correctness*).

The above definition guarantees that no information on the values of Private Attributes is disclosed beside the result of the condition evaluation on that attribute. At the same time, the *Zero-Knowledge* property guarantees that policies can still be evaluated on Private Attributes by third parties without knowing the actual attribute value, obtaining the same result as if they knew it. In other words, the evaluation of a condition involving the values of Private Attributes can be independently computed by a third party, without any additional information, and always obtain the same result.

All attributes that are not Private Attributes are named Public Attributes.

5.1.3. Trust assumptions

The proposed model works on the following trust assumptions. Subjects are not trusted (by any entity in the system) as they might attempt to claim unduly access to the resources, for example, by providing tampered values for their Private Attributes during a policy evaluation or by maliciously claiming a forged policy evaluation result. The Resource Owner is not trusted (by any one, except the Resource) as they might attempt to unduly deny or allow access to resources to Subjects. In either case, the policy decision would be tampered. Moreover, the Resource Owner might be interested in violating the confidentiality of Private Attributes.

Attribute Managers are, instead, considered trusted entities. We point out how the Attribute Managers are the entities who manage Private Attributes, meaning that they have full control over the values of said attributes, but not on their disclosure to third parties. Acting as an Issuer in SSI, an Attribute Manager is trusted by the Subjects who decide to request them the Verifiable Credentials embedding the current value of their attributes, and by the Resource Owners who decide to use such attributes as decision factors in the conditions of their SmartPolicies. In other words, if a given Attribute Manager is not trusted by some actors, such actors would not involve them in their operations, i.e., Subjects will not use attributes from such Attribute Manager in their access requests, while Resource Owners will not employ such attributes in their SmartPolicies. For example, the Internal Revenue Agency is considered to be trusted by most of the population and by all the public offices to provide the date of birth of a person, or a University is considered to be trusted by other educational facilities or by companies to issue educational certifications.

The Smart Contracts, including SmartPolicies, are considered trusted entities due to their nature. Any blockchain protocol is, in general, tasked with creating trust in a common state, including smart contracts execution, among mutually untrusted parties. This is why we deem smart contracts trusted *by design*, their execution is guaranteed to be correct by the distributed consensus algorithm behind the blockchain they are deployed on.

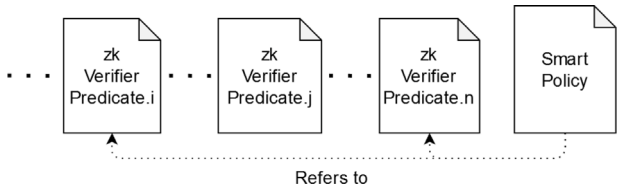


Fig. 5. Relationship between the SmartPolicy and the zkVerifiers.

Algorithm 1: An outline of ZoKrates program including the code to link the Subject to their proofs

```
def main(val: private field, nonce: private field, hash1: field, hash2: field):
    assert(/* Predicate on private field val */)
    field[2] h = sha256packed([0, 0, val, nonce])
    assert(hash1 == h[0])
    assert(hash2 == h[1])
    return true
```

5.2. Architecture

This section describes the architecture we designed to address the problem described in Section 5.1. It employs non interactive zero-knowledge proofs (see Section 2.4), encapsulated inside a blockchain transaction, which allows the implementation of a privacy preserving evaluation of SmartPolicies involving conditions on Private Attributes. In a traditional XACML based Access Control system, in order to evaluate a request to access a Resource, the PIP would query the Attribute Manager to collect the current attribute values to allow the PDP to evaluate the conditions in the policy rules, leaving no role for the Subject the attributes refer to. Following SSI, instead, the Attribute Manager provides the Subject with a Verifiable Credential, and the Subject, in turn, uses it to prove some property related to their attributes in order to satisfy the policy defined by the Resource Owner. The Attribute Manager stores metadata about Private Attributes, in form of hash values, which the PIP might fetch during the policy evaluation process. Not only the attribute value is never disclosed, but the evaluation burden of a condition is also shifted towards the Subjects. To achieve this, we first present how we propose to support Private Attributes through zero-knowledge proofs in Section 5.2.1. Section 5.2.2, instead, describes the need for metadata stored by the Attribute Manager. We then show in Section 5.2.3 how the SmartPolicies proposed in Di Francesco Maesa et al. (2019) can be extended to support this new model based on SSI compliant representation of attributes.

5.2.1. Zero-knowledge proofs for Private Attribute Predicates

In Section 2.4 we have shown that, in general, it is possible to create a succinct non interactive zero-knowledge proof to prove the result of a given boolean function over the value of a given attribute, without the need to disclose such value. ZoKrates is a tool implementing such zero-knowledge proofs on Ethereum. In this paper we have chosen to rely on ZoKrates to build zero-knowledge proofs that prove that some conditions over Private Attributes are true without breaking their confidentiality. ZoKrates allows to compute off-chain zkSNARK proofs concerning the results of a given condition, and to verify such proofs on-chain, thus making such result trusted. In our system, the proofs and the related on-chain verifications concern the evaluation of the conditions regarding Private Attributes that are present in the SmartPolicies, therefore obtaining trustworthy evaluation results.

Being a trusted entity in the system, each Attribute Manager is in charge of carrying out the trusted setup of ZoKrates, the process generating the zero-knowledge circuit, the proving key, the zkVerifiers related to the conditions over the attributes it manages, and deleting the toxic waste. Recall these steps are important to guarantee the integrity and correctness of the system because whoever generates the toxic waste has the power to craft fake proofs (but not to disclose the private value).

In order to avoid clash of names with XACML conditions, we define as a Private Attribute Predicate a combination (AND/OR) of logical or mathematical conditions, named Value Checks, involving a number of Private Attributes. A Private Attribute Predicate is defined by the Attribute Manager on one, or more, Private Attributes it manages. An example of Private Attribute Predicate defined by the Internal Revenue Agency could be *eligible for pension*, which consists of the conjunction of two Value Checks, the first on the age, which must be above a given threshold, and the second on the number of years the Subject paid the related contributions, where age and number of contribution years are Private Attributes. A Private Attribute Predicate is implemented as a zkVerifier and is deployed on the blockchain to enable Resource Owners to integrate such zero-knowledge checks in their SmartPolicies. Fig. 5 shows a SmartPolicy integrating the zkVerifiers implementing the predicates i and n. Finally, the Attribute Manager needs to advertise the proving keys and the *out* circuits returned by ZoKrates to the Subjects to allow them to compute the zero-knowledge proofs alongside with the Private Attributes' values. Such proofs will be sent, by the Subject, to the SmartPolicy for evaluation during an access request.

5.2.2. Private Attributes metadata

To fully trust the result returned by the zkVerifiers it is necessary to define a linking between Private Attribute values and the Subjects these attributes refer to. For instance, malicious Subjects could make up fake values for their Private Attributes instead of using the ones released by the Attribute Manager through Verifiable Credentials, in order to build fraudulent, but correctly verifiable, proofs with them. Another attack, known as replay attack, is possible because, due to the transparency of blockchain transactions, a malicious Subject could reuse proofs known to be valid but associated to attributes of other users. A further exploit consists of re-using proofs produced in the past but related to out-of-date values of the attributes of the Subject, i.e., revoked values from Verifiable Credentials that are no longer valid.

Hence, a tamper evident linking between the current Private Attribute values and the Subjects they refer to is needed to avoid such kinds of proof forging attacks. A simple and effective way in providing Subject identity linking involves hashing (see Yeh and Tan (2020)) and works as follows:

(a) Every time an Attribute Manager issues a value val_A for a given attribute $attr$ to a Subject A, it also chooses a random nonce $nonce_A$ and it computes the hash of the attribute value concatenated with the nonce, i.e., $h_A^{attr} = H(val_A, nonce_A)$, with $H()$ a chosen cryptographic hash function. The hash h_A^{attr} represents a public *metadata* linking the Subject A to the proofs involving $attr$ they will submit.

(b) Each nonce is privately shared with the corresponding Subject in the Verifiable Credential, while the hash metadata is publicly readable (i.e., kept in a public bulletin board, such as a map inside an Attribute Manager controlled smart contract called AMContract, see Section 4) alongside the Subject identifier and attribute identifier, i.e., the tuple $(A, attr, h_A^{attr})$;

(c) While verifying the correctness of a zero-knowledge proof about a value val_A of $attr$, it is also required to verify that val_A has been issued to and links to the Subject A, i.e., to check that $H(val_A, nonce_A)$ equals to h_A^{attr}.

The two checks listed in point (c) need to be performed together as an atomic operation, i.e., the private attributes check and the identity linking verifications cannot be performed separately, and therefore should be implemented by the same zkVerifier. Moreover, to increase

the security of the proposed solution (for instance, to prevent a Resource Owner from using the value of nonces to conduct pre-image attacks) the values of nonces are kept secret as well. As a consequence, the nonce becomes an additional private value of the Subject when computing the proof, and needs to be treated as such, i.e. not shared with the Resource Owners. It is, in general, necessary to introduce a randomization factor such as the nonce, especially for attributes whose values range within small intervals (e.g., year of birth), for whom brute force attacks on the hash would be viable. The management of nonces could be approached as follows.

The first approach is the simplest, and requires that the Attribute Manager generates a single nonce for each Subject. The unique nonce paired to a Subject will be used for hashing all the values of all the Private Attributes concerning such Subject, similarly to a personal key. This could even be extended to have an unique nonce per Subject over all the different Attribute Managers. The second approach requires a distinct nonce for each Private Attribute of each Subject. Hence, all the values taken by a given Private Attribute of a given Subject are hashed with the same nonce, while the values of two distinct Private Attributes, even if they belong to the same Subject, are hashed with two different nonces. The third approach, instead, requires that the Attribute Manager generates a new nonce each time the value of a Private Attribute changes, and that nonce will be used exclusively to hash that value. Consequently, adopting the third approach, each value of each Private Attribute of each Subject is hashed with a distinct nonce.

The most desirable approach depends on the privacy, security, and storage requirements of a deployed system. For instance, using a single nonce for all attributes of the same Subject on the same Attribute Manager (or even all Attribute Managers), would provide the *all-or-nothing non transferability* side effect (Camenisch and Lysyanskaya, 2001). In fact, let us consider the case *Alice* shares her secret nonce (and private key) to another Subject, say *Bob*, to allow him to submit a request on her behalf to a given resource *R*. This would not only allow *Bob* to access R pretending to be *Alice*, but it would also have the side effect (unwanted by Alice) of potentially allowing *Bob* to access all other resources *Alice* attribute values allow to access as well. Hence, this side effect would discourage the nonce (and key) sharing in the first place and, consequently, the adoption of a single nonce for all attributes would somehow positively affect the degree of confidence in the claimed identity of the Subject (which is called Level of Assurance in the eIDAS Regulation (EU) 910/2014, European Union (2014)). The reader interested on all-or-nothing non transferability can refer to Feulner et al. (2022) for additional insights.

Algorithm 1 shows an example of ZoKrates program implementing this approach using the *sha256Packed()* hashing function. The function takes a list of 4 *fields* of 128 bits each, and since a *field* cannot contain the whole 256 bits returned by the hash, the result is split into two *fields* each storing 128 bits of the hash. Therefore, using ZoKrates the hash metadata consists of a pair of values. Such code is then compiled to generate the *out* circuit, the proving key, and the zkVerifier to deploy on the blockchain.

Every time the SmartPolicy is evaluated, and before querying the zkVerifier, the hash metadata concerning the involved Private Attributes are automatically fetched from the Attribute Manager's AMContract using the Subject's address and the attribute name. A successful evaluation guarantees that the Subject invoking the SmartPolicy is actually the Subject associated to the hash metadata in the AMContract since the addresses coincide, and only a Subject with a private key generating that address can submit transactions from that address. To revoke, or modify, anPrivate Attribute value about a Subject, the Attribute Manager needs to remove, or update, the hash metadata from the AMContract. Therefore, at access request time the SmartPolicy might retrieve hash values that do not match anymore a proof built on outdated values, denying the request. The revocation of a credential is an open problem in SSI, and the chosen approach will be compared to alternative solutions in Section 8.3.

Summarizing, the proposed solution based on hash metadata brings the following benefits: it links a Private Attribute value to a Subject and prevents impersonification or replay attacks enabled by blockchain transparency; it allows a SmartPolicy, acting as SSI's Verifier, to check for the revocation of a Private Attribute value, which was originally issued off-chain via a Verifiable Credential. We note that this solution design is driven by the will of being as compatible as possible with the base architecture of the blockchain based access control system presented in Di Francesco Maesa et al. (2019).

5.2.3. Smart Policy integration

The previous sections explained how to support a privacy preserving and auditable evaluation of Private Attribute Predicates, i.e., conditions regarding Private Attributes. This section shows how we propose to expand the SmartPolicy model presented in Section 4 to support it.

The first step is to specify in the XACML policy which attributes are Private Attributes, and which Private Attribute Predicates (i.e., zkVerifiers) must be used on such attributes.

We have shown in Section 2.3 how an Access Control policy logic in the XACML model can be broken down into a set of rules, each consisting of a target and a set of conditions. The *target* of a rule is used as in Di Francesco Maesa et al. (2019), i.e., for expressing predicates on Public Attributes only, hence we do not focus on it in the following. Instead, *conditions* within the rules are used to express operations involving attributes that can be both Public Attributes and Private Attributes. To allow the declaration of Private Attributes within rule conditions, we introduce a minor extension of the XACML standard. In particular, we add an optional field, i.e., `Private=''true''`, in the `AttributeDesignator` XACML tag, which is the tag used to declare an attribute within a condition. Both public and private attributes declarations also contain the address of the related AMContracts. This address is specified using the existing (and optional) field `Issuer` of the `AttributeDesignator` XACML tag. Instead, to specify the name of the zkVerifiers that must be used for the privacy preserving evaluation of Private Attribute Predicates on such Private Attributes, we simply exploit the standard `Apply FunctionId` XACML tag. A new function name is defined for each existing zkVerifier, and this name includes the zkVerifier address, thus uniquely identifying it. Listing 1 shows an example of XACML condition exploiting the zkVerifier defined in Section 5.2.1. This example involves two Private Attributes, `urn:it:internalrevenueagency:age` and `urn:it:internalrevenueagency:contributionyears`, which are evaluated by zkVerifier represented by the function having Id `urn:it:internalrevenueagency:verifiers:eligibleforpension:0x1482aDFDC2A33983EE69F9F8e4F852c467688Ea0`. We expand this example in a complete XACML policy including Private Attributes and zkVerifiers in Section 6.

```
1  ...
2  <Condition>
3  ...
4  <Apply FunctionId="urn:it:internalrevenueagency:verifiers:
       eligibleforpension:0
       x1482aDFDC2A33983EE69F9F8e4F852c467688Ea0">
5    <AttributeDesignator AttributeId="urn:it:
         internalrevenueagency:attributes:age" Category="urn:
         oasis:names:tc:xacml:1.0:subject-category:access-
         subject" DataType="http://www.w3.org/2001/XMLSchema#
         integer" Issuer="0
         xd9145CCE52D386f254917e481eB44e9943F39138"
         MustBePresent="true" Private="true"></
         AttributeDesignator>
6   <AttributeDesignator AttributeId="urn:it:
       internalrevenueagency:contributionyears" Category="urn
       :oasis:names:tc:xacml:1.0:subject-category:access-
       subject" DataType="http://www.w3.org/2001/XMLSchema#
       integer" Issuer="0
       xd9145CCE52D386f254917e481eB44e9943F39138"
       MustBePresent="true" Private="true"></
       AttributeDesignator>
7  </Apply>
```

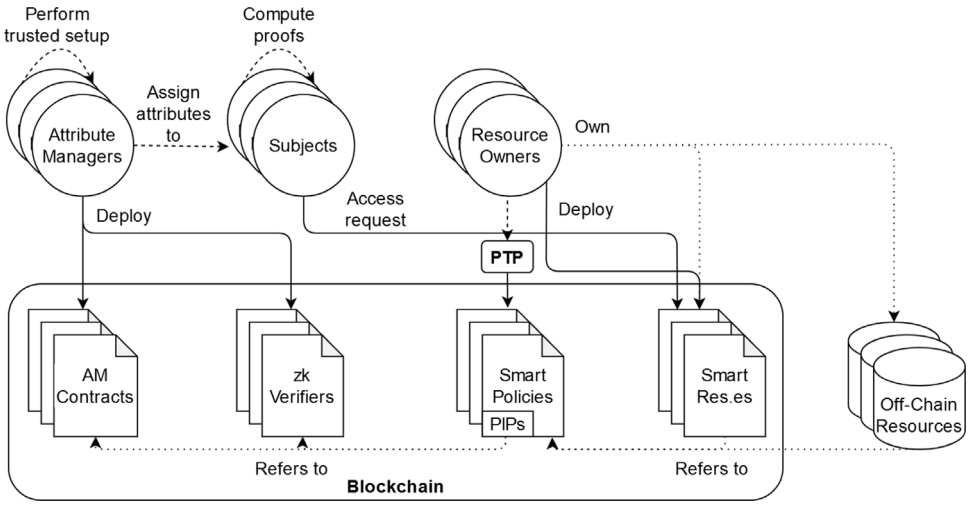


Fig. 6. Overall system architecture. Plain arrows represent on-chain operations, dashed arrows off-chain operations, and dotted arrows other kind of relationships.

```
  ...
</Condition>
  ...
```

Listing 1: Example of Private Attribute and zkVerifier usage in XACML policies.

The second step is performed by the Policy Translation Point (PTP), which takes the XACML policy as input and produces the corresponding SmartPolicy. For what concerns Public Attributes, the PTP works in the same way as described in Di Francesco Maesa et al. (2019). Instead, to support Private Attributes in a seamless manner, we exploit the modularity of the XACML reference architecture. More precisely, Private Attributes are managed in the SmartPolicy by treating the zkVerifiers as Attribute Managers, thus defining a customized type of Policy Information Points meant to interact with them. We recall that the aim of Policy Information Points in XACML is exactly the one to provide a standard interface towards different types of Attribute Managers, each having its own protocols. Hence, the code implementing such Policy Information Points in the SmartPolicy, upon receiving proofs as inputs of an access request, invokes the AMContract of the Private Attributes' issuers to fetch the hash metadata. The resulting system architecture is shown in Fig. 6, while Section 6 shows an example of SmartPolicy.

To summarize, the procedure performed by the SmartPolicy to evaluate an XACML condition, say C, involving Public Attributes and/or Private Attributes is outlined as follows:

- **Public Attribute:** for each Public Attribute in C, the SmartPolicy fetches its current value invoking the corresponding AMContract, and evaluates the boolean function in C involving such attribute (possibly using constants and other attribute values). We recall that Public Attribute values are fetched in plaintext from their Attribute Managers (thus being visible to all the blockchain users);
- **Private Attribute:** the SmartPolicy expects, as input from the Subject, a zero-knowledge proof for each Private Attribute Predicate in C. For each proof, the SmartPolicy retrieves from the AMContract deployed by the issuer of the Private Attribute the associated hash metadata value(s), and invokes the zkVerifier implementing such Private Attribute Predicate passing to it the proof and the hash metadata.

The results obtained from the evaluation of the boolean functions on Public Attributes and from the execution of zkVerifiers implementing the Private Attribute Predicates on the provided proofs are then combined through the logical operators present in C. The two types of attributes can be combined freely in conditions, as much as the available Private Attribute Predicates allow.

6. A reference example

This section presents a reference example to showcase our proposal. We consider a use case where the Resource to be protected is a SmartResource, i.e., a smart contract. Without loss of generality, we make this choice because all actors (e.g., Subjects, SmartPolicy, and Attribute Managers) as well as the Access Control System reside and take actions on the same chain. In our example, the Resource Owner is an educational program facility that assigns prizes, in the form of ERC20 tokens, to bachelor students with an average grade above a given threshold, which we suppose to be 27,[1] and being enrolled in the university for no more than three years. We consider the role of a student to be public information, hence being represented by the Public Attribute *subjectRole*, while the average grade and the current year of enrollment to be Private Attributes, namely *avgGrade* and *enrollmentYear* respectively. Moreover, we assume the Attribute Manager uses an unique nonce for all the Private Attributes of the same Subject. To get the token, each student invokes the *Student Prizes* smart contract which, in turn, invokes the SmartPolicy to determine the right to receive the prize.

Algorithm 2: The ZoKrates program generating the RegularlyEnrolledVerifier.

```
def main(enrollmentYear: private field, nonce: private field, hash1: field,
  hash2: field):
    assert(enrollmentYear > 0 && enrollmentYear ≤ 3)
    field[2] h = sha256packed([0, 0, enrollmentYear, nonce])
    assert(hash1 == h[0])
    assert(hash2 == h[1])
    return true
```

We assume the Attribute Manager, which is the university in our scenario, provides the following zkVerifiers for the Private Attributes: *(i)* the *RegularlyEnrolledVerifier*, generated by Algorithm 2, which checks the student is enrolled within the expected timeframe in the bachelor degree, i.e. checks the secret value *enrollmentYear* is less than "3"; *(ii)* the *AvgGradeGreaterOrEqVerifier*, generated by Algorithm 3, which checks the secret value *avgGrade* is greater or equal than an input variable *threshold* that can be decided by the Resource Owner, e.g., the

[1] We use the Italian grading system range, which goes from 18 (~6/10 or *D*) to 30 (~10/10 or *A*).

Algorithm 3: The ZoKrates program generating the AvgGrade-GreaterOrEqVerifier.

```
def main(avgGrade: private field, nonce: private field, hash1: field,
  hash2: field, threshold: field):
    assert(avgGrade ≥ threshold && avgGrade ≤ 30)
    field[2] h = sha256packed([0, 0, avgGrade, nonce])
    assert(hash1 == h[0])
    assert(hash2 == h[1])
    return true
```

educational program facility. The Attribute Manager deploys the zkVerifiers on the blockchain and advertises their addresses so that Resource Owners can use them in their XACML policies.

```
1  <Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
       PolicyId="2" RuleCombiningAlgId="urn:oasis:names:tc:xacml
      :1.0:rule-combining-algorithm:first-applicable" Version="
      1.0">
2   <Description>Prize policy</Description>
3   <Target></Target>
4   <Rule Effect="Permit" RuleId="prize">
5    <Target>
6     <AnyOf>
7      <AllOf>
8       <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:
            string-equal">
9        <AttributeValue DataType="http://www.w3.org/2001/
           XMLSchema#string">bachelor student</AttributeValue>
10       <AttributeDesignator AttributeId="urn:it:uniPisa:
           attributes:subjectRole" Category="urn:oasis:names:
           tc:xacml:1.0:subject-category:access-subject"
           DataType="http://www.w3.org/2001/XMLSchema#string"
           Issuer="0xd9145CCE52D386f254917e481eB44e9943F39138"
            MustBePresent="true"></AttributeDesignator>
11      </Match>
12     </AllOf>
13    </AnyOf>
14   </Target>
15   <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:
        and">
16    <Apply FunctionId="urn:it:uniPisa:verifiers:
         AvgGradeGreaterOrEqVerifier:0
         xd8b934580fcE35a11B58C6D73aDeE468a2833fa8">
17     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#
          integer">27</AttributeValue>
18     <AttributeDesignator AttributeId="urn:it:uniPisa:attributes
          :avgGrade" Category="urn:oasis:names:tc:xacml:1.0:
          subject-category:access-subject" DataType="http://www.
          w3.org/2001/XMLSchema#integer" Issuer="0
          xd9145CCE52D386f254917e481eB44e9943F39138"
          MustBePresent="true" Private="true"></
          AttributeDesignator>
19    </Apply>
20    <Apply FunctionId="urn:it:uniPisa:verifiers:
         RegularlyEnrolledVerifier:0
         xf8e81D47203A594245E36C48e151709F0C19fBeE">
21     <AttributeDesignator AttributeId="urn:it:uniPisa:attributes
          :enrollmentYear" Category="urn:oasis:names:tc:xacml
          :1.0:subject-category:access-subject" DataType="http
          ://www.w3.org/2001/XMLSchema#integer" Issuer="0
          xd9145CCE52D386f254917e481eB44e9943F39138"
          MustBePresent="true" Private="true"></
          AttributeDesignator>
22    </Apply>
23   </Condition>
24  </Rule>
25  <Rule Effect="deny" RuleId="denyRule"></Rule>
26 </Policy>
```

Listing 2: XACML policy of the reference example.

Listing 2 shows the XACML policy using the previously defined zkVerifiers. The policy has two rules. The first rule defines a *target* checking that the value of the Public Attribute *subjectRole* is equal to *"bachelor student"*, and a *condition* embedding the two zkVerifiers previously described. The first verifier, *AvgGradeGreaterOrEqVerifier*, is set at line 16 of the policy exploiting the `Apply FunctionId` XACML tag. Line 18 specifies the name of the attribute the zkVerifier refers to, *avgGrade*, and labels the attribute as private with the element `Private=''true''` of the `AttributeDesignator` XACML tag. Finally, line 17 specifies the threshold parameter about the grade, which is 27, to be passed to the zkVerifier. Similarly, the second zkVerifier in the condition, *RegularlyEnrolledVerifier*, is specified in line 20, while the related Private Attribute, *enrollmentYear*, is specified in line 21. The address of the Attribute Manager is specified in the field *Issuer* of the `AttributeDesignator` XACML tag, i.e., in lines 10, 18, and 21 of the XACML policy. The second XACML rule of the policy (line 25) is a default-deny rule that only returns DENY. The two rules are combined by the `first-applicable` combining algorithm that returns the result of the first applicable rule.

Algorithm 4: The pseudocode of the example Smart Policy

```
constructor():
    AM = AMContract(0xd91...138)
    V^geq = AvgGradeGreaterOrEqVerifier(0xd8b...fa8)
    V^eY = RegularlyEnrolledVerifier(0xf8e...BeE)
    threshold_avgGrade = 27
    targetRole ="bachelor student"

function evaluateTarget_prize(subject: address):
    role = AM.getPublicAttributeOf(subject,"subjectRole")
    if(role == targetRole) return true
    return false

function evaluateCondition_prize_AvgGradeGreaterOrEq(proof^grade: proof,
  subject: address):
    h^grade = AM.getMetadataOf(subject,"avgGrade")
    if(V^geq.verifyTx(proof^grade, h^grade, threshold_avgGrade)) return true
    return false

function evaluateCondition_prize_RegularlyEnrolled(proof^eY: proof,
  subject: address):
    h^eY = AM.getMetadataOf(subject,"enrollmentYear")
    if(V^eY.verifyTx(proof^eY, h^eY)) return true
    return false

function evaluateRule_prize([proof^grade, proof^eY]: proof[], subject:
  address):
    if(!evaluateTarget_prize(subject)) return NOTAPPLICABLE
    bool b1 =
      evaluateCondition_prize_AvgGradeGreaterOrEq(proof^grade, subject)
    bool b2 = evaluateCondition_prize_RegularlyEnrolled(proof^eY,
      subject)
    if(b1 ∧ b2) return PERMIT
    else return NOTAPPLICABLE

function evaluateRule_denyRule():
    return DENY

function first_applicable(evaluations: RuleReturn[]):
    for(eval in evaluations)
      if(eval == DENY) return false
      else if(eval == PERMIT) return true
    return false

function evaluate([proof^grade, proof^eY]: proof[]):
    RuleReturn prize = evaluateRule_prize([proof^grade, proof^eY],
      msg.sender)
    RuleReturn deny = evaluateRule_denyRule()
    return first_applicable([ prize, deny ])
```

The SmartPolicy derived from the XACML policy has a code similar to Algorithm 4, a Solidity-like code that shows the functions used to evaluate the XACML target, the conditions, the evaluation of the policy rules, the combining algorithm, and the *evaluate* function that triggers the policy evaluation. To improve the readability, the algorithms assign names to variables of the contract in the constructor, an action that the

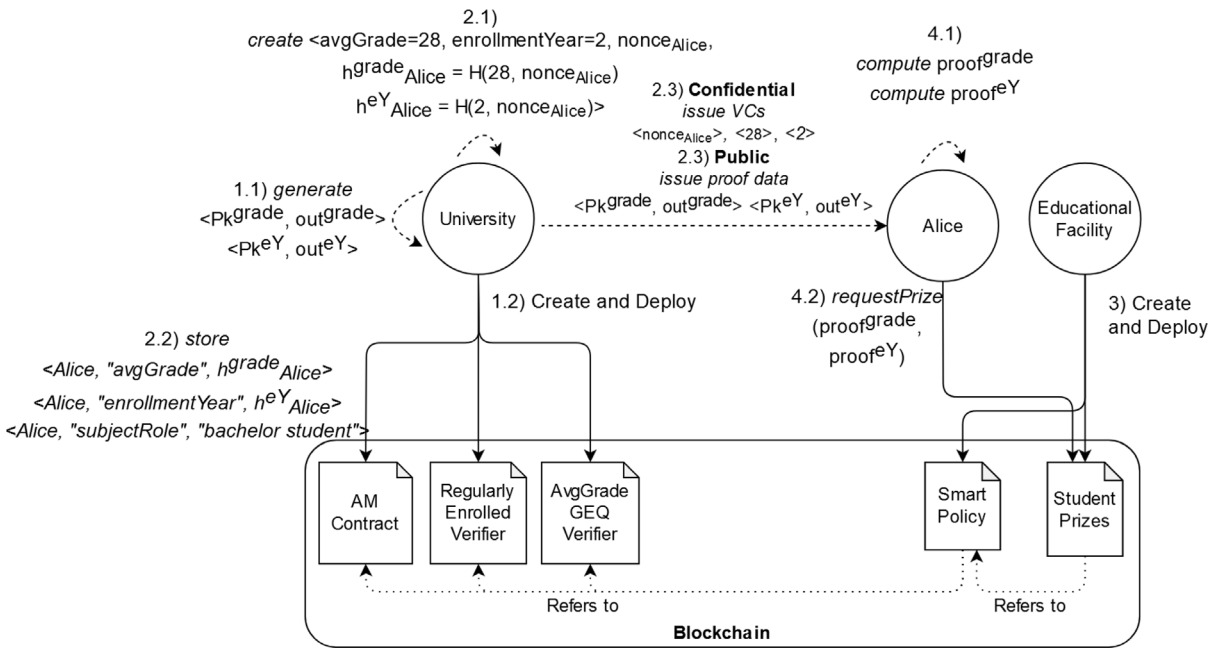


Fig. 7. Workflow overview.

Policy Translation Point may skip. An optimization procedure might be required to implement lazy evaluation of boolean expressions or to reduce the SmartPolicy code, e.g., the number of functions, to lower the deployment cost.

The operation workflow of the reference example follows the steps shown in Fig. 7.

Step (1) Initialization

(Step 1.1) The Attribute Manager codes the ZoKrates programs, shown in Algorithms 3 and 2, for the verification of the Private Attributes $avgGrade$ and $enrollmentYear$, it generates the zkSNARK proving keys, Pk^{grade} and Pk^{eY}, and the circuits, out^{grade} and out^{eY}, respectively.

(Step 1.2) The Attribute Manager exports the ZoKrates verifier smart contracts *AvgGradeGreaterOrEqVerifier* and *RegularlyEnrolledVerifier* and deploys them on the blockchain to be used by SmartPolicies. In addition, the Attribute Manager deploys their AMContract, which manages the Public Attributes, i.e., the *subjectRole* of the Subjects and the hash metadata related of the Private Attributes.

Step (2) Issue credentials

(Step 2.1) The Attribute Manager manages three attributes of the Subject: *subjectRole*, *avgGrade*, and *enrollmentYear*. Assuming that, for a given Subject, say *Alice*, the value of the Public Attribute *subjectRole* is equal to "bachelor student", the value of the Private Attribute *avgGrade* is equal to 28, and the value of the Private Attribute *enrollmentYear* is equal to 2, the Attribute Manager generates a random nonce $nonce_{Alice}$ for *Alice*, computes the hash metadata of the values of her Private Attributes, namely: $h^{grade}_{Alice} = H(28, nonce_{Alice})$ and $h^{eY}_{Alice} = H(2, nonce_{Alice})$, where *nonce* is a unique and secret value paired to *Alice*.

(Step 2.2) Afterwards, the Attribute Manager stores the tuples (*Alice*, "avgGrade", h^{grade}_{Alice}), (*Alice*, "enrollmentYear", h^{eY}_{Alice}), and (*Alice*, "subjectRole", "bachelor student") its AMContract.

(Step 2.3) Finally, the Attribute Manager issues to *Alice* the Verifiable Credentials embedding the values of the Private Attributes, $avgGrade = 28$ and $enrollmentYear = 2$, as well as $nonce_{Alice}$ assuming this is the first communication between the twos, through a secure channel that provides confidentiality. To allow *Alice* to compute the proofs, the Attribute Manager provides her the proving keys and the circuits, data that can safely be public.

Step (3) Smart resource and policy deployment

The Resource Owner generates the SmartPolicy and deploys it to the blockchain and stores the policy's address in the *Student Prizes* contract, the SmartResource in this scenario.

Step (4) Prize request

(Step 4.1) *Alice*, whose attribute values satisfy the policy, wishes to claim a prize from the *Student Prizes* contract. To do so, *Alice* creates the zero-knowledge proof for each zkVerifier referred by the SmartPolicy. To create a proof, *Alice* utilizes the proving key and the circuit for each zkVerifier she wants to challenge. The witness to generate the proof consists in the Private Attribute values, the *nonce*, and the hash metadata. *Alice* computes, on her premise, the proofs $proof^{grade}$ and $proof^{eY}$.

(Step 4.2) Finally, *Alice* submits the proofs to the *Student Prizes* contract that, in turn, invokes the *evaluate* function of the SmartPolicy. As shown in Algorithm 4, the *evaluate* function of the SmartPolicy invokes the internal functions implementing the two rules, namely *evaluateRule_prize* and *evaluateRule_denyRule*. The *evaluateRule_prize* function, at first invokes another internal function, *evaluateTarget_prize*, which implements the evaluation of the policy target retrieving from AMContract the current value of the *subjectRole* attribute and compares such value with the required one, i.e., "bachelor student". If the target is satisfied, the SmartPolicy invokes the functions implementing the condition section, namely *evaluateCondition_prize_AvgGradeGreaterOrEq* and *evaluateCondition_prize_RegularlyEnrolled*. Since those Value Checks concern Private Attributes, the two functions at first retrieve from AMContract the metadata, i.e., h^{grade}_{Alice} and h^{eY}_{Alice} respectively, and then each invokes the proper zkVerifier. Finally, *evaluate* combines the results of the rules through the combining algorithm chosen in the XACML policy in order to compute the final decision.

To interact with the blockchain infrastructure Alice should use crypto wallet applications conveniently upgraded to support ZoKrates-like functionalities to create the proofs, which should be carefully designed to not complicate furthermore such applications. We point to papers such as Voskobojnikov et al. (2021) for more information about usability issues of wallets.

6.1. Prototype

We developed a prototype on Ethereum given its leading market position. Nonetheless, the overall architecture and design of the solution is applicable to other EVM-based blockchains, such as Polygon chains (Polygon Technology, 2022), or other Turing-complete blockchains as well, such as EOS.IO-based (EOSIO, 2021). The considered SmartPolicy, shown in Algorithm 4, is implemented as a Solidity smart contract. The two zkVerifiers, *RegularlyEnrolledVerifier* and *AvgGradeGreaterOrEqVerifier*, are generated from the ZoKrates programs, version 0.7.7, shown in Algorithms 2 and 3.

The input of a ZoKrates program consists of private inputs, labeled with *private*, and public inputs, which have no label. The ZoKrates private inputs are the Private Attribute value and the Subject's nonce, while the public inputs are the hash metadata, divided into *hash1* and *hash2* due to ZoKrates types, and, optionally, threshold values (e.g., grade of 27 for *AvgGradeGreaterOrEqVerifier*). The ZoKrates public inputs are required by the *verifyTx* function exposed by the ZoKrates verifiers, while the private inputs are embedded in the zero-knowledge proof. The SmartPolicy invokes the *verifyTx* of each zkVerifier it refers to.

```
payload = {
    "sub": subject.publicKey,
    "vc": {
        "@context": ["https://www.w3.org/2018/credentials/v1"],
        "type": ["VerifiableCredential"],
        "credentialSubject": {
            "private": {
                "attribute_value": 28,
                "nonce": 123...567
            },
            "public": {
                "hash1": 123...4,
                "hash2": 678...9
            }
        }
    }
}

options = {
    "header": {
        "typ": "JWT",
        "alg": "ES256K"
    }
}

jwt = await createVerifiableCredentialJwt(payload,
      attributeManager.privateKey, options)
```

Listing 3: Structure of the payload of a Verifiable Credential sent by the Attribute Manager to the Subject.

The Verifiable Credentials embedding the Private Attribute values sent by the Attribute Manager to the Subject are implemented with the did-jwt-vc library (Decentralized Identity Foundation, 2022). The payload of a credential contains the Private Attribute values and the nonce of the Subject and it is sent as a JSON Web Token (JWT), the data structure used by the library to send credentials over the internet. Listing 3 shows the code snippet creating the data *payload* of the credential, which contains the attributes and metadata about the average grade of the subject, the *options* header used for the creation and verification of the JWT. The payload of the credential contains, in the "credentialSubject" field, the data sent by the Attribute Manager separated between private data, e.g., the Private Attribute value, 28 in the example, and public data, e.g., the hash metadata. All of this data will be used by the Subject to compute the zero-knowledge proof with ZoKrates. Note that the *AvgGradeGreaterOrEqVerifier* requires an extra public input that is the threshold grade to compare the average with, called *threshold* in Algorithm 3, and is required to build the proof. Since this value may change between SmartPolicies, it has to be exposed by the Resource Owner and is the responsibility of the Subject to retrieve it, therefore it is not present in the credential sent by the Attribute Manager. Moreover, the *payload* requires the Subject's DID, built from their Ethereum public key, in the "sub" field. Finally, the Attribute Manager builds the credential's JWT signing it with their private key and inserting an *options* parameters stating that the JWT is signed with ES256K signing algorithm.

7. Experimental evaluation

This section shows a set of measurements evaluating the feasibility of our proposal. We measure the main costs involving the SmartPolicy and the zkVerifiers as their complexity increases, and how these costs are distributed among the actors in the scenario.

7.1. Setup

The experimental evaluation presented in this section will focus on SmartPolicies implementing conditions over Private Attributes only, as those are the innovative contribution of this work. Focusing on Private Attributes, the complexity of a SmartPolicy mainly depends on the Value Checks it implements (both their number and individual complexity). This is why we use as approximated measure of complexity of a policy the number of Value Checks it contains. Value Checks are encapsulated inside the Private Attribute Predicates provided by Attribute Managers, and each Private Attribute Predicate can include one or more Value Checks, as explained in Section 5.2.1. A single policy may involve one or more Private Attribute Predicates. To avoid introducing assumptions on the Private Attribute Predicates structure we have chosen to consider the two possible extremes, i.e., policies with only Private Attribute Predicates containing a single Value Check, and policies with a single Private Attribute Predicate containing all needed Value Checks. We name this two scenarios **Modular** and **Monolithic** respectively.

The reference XACML policy shown in Listing 2 follows a purely Modular approach, as the *condition* section includes the invocation of two zkVerifiers (lines 16 and 20) each implementing a Private Attribute Predicate performing one Value Check only. The results of the evaluation of the two zkVerifiers are combined in the XACML policy applying the chosen XACML function, i.e., `urn:oasis:names:tc:xacml:1.0:function:and` (line 15), to obtain the overall result. If the policy would have wanted to follow the Monolithic approach, instead, it should have contained a single condition calling a single zkVerifier implementing a Private Attribute Predicate including both the Value Checks on the *avgGrade* and *enrollmentYear* Private Attributes. In such a case, the Private Attribute Predicate would also internally have to combine the results obtained from the two Value Checks with the intended logical operator, i.e., the AND operator in our policy example. We remark how zkVerifiers are managed by Attribute Managers, and so they alone choose what Private Attribute Predicates are supported for their data. This means that the above example could have followed a Monolithic approach only if the needed Private Attribute Predicate was already provided by the considered Attribute Manager.

In our experiments we have tested SmartPolicies with an increasing number of Value Checks (from 1 to 70) for both approaches, using a different zkVerifier for each Value Check in the Modular scenario and a single zkVerifier containing all Value Checks for the Monolithic one. The considered Value Checks perform the comparison of a private value against a target number similarly to the *AvgGradeGreaterOrEqVerifier* of the reference example shown in Section 6. To avoid short circuiting savings, all the proofs used in the experiments are produced using attribute values that satisfy all the zkVerifiers, i.e., all zkVerifier invocations return true. Note that the maximum number of Value Checks in our experiments is set to 70 because, above that, zkVerifiers in the Monolithic model could not be deployed due to the code size exceeding 24 kB, which is a hard limit set by the current Ethereum protocol (Ethereum, 2016). All test scripts can be found on the project GitHub repository.[2]

[2] https://github.com/0Alic/zkABAC.

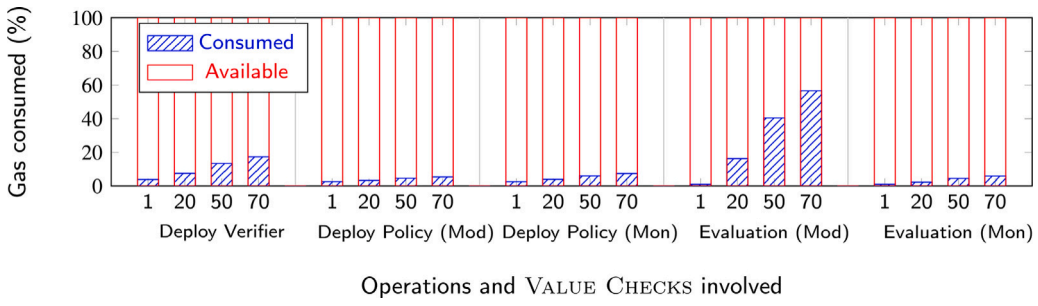


Fig. 8. Consumption compared to an Ethereum block. (Mod) = Modular, (Mon) = Monolithic.

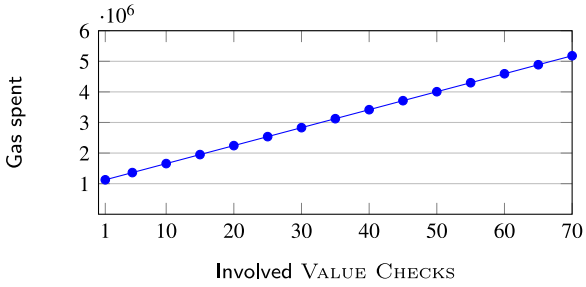


Fig. 9. Costs to deploy a zkVerifier.

7.2. Measurements performed and results

In the Ethereum protocol the "cost" of smart contracts is measured in units of gas, i.e., an approximation of the number of OPCODES invoked during any distributed code execution or deployment (Wood, 2017). The main operations we consider are deployment of the zkVerifiers, deployment of the SmartPolicy, and evaluation of such SmartPolicy. We present in Fig. 8 the relative cost of our main operations with respect to the maximum size of an Ethereum block (30M units of gas at the time of writing).

7.2.1. zkVerifiers deployment

The zkVerifier deployment cost is sustained by the Attribute Manager. Fig. 9 shows the cost of deploying a ZoKrates generated verifier smart contract varying the number of Value Checks performed. The deployment cost is clearly linear in the number of Value Checks, starting from 1.1M units of gas for the deployment of the smaller zkVerifier performing a single Value Check (approximately 3.75% of the block capacity, see Fig. 8), and increasing by approximately 60K units of gas for each additional Value Check. We note that, in our experiments, the ZoKrates programs implementing the zkVerifiers use the same nonce to hash all the Private Attribute values related to the same Subject.

Considering a SmartPolicy with n Value Checks, following the Monolithic approach would mean paying the cost shown in Fig. 9 for $x = n$. Following the Modular approach would instead require to deploy n zkVerifiers with a single Value Check, paying n times the cost associated to $x = 1$ in the figure. If we name the function depicted in Fig. 9 as $cost(x)$, then $cost_{Monolithic}(n) = cost(n)$, while $cost_{Modular}(n) = n * cost(1)$. If we use the aforementioned observation that each additional Value Check increases the deployment cost of approximately 60K units of gas, we can approximate $cost(n) \approx cost(1)+60k * (n-1)$. Since $cost(1) = 1.1M$ then always holds that $cost_{Modular}(n) = n * cost(1) = n * 1.1M = 1.1M + 1.1M * (n-1) > 1.1M + 60k * (n-1) = cost(1) + 60k * (n-1) = cost_{Monolithic}(n)\ \forall n > 1$ and so the cost of zkVerifiers deployment in the Monolithic approach will always be lower (or equal in case $n = 1$) than the Modular approach. This is intuitively correct as the cost of deploying a single slightly more complex contract, monolithic approach, is expected to be lower than the overhead of deploying many individual contracts of the Modular approach.

7.2.2. Smart Policy deployment

The deployment cost of a SmartPolicy is paid by the Resource Owner, and it is depicted in Fig. 10(a) for the two considered approaches.

Following the Modular approach, the number of Value Checks is the same as the number of zkVerifiers invocations. We observe that the deployment of a SmartPolicy referring to a single zkVerifier costs about 750K units of gas (2.52% of a block as shown in Fig. 8), and the cost increases fairly linearly of about 12K units of gas on average for each additional Value Check, which requires an additional invocation of the corresponding zkVerifier in the SmartPolicy. For example, the cost of a SmartPolicy referring to 50 Value Checks takes about 1.3M units of gas to deploy (4.53% of a block).

Following the Monolithic approach, there is a single zkVerifier V referred by the XACML policy, and so the related SmartPolicy includes a single corresponding function *evaluateCondition_V*. Despite that, the cost of deployment increases faster in the Monolithic model compared to the Modular one as shown in Fig. 10(a).

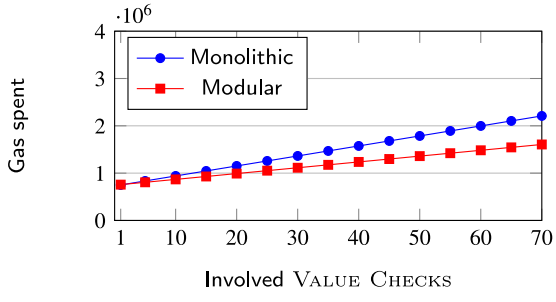
We note that a SmartPolicy deployment cost is lower than a zkVerifier deployment involving the same number of Value Checks. This is caused by the relatively simpler operations contained in a SmartPolicy compared to the heavy cryptographic operations carried on by zkVerifiers. As such, the bytecode size (that mainly influences deployment costs) is lower for SmartPolicies. This also means that, differently from the zkVerifiers, policies referring to more than 70 Value Checks (through multiple zkVerifiers) could be deployed. However we stopped at 70 to keep the results consistent.

7.2.3. Smart Policy evaluation

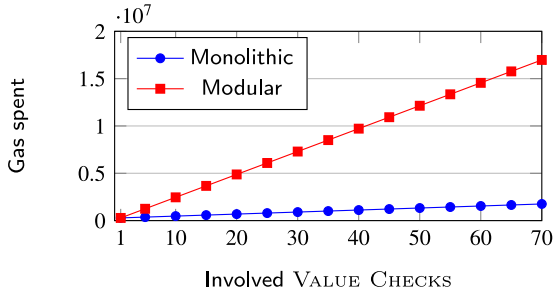
The SmartPolicy evaluation cost is charged to the Subject who requests the access and it is shown in Fig. 10(b) depending on the number of Value Checks performed by the policy.

First, we note that the cost of the Modular approach is always greater than the cost of the Monolithic one. In both cases the cost to evaluate a policy including one Value Check costs about 280K units of gas (1% of a block as shown in Fig. 8). For every additional Value Check in the policy, the Modular approach takes about additional 240k units of gas, while the Monolithic approach takes about 22K units of gas. Indeed, it costs about 12M units of gas (40% of a block) and 1.3M units of gas (4.49% of a block) to evaluate a policy containing 50 Value Checks with the Modular and Monolithic approaches, respectively. The motivation of the difference of costs is clear and is caused by the execution of multiple zkVerifiers, one for each Value Check, in the Modular model. This behavior raises the cost of the Modular approach by a reasonable margin when the SmartPolicy involves many Value Checks.

We remark that each zkVerifier called in the SmartPolicy would require a proof to be evaluated that has to be created and passed as parameter by the subject submitting the access request. As such, another advantage of the Monolithic model over the Modular one is that a single proof needs to be passed. We note that proofs have constant size, about 700 bytes, independently from the complexity of the zkVerifier. However, to generate proofs users need to know the

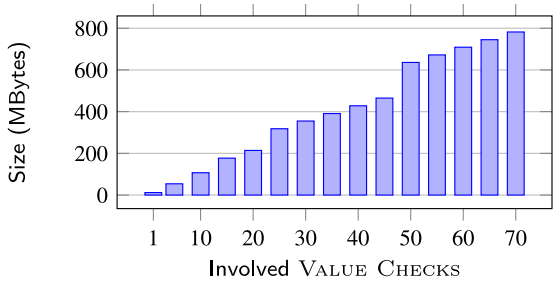


(a) SmartPolicy deployment cost.

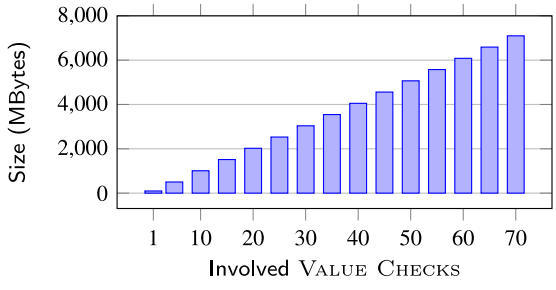


(b) SmartPolicy evaluation cost.

Fig. 10. Costs to deploy and evaluate a SmartPolicy.



(a) Proving key size in MB.



(b) Circuit size in MB.

Fig. 11. Size of the proving key and circuit.

proving key and the circuit *out* generated by the Attribute Manager, whose size is not constant. Hence, we show in Fig. 11 their sizes. A proving key associated to a zkVerifier with a single Value Check occupies about 11 MB while the circuit *out* about 97 MB of space. The same data takes about 636 MB and 5 GB when considering a zkVerifier with 50 Value Checks. Even if such data size may seem daunting, we remark this communication happens only once for each zkVerifier involved, and it is only needed to Subjects to generate proofs off-chain, while only resulting proofs are ever passed on chain, whose size is fixed.

7.2.4. Overall policy examples evaluation

To assess the costs of our proposal, we conclude the experimental evaluation with a breakdown of the costs, in terms of gas and fees, of our reference example and a more general realistic scenario. We compute the costs in both Ethereum's cryptocurrency as well as Polygon's for comparison given their similarity as both are EVM compatible. The Fee in cryptocurrency of a transaction in both protocols can be computed as $gas \times gasprice$ where $gasprice$ is the amount of cryptocurrency paid for each unit of gas. The following parameters have been taken into consideration to compute the fee in USD on the 10th of October 2022[3]:

- Ethereum: cryptocurrency ETH, exchange rate 1 ETH = 1302.72 USD, and average gas price $32 * 10^{-9}$ ETH per unit of gas;
- Polygon: cryptocurrency MATIC, exchange rate 1 MATIC = 0.83 USD, and average gas price $31 * 10^{-9}$ MATIC per unit of gas.

The reference example presented in Section 6 follows the Modular approach, and the policy involves 2 Value Checks. The greatest burden is borne by the Attribute Manager to deploy the two zkVerifiers costing 2.2M units of gas, which translates in about 91 USD on Ethereum or 0.06 USD on Polygon. Such a cost will be amortized in the long run, as the same two zkVerifiers will be referenced by a (possibly large) number of other SmartPolicies. The SmartPolicy deployment requires the Resource Owner to spend about 769K units of gas, i.e., about 32 USD in Ethereum or 0.02 USD in Polygon. The proof generation requires the Subject to retrieve from the Attribute Managers 22 MB of proving keys and 190 MB of circuits, and takes about 3 s per proof (6 s in total with 2 proofs) of computational time on a PC running Windows 10 equipped with an Intel Core i7-8750H (2.20 GHz) CPU and 16 GB or RAM memory. Finally, the cost that is paid by a Subject every time they need to evaluate the SmartPolicy to check their right to invoke the *Student Prizes* smart contract amounts to 530K units of gas, i.e., about 22 USD on Ethereum and 0.01 USD on Polygon.

Moving beyond our reference example, we can consider policies with 10 Value Checks as the reasonably bigger policies (concerning Private Attributes only) to expect in most real scenarios. Table 1 reports the corresponding deployment costs of zkVerifiers and deployment and evaluation costs of SmartPolicies, comparing the Modular and the Monolithic approaches. We observe that the only measure that is comparable is the SmartPolicy deployment cost, which is also, arguably, the less important as it is paid once for each new policy. Instead, for what concerns the deployment of zkVerifiers, the Monolithic approach is considerably cheaper than the Modular one. As a matter of fact, the cost for the deployment of 10 zkVerifiers following the Modular approach is about seven times the cost for deploying a single zkVerifier implementing the same Value Checks following the Monolithic approach. However, once deployed, those zkVerifiers will keep being used by other SmartPolicies in the Modular approach, and so the cost will be paid only once. In the Monolithic approach, instead the zkVerifier cost would have to be paid again for any new policy. An unequivocal advantage of the Monolithic approach over the Modular is in the five times lower evaluation cost of the SmartPolicy. This is especially relevant as a SmartPolicy is expected to be evaluated often during its life cycle.

The generation of the ten proofs for the Modular approach requires the Subject to retrieve from the Attribute Managers 110 MB of proving keys and 970 MB of circuits, and takes about 3 s per proof, 30 s in total, of computational time on the Subject's premise. In the Monolithic approach, instead, the Subject requires to retrieve from the Attribute Manager a single proving key of 107 MB and a single circuit of 1009 MB, and the generation of a proof requires about 34 s of computational time. This means that, from the point of view of the Subject, the two approaches are comparable concerning proof management.

[3] Sources: https://etherscan.io/ and https://polygonscan.com/.

Table 1
Costs related to a policy involving 10 Value Checks on Private Attributes.

Operation	Approach	Paid by	Gas	Fee (ETH)	Fee (MATIC)	USD (Ethereum)	USD (Polygon)
Deploy zkVerifier	Modular	Attribute Manager	11.2M	0.358	0.347	466.89	0.29
Deploy zkVerifier	Monolithic	Attribute Manager	1.6M	0.051	0.050	66.70	0.04
Deploy SmartPolicy	Modular	Resource Owner	867K	0.028	0.027	36.14	0.02
Deploy SmartPolicy	Monolithic	Resource Owner	940K	0.030	0.029	39.19	0.02
Evaluate SmartPolicy	Modular	Subject	2.4M	0.077	0.074	100.05	0.06
Evaluate SmartPolicy	Monolithic	Subject	474K	0.015	0.015	19.76	0.01

We have presented the costs considering two different protocols to show how the blockchain choice might influence the overall applicability depending on the considered use case, i.e. the value of resource whose access is to be protected. While the costs in USD are quite high in Ethereum (e.g., 22 USD to claim a prize for students in our reference example), the same costs are, at the time of writing, much lower in Polygon (0.01 USD to do the same). This is likely caused by the popularity and large adoption of Ethereum, reflected by the corresponding high cryptocurrency exchange rate.

8. Discussion

8.1. On the privacy of the system

We observe that the values of Private Attributes, despite their name, are not necessarily kept private. The *Zero-Knowledge* property guarantees that no additional information is disclosed about a Private Attribute value during a Private Attribute Predicate evaluation, *beside the evaluation result itself*, and in some cases the evaluation result alone may provide enough information to narrow down considerably the set of possible values for a Private Attribute.

For example, let us consider two Private Attributes: age, a numerical attribute representing the age of an employee, and $hasClearanceA$, a boolean attribute stating if the employee has reached the clearance level denoted A or not. Evaluating a policy allowing the access *if* $age \geq 40$ would disclose some information, by allowing an external observer to infer if the employee is over 40 years old. As such the actual privacy of the attribute value is decreased, as the possible set of values is shrunk. Evaluating a policy allowing the access *if* $hasClearanceA$ *is true* would, instead, disclose the attribute value entirely, as the attribute is binary.

This behavior is, in general, unavoidable in any Access Control system with publicly visible policy decisions. This is why, in such a model, the *Zero-Knowledge* property guarantees the best level of privacy achievable. Moreover, by employing an SSI model we allow users to retain control on which attribute values to potentially disclose. This is possible as users are the ones in charge of building and submitting proofs for policy evaluations. In the traditional model, instead, Attribute Managers interact directly with Resource Owners, granting all needed values (in plain text) upon request.

An important property protecting the privacy of a Subject in the long run is the unlinkability of the requests they submit. The proposed system is based on technologies, SSI, VCs, and blockchain, that only grant their users pseudonymity. Therefore, as long as the Subjects use the same identifiers, DIDs or blockchain addresses, when requesting access to different Resources, an analysis of the ledger easily links together requests of the same user. Consequently, unlinkability can only be granted if a Subject is careful enough to use different identifiers through their operations.

Finally, we remark that our proposal might leak information about Private Attribute values only upon successful proofs evaluation, i.e. only when the zkVerifiers return true. In other words no information is obtained by failed proofs, as they might fail for many reasons, not only because the values used do not satisfy the desired condition (e.g., they might be intentionally malformed or use values not matching the correct metadata).

8.2. On the structure of zkVerifiers

In Section 7 we have shown two different approaches to implement SmartPolicies: Modular (when each Value Check of the policy is contained in a distinct Private Attribute Predicate) and Monolithic (when all Value Checks of the policy are contained in the same unique Private Attribute Predicate). In this section we compare the two with respect to three main aspects: flexibility, privacy, and cost.

Flexibility

We remark that all zkVerifiers are deployed by Attribute Managers, as they are the sole entities in charge of deciding which Value Checks are supported on the Private Attributes they manage. In the Modular approach different existing Value Checks, one per Private Attribute Predicate, can be easily combined by a policy to express its logic. In the Monolithic one, a SmartPolicy can use Private Attributes only if there already exists a Private Attribute Predicate expressing the combination of Value Checks needed by the policy exactly. This means that, for any practical application, to support arbitrary Monolithic SmartPolicies the system should support the option for Attribute Managers to deploy custom zkVerifiers upon Resource Owners request. Not only the Monolithic approach complicates the entities interaction, but it also shifts the design of supported zkVerifiers from Attribute Managers towards Resource Owners. Attribute Managers still retain the ultimate decision on what zkVerifiers to actually deploy.

Moreover, having dedicated zkVerifiers reduces code reusability, increasing the probability of bugs and unwanted behaviors. As the Modular approach is based on a fixed set of Value Checks and corresponding zkVerifiers provided by any Attribute Manager, such set could be thoroughly audited before SmartPolicies creation.

Finally, another shortcoming of the Monolithic approach is that it might require a single zkVerifier regarding attributes managed by different Attribute Managers. This would require mutually, and possibly untrusted, cooperation between Attribute Managers, complicating the system even further. The Modular approach, instead, seamlessly supports the combination of zkVerifiers generated by different Attribute Managers. To conclude, for all this reasons the Modular approach is more flexible than the Monolithic one.

Privacy

Full modularity of zkVerifiers introduces an increased risk of disclosure of Private Attribute values. In the Modular approach, all Value Checks results are first obtained, through the evaluation of the corresponding zkVerifiers, and then combined within the SmartPolicy. This means that each single result becomes known to all. In the Monolithic approach, instead, only the combined final result is disclosed, potentially strengthening users privacy. For example, consider a policy that is the disjunction of two or more Value Checks. In the Modular approach they would all be computed by distinct zkVerifiers, and so all results would be disclosed. In the Monolithic approach, instead, only the final (i.e., already combined) result would be returned by the single zkVerifier, and hence disclosed. Assuming such result is true, any observer would only know that at least one of the Value Checks involved returned true, but not which or how many.

Let us consider the interaction between Attribute Managers and Resource Owners following the Monolithic approach as discussed in the previous **Flexibility** paragraph, i.e. supporting custom zkVerifiers creation by Attribute Managers upon Resource Owners suggestion. This

changes the role of Attribute Managers from an active one, of designing Private Attribute Predicates, to a passive one, of checking and approving the Private Attribute Predicates proposed by Resource Owners. Since Resource Owners are untrusted, they may request zkVerifiers structured to reveal the value of the Private Attributes despite being hidden behind zero-knowledge proofs. Therefore, Attribute Managers would no longer have the sole responsibility on deciding the level of privacy and flexibility of the operations allowed on Private Attribute values (balancing expressiveness and privacy to prevent misuse), but also on auditing the proposed Private Attribute Predicates privacy provisions. This might be exploited for a privacy disclosure attack as outlined in Section 8.4.

Cost

We have observed in Section 7 that the Modular approach costs more, in terms of blockchain fees, than the Monolithic one during policy evaluation, which is, supposedly, the most frequent operation performed. The opposite is true for deployment cost. Deployment cost is, in general, less relevant than execution cost, as it paid only once during setup, but this is only truly correct for the Modular approach. As explained above, all SmartPolicies following a Modular approach, would keep reusing the same set of zkVerifiers already deployed. Instead, Monolithic SmartPolicies would likely require a new custom zkVerifier to be deployed just for them. This means that the initial deployment cost of all zkVerifiers in the Modular approach is amortized through the lifetime of all SmartPolicies that use them. The cost of a Monolithic induced zkVerifier would, instead, only be amortized by all calls to that specific SmartPolicy. A counter argument is that Attribute Managers may deploy zkVerifiers that are later never used by SmartPolicies in the Modular approach, while this never happens in the on demand model of the Monolithic approach. In practice, however, in the Modular approach the number of zkVerifiers is constant and fixed at the beginning. Instead, in the Monolithic approach, the number of zkVerifiers is likely to be linear in the number of policies. All this implies that, since the number of policies is likely to increase over time, the relative zkVerifiers deployment cost of the Modular approach will get smaller and smaller, thus resulting much cheaper compared to the deployment cost of the policy dependent zkVerifiers of the Monolithic approach.

To evaluate how the heavier deployment costs of a zkVerifier of the Monolithic model compare to the heavier SmartPolicy execution costs of the Modular model, we consider the global runtime costs of policy evaluation for both models. In the Monolithic model we potentially need a new custom zkVerifier for each policy, so we take into account its original deployment cost. In the Modular model all zkVerifiers are deployed only once and then reused by all subsequent policies, so, if they are used by a high enough number of policies, their initial deployment cost is proportionally negligible for a single policy using them. This is why, in the experiment, we only consider the initial zkVerifier deployment cost for the Monolithic model. We show in Fig. 12 how the global policy verification cost would be depending on the number of policy evaluation performed during the policy life cycle, considering three different numbers of Value Checks in the policies. We only plot up to ten policy evaluations as the trend is already clear. We can see from the plot that, even for a relatively small number of Value Checks, greater than one, the less expensive evaluation cost in the Monolithic model greatly outperforms the Modular one despite the initial deployment costs in terms of overall gas costs. This is due to the fact that executing twice a verifier is way more costly than executing once a single verifier twice as complex.

We also note that what would be a single zkVerifier contract in the Monolithic model, would, instead, be divided among several zkVerifier contracts in the Modular model. This modularity helps in making the needed verifier contracts actually deployable, as a single big contract would require more consecutive free space in a single block (hence be more penalized by block congestion), as well as exceeding block limitations such as the block gas limit, or the maximum deployable code size enforced by the Ethereum protocol (Ethereum, 2016). On the other end, the Modular approach requires a more costly SmartPolicy evaluation transaction, risking to exceed the block gas limit at evaluation time, rather than deployment.

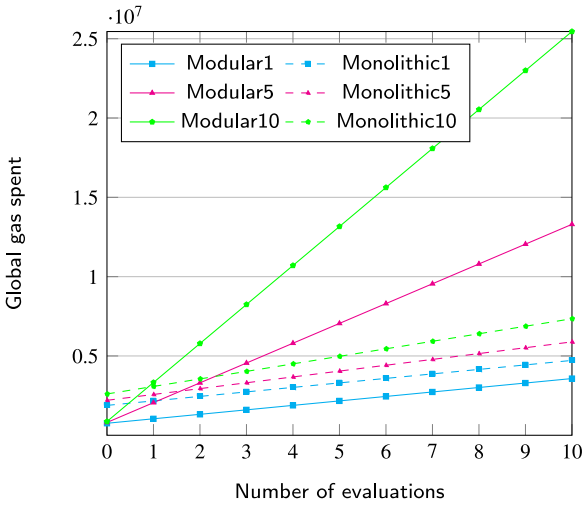


Fig. 12. Global cost of a SmartPolicy life cycle, including initial deployment cost and subsequent evaluation costs, for an increasing number of policy evaluations. For the Monolithic approach the initial deployment of the needed zkVerifier is counted as well. We consider both Modular and Monolithic models for 1, 5, and 10 Value Checks.

8.3. Credentials revocation

As introduced in Section 2.2, the revocation of a Verifiable Credential is still an open problem and the typical approaches involve either issuing credentials that include an expiration date or querying a revocation registry (Abraham et al., 2020). The former approach, requires less resources but it is less flexible: if the credential lifespan is too long, it is possible for the Subject to utilize the credential even if the embedded claims are not valid anymore; instead, if the lifespan is too short, the Subject needs to ask for new credentials too frequently. This paper exploits the latter approach, implemented with the hash metadata described in Section 5.2.2, which has been considered as more suitable in Access Control scenarios. Nevertheless, an expiration date check could be integrated in the system as follows: as an additional public parameter associated to an attribute stored in the AMContract and retrieved by the SmartPolicy; or as a Private Attribute and checked as such with a dedicated zkVerifier.

8.4. Security analysis

We end the discussion of the paper summarizing and analyzing the security of the system against some relevant attacks. In particular, in the following we focus solely on attacks related to the main contribution of this work, i.e., expanding SmartPolicies to support Private Attributes.

First of all, we recall that, considering the trust assumptions of our model defined in Section 5.1.3, the relevant untrusted entities are the Subject and Resource Owner. Assuming that such actors behave maliciously, in the following we describe a number of relevant attacks they could attempt and how our system protects against them.

- Attack 1: a malicious Subject S could try to obtain the access to a SmartPolicy protected Resource R by submitting to R a proof produced using a forged value val_S^f of their Private Attribute *pattr*, i.e., a value that is different from the one present in the VC sent to S by the Attribute Manager, val_S. This attack cannot succeed because, although S could easily produce a proof $P^{val_S^f}$ using the forged value val_S^f of *pattr*, this proof will not be validated by the Verifier because it will not match the hash metadata paired to *pattr* and S retrieved from the AMContract, which are related the to real value of such attribute i.e., val_S.

- Attack 2: a malicious Subject S could try to obtain the access to a SmartPolicy protected Resource R by submitting to R an old proof $P^{val_S^{old}}$ that was produced using a previously correct value val_S^{old} of their Private Attribute *pattr* which is now out-of-date. However, similarly to the previous case, this attack would fail. As a matter of fact, the Verifier would not validate $P^{val_S^{old}}$ since the hash metadata associated to the new value $P^{val_S^{new}}$ of *pattr* for S, which is retrieved from the AMContract, does not correspond to the (out-of-date) value used to produce $P^{val_S^{old}}$.
- Attack 3: a malicious Subject S could try to obtain the access to a SmartPolicy protected Resource R by submitting to R a valid proof P_Q computed by another Subject Q, and hence related to the value of the Private Attribute *pattr* associated to Q. This proof might have been retrieved from the blockchain exploiting a past interaction of Q with the same SmartResource. In this case as well, the attack would fail. As a matter of fact, the zkVerifier would not validate P_Q because the SmartPolicy will retrieve from the AMContract and submit to the zkVerifier the hash metadata associated to the address of S for the Private Attribute *pattr* which, obviously, was not the one used to compute P_Q.
- Attack 4: a malicious Resource Owner RO (or another blockchain participant) could try to discover the value val_S of a Private Attribute *pattr* issued to Subject S from the entries publicly available on the AMContract for that attribute. This attack is futile because such entries are the result of a cryptographic hash of the value concatenated to a secret nonce. Since the adopted hashing function is not invertible, and given that the value of the nonce is kept secret, the longer are the nonce and value, the smaller is the probability of guessing val_S through brute force attacks.
- Attack 5: a malicious Resource Owner RO could try to discover the value val_S of a Private Attribute *pattr* issued to Subject S by creating SmartPolicies which maliciously combine (if possible) the Value Checks implemented by the zkVerifiers. For instance, let us suppose that the Attribute Manager M defines a Private Attribute called *age*, representing the age of Subjects. Let us also suppose that M provides two zkVerifiers concerning this attribute, one implementing the Value Check $age \geq 18$, and the other implementing the Value Check $age \leq 18$. RO could define a SmartPolicy for one of their resources where both the previous zkVerifiers are evaluated. If this SmartPolicy returns PERMIT when invoked by S, we know that S is 18 years old, i.e., the value of the Private Attribute *age* has been disclosed. The same holds if we consider one single zkVerifier embedding both Value Checks. This attack can be mitigated if the Attribute Managers, for each Private Attribute, make available to Resource Owners a set of zkVerifiers which, even if all evaluated on the same Subject at the same time, do not disclose the value of the Private Attribute. In some cases, even restricting the effective range of possible values of a Private Attribute to a small enough set might result in a successful attack. In the previous example, to avoid the attack, M should not have made available both zkVerifiers, because their combination allows Resource Owners to disclose the age of Subjects exactly 18 years old.

9. Conclusions

In this paper we have presented a general XACML Attribute Based Access Control system that performs transparent policies evaluation while allowing for chosen attributes, that we called private attributes, to remain confidential. The proposed system is compliant with the XACML standard and employs a Self Sovereign Identity model putting subjects in control of private attributes management. To achieve this, our proposal is based on smart contracts enabled by blockchain technology and uses zero-knowledge Succint Non-interactive ARguments of Knowledge.

After showing that our proposal is the first to achieve such level of generality in the related literature, we have presented the SmartPolicy model we base our system upon. We have detailed the actors involved, the trust assumptions employed, the novel types of attributes we allow to support, and the architecture of our system, including a reference example. Before discussing more in depth three key points of our proposal at the end of the paper, we have presented our experimental evaluation of our working proof of concept implementation. The experimental results prove the feasibility of our approach in the currently most popular smart contacts enabling public blockchain, i.e., Ethereum.

9.1. Future work

The work can be expanded in both theoretical and experimental directions. From a theoretical point of view, different techniques could be considered to enable transparent policies evaluation over private attributes. The employed zero-knowledge proofs well satisfy our requirement but we have shown with the experimental results the considerable size of the off chain data they require. Other privacy preserving techniques could be considered to see if they offer comparable levels of compliance at a cheaper cost. Even by maintaining the same zero-knowledge proofs approach, the proposal could be enriched by further evaluating the Modular versus Monolithic models comparison outlined in Section 8.2. A potential idea to unify the two models could be to employ an automatic translation step from policies to verifiers, such as the Cinderella approach (Delignat-Lavaud et al., 2016), to translate any policy in an equivalent monolithic representation. This would ease the policy design burden on Resource Owners, by flattening policies and encoding the associated zkVerifiers automatically. Moreover, the preliminary security analysis presented in Section 8.4 could be expanded in a thorough formal analysis evaluating all potential security issues, including the need for a formal verification of the chosen underlying ZoKrates zero-knowledge engine.

From a practical point of view, the prototype implemented could be expanded to support other non EVM based blockchains as well as permissioned protocols. It would be especially interesting to evaluate our system in a permissioned scenario due to the lower costs involved.

Moreover, the preliminary proof of concept implementation presented in this paper could be improved, for example regarding the storage of the hash metadata. In our proposal the metadata hashes are stored in an explicit map maintained inside the AMContracts. Such map provides constant time retrievals but is also linear in space with the number of Private Attributes and Subjects. More advanced data structures could be employed to improve efficiency by saving precious, and expensive, on chain space. To do so, constant size structures, such as Merkle Tree roots or cryptographic accumulators could be employed. The trade off is that the verification step performed at policy evaluation time would be more complex. For example, checking that an element belongs to an accumulator is a much more costly operation than retrieving an element from a map. Similarly, employing Merkle trees would require Merkle paths to be provided when checking for elements inclusion. The trade off between space efficiency and execution complexity should be properly evaluated. Furthermore, we chose to rely on ZoKrates to manage zero-knowledge proofs, but more advanced (Grassi et al., 2021) or efficient (rapidsnark, 2022) tools could be adopted, instead, to improve the implemented proof of concept.

A more ambitious line of future research could further analyze the concept of "real privacy level" of the system informally discussed in Section 8.1. Arbitrary complex constraints on the potential disclosure level of attribute values might be put in place to guarantee the desired levels of privacy.

CRediT authorship contribution statement

Damiano Di Francesco Maesa: Conceptualization, Methodology, Writing – original draft, Writing – review & editing, Validation. **Andrea Lisi:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing, Software. **Paolo Mori:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing, Supervision. **Laura Ricci:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing, Project administration. **Gianluca Boschi:** Conceptualization, Methodology, Software.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The code and data is shared with a link to Github present in the paper.

References

Abraham, A., More, S., Rabensteiner, C., Hörandner, F., 2020. Revocable and offline-verifiable self-sovereign identities. In: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). pp. 1020–1027. http://dx.doi.org/10.1109/TrustCom50675.2020.00136.

Backes, M., Camenisch, J., Sommer, D., 2005. Anonymous yet accountable access control. In: Atluri, V., di Vimercati, S.D.C., Dingledine, R. (Eds.), Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, WPES 2005, Alexandria, VA, USA, November 7, 2005. ACM, pp. 40–46. http://dx.doi.org/10.1145/1102199.1102208.

Belchior, R., Putz, B., Pernul, G., Correia, M., Vasconcelos, A., Guerreiro, S., 2020. SSIBAC: Self-sovereign identity based access control. In: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). pp. 1935–1943. http://dx.doi.org/10.1109/TrustCom50675.2020.00264.

Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M., 2018. Scalable, transparent, and post-quantum secure computational integrity. Cryptol. ePrint Arch..

Bitansky, N., Canetti, R., Chiesa, A., Tromer, E., 2012. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. ITCS '12, Association for Computing Machinery, New York, NY, USA, pp. 326–349. http://dx.doi.org/10.1145/2090236.2090263.

Blum, M., De Santis, A., Micali, S., Persiano, G., 1991. Noninteractive zero-knowledge. SIAM J. Comput. 20 (6), 1084–1118. http://dx.doi.org/10.1137/0220068.

Camenisch, J., Lysyanskaya, A., 2001. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (Ed.), Advances in Cryptology — EUROCRYPT 2001. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 93–118. http://dx.doi.org/10.1007/3-540-44987-6_7.

Chadwick, D., Longley, D., Sporny, M., Terbu, O., Zagidulin, D., Zundel, B., 2019. Verifiable credentials implementation guidelines 1.0 (W3C working group note 24 september 2019). Online https://www.w3.org/TR/vc-imp-guide/ [Accessed on 30 November 2022].

Chase, M., Ganesh, C., Mohassel, P., 2016. Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials. In: Robshaw, M., Katz, J. (Eds.), Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III. In: Lecture Notes in Computer Science, vol. 9816, Springer, pp. 499–530. http://dx.doi.org/10.1007/978-3-662-53015-3_18.

De Salve, A., Lisi, A., Mori, P., Ricci, L., 2022. Selective disclosure in self-sovereign identity based on hashed values. In: 2022 IEEE Symposium on Computers and Communications. ISCC, pp. 1–8. http://dx.doi.org/10.1109/ISCC55528.2022.9913052.

Decentralized Identity Foundation, 2022. did-jwt-vc. https://github.com/decentralized-identity/did-jwt-vc [Online, accessed on 30 November 2022].

Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Parno, B., 2016. Cinderella: Turning shabby X.509 certificates into elegant anonymous credentials with the magic of verifiable computation. In: IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016. IEEE Computer Society, pp. 235–254. http://dx.doi.org/10.1109/SP.2016.22.

Di Francesco Maesa, D., Lunardelli, A., Mori, P., Ricci, L., 2019. Exploiting blockchain technology for attribute management in access control systems. In: Djemame, K., Altmann, J., Bañares, J.A., Agmon Ben-Yehuda, O., Naldi, M. (Eds.), Economics of Grids, Clouds, Systems, and Services. Springer International Publishing, Cham, pp. 3–14. http://dx.doi.org/10.1007/978-3-030-36027-6_1.

Di Francesco Maesa, D., Mori, P., Ricci, L., 2017. Blockchain based access control. In: Chen, L.Y., Reiser, H.P. (Eds.), Distributed Applications and Interoperable Systems. Springer International Publishing, Cham, pp. 206–220. http://dx.doi.org/10.1007/978-3-319-59665-5_15.

Di Francesco Maesa, D., Mori, P., Ricci, L., 2018. 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. http://dx.doi.org/10.1109/Cybermatics_2018.2018.00237.

Di Francesco Maesa, D., Mori, P., Ricci, L., 2019. A blockchain based approach for the definition of auditable access control systems. Comput. Secur. 84, 93–119. http://dx.doi.org/10.1016/j.cose.2019.03.016.

Ding, S., Cao, J., Li, C., Fan, K., Li, H., 2019. A novel attribute-based access control scheme using blockchain for IoT. IEEE Access 7, 38431–38441. http://dx.doi.org/10.1109/ACCESS.2019.2905846.

Eberhardt, J., Tai, S., 2017. On or off the blockchain? Insights on off-chaining computation and data. In: European Conference on Service-Oriented and Cloud Computing. Springer, pp. 3–15. http://dx.doi.org/10.1007/978-3-319-67262-5_1.

Eberhardt, J., Tai, S., 2018. Zokrates-scalable privacy-preserving off-chain computations. 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). IEEE, pp. 1084–1091. http://dx.doi.org/10.1109/Cybermatics_2018.2018.00199.

Enge, A., Satybaldy, A., Nowostawski, M., 2022. An offline mobile access control system based on self-sovereign identity standards. Comput. Netw. 219, 109434. http://dx.doi.org/10.1016/j.comnet.2022.109434, URL https://www.sciencedirect.com/science/article/pii/S1389128622004686.

EOSIO, 2021. EOS. https://github.com/EOSIO/eos [Online, accessed on 30 November 2022].

Ethereum, 2016. EIP-170: Contract code size limit. https://eips.ethereum.org/EIPS/eip-170 [Online, accessed on 30 November 2022].

EurLex, 2016. General data protection regulation. Online, https://eur-lex.europa.eu/eli/reg/2016/679/oj [Online, accessed on 30 November 2022].

European Commission, 2022. The digital services act package. https://digital-strategy.ec.europa.eu/en/policies/digital-services-act-package [Online, accessed on 30 November 2022].

European Union, 2014. Regulation (EU) no 910/2014 of the European parliament and of the council of 23 july 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing directive 1999/93/EC. Online: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32014R0910 [Accessed on 30 November 2022].

Extropy.IO on Medium, 2018. Zokrates tutorial with truffle. https://medium.com/extropy-io/zokrates-tutorial-with-truffle-41135a3fb754 [Online, accessed on 30 November 2022].

Feige, U., Fiat, A., Shamir, A., 1988. Zero-knowledge proofs of identity. J. Cryptol. 1 (2), 77–94. http://dx.doi.org/10.1007/BF02351717.

Ferdous, M.S., Chowdhury, F., Alassafi, M.O., 2019. In search of self-sovereign identity leveraging blockchain technology. IEEE Access 7, 103059–103079. http://dx.doi.org/10.1109/ACCESS.2019.2931173.

Feulner, S., Sedlmeir, J., Schlatt, V., Urbach, N., 2022. Exploring the use of self-sovereign identity for event ticketing systems. Electron. Mark. 1–19. http://dx.doi.org/10.1007/s12525-022-00573-9.

Fotiou, N., Siris, V.A., Polyzos, G.C., Kortesniemi, Y., Lagutin, D., 2022. Capabilities-based access control for IoT devices using verifiable credentials. In: 43rd IEEE Security and Privacy, SP Workshops 2022, San Francisco, CA, USA, May 22-26, 2022. IEEE, pp. 222–228. http://dx.doi.org/10.1109/SPW54247.2022.9833873.

Garrido, G.M., Sedlmeir, J., Babel, M., 2022. Towards verifiable differentially-private polling. In: ARES 2022: The 17th International Conference on Availability, Reliability and Security, Vienna,Austria, August 23 - 26, 2022. ACM, pp. 6:1–6:11. http://dx.doi.org/10.1145/3538969.3538992.

Goldwasser, S., Micali, S., Rackoff, C., 1989. The knowledge complexity of interactive proof systems. SIAM J. Comput. 18 (1), 186–208. http://dx.doi.org/10.1137/0218012.

Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., Schofnegger, M., 2021. Poseidon: A new hash function for Zero-Knowledge proof systems. In: 30th USENIX Security Symposium (USENIX Security 21). USENIX Association, pp. 519–535, URL https://www.usenix.org/conference/usenixsecurity21/presentation/grassi.

Hu, V.C., David, F., Rick, K., Adam, S., Sandlin, M., Karen, S., 2014a. Guide to attribute based access control (ABAC) definition and considerations. In: NIST Special Publication 800-162.

Hu, V.C., Ferraiolo, D., Kuhn, R., Friedman, A.R., Lang, A.J., Cogdell, M.M., Schnitzer, A., Sandlin, K., Miller, R., Scarfone, K., et al., 2014b. Guide to attribute based access control (ABAC) definition and considerations. NIST Spec. Publ. 800 (162).

Hyperledger, 2018a. Hyperledger Indy Credential Revocation. https://hyperledger-indy.readthedocs.io/projects/hipe/en/latest/text/0011-cred-revocation/README.html [Online, accessed on 30 November 2022].

Hyperledger, 2018b. Hyperledger Indy Documentation. https://hyperledger-indy.readthedocs.io/en/latest/ [Online, accessed on 30 November 2022].

Joinup, 2022. About SSI eIDAS bridge. Online: https://joinup.ec.europa.eu/collection/ssi-eidas-bridge/about [Accessed on 30 November 2022].

Li, Q., Xue, Z., 2020. A privacy-protecting authorization system based on blockchain and zk-SNARK. In: Tian, Z., Yin, L., Gu, Z. (Eds.), CIAT 2020: International Conference on Cyberspace Innovation of Advanced Technologies, Virtual Event / Guangzhou, China, December 5, 2020. ACM, pp. 439–444. http://dx.doi.org/10.1145/3444370.3444610.

Loffreto, D., 2012. The moxy tongue, what is sovereign source authority? https://www.moxytongue.com/2012/02/what-is-sovereign-source-authority.html [Online, accessed on 30 November 2022].

Monero community, 2022. Monero. Online: https://www.getmonero.org/ [Accessed on 30 November 2022].

Mukta, R., Martens, J., Paik, H.-y., Lu, Q., Kanhere, S.S., 2020. Blockchain-based verifiable credential sharing with selective disclosure. In: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). pp. 959–966. http://dx.doi.org/10.1109/TrustCom50675.2020.00128.

Nakamoto, S., 2008. Bitcoin: A peer-to-peer electronic cash system.

Ouaddah, A., Abou Elkalam, A., Ait Ouahman, A., 2016. FairAccess: a new blockchain-based access control framework for the internet of things. Secur. Commun. Netw. 9 (18), 5943–5964. http://dx.doi.org/10.1002/sec.1748.

Partala, J., Nguyen, T.H., Pirttikangas, S., 2020. Non-interactive zero-knowledge for blockchain: A survey. IEEE Access 8, 227945–227961. http://dx.doi.org/10.1109/ACCESS.2020.3046025.

Polygon Technology, 2022. Polygon documentation. Online, https://docs.polygon.technology/ [Online, accessed on 30 November 2022].

Poon, J., Dryja, T., 2016. The bitcoin lightning network: Scalable off-chain instant payments. Online https://www.bitcoinlightning.com/wp-content/uploads/2018/03/lightning-network-paper.pdf [Online, accessed on 30 November 2022].

Preukschat, A., Reed, D., 2021. Self-Sovereign Identity. Manning Publications.

rapidsnark, 2022. Rapidsnark repository. Online: https://github.com/iden3/rapidsnark [Accessed on 30 November 2022].

Rosenberg, M., White, J., Garman, C., Miers, I., 2022. Zk-creds: Flexible anonymous credentials from zksnarks and existing identity infrastructure. IACR Cryptol. ePrint Arch. 878, URL https://eprint.iacr.org/2022/878.

Rouhani, S., Deters, R., 2019. Blockchain based access control systems: State of the art and challenges. In: IEEE/WIC/ACM International Conference on Web Intelligence. pp. 423–428. http://dx.doi.org/10.1145/3350546.3352561.

Schanzenbach, M., Kilian, T., Schütte, J., Banse, C., 2019. ZKlaims: Privacy-preserving attribute-based credentials using non-interactive zero-knowledge techniques. In: Obaidat, M.S., Samarati, P. (Eds.), Proceedings of the 16th International Joint Conference on E-Business and Telecommunications, ICETE 2019 - Volume 2: SECRYPT, Prague, Czech Republic, July 26-28, 2019. SciTePress, pp. 325–332. http://dx.doi.org/10.5220/0007772903250332.

Sedlmeir, J., Smethurst, R., Rieger, A., Fridgen, G., 2021. Digital identities and verifiable credentials. Bus. Inf. Syst. Eng. 63 (5), 603–613. http://dx.doi.org/10.1007/s12599-021-00722-y.

Sharma, B., Halder, R., Singh, J., 2020. Blockchain-based interoperable healthcare using zero-knowledge proofs and proxy re-encryption. In: 2020 International Conference on COMmunication Systems NETworkS. COMSNETS, pp. 1–6. http://dx.doi.org/10.1109/COMSNETS48256.2020.9027413.

Song, L., Ju, X., Zhu, Z., Li, M., 2021. An access control model for the internet of things based on zero-knowledge token and blockchain. EURASIP J. Wireless Commun. Networking 2021 (1), 1–20. http://dx.doi.org/10.1186/s13638-021-01986-4.

Sonnino, A., Al-Bassam, M., Bano, S., Meiklejohn, S., Danezis, G., 2019. Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. In: 26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019. The Internet Society.

Sporny, M., Longley, D., Chadwick, D., 2022a. Verifiable credentials data model v1.1 (W3C proposed recommendation 03 march 2022). https://www.w3.org/TR/vc-data-model/ [Online, accessed on 30 November 2022].

Sporny, M., Longley, D., Sabadello, M., Reed, D., Steele, O., Allen, C., 2022b. Decentralized identifiers (DIDs) v1.0 (W3C proposed recommendation 19 july 2022). https://www.w3.org/TR/did-core/ [Online, accessed on 30 November 2022].

StarkNet, 2021. StarkNet documentation. Online: https://starknet.io/docs/ [Accessed on 30 November 2022].

The OASIS Technical Committee, 2013. eXtensible access control markup language (XACML) version 3.0 oasis standard.

uPort project, 2019. Revocation registry. https://github.com/uport-project/revocation-registry [Online, accessed on 31 May 2022].

Voskobojnikov, A., Wiese, O., Mehrabi Koushki, M., Roth, V., Beznosov, K.K., 2021. The u in crypto stands for usable: An empirical study of user experience with mobile cryptocurrency wallets. In: Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems. CHI '21, Association for Computing Machinery, New York, NY, USA, http://dx.doi.org/10.1145/3411764.3445407.

Web3, 2022. The Web3 foundation. https://web3.foundation/about/ [Online, accessed on 30 November 2022].

Wood, G., 2017. Ethereum: a secure decentralised generalised transaction ledger.

Yang, X., Li, W., 2020. A zero-knowledge-proof-based digital identity management scheme in blockchain. Comput. Secur. 99, 102050. http://dx.doi.org/10.1016/j.cose.2020.102050.

Yeh, R., Tan, M., 2020. STACK-X webinar — Introduction to zero-knowledge proof. https://www.developer.tech.gov.sg/communities/events/stack-x-meetups/past-webinars/zero-knowledge-proof-part-2.html [Online, accessed on 30 November 2022].

ZCash community, 2021. Zcash. Online: https://z.cash/ [Accessed on 30 November 2022].

ZCash community, 2021. Zcash: Parameter generation. Online: https://z.cash/technology/paramgen/ [Accessed on 30 November 2022].

Zhu, Y., Qin, Y., Zhou, Z., Song, X., Liu, G., Chu, W.C.-C., 2018. Digital asset management with distributed permission over blockchain and attribute-based access control. In: 2018 IEEE International Conference on Services Computing. SCC, pp. 193–200. http://dx.doi.org/10.1109/SCC.2018.00032.

Damiano Di Francesco Maesa is an Assistant professor at the department of computer science of the University of Pisa, Italy. His past positions include College Research Associate at Clare College of the university of Cambridge, Research Associate at the Innovation and Intellectual Property Management Laboratory at the University of Cambridge, Research Associate at the Computer Lab of the University of Cambridge, Research Associate at King's college London, and Research Assistant at the National Research Institute in Pisa, Italy. He has a Ph.D. in Computer Science from the University of Pisa and specializes in the analysis and study of novel applications of blockchain and Distributed Ledger Technologies (DLT). Beside his academic contributions, he has held several guest lectures, seminars, and workshops to spread awareness on blockchain technology.

Andrea Lisi received B.Sc. and M.Sc. degrees in computer science in 2016 and 2019 at the University of Pisa, Italy, department of computer science. Currently, he is a Ph.D. candidate in computer science at the University of Pisa. His current research interests are in the field of cryptocurrency and blockchain technology, in particular he focuses on layer-2 techniques to reduce the costs of an application while preserving the benefits of blockchains.

Paolo Mori received the M.Sc. degree in Computer Science from the University of Pisa and the Ph.D. from the same University. He is currently a senior researcher at IIT-CNR. His main research interests involve trust, security and privacy in distributed systems, focusing on access and usage control, data privacy aspect of (Decentralized) Online Social Networks, and Blockchain technology and its applications. He usually serves in the organization and Program Committees of international conference/workshops. He is (co-)author of 140+ papers published in international journals and conference/workshop proceedings. He is usually actively involved in research projects on information and communication security, such as the European Commission funded E-CORRIDOR, SIFIS-Home, SPARTA, C3ISP, NeCS, CoCoCloud, CONTRAIL, NESSOS, GridTrust, S3MS.

Laura Ricci received the M.Sc. degree in Computer Science from the University of Pisa and the Ph.D. from the same University. She is currently Full Professor at University of Pisa. Her research interests include blockchains and cryptocurrencies, distributed systems, peer-to-peer networks, and social network analysis. In these fields, she has co-authored 150+ papers published in international journals and conference/workshop proceedings. She has served as program committee member and chair of several conferences and has been the local coordinator of the H2020 European Project "Helios: A context aware Distributed Networking Framework". In 2019 she joined the group of experts responsible for defining the italian national strategy for blockchains.

Gianluca Boschi received his B.Sc.degree in Computer Science in 2020 with a thesis titled "zk-ABAC: an access control system enabling privacy with zero-knowledge proofs", and he is currently a Ms.C. student at the Cybersecurity degree at the University of Pisa.