Blockchain: Research and Applications 5 (2024) 100196

Contents lists available at ScienceDirect

Blockchain: Research and Applications

journal homepage: www.journals.elsevier.com/blockchain-research-and-applications

Research Article

How can the holder trust the verifier? A CP-ABPRE-based solution to control the access to claims in a Self-Sovereign-Identity scenario

Francesco Buccafurri [a,*], Vincenzo De Angelis [b], Roberto Nardone [c]

[a] University of Reggio Calabria, 89124 Reggio Calabria, Italy
[b] University of Calabria, 87036 Arcavacata di Rende (CS), Italy
[c] University of Naples "Parthenope", 80143 Naples, Italy

ARTICLE INFO

Keywords:
Blockchain
IPFS
Self Sovereign Identity
Smart contract
Access control

ABSTRACT

The interest in Self-Sovereign Identity (SSI) in research, industry, and governments is rapidly increasing. SSI is a paradigm where users hold their identity and credentials issued by authorized entities. SSI is revolutionizing the concept of digital identity and enabling the definition of a trust framework wherein a service provider (verifier) validates the claims presented by a user (holder) for accessing services. However, current SSI solutions primarily focus on the presentation and verification of claims, overlooking a dual aspect: ensuring that the verifier is authorized to access the holder's claims. Addressing this gap, this paper introduces an innovative SSI-based solution that integrates decentralized wallets with Ciphertext-Policy Attribute-Based Proxy Re-Encryption (CP-ABPRE). This combination effectively addresses the challenge of verifier authorization. Our solution, implemented on the Ethereum platform, enhances accountability by notarizing key operations through a smart contract. This paper also offers a prototype demonstrating the practicality of the proposed approach. Furthermore, it provides an extensive evaluation of the solution's performance, emphasizing its feasibility and efficiency in real-world applications.

1. Introduction

Self-Sovereign Identity (SSI) is an emerging paradigm in digital identity management, where users, rather than organizations, have complete control over their digital identity and personal data [1]. This paradigm empowers users to manage their identity independently, utilizing Verifiable Credentials (VCs) issued by trusted entities (*issuers*) and stored in personal digital *wallets*, usually stored locally. These VCs, when required, are presented as Verifiable Presentations (VPs) to a *verifier* for service access. For the sake of simplicity, this paper uses the terms VCs and VPs interchangeably.

To clarify this concept, consider a practical example in a car-sharing context:

Example 1.1. Alice needs to rent a car. The car rental company (verifier) must be sure that Alice has her driving license. Therefore, Alice must contact the Motor Vehicle Office (issuer) to obtain a driver's license attestation (VC) and provide it to the car rental company.

Most existing SSI solutions, as highlighted in the literature [2–4], mainly focus on the problem of the trust required of the user to obtain a service. They leverage blockchain technology to check the correctness of VCs, without compromising personal information. For example, in Ref. [5], the issuer stores on the blockchain only cryptographic proofs of the VCs, which do not contain personal information about the users. These proofs are used by the verifier to check, through cryptographic mechanisms, that the VCs presented by the user are valid and released by an authorized issuer.

However, existing solutions primarily focus on trust establishment for service access, overlooking a critical aspect: ensuring the verifier's right to access the user's VCs. Our paper addresses this dual issue by posing and answering a fundamental question: "How can the holder be sure that the verifier is authorized to access users' VCs?". In other words, we want to implement an innovative access control mechanism within the SSI framework, so that the verifier (which we often call the *service provider*) has to prove that it satisfies some requirements before accessing the VCs of the holder. We argue that this is a relevant problem in practice because VCs might contain very sensitive data (e.g., health

* Corresponding author.
E-mail address: bucca@unirc.it (F. Buccafurri).

https://doi.org/10.1016/j.bcra.2024.100196
Received 9 June 2023; Received in revised form 8 January 2024; Accepted 2 April 2024
Available online 4 April 2024

data). So the disclosure of such data should be controlled. Hence, the original contribution of this paper is to integrate an encryption-enforced access control mechanism into an SSI framework to complete it.

To clarify the main goal of this paper, let us consider the following example:

Example 1.2. Bob needs a cardiologist appointment at a given hospital. Bob should provide the hospital with his identity document along with some medical records previously released by different health facilities, describing his health status. Even though this information (apart from the identity document) does not represent a *credential* in the strict sense, it is needed to obtain the service. Therefore, it plays the role of the VC in an SSI scenario. While the standard problem is to allow its secure, privacy-preserving, and verifiable presentation, according to our approach, Bob's problem also includes being sure that the hospital owns an accredited cardiology ward before disclosing his data. Metaphorically, we can say that Bob would like to be sure that he does not open his wallet in front of a thief.

Looking at the previous example, it is rather clear that we cannot adopt any other access control model than Attribute-Based-Access-Control (ABAC) [6]. Indeed, access rights can only derive from the possession of some attributes, and further, we are in an open context in which subjects are not a priori known. As a matter of fact, the holder has no reason to allow or deny access to personal data to a specific service provider, but they are more interested in protecting their data by stating attribute-based policies that have to be satisfied to access such data.

An effective way to implement ABAC is Ciphertext-Policy Attribute-Based Encryption (CP-ABE) [7]. CP-ABE is a type of public-key encryption, in which a policy is associated with ciphertext and each user owns a private key associated with the attributes they own. If such attributes satisfy the policy, the user can decrypt the ciphertext with the private key. In our context, the data of the user (i.e., the VCs) can be encrypted under a given policy and the service providers have to prove to satisfy such a policy to access them. From the side of the specification of the policies, our proposal includes an additional abstraction layer, allowing the definition of policies with hierarchical attributes.

An issue that should be taken into account is that the user might own a large amount of data and might want to change the associated policies very frequently. In this case, the user cannot locally store the wallet in their device and has to rely on an external database (where VCs are stored in encrypted form). However, if the user wants to change the policy of a VC, they would have to download the VC, re-encrypt it under a new policy, and store it again in the database. This can result in an excessive effort on the client-side, especially if the user relies on resource-constrained devices in terms of storage and computational capabilities.

To solve this, we leverage an extension of CP-ABE, called Cipher-Policy Attribute-Based Proxy Re-Encryption (CP-ABPRE) [8], which allows a *semi-trusted* proxy, which we call *consensus provider*, to transform a ciphertext encrypted under a given policy into another ciphertext with a new policy without learning anything about the plaintext.

In our solution, the consensus provider acts as an intermediary in the communication between the user and the service provider by providing the latter (if authorized) with all the information needed to access the VCs. The benefit of the introduction of this intermediary is the reduction of the client-side effort without threatening the VCs' confidentiality.

A small price to pay is in terms of privacy since the consensus provider knows some metadata (but not the content) about the users' VCs and who is the service provider interacting with the user (but not the required service).

However, this price is compensated by the above benefit and another advantage in terms of accountability that our solution offers. Indeed, the introduction of the consensus provider allows us to notarize some critical operations and to disclose them to an agent authorized by the law in case of need [9,10]. To achieve this, we leverage the smart contracts offered by Ethereum [11], by referring to an integration into an SSI solution leveraging the Ethereum blockchain.

The contributions of this paper are as follows:

- We propose a user-centric access control mechanism in SSI, preventing unauthorized access to holder credentials.
- The solution integrates CP-ABPRE, Ethereum blockchain, and IPFS, adhering to W3C standards for credentials and Decentralized Identifiers (DIDs) [12,13].
- The proposal also offers a notarization mechanism based on smart contracts to obtain accountability for the relevant operations.
- We provide mechanisms to manage hierarchical attributes in the CP-ABPRE schemes and solve some practical problems such as the management of identities as attributes, the inclusion of file labels, and the setting of a validity time for the policies.

This work significantly extends our preliminary ideas presented in Ref. [14], offering a detailed architecture, a specific CP-ABPRE scheme and blockchain application, a prototype, and solutions to practical challenges not covered in the initial presentation. In this work, we adopt the Ethereum blockchain and the CP-ABPRE scheme [15] to define a concrete way to notarize critical operations through smart contracts and to develop a prototype of the solution. These aspects are not considered in the previous work, in which notarization is only mentioned at a high level, and no validation is provided. Furthermore, in this paper, the architecture is entirely revised by introducing new components and features. Finally, no improvement in solving practical problems of CP-ABPRE schemes is mentioned in the previous work.

The structure of this paper is as follows. Sections 2 and 3 review related literature and provide the necessary technical background. Section 4 discusses the motivation behind this work. Sections 5 and 6 delve into managing CP-ABPRE schemes' practical aspects. The proposed solution is then detailed in Section 7, with its security analyzed in Section 9. A prototype and performance evaluation are presented in Section 8. Finally, Section 10 ends the paper by drawing final remarks.

2. Related work

In this section, we explore the relevant literature on SSI-based solutions. Furthermore, we introduce the techniques of attribute-based encryption employed in this paper to implement an attribute-based access control mechanism.

2.1. Self-Sovereign Identity

In the European Union, the General Data Protection Regulation (GDPR) [16] addresses the management of personal data and aims to provide users with full control and rights over their data. The compliance of the SSI paradigm with GDPR [17,18] makes it very attractive for governments and business enterprises. In particular, some domains that will benefit from the SSI paradigm are the healthcare [19,20] and the industrial IoT [21].

An interesting survey on the components of an SSI architecture is provided in Ref. [22]. Currently, several implementations of SSI are available in the literature [2,3,23–25].

A common element of all the implementations is that they are based on blockchain technology [26,27].

For example, uPort [2] is an identity management system which implements the SSI paradigm and is based on the Ethereum blockchain [11], in which an identity is a simple smart contract. The uPort project has evolved into the Veramo project [28], a JavaScript framework that makes it easier to implement SSI solutions that are fully compliant with the W3C standard in terms of the format of DIDs and verifiable credentials [12,13]. Veramo also supports the Ethr-DID library [29], which

Table 1
Comparison table.

Proposal	Trust in the holder	Trust in the verifier	Trust in the issuer	Access control mechanism	Implementation available	Blockchain adopted
[3]	yes	no	yes	N/A	yes	Hyperledger Indy
[28]	yes	no	no	N/A	yes	Ethereum
[30]	yes	no	no	N/A	yes	Ethereum
[31]	yes	no	no	N/A	yes	Ethereum
[32]	yes	no	no	N/A	yes	Hyperledger Indy
[34] (Non-SSI)	no	yes	N/A	Mandatory	no	N/A
[35] (Non-SSI)	no	yes	N/A	Mandatory	no	N/A
[36]	yes	no	yes	N/A	yes	Hyperledger Besu / Ethereum
Our proposal	yes	yes	no	ABAC	yes	Ethereum

allows Ethereum addresses to be used as fully self-managed DIDs that comply with the W3C specification.

Other solutions using the Ethereum blockchain can be found in Refs. [30,31]. Although several types of blockchain can be adopted in our solution, we also rely on Ethereum in order to provide a practical implementation.

On the other hand, other solutions, such as Refs. [3,32], use a permissioned blockchain.

Due to its accountability features, blockchain remains the prevalent approach to implementing SSI-based solutions. However, recent promising approaches, such as Ref. [33], trace the road towards the adoption of Peer-to-Peer (P2P) solutions not relying on blockchain. The advantage of Ref. [33] is that it does not require blocks, chains, or miners to provide the proofs for P2P transactions. It is also advantageous in terms of scalability and throughput. However, it is still not clear how to apply this approach in the SSI framework and in particular, how to replace the role of smart contracts. In future work, we plan to study the possibility of transforming our solution into a blockchain-free version.

All the above solutions focus on the problem of the trust required of the user to obtain a service. To the best of our knowledge, no proposal in the literature deals with the problem of the trust required of the service provider to obtain users' credentials and data.

Some similarities with our work, in terms of objectives to reach, can be found in Ref. [34], in which the authors proposed a personal data management system, which uses blockchain as an automated access-control manager. Anyway, this proposal is not tailored to the SSI context. Furthermore, the access control mechanism is totally different from ours. Indeed, in Ref. [34], the user knows the service provider and defines the data it can access to. In our setting, the user does not know in advance the service provider and wants to be sure that it satisfies some requirements.

Similar considerations can be made for Ref. [35], in which a blockchain-based information management system to handle medical records is provided. Anyway, the access control mechanism is for specific users known by the owner of a file and no attributed-based policy can be set.

Finally, we highlight a very recent paper [36] focusing on another relevant aspect that is not treated in standard SSI approaches. In particular, the current paradigm suffers from trust issues between the verifier and the issuer of a verifiable credential. Then, Salve et al. [36] proposed a new multi-layer framework that exploits the web of trust concept to solve this problem. This aspect is also partially treated in Ref. [3] in which the authors provided a quantitative model for computing reputation scores for issuers.

To conclude this section, in Table 1, we summarize the solutions discussed in this section and compare them with our approach in terms of 6 features. Specifically, we consider:

1. Trust in the holder, i.e., the verifier is able to trust the holder. This is the standard objective of SSI.
2. Trust in the verifier, i.e., the holder is able to trust the verifier. This is the objective of this proposal.
3. Trust in the issuer, i.e., the verifier is able to trust the issuer.
4. Access control mechanism adopted to trust the verifier.
5. Availability of a code implementation of the solution.
6. Blockchain adopted to implement the solution.

From the table, it arises that our proposal is the first SSI-based approach achieving trust in the verifier with the ABAC mechanism.

2.2. ABE for access control

In this paper, we implement an ABAC [6] mechanism, by exploiting Attribute-Based Encryption (ABE) [37].

ABE represents a generalization of the standard public-key encryption. It is based on attributes and policies: if a policy is satisfied by a set of attributes, decryption is allowed. In the literature, ABE schemes are divided into two categories: Key-Policy ABE (KP-ABE) and CP-ABE. In KP-ABE, a ciphertext is labeled with a set of attributes (characterizing the ciphertext itself), and a user receives a private key with an embedded policy. The policy defines which types of ciphertext the user can decrypt. On the contrary, in CP-ABE, the policy is associated with the ciphertext while the attributes are associated with the private key of the user. In this paper, we refer to the latter. The first ABE scheme has been proposed in Ref. [37]. Actually, Sahai and Waters [37] presented a new type of Identity-Based Encryption (IBE) called Fuzzy-IBE (FIBE) in which the identities of the users are represented by sets of attributes. In particular, a user owns a private key associated with the attributes w. Similarly, the ciphertext is encrypted with attributes w' and it can be decrypted if $|w \bigcap w'| > d$, where d is an appropriate threshold. This approach is suitable for biometric applications since it allows a sort of error tolerance. The first KP-ABE and CP-ABE schemes have been proposed in Ref. [38] and Ref. [7], respectively. In both proposals, the policies are expressed as monotonic access trees where the leaves represent attributes and the internal nodes are threshold gates. This allows the users to set very expressive policies including AND and OR connectives. Successively, several improved schemes (KP-ABE and CP-ABE) have been proposed [39–42]. For example, in Refs. [40,42], the authors proposed CP-ABE and KP-ABE schemes, respectively, where the length of the ciphertext is constant and does not depend on the number of attributes. An interesting CP-ABE scheme where the policy is hidden to a potential decryptor is discussed in Ref. [41]. This way, regardless of the satisfaction of a policy, they learn nothing about the policy itself from the ciphertext except the fact that they are or are not able to decrypt the data.

The disadvantage of ABE schemes is that they are based on pairing-based cryptography [43], which is time-expansive [44]. To solve this, alternative solutions [45,46] propose ABE schemes without pairing. These solutions appear suitable for the IoT, where constrained devices cannot perform high-intensive computations.

Parallel to the development of ABE, other advanced cryptography schemes, called Proxy Re-Encryption (PRE) schemes, have emerged in Ref. [47]. In PRE, a user delegates a semi-trusted proxy intended for them to re-encrypt a ciphertext into another ciphertext for a different

user. The main advantage of this approach is that the user saves the computational effort to re-encrypt the ciphertext, but the proxy learns nothing about the content of this latter. The first PRE scheme is introduced in Ref. [47]. However, this scheme is bidirectional (if the proxy is able to re-encrypt a ciphertext from Alice to Bob, it is also able to re-encrypt a ciphertext from Bob to Alice) and transitive (if the proxy is able to re-encrypt a ciphertext from Alice to Bob and from Bob to Charlie, it is able to re-encrypt a ciphertext from Alice to Charlie). An interesting discussion about the properties that a PRE scheme should guarantee is presented in Ref. [48].

Successively, more efficient and secure solutions have been proposed in Refs. [48–50].

PRE approaches can be integrated with ABE to obtain Attribute-Based Proxy Re-Encryption (ABPRE) [51]. In this paper, we focus our attention on CP-ABPRE where the re-encryption procedure consists of replacing the policy associated with a ciphertext with a new policy. The first CP-ABPRE scheme is presented in Ref. [8] and supports AND gates between positive and negative attributes. Then, other works have been proposed [15,52,53]. Further details are provided in Section 3.3, in which we investigate the CP-ABPRE schemes available in the literature and select the scheme for our proposal.

We observe that ABE schemes, possibly in combination with proxy re-encryption, have been extensively used in the literature to implement ABAC [38,54–59] but not within a blockchain-based SSI architecture.

3. Background

In this section, we provide the technical background useful for the comprehension of the rest of this paper.

3.1. Self-Sovereign Identity (SSI)

We provide a brief overview of the SSI approach. We do not aim to be exhaustive since SSI is a very broad topic in which several implementations are possible. Therefore, we focus our attention only on the aspects and implementations relevant to our purpose and show how these concepts are adopted in our proposal.

In SSI, each user (also called the *holder*) generates and maintains some DIDs, which they use to interact with other entities. This allows users to have full control of their identity (and of their data) without requiring the collaboration of a trusted third party such as an identity provider.

A VC is a set of claims about a subject that represent information issued by a certain authority. Such authority is called the *issuer*. At a high level, according to the W3C specifications [12,13], a VC includes three sections:

- VC's metadata (e.g., date of issue, expiration date, state of the credential, and so on);
- VC's claims about the subject;
- Proof, i.e., the digital signature made by the issuer of the credential.

The VCs are stored by the holder in a *digital wallet*. The holder can prove something about themselves by presenting a *Verifiable Presentation* (VP) to a *verifier* that consists of data derived from one or more VCs. VCs often coincide with VPs.

As a standard technology, the DIDComm [60] protocol is used to exchange the credentials between the issuer and the holder and between the holder and the verifier.

Blockchain technology is largely adopted in SSI. Indeed, when the issuer releases the VCs to the user, it also stores cryptographic proofs of the VCs on the blockchain. These proofs do not contain any personal information about the user but can be used by the verifier to check the correctness of the VPs. This way, the issuer is not directly involved in the process when the user contacts the verifier. Furthermore, it is the user who chooses the information to disclose to the verifier. The

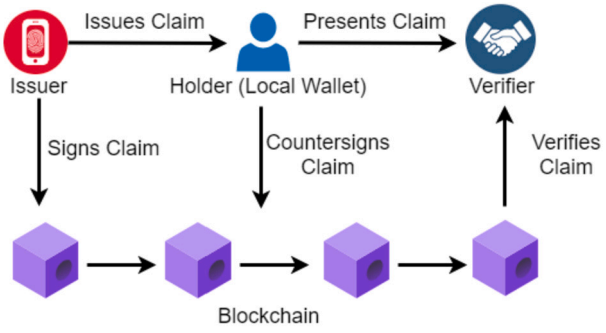


Fig. 1. Self-Sovereign Identity (SSI) paradigm.

blockchain also stores the DID documents. Each DID points to a DID document that contains a set of data describing the DID subject, including mechanisms, such as cryptographic public keys, that the DID subject or a DID delegate can use to authenticate itself and prove its association with the DID [12].

On the contrary, in a traditional identity management model, an identity provider is involved during the user's request to the verifier by disclosing the information on behalf of the user. This leads to an unnecessary privacy leakage since the identity provider discovers that the user has required a certain service from a certain verifier. In the SSI model, the role of the identity provider is reduced to that of an identity issuer that releases some information about the identity of the user but does not manage it.

Another advantage of the blockchain is that it represents a natural way to generate DIDs. Indeed, blockchain addresses can be generated and exploited by users as DIDs [29]. Furthermore, each user can generate as many blockchain addresses as they need and use them to perform independent requests.

Anyway, when blockchain addresses are used as DIDs, a linkage between the real identity and the DID may be necessary [61], otherwise the user may repudiate the ownership of such DIDs. This happens, for example, when the juridical regulation requires that the parties interacting with the user know who actually the user is. Furthermore, under certain circumstances, the accountability of some critical operations performed by the user is required so that an agent authorized by the law (also with the collaboration of some entities) can identify the user.

The mapping between the DID and the real identity of the user can be guaranteed by a certification authority.

To certify such a mapping, there are some ways. For example, the user could generate a transaction from their blockchain address containing a challenge provided by the certification authority. Alternatively, the user can sign a message with the private key corresponding to the public key associated with their blockchain address.

In our application, the certification authority coincides with an *identity issuer* [61] (who already knows the real identity of the user). In particular, by adopting the SSI principles, after verifying the mapping between the Ethereum address and the real identity of a user, the identity issuer releases a VC to the user, certifying the ownership of the blockchain address.

However, the identity issuer alone should not be able to know the actions performed by the user, but only with the collaboration of another entity willing to prove to a legal authority that the user performed a given action. This aspect is better discussed in Section 9.

In this paper, we refer to a service provider (playing the role of verifier), an identity issuer, and a set of attribute providers (playing the role of *attribute issuers*).

A high-level representation of the SSI paradigm is represented in Fig. 1.

3.2. Ciphertext-policy attribute-based proxy re-encryption (CP-ABPRE)

CP-ABPRE is a public-key encryption scheme that allows a semi-trusted proxy to transform a ciphertext encrypted under a given policy into another ciphertext under a different access policy without learning anything about the plaintext. We introduce the following definitions.

Definition 3.1 *(Access structure* [62]*).* Let $\{P_1, P_2 \ldots, P_n\}$ be a set of parties. A set $\mathbb{A} \subseteq 2^{\{P_1, P_2 \ldots, P_n\}}$ is *monotone* if $\forall B, C$: if $B \in \mathbb{A}$ and $B \subseteq C$ then $C \in \mathbb{A}$. An access structure (monotone access structure) is a set (monotone set) $\mathbb{A}$ of non-empty subsets of $\{P_1, P_2 \ldots, P_n\}$, i.e., $\mathbb{A} \subseteq 2^{\{P_1, P_2 \ldots, P_n\}} \setminus \{\emptyset\}$. The sets in $\mathbb{A}$ are called the *authorized sets*, whereas the other sets are called the *unauthorized sets*. $\mathbb{A}$ is also called *policy*.

Definition 3.2. A CP-ABPRE scheme consists of the following algorithms:

1. Setup(k): This algorithm receives a security parameter k and returns a public parameter PK and a master secret key MSK.
2. Encrypt($PK, M, \mathbb{A}$): This algorithm encrypts a message M under the policy (access structure) $\mathbb{A}$ by using PK. It outputs a ciphertext CT, which can be decrypted only by a user who owns the attributes that satisfy $\mathbb{A}$.
3. KeyGen(MSK, S): This algorithm takes as input a set of attributes S and the master secret key MSK. It outputs a private key SK associated with S.
4. Decrypt(CT, SK, PK): This algorithm takes as input a public parameter PK, a private key SK associated with a set of attributes S, and a ciphertext CT encrypted under an access structure $\mathbb{A}$. If S satisfies $\mathbb{A}$, then the algorithm outputs a plaintext M.
5. ReKeyGen($PK, SK, \mathbb{A}'$): This algorithm takes as input a private key SK associated with a set of attributes S and an access structure $\mathbb{A}'$. It outputs a re-encryption key RK that can be used, by a proxy, to re-encrypt a ciphertext CT, encrypted under a policy $\mathbb{A}$, into a new ciphertext CT' encrypted under the policy $\mathbb{A}'$. The re-encryption is allowed only if S satisfies $\mathbb{A}$.
6. ReEncrypt(PK, CT, RK): This algorithm uses a re-encryption key RK to re-encrypt a ciphertext CT, encrypted under a certain policy, into another ciphertext CT' under a new policy.

Moreover, a trusted third party, called Private Key Generator (PKG), is also present. The PKG invokes Setup(k) to obtain PK and MSK. It shares PK with all the users and keeps MSK secret. Finally, another entity, called the *proxy*, has the role of invoking ReEncrypt(PK, CT, RK) and changing the policy associated with the ciphertexts.

The scheme works as follows.

Consider two users Alice and Bob. Suppose that Alice wants to encrypt a file f so that only the users satisfying a policy $\mathcal{P}$ can decrypt it. She invokes $C = \text{Encrypt}(PK, f, \mathcal{P})$ and sends C to the proxy that manages Alice's files without accessing their content. If Bob wants to access the file f, he first contacts the PKG to obtain a private key associated with his attributes S. The PKG verifies that Bob really owns such attributes (often this step requires the collaboration of some attribute providers), invokes $SK_B = \text{KeyGen}(MSK, S)$, and provides Bob with SK_B. If the set S satisfies the policy $\mathcal{P}$, Bob asks the proxy for C and accesses the file $f = \text{Decrypt}(C, SK_B, PK)$.

Until now, we have seen only the ABE part of the ABPRE scheme, and the proxy plays just the role of a simple database storing Alice's data.

Now, suppose that Alice wants to change the policy associated with the file f from $\mathcal{P}$ to $\mathcal{P}'$. She can download the encrypted file from the proxy, decrypt it, re-encrypt it under $\mathcal{P}'$, and upload the file again on the proxy. However, if she owns many files and/or the policy changes are frequent, the encryption/decryption operations (and also download/upload) may result in prohibitive costs, especially in the case of resource-constrained devices. Therefore, Alice generates a re-encryption key $RK = \text{ReKeyGen}(PK, SK_A, \mathcal{P}')$, where SK_A is the private key associated with Alice's attributes (in this example, we suppose that Alice satisfies the policy $\mathcal{P}$), and sends it to the proxy. When another user (possibly, Bob) contacts the proxy to obtain f, the proxy invokes $C' = \text{ReEncrypt}(PK, C, RK)$ and sends C' to the user. The user can retrieve f only if they own a private key satisfying the policy $\mathcal{P}'$.

To conclude this section, we observe that when CP-ABPRE is applied in real-life contexts, there are some issues that need to be considered.

1. First, consider another user, Charlie, who stores a file f' with the same policy $\mathcal{P}$ chosen by Alice. If he provides a re-encryption key to the proxy to change the policy of f' from $\mathcal{P}$ to $\mathcal{P}''$, the proxy can use this key to change the policy associated with the file f of Alice. This issue is analyzed in Section 6.1.
2. A similar problem occurs if Alice encrypts multiple files under $\mathcal{P}$ and wants to change the policy of just some of them. Once the re-encryption key is obtained, the proxy can change the policies of all the other files of Alice encrypted under the same policy. Even though in some cases this can be considered a positive feature of the system (since Alice can provide a single re-encryption key for many files), we are interested in providing a mechanism such that Alice can avoid the re-encryption of non-intended files even under already used policies. This issue is analyzed in Section 6.2.
3. Another issue to solve is that a policy should have a time validity, in such a way that a file can be accessed only during a time window. This issue is analyzed in Section 6.3.
4. Finally, a mechanism to manage hierarchical attributes is desirable. This aspect is addressed in Section 5.

3.3. Selection of the CP-ABPRE scheme

Our solution requires the application of a CP-ABPRE scheme to implement the access control mechanism. Even though our solution works with any underlying scheme (also by applying the approaches discussed in Sections 5 and 6), we investigated the literature and selected a specific scheme (i.e., Ref. [15]) to provide a more concrete proposal. Furthermore, we also provide our Java implementation of such a scheme and integrate it in a complete prototype described in Section 8.

The Ref. [15] is widely referred to in the literature. The scheme therein presented supports any monotonic access policy represented by an LSSS matrix (M, ρ) [62,63], where M is a matrix and ρ is a mapping between the rows of M and a set of attributes. There are two issues to take into consideration when using this scheme. The first is that it does not support policies with negation. This is not a limitation in our setting. Indeed, the satisfaction of a negative attribute, in practice, would be implemented according to a closed-world assumption (otherwise, it would be infeasible for a PKG to test the universe of attribute providers). Therefore, an attribute not provided by the user would be satisfied if negated. This would be clearly incorrect for practical purposes. If we want to require that an attribute is not fulfilled by a user (e.g., no criminal conviction), we set a positive attribute with this meaning.

The second issue is that, in this scheme, the re-encryption requires that the old access structure (M, ρ) and the new access structure (M', ρ') are disjoint, i.e., if an attribute x satisfies (M, ρ), then x does not satisfy (M', ρ'). This is not suitable in several real scenarios where the policies have slight changes (e.g., only a new attribute is required or a single attribute is no longer required).

However, as it will be clear in Section 7, in our solution, the re-encryption is done from a policy including only the identity of the user and a label (see Section 6) to a new policy requiring some attributes to a service provider. Therefore, the two associated access structures are always disjoint.

The main advantages of this scheme are: (1) it uses LSSS matrix that allows very expressive policies, and (2) the performance (the size of

keys and ciphertext and the number of computations) does not depend on the size of the attribute universe.

Now, we see the other possible schemes. The schemes in Refs. [8,52] allow only policies with AND connectives between positive and negative attributes, thus less expressive policies. Furthermore, their performance depends on the size of the attribute universe. The scheme in Ref. [53] is, basically, very similar to that in Ref. [15]. In Ref. [64], a ciphertext is associated with an attribute-based access structure and an access time that defines when a user can access the data. This is an interesting feature, and we implement it through the mechanism described in Section 6.3. Unfortunately, in the Ref. [64] scheme, the re-encryption regards only the access time and not the attribute-based access structure.

Other schemes in Refs. [65,66] do not meet our requirements.

4. Motivation

In this section, we explain the rationale of our proposal by highlighting the advantages we achieve with respect to a standard solution. The idea at the basis of our proposal is that the holder is able to check if some credentials of the verifier are compliant with self-defined access control policies.

Our solution extends the standard SSI paradigm by introducing new actors and new mechanisms. Basically, they are listed as below.

- A distributed wallet (instead of the standard client-side wallet) that gathers all encrypted verifiable credentials. The symmetric key enabling their decryption is generated by the user and encrypted with an ABPRE scheme to embed access control policies.
- A consensus provider whose role is twofold: (1) eliminating direct interaction between the holder and the verifier until trust is established and (2) implementing accountability for relevant actions by the holder and the verifier.
  To achieve the objective (1), the consensus provider delivers symmetric keys (using re-encryption) to the verifier. We observe that, due to the proxy-re-encryption mechanism, the consensus provider has no additional rights to access symmetric keys beyond those allowed by embedded access control policies. Concerning objective (2), the consensus provider interacts with a smart contract (that we will mention below) to notarize some relevant actions of the holder and the verifier without compromising their privacy. It is worth noting that the presence of the consensus provider seems to move us away from the SSI principles. This is true, but, as explained below, this is a necessary price we have to pay to achieve our goals.
- A smart contract, whose functions are invoked by the holder, the verifier, and the consensus provider, notarizes the relevant actions performed by the various actors, thus achieving the aimed accountability features.

The main objective of our extended SSI architecture is to empower the holder with greater control over their data, aligning seamlessly with the principles of SSI. This enhanced control is achieved through two interrelated mechanisms. First, the implementation of an access control mechanism restricts access to the holder's data solely to authorized and trusted parties. Second, accountability plays a crucial role by providing the holder with digital (forensic) proofs. These proofs enable verification of whether data access aligns correctly with the established access policies.

To underscore the significance of accountability, it is important to note that in standard SSI, an untrusted verifier receiving verifiable credentials could potentially share them with unauthorized parties. In the absence of accountability mechanisms, the holder lacks control over such data leakage in traditional SSI frameworks.

Now we show that our extended SSI architecture is necessary to achieve the general objective described above. In other words, we argue that a standard SSI solution in which the role of the holder and the verifier is reversed (i.e., the holder requires the verifier's verifiable credentials to check if they are compliant with the access control policies) would not reach the achievement we are looking for and other advantages of our solution.

First, no accountability mechanism could be implemented (at least in a trivial way) without the consensus provider. In our solution, we reach accountability, addressing the concern of unauthorized forwarding of credentials by using a smart-contract-based approach that relies on the presence of a distributed wallet, to access to which no action is required with the holder. An additional point is that in scenarios where access control is managed locally by the user (as is the case with a standard wallet), relevant information such as policy changes may lack accountability. In contrast, our approach using the smart contract and the distributed wallet ensures accountability even for policy changes, providing a more comprehensive and secure solution.

Second, for the exchange of verifier credentials with the holder, the establishment of a secure channel, typically P2P [60], is required. However, this introduces a potential risk to the holder's identity, which should remain undisclosed until the verifier is authorized to receive the credential. In contrast, our solution eliminates the need for a bidirectional connection. The holder initiates the process by sending a request to the verifier, possibly through a proxy to conceal their IP address, and no response is required from the verifier. This approach ensures a more secure and privacy-preserving credential exchange.

Finally, we address a scenario where a multitude of credentials and policies may be present. In such cases, it becomes crucial to minimize the burden on the management of an access control mechanism, especially considering that the holder device is often a smartphone with limited computational and memory resources.

5. Hierarchical attributes in CP-ABPRE schemes

One of the contributions of this work is to provide a mechanism to manage hierarchical attributes. Indeed, this feature is not addressed directly by any CP-ABPRE scheme available in the literature. For example, consider an academic domain in which there are three types of professors: full, associate, and assistant. Some files may be encrypted only for full professors, while other files may be encrypted for any kind of professor. In the first case, the files should be encrypted under the policy $\mathcal{P} = \{\texttt{Full}\}$. In the second case, the files should be encrypted under the policy $\mathcal{P}' = \{\texttt{Full} \vee \texttt{Associate} \vee \texttt{Assistant}\}$, where $\vee$ denotes the OR operator. Anyway, in the latter case, it would be more practical, for the user encrypting the files, to select a simple policy $\mathcal{P}'' = \{\texttt{Professor}\}$ equivalent to $\mathcal{P}'$.

Unfortunately, in the state-of-the-art CP-ABPRE schemes, to support all the policies $\mathcal{P}$, $\mathcal{P}'$, and $\mathcal{P}''$ simultaneously, the attribute `Professor` has to be maintained separately by the attributes `Full`, `Associate`, and `Assistant`. This means that the attribute universe should include four attributes in place of three. Furthermore, each professor should maintain (in their private key) two attributes (`Professor` and one among `Full`, `Associate`, and `Assistant`) in place of a single attribute.

As discussed at the end of this section, the total number of attributes in the attribute universe and/or maintained (in the private key) by the users should be as low as possible for efficiency reasons.

It would be desirable to have a CP-ABPRE scheme in which a professor with just one attribute among the attributes `Full`, `Associate`, and `Assistant`, can satisfy $\mathcal{P}''$ without explicitly maintaining the attribute `Professor`.

In this section, we do not have the ambition to provide a new CP-ABPRE scheme with the above feature. Instead, we provide a way to allow the user to express a policy with *high-level* attributes (then, more declarative), which is automatically translated into a policy with *low-level* attributes. Only these low-level attributes will be included in the attribute universe of the chosen CP-ABPRE scheme.

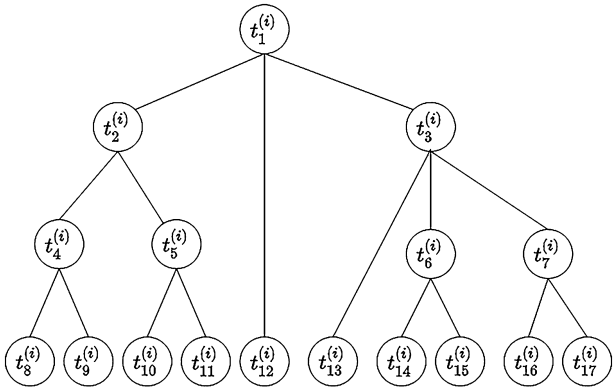


Fig. 2. Example of Tree T_i with $|T_i| = 17$.

We see, in detail, how this mechanism works.

Consider a set of *attributes* $A = \{a_1, \ldots, a_n\}$ and a partition $T = \{T_1, \ldots, T_k\}$ of A (i.e., $\bigcup_{i=1}^{k} T_i = A$ and $T_i \cap T_j = \emptyset$ with $1 \leq i < j \leq k$).

The set $T_i = \{t_1^{(i)}, \ldots, t_{|T_i|}^{(i)}\} \in T$ is a *hierarchical domain* arranged as a *Tree*, where each attribute $t_j^{(i)} \in T_i$ represents a *node* of the Tree. W.l.o.g., we assume that the element $t_1^{(i)}$ is the *root* of the Tree T_i. An example of a Tree T_i with $|T_i| = 17$ is depicted in Fig. 2.

Given a node $t_j^{(i)} \in T_i$, we denote by C_{ij} the set of all the children nodes of $t_j^{(i)}$. Furthermore, with each node $t_j^{(i)} \in T_i$, we associate a bit b_{ij} with the following *integrity constraint*: $b_{ij} = \bigvee_{t_h^{(i)} \in C_{ij}} b_{ih}$, where the symbol $\bigvee$ denotes the OR operator.

Finally, we define a function *Node Policy* $P(t_j^{(i)})$ for each node $t_j^{(i)}$:

$$P(t_j^{(i)}) = \begin{cases} t_j^{(i)} \wedge b_{ij}, & \text{if } t_j^{(i)} \text{ is a leaf} \\ \bigvee_{t_h^{(i)} \in C_{ij}} (P(t_h^{(i)}) \wedge b_{ih}), & \text{otherwise} \end{cases}$$

where $\wedge$ denotes the AND operator. The policy $P(t_1^{(i)})$ (i.e., the policy of the root of T_i), is called *Root Policy*.

We define the *Tree Policy* as $P(T_i) = P(t_1^{(i)}) \vee (\neg b_{i1})$.

Clearly, given a Tree, we can have different Tree Policies according to the bits associated with the nodes. For example, some possible Tree Policies for the Tree in Fig. 2 are depicted in Fig. 3.

We define the *Total Policy* $\mathcal{P}$ as the conjunction of all the Tree Policies: $\mathcal{P} = \bigwedge_{i=1}^{k} P(T_i)$.

The Total Policy is in *Conjunctive Normal Form*.

At this point, we illustrate how a user can leverage the above structure to define the policies at a high level, with no need to specify low-level attributes.

We define the following two *semantic constraints*.

1. A father-child relationship in a Tree corresponds to an *Is-A* relationship so that, if a user owns an attribute associated with a node t of the Tree, then they implicitly own all the attributes associated with the nodes in the path from t to the root.
2. If a user owns an attribute associated with a non-leaf node, then they own at least one attribute associated with one of its children nodes.

Regarding the management of access control, setting a policy means assigning bits to the nodes of the trees. In detail, each user sets such bits according to the following *operative rules*:

1. By default, the bits of all the nodes of all the trees are set to 0.
2. If a user sets the bit of a node t to 1, then all the bits of the nodes in the path from t to the root are set to 1. Moreover, all the bits of the nodes in the sub-Tree with t as root are set to 1.
3. If a user sets the bit of a node t to 0, then all the bits of the nodes in the sub-Tree with t as root are set to 0. Moreover, if all bits of the sibling nodes of t (i.e., the nodes with the same parent node) are set to 0, then the parent of t is set to 0 recursively according to this rule.

Rule (1) implies that, by default, the Total Policy is equal to 1 i.e., it is always satisfied. Rules (2) and (3) are sufficient to guarantee the integrity constraint for each node of the Tree. Actually, regarding Rule (2), it is not necessary from a logical point of view to set all the bits of the nodes in the sub-Tree to 1 to guarantee the integrity constraint. However, this has an impact on the way the policy is expressed. Indeed, it allows us to set the policy in terms of high-level attributes without specifying the leaf attributes.

We illustrate how these operative rules work through an example.

Example 5.1. Consider again the academic context described by the Tree in Fig. 4. Initially, all the nodes have the bits set to 0 (Rule (1)). If a user wants to set a policy that can be satisfied by any professor, they simply change the bit only for the attribute `Professor`, and due to Rule (2), the bits of the nodes `University`, `Full`, `Associate`, and `Assistant` become 1.

We observe that, since the high-level attributes are translated into the corresponding low-level attributes, the Total Policy includes only the low-level attributes. Therefore, the used CP-ABPRE scheme only needs to include, in the attribute universe, the attributes associated with the leaves of the trees.

We observe that the hierarchical organization of the attributes gives an advantage in terms of the space required for the universe with respect to the trivial flat enumeration of all possible attributes (including also high-level attributes). Indeed, the attribute universe can include only the attributes associated with the leaves of the trees.

This advantage is effective, especially in those schemes, such as schemes in Refs. [8,52], in which the size of ciphertexts, public keys, private keys, re-encryption keys, and the computational costs of CP-ABPRE algorithms increase as the number of elements of the attribute universe increases.

Furthermore, for the schemes in which the above sizes and computational costs do not depend on the size of the attribute universe (e.g., Ref. [15]), we have benefits. Indeed, if high-level attributes are included, for each attribute a user owns, they would maintain all the ancestor attributes.

In the above example, an associate professor would maintain the attributes `Associate`, `Professor`, and `University` in their private key. Therefore, in this case, the size of this key and all the computations involving such a key increase with respect to the case in which the professor maintains only the attribute `Associate`.

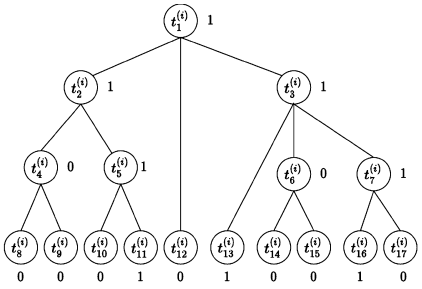
6. Identities, label, and time management in CP-ABPRE schemes

Throughout this section, we face the three problems mentioned in Section 3.2. As in Section 5, we distinguish between the schemes in which the performance does not depend on the size of the attribute universe, which we identify by (i), and the other schemes, which we identify by (ii).

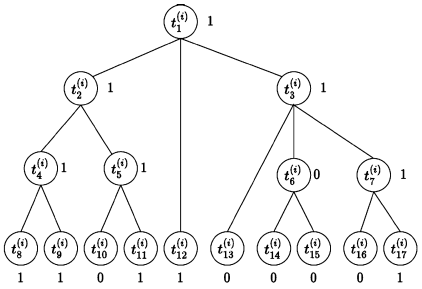
6.1. Identities management

The first treated problem regards the management of the identities of users.

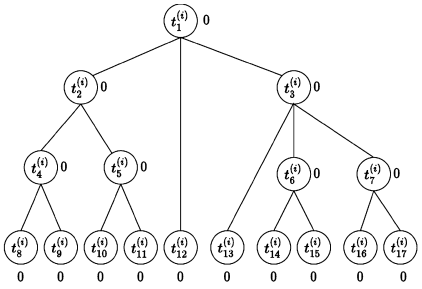
Our solution requires that the files (i.e., the VCs) are encrypted under a policy that only a specific user u (owner of the files) can satisfy,



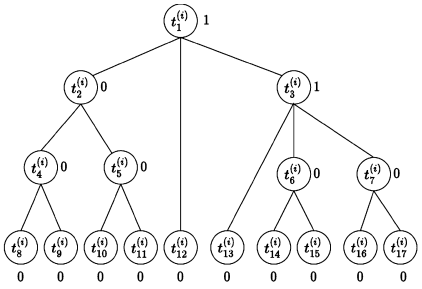
(a) $P(T_i) = t_{11}^{(i)} \vee t_{13}^{(i)} \vee t_{16}^{(i)}$.



(b) $P(T_i) = t_{8}^{(i)} \vee t_{9}^{(i)} \vee t_{11}^{(i)} \vee t_{12}^{(i)} \vee t_{17}^{(i)}$.



(c) $P(T_i) = 1$.



(d) "Malformed" Policy since the integrity constraint is violated for the bit b_{i3}.

Fig. 3. Examples of Root Policies.

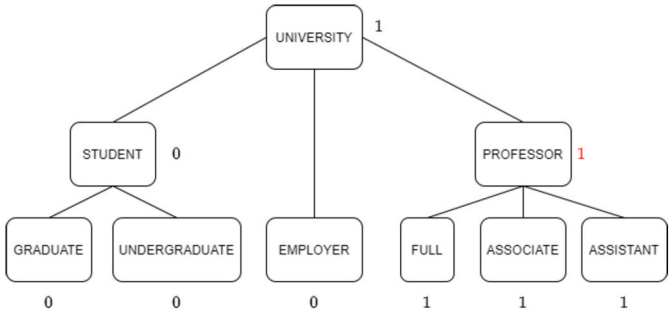


Fig. 4. Tree for an academic context.

i.e., a policy requiring the user's identity as an attribute. This way, the re-encryption keys used to change the policies of files of other users cannot be used to change the policies of the files of u.

For schemes of type (i), the way to accomplish this is straightforward: just add an attribute ID_u in the attribute universe for each user u of the system. When a user u wants to encrypt a file so that only himself/herself can decrypt it, he/she defines a policy requiring ID_u to access the file. Clearly, ID_u will be included in the private key provided to u by the PKG.

However, this approach is not applicable for schemes of type (ii), since the cardinality of the attribute universe would be too high. Thus, for such schemes, we consider a set $ID = \{id_1, id_2, \ldots, id_n\}$ and define the *Identity Set* ID_S as the set of all the subsets of ID with cardinality equal to a fixed value $0 < k < n$. We denote by $ID_u \in ID_S$ the *identity* of the user u. Note that $ID_x \not\subset ID_y$ for any pairs of users x, y. This is necessary; otherwise, if an identity $ID_x \subset ID_y$, then the user y can satisfy the policies requiring ID_x. The cardinality of ID_S is $\binom{n}{k}$. Therefore, to maximize such a cardinality, we set $k = \lfloor \frac{n}{2} \rfloor$. This way, as the binomial coefficient $\binom{n}{\lfloor \frac{n}{2} \rfloor}$ grows exponentially with n, even small values of n are enough to have very large domains of users. For example, with only $n = 33$ attributes, we represent $2,333,606,220$ users.

6.2. Label management

The second treated problem is to have a *label* associated with the encrypted files so that the proxy can re-encrypt only the files of a user with the same label.

For both schemes of types (i) and (ii), we use the same approach.

We add in the attribute universe a set of labels $L = \{l_1, \ldots, l_t\}$, for a given value t. For each label $l \in L$, u receives, from the PKG, a private key including such a label (along with the identity of the user, as explained in Section 6.1). When u encrypts a file, he/she include his/her identity ID_u together with a label $\bar{l}$ in the policy of the file (this makes the use of a given label unique despite the shared set of labels). Then, u generates the re-encryption key for the proxy by using the private key associated with $(ID_u, \bar{l})$. This way, the proxy can only change the policies of the files of u associated with $\bar{l}$.

For both schemes of types (i) and (ii), the number of private keys required is t (according to the needs of the user) and each private key includes just two attributes, that is, the identity of the user and a file label.

The above mechanism solves the issues described in items 1 and 2 at the end of Section 3.2.

We will return to these aspects in Section 7.

6.3. Time window

The last treated issue is how to set a validity time for the rights of access to a ciphertext. This approach is partially inspired by the approach proposed in Ref. [7], but it is more general and includes some practical aspects.

In detail, the problem is how to set a time window within which the files can be accessed.

For schemes of type (i), we include in the attribute universe some integers corresponding to the UNIX timestamps (they do not need to be materialized or stored). Specifically, when a user u receives a private key from the PKG, it includes in such a key (as an attribute) the current UNIX timestamp t. When another user y wants to encrypt a file so that it can be accessed between t_1 and t_2, it sets a policy in the form $\mathcal{P} = \{t_1 < \texttt{Time} < t_2\}$. If $t_1 < t < t_2$, u can decrypt the file.

As illustrated in Ref. [7], through the connectives "OR" and "AND", it is possible to efficiently implement the operation $<$ and $>$ at a small price to express the UNIX timestamps in binary (thus, the user will receive a private key by the PKG, including as many attributes as the bits of the current UNIX timestamp t).

Regarding schemes of type (ii), the approach is the same, but the size of the attribute universe would be too high with the inclusion of the UNIX timestamps. Anyway, in a practical context, we do not need a time granularity at the level of the second. Therefore, we can lower the granularity at the level of the day or the month, thus reducing the size of the attribute universe.

7. The proposed protocol

7.1. Overview of the protocol

We start by providing an overview of the proposed protocol. Each user is the owner of a set of VCs released by some issuers. We do not have any requirement in the way in which the VCs are exchanged and their format. Then, standard technologies such as DidComm [60] (for the exchange) and JSON-LD (W3C compliant format [13]) should be adopted.

On the other hand, to implement our solution, the interaction between the user and the service provider requires some modifications, while the format of the credential is kept standard.

To satisfy the request of a user, a service provider needs to access some of their VCs. However, not all VCs are accessible by the service provider, only those for which it meets a certain policy chosen by the user. In general, the user does not know which attributes the service provider owns and which policies it fulfills. The management and enforcement of the user's policies are delegated to a semi-trusted third party which we call the *consensus provider*. It acts as an intermediary in the communication between the user and the service provider by providing the latter (if authorized) with all the information needed to access the VCs. The consensus provider is semi-trusted in the sense that it collaborates in the execution of the protocol, but it is not able to access the content of users' VCs so that confidentiality is guaranteed.

As in standard SSI-based solutions, to certify the possession of some attributes and identities of users and service providers, our solution includes some attribute issuers and identity issuers that release some credentials to users and providers. However, according to the SSI principles, they are not involved during the interaction between the user and the service provider.

Regarding the technologies, we rely on the blockchain to notarize critical operations. To be concrete, we consider the Ethereum blockchain, which supports smart contracts. Through them, we offer accountability guarantees by *post-mortem* detecting anomalous behavior of users, service providers, and consensus providers.

We also consider a distributed wallet in which the VCs are stored in encrypted form and the consensus provider maintains only some symmetric keys encrypted with CP-ABPRE that allow access to the VCs. Such a distributed wallet is implemented through the InterPlanetary File System (IPFS) [67], which is a distributed file system in which the VCs are stored and easily accessible by any involved entity. Ethereum [29] and IPFS [68] are two technologies supported by the Veramo project [69,28] adhering to W3C standard. Moreover, as reported in Ref. [70], "over a dozen of the DID methods registered in the W3C DID Specification Registries are designed to work with either Bitcoin or Ethereum".

Finally, as discussed in Section 3.3, we rely on the CP-ABPRE scheme [15].

7.2. Notations

We introduce the notations we use in the following.

- We denote by $E_k(M)$ the application of a symmetric encryption function (e.g., 3DES or AES) on a message M with symmetric key k.
- We denote by $D_k(C)$ the decryption of a ciphertext C with the symmetric key k.
- We denote by Setup(z), Encrypt($PK, M, \mathcal{P}$), KeyGen(MSK, S), Decrypt(CT, SK, PK), ReKeyGen ($PK, SK, \mathcal{P}'$), and ReEncrypt(PK, CT, RK) the algorithms of the selected CP-ABPRE scheme selected as defined in Section 3.
- We model an Ethereum transaction as $T = \langle id_T, Eth_{src}, Eth_{dest}, data \rangle$, where id_T is the identifier of the transaction, Eth_{src} is the source address, Eth_{dest} is the destination address and $data$ is the payload. The source and the destination of a transaction can be a user, a service provider, a consensus provider, or a smart contract. The Ethereum addresses play the role of DIDs (see Section 3.1).

7.3. Entities

We introduce the following entities of our system.

- The user u requiring a service s.
- The service provider SP delivering the service s to u.
- The consensus provider CP (which plays the role of a proxy re-encryptor).
- The PKG, i.e., a trusted third party that generates the ABE private keys for users and service providers. It relies on identity providers and attribute providers to certify the identity of the users and the attributes of the service providers, respectively.
- The identity issuer II, which associates each user u with an identity ID_u. In addition, II is in charge of certifying the linkage between the DID of a user or a service provider and their real identity. II can verify such a linkage through one of the techniques proposed in Section 3.1. Observe that II does not manage the DIDs of CP since we assume they are public.
  W.l.o.g., we assume a single II for all users. In the case of multiple IIs, they have to guarantee the uniqueness of each identity in the system.
- The attribute issuer $AI(a_i)$, which is in charge of certifying that the service providers own the attribute a_i. In general, an attribute issuer can certify multiple attributes.
- $IPFS$ is the InterPlanetary File System, which is used as a repository to store the (encrypted) users' VCs. When a $VC(c)$ is stored in the $IPFS$, it returns an index $i_{IPFS}(c)$, which allows the users to retrieve the VC at a later time.
- The SC is the smart contract aimed at achieving accountability.

Now, we describe the phases that the entities involved in our proposal perform to implement the required functionalities.

7.4. Credentials issuing

In this phase, some credentials needed to implement our solution are released by some issuers. This can be performed by leveraging any standard technology and protocol implemented for SSI solutions. For reference, we assume that the credentials are exchanged through the DIDComm protocol [60] and they are in JSON-LD format according to the W3C specification [13]. They are standard technologies in the SSI domain [70].

1. A generic issuer releases the credential $VC(f)$ to u. $VC(f)$ represents the actual credential that u needs to access some service of SP.
2. II releases a credential $VC(u)$ to u. $VC(u)$ contains two attributes. The first attribute is ID_u and is used to certify the real identity of u. The second attribute is Eth_u and is used to certify that u owns

this Ethereum address and that II knows the mapping between ID_u and Eth_u.
3. II releases a credential $VC(SP)$ to SP. Similar to the previous step, $VC(SP)$ contains two attributes, the real identity ID_{SP} of SP and its Ethereum address Eth_{SP}.
4. SP obtains, for each attribute a_i it owns, a credential $VC(a_i)$ from the attribute issuer $AI(a_i)$. This credential certifies that SP owns the attribute a_i.

7.5. Registration phase, key generation, and smart contract deployment

In this phase, u and SP register with PKG to obtain the CP-ABPRE private keys. Furthermore, u and SP perform a registration with CP, which deploys a smart contract used to notarize some critical operations performed by u, SP, and CP itself.

All these steps are performed only once for each entity of the system.

1. Preliminary, PKG invokes Setup(z), where z is a security parameter, to obtain PK and MSK (see Section 3.2). PK is made publicly available while MSK is maintained private by PKG.
2. User u sends PKG the credential $VC(u)$ and a number of labels v they need. Observe that, differently from the description given in Section 6.2, to be more practical, we allow u to select a number of labels $v \leq t$ in the case that they do not need all the t possible labels. PKG verifies the credential and then the identity ID_u of u. PKG randomly selects, from the set $L = \{l_1, \ldots, l_t\}$, a set L_u of cardinality v. For each label $l_i \in L_u$, PKG generates a private key $SK_u^{l_i} = \text{KeyGen}(MSK, \{l_i, ID_u\})$ and forwards it to u.
3. Similarly, SP contacts PKG to obtain the private key SK_{SP} associated with its attributes A_{SP}. SP provides PKG with the set $VC(A_{SP})$ containing all the credentials $VC(a_i)$ for each attribute $a_i \in A_{SP}$.
   After verifying all the credentials (for each attribute a_i), PKG generates the private key $SK_{SP} = \text{KeyGen}(MSK, A_{SP} \cup \{UT\})$ and sends it to SP. According to the mechanism described in Section 6.3, PKG includes in the private key the current UNIX timestamp UT (along with the set A_{SP}) when SP requires the key. The three steps described above, involving the interactions of u and SP with PKG, are summarized in the sequence diagram in Fig. 5.
4. In the next step, u and SP register with CP by providing their credentials $VC(u)$ and $VC(SP)$ containing ID_u and ID_{SP}, respectively.
5. This phase ends with the deployment by CP of the smart contract SC represented in Fig. 6 (written in Solidity [71]). For space limitation, we represent only a partial version in the figure. The complete smart contract, which also manages anomalous situations, is reported at https://github.com/vincenzodeangelisrc/Self-Sovereign-Identity-Solution-UNIRC/blob/main/Self_Sovereign_Identity_Solution/src/it/unirc/CP/DeploySC/Contract.sol.
   It contains three mappings between an Ethereum address (of a user or a service provider) and a struct called `NotarizationData` containing an array `hash_list` to store the actual data to notarize, and two temporary variables `temp_hash` and `sent_time` (their meaning will be clear in the following). The first mapping (`StorageMap`) is used to notarize the storage of a VC of u. The second mapping (`PolicyChangeMap`) is used to notarize a policy change performed by u. Finally, the third mapping (`FileAccessMap`) is used to notarize the access to the VCs by SP.

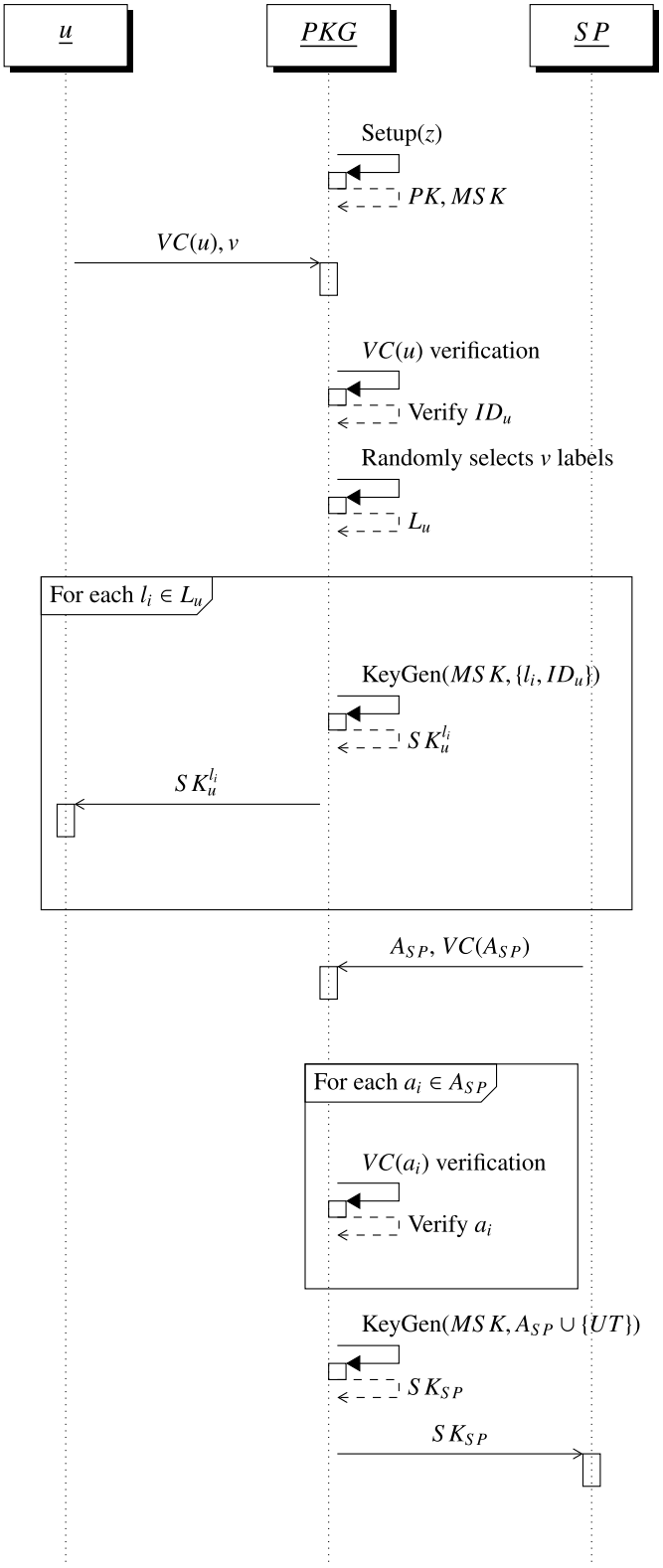


Fig. 5. Interactions of user u and the service provider SP with the Private Key Generator PKG.

7.6. VC storage

This phase starts when u wants to store the credential $VC(f)$ in their distributed wallet.

1. User u selects a symmetric key k_f (for $VC(f)$) and encrypts it, thus obtaining $c_f = E_{k_f}(VC(f))$. Then, c_f is stored in the $IPFS$, which returns the index $i_{IPFS}(c_f)$.
2. User u selects a label $l_i \in L_u$ to be associated with f, defines a *Basic Policy* $\mathcal{B} = \{ID_u \wedge l_i\}$, and encrypts k_f under $\mathcal{B}$ (with CP-

```
pragma solidity 0.6.11;
contract Notarization{

    mapping(address => NotarizationData ) StorageMap;
    mapping(address => NotarizationData ) PolicyChangeMap;
    mapping(address => NotarizationData ) FileAccessMap;
    struct NotarizationData{
        bytes32 [] hash_list;
        bytes32 temp_hash;
        uint256 sent_time;}
    function startNotarization(uint8 _type, address _ETHr,
         bytes32 _hash){
    if(_type==1){
        if((StorageMap[_ETHr].sent_time==0)&&(StorageMap[_ETHr
            ].temp_hash==0x0000000000000000000000000000)){
        StorageMap[_ETHr].temp_hash=_hash;
        StorageMap[_ETHr].sent_time=block.timestamp;}
      ......  //omitted for readability }
    else{  //type 2 and type 3
    ....... //omitted for readability
    }}
    function confirmNotarization(uint8 _type, bytes32 _hash){
    if(_type==1){
        if((block.timestamp-StorageMap[msg.sender].sent_time)
            <=10){
            if(_hash==StorageMap[msg.sender].temp_hash){
            StorageMap[msg.sender].hash_list.push(_hash);}
            else{emit "Event"}}
        StorageMap[msg.sender].sent_time=0;
        StorageMap[msg.sender].temp_hash=0
            x0000000000000000000000000000000000000000;}
    else{ //type 2 and type 3
    ....... //omitted for readability
    }}
}
```

Fig. 6. A portion of the source code of the smart contract.

ABPRE), thus obtaining $s_f = \text{Encrypt}(PK, k_f, \mathcal{B})$. Observe that, at the moment, only u is able to decrypt k_f through $SK_u^{l_i}$, since only u owns the attribute ID_u.

3. User u authenticates with CP (by providing its credential $VC(u)$) and sends the tuple $(s_f, i_{IPFS}(c_f), l_i, Eth_u)$. Through $VC(u)$, CP verifies the linkage between Eth_u and ID_u and stores this mapping (if it does not already exist) in the *User Association Table* in Fig. 7. Moreover, CP stores the tuple $(s_f, i_{IPFS}(c_f), l_i, ID_u)$ in the *VC Table* in Fig. 8. It is to be observed that external users observing the blockchain can know that the same user (with the same Ethereum address) performs the VC Storage operation multiple times. Anyway, they do not know the real identity of the user, so that it does not result in any privacy leakage.
4. To notarize this operation, CP generates a transaction $T_1 = \langle id_{T_1}, Eth_{CP}, Eth_{SC}, data\rangle$ towards the SC. The $data$ field is set to $(type = 1||Eth_u||H(s_f||i_{IPFS}(c_f)||l_i))$, where H denotes a cryptographic hash function (e.g., SHA256) and $type = 1$ represents the type of notarization to perform (in this case, the notarization of the storage of a VC). According to the value of $type$, the proper mapping of SC will be used.
   T_1 triggers the function `startNotarization` of SC which finds in the `StorageMap` (since $type = 1$) the struct `NotarizationData` associated with Eth_u. In such a struct, SC stores the content of $H(s_f||i_{IPFS}(c_f)||l_i)$ in `temp_hash` and the current timestamp in `sent_time`.
   If there is no struct associated with Eth_u in the `StorageMap`, then a new struct `NotarizationData` is created (and associated with ETH_u), where `temp_hash` and `sent_time` are set as above.
5. Similarly, u generates a transaction $T_2 = \langle id_{T_2}, Eth_u, Eth_{SC}, data\rangle$ towards SC, where the $data$ field is set to $(type = 1||H(s_f||i_{IPFS}(c_f)||l_i))$.
   T_2 triggers the function `confirmNotarization`, which finds, in the `Storage Map`, the struct `NotarizationData` associated with Eth_u and checks that `temp_hash` $= H(s_f||i_{IPFS}(c_f)||l_i)$ and that the difference between the current timestamp and the timestamp stored in `sent_time` is less than a certain threshold. The second step ensures that the VC storage occurs within a short period. If both the checks pass, the digest $H(i_{IPFS}(c))$ is added to the `hash_list` array contained in the struct `NotarizationData`. Otherwise, the SC publishes an event to warn CP about the failure of the operation and its originating fault. Finally, SC also resets both `temp_hash` and `sent_time` variables.

User Identity	Ethereum Address
ID_u	Eth_u
...	...

Fig. 7. User Association Table.

ABPRE-Encrypted symmetric key	IPFS index	Label	Identity
s_f	$i_{IPFS}(c_f)$	l_i	ID_u
...	...	...	...

Fig. 8. Verifiable Credential (VC) Table.

The VC storage procedure is summarized in the sequence diagram of Fig. 9.

7.7. Policy setting and policy change

In this phase, u sets a new policy $\tilde{\mathcal{P}}$ for the VCs associated with the label l_i.

1. User u defines the new policy $\tilde{\mathcal{P}}$. We recall that, in our application, the domain of the attributes is organized in a set of trees. Therefore, setting a policy means assigning the bits to the nodes of the trees according to the operative rules described in Section 5. These rules enable the translation of $\tilde{\mathcal{P}}$, possibly containing high-level attributes, into the policy $\mathcal{P}$ containing only low-level attributes. Observe that, since $\mathcal{P}$ is a conjunctive normal form of positive attributes, it is supported by the selected CP-ABPRE scheme [15]. We suppose $\mathcal{P}$ may include a validity time as described in Section 6.3. At this point, u generates the re-encryption key $RK_{l_i} = \text{ReKeyGen}(PK, SK_u^{l_i}, \mathcal{P})$. It allows CP to change the basic policy $\mathcal{B}$ (associated with the VCs with label l_i) into $\mathcal{P}$.
2. User u authenticates with CP (by providing $VC(u)$) and sends (RK_{l_i}, l_i, Eth_u). CP finds an entry (ID_u, Eth_u) in the User Association Table to verify such a mapping. If such an entry does not exist, it is added after verifying $VC(u)$.
3. Successively, CP finds an entry, if any, in the *Policy Table* (represented in Fig. 10) with the pair (ID_u, l_i) and updates the re-encryption key field with RK_{l_i}. If the entry does not exist, then CP creates a new tuple (ID_u, l_i, RK_{l_i}) and inserts it in the Policy Table.
4. The notarization of this phase is analogous to the notarization of the VC storage phase but it involves another mapping. Specifically, CP generates a transaction $T_3 = \langle id_{T_3}, Eth_{CP}, Eth_{SC}, data\rangle$ towards SC, where $data$ is set to $(type = 2||Eth_u||H(l_i||RK_{l_i}))$.
   T_3 triggers the function `startNotarization` of SC, which finds in the `PolicyChangeMap` (since $type = 2$) the struct `NotarizationData` associated with Eth_u. In such a struct, SC stores the content of $H(l_i||RK_{l_i})$ in `temp_hash` and the current timestamp in `sent_time`.
5. User u generates a transaction $T_4 = \langle id_{T_4}, Eth_u, Eth_{SC}, data\rangle$ towards SC, where $data$ is set to $(type = 2||H(l_i||RK_{l_i}))$.
   T_4 triggers the function `confirmNotarization`, which finds, in the `PolicyChangeMap`, the struct `NotarizationData` associated with Eth_u and checks that `temp_hash` $= H(l_i||RK_{l_i})$ and that the difference between the current timestamp and the timestamp stored in `sent_time` is less than a certain threshold. If both

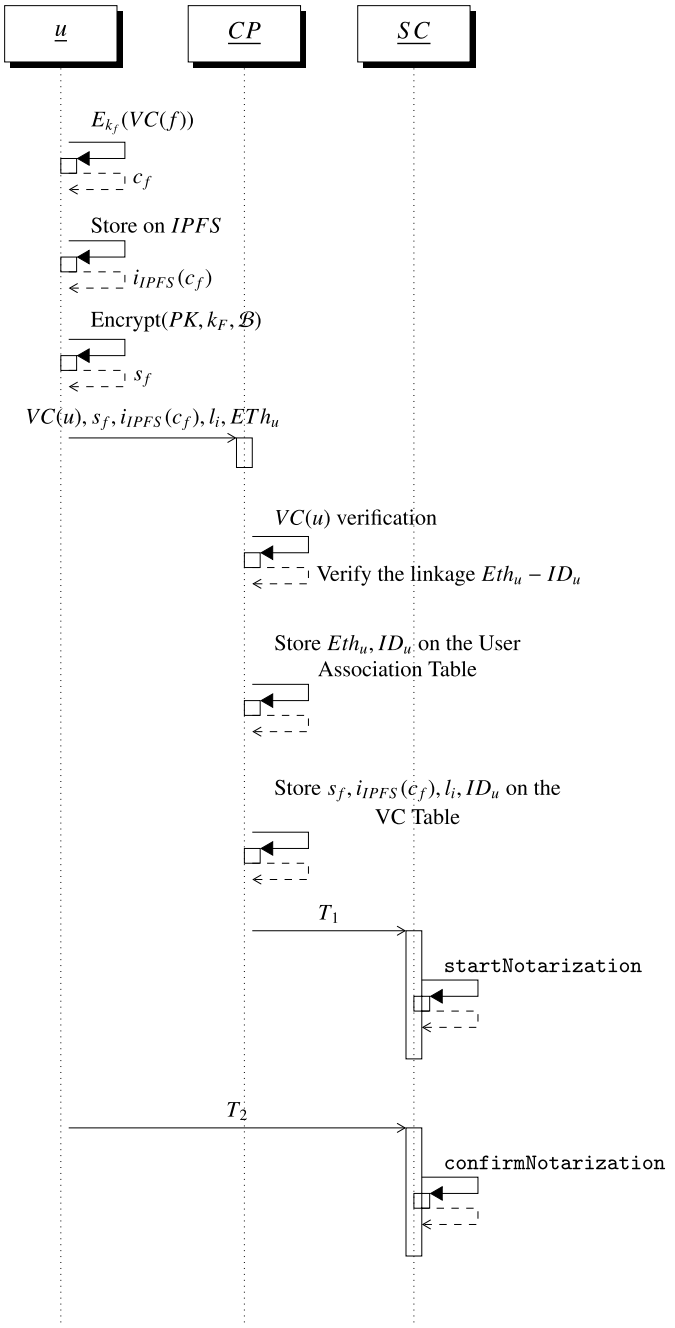


Fig. 9. Verifiable Credential (VC) storage.

Identity	Label	Re-Encryption key
ID_u	l_i	RK_{l_i}
...	...	...

Fig. 10. Policy Table.

the checks pass, then $H(l_i||RK_{l_i})$ is added to the `hash_list` array contained in the struct `NotarizationData`. Otherwise, SC publishes an event to warn CP about the failure of the operation and the originating fault. Finally, SC also resets both `temp_hash` and `sent_time` variables.

The above steps are summarized in the sequence diagram of Fig. 11.

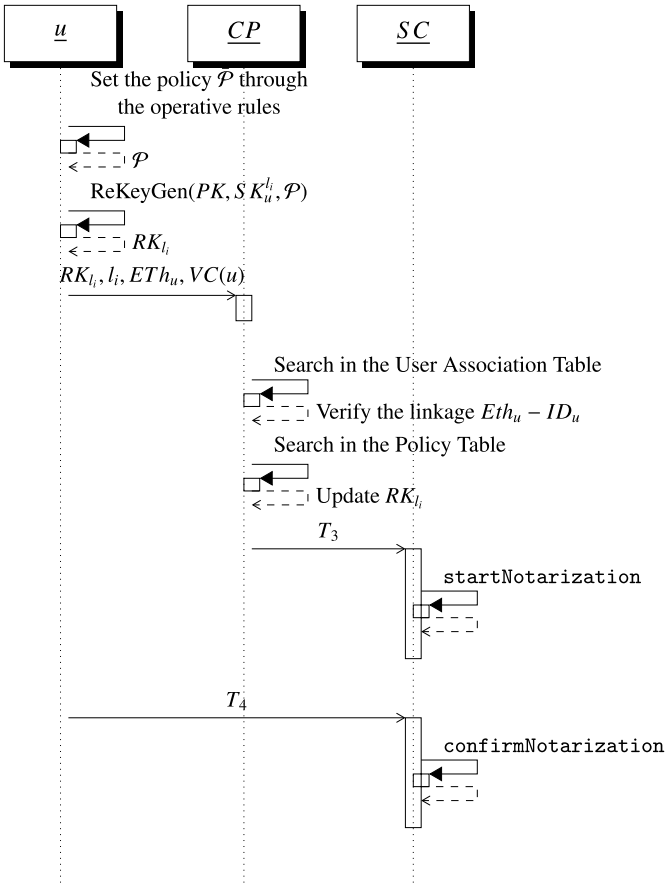


Fig. 11. Policy Setting and Policy Change.

SP Identity	Ethereum Address
ID_{SP}	Eth_{SP}
...	...

Fig. 12. Service Provider Association Table.

7.8. Service request

In this phase, the user u requires a service s to SP, which needs access to the VCs of u.

1. u contacts SP to obtain s. In this step, we do not require any authentication for u. Indeed, until SP does not prove that it satisfies the policies and accesses u's VCs, it does not need to know u's identity. Observe that, in this step, no response from SP is needed; thus, no bidirectional connection has to be established. To avoid a possible linkage between the IP address of u and his/her real identity, the request of u can be performed behind a proxy, or through anonymous standard services such as the VPN and Tor [72].
2. SP authenticates with CP by providing $VC(SP)$. Through it, CP verifies the mapping between the identity ID_{SP} and Eth_{SP} and stores this mapping in the *Service Provider Association Table* in Fig. 12.
3. Meanwhile, u authenticates with CP and confirms that they have required the service s to SP. Furthermore, u sends a set of labels $L_u^s \subseteq L_u$ to CP. L_u^s identifies the VCs that u wants to send to SP to obtain the service s. Observe that, u does not know a priori if SP is authorized to access their VCs. Therefore, it is possible that SP does not satisfy all the policies associated with the labels in L_u^s.
4. For each $l_i \in L_u^s$, CP retrieves the re-encryption key RK_{l_i} from the Policy Table. Furthermore, for each $l_i \in L_u^s$, it retrieves all the tuples in the form $(s_f, i_{IPFS}(c_f), l_i, ID_u)$ from the VC Table (i.e.,

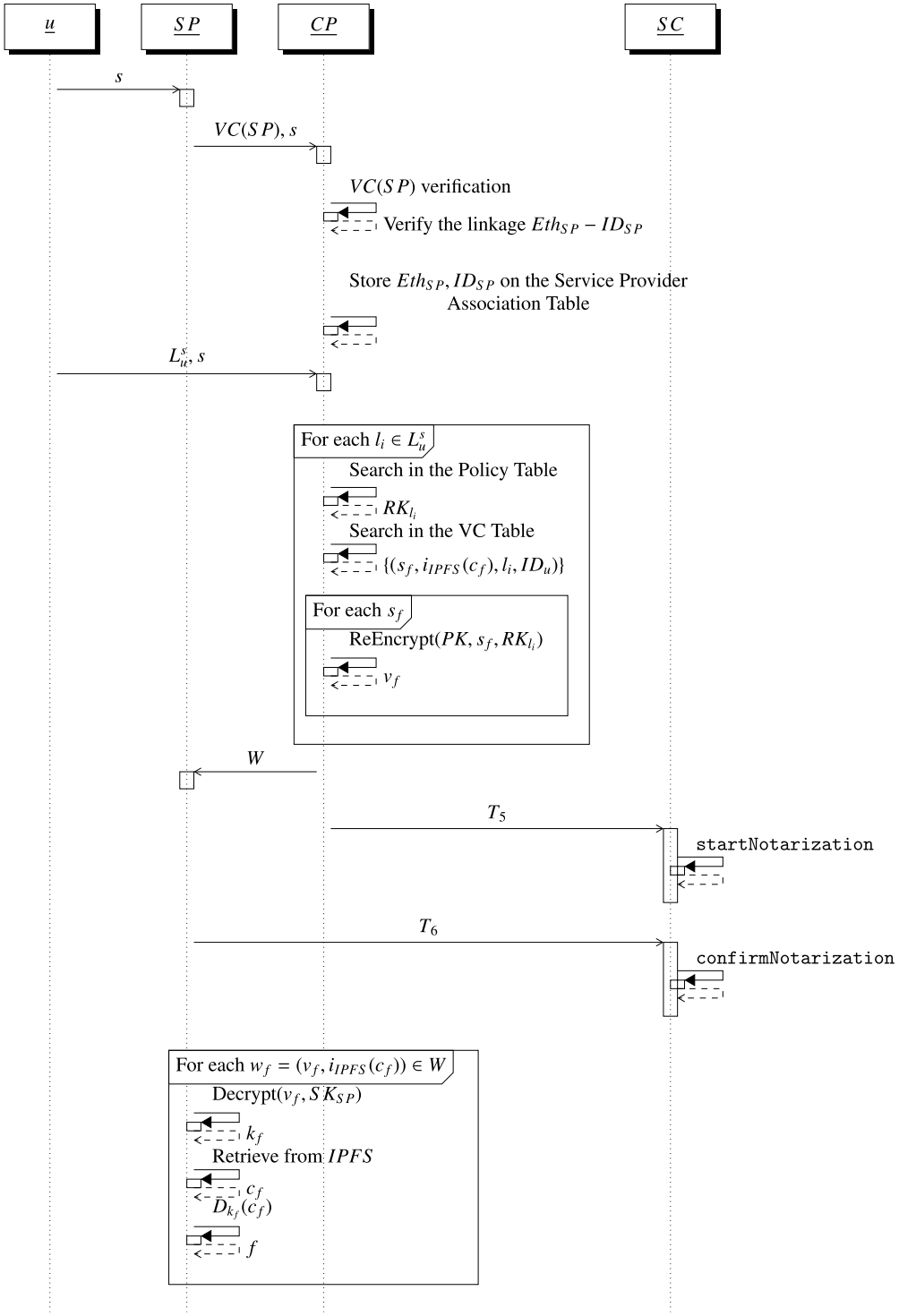


Fig. 13. Service request.

all the tuples associated with l_i of the user u). For each of them, CP invokes $v_f = \text{ReEncrypt}(PK, s_f, RK_{l_i})$, where v_f represents a symmetric key k_f encrypted under a policy $\mathcal{P}$ associated with the re-encryption key RK_{l_i}. We denote by w_f the pair $(v_f, i_{IPFS}(c_f))$ and by W the set of all the w_f (obtained for all the $l_i \in L_u^s$). CP forwards the set W to SP.

5. For the notarization, again, we follow the same procedure as the previous sections but with SP in place of u. CP generates a transaction $T_5 = \langle id_{T_5}, Eth_{CP}, Eth_{SC}, data \rangle$ towards SC, where $data$ is set to $(type = 3 || Eth_{SP} || H(W))$.
   T_5 triggers the function `startNotarization` of SC, which finds in the `FileAccessMap` (since $type = 3$), the struct `NotarizationData` associated with Eth_u. In such a struct, SC stores the content of $H(W)$ in `temp_hash` and the current timestamp in `sent_time`.
6. SP generates a transaction $T_6 = \langle id_{T_6}, Eth_{SP}, Eth_{SC}, data \rangle$ towards SC, where $data$ is set to $(type = 3 || H(W))$.
   T_6 triggers the function `confirmNotarization`, which finds in the `FileAccessMap`, the struct `NotarizationData` associated with Eth_{SP} and checks that `temp_hash` $= H(W)$ and that the dif-

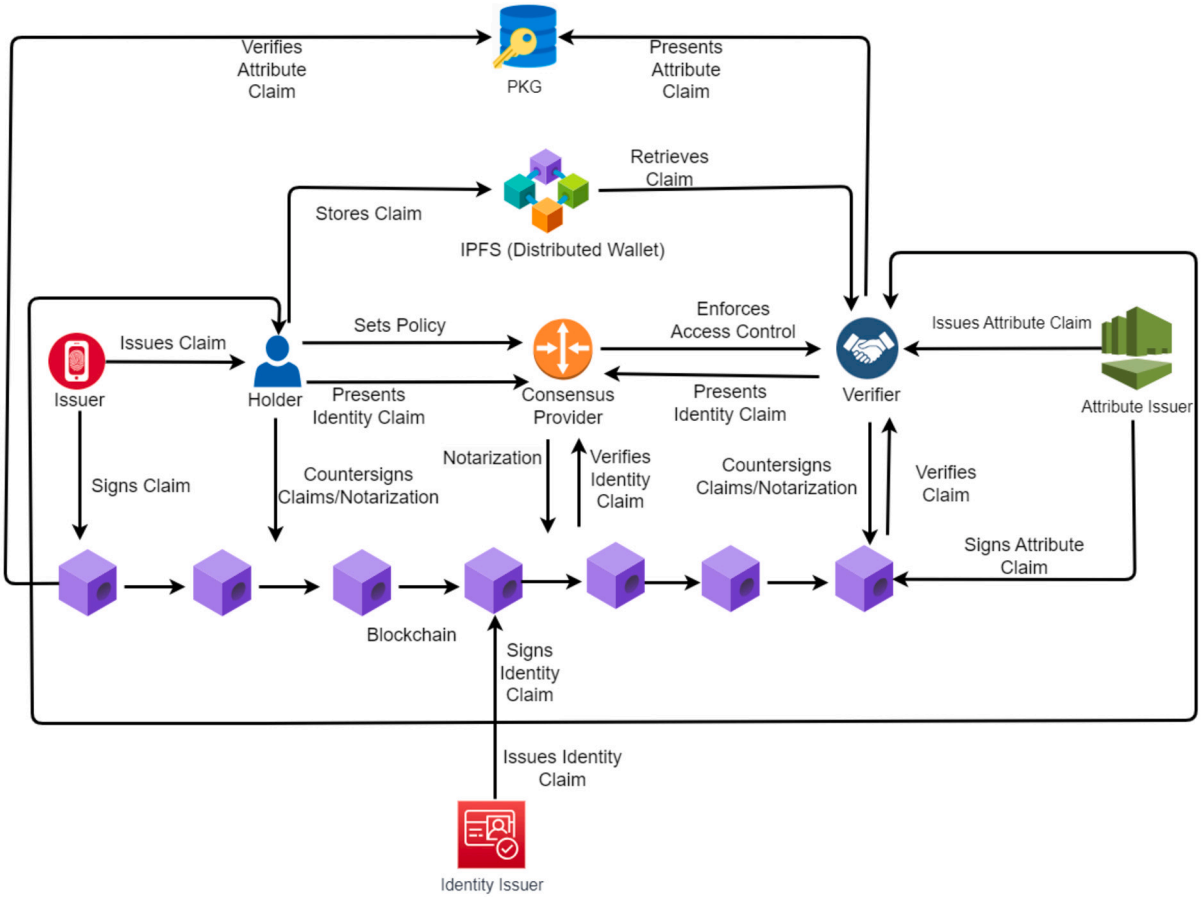


Fig. 14. Extended Self-Sovereign Identity (SSI) paradigm.

ference between the current timestamp and the timestamp stored in `sent_time` is less than a certain threshold. If both the checks pass, $H(W)$ is added to the `hash_list` array contained in the struct `NotarizationData`. Otherwise, SC publishes an event to warn CP of the failure of the operation and the originating fault. Finally, SC also resets both `temp_hash` and `sent_time` variables.

7. Once W is obtained, for each pair $w_f = (v_f, i_{IPFS}(c_f)) \in W$, SP invokes Decrypt(v_f, SK_{SP}). If SP satisfies the policy $\mathcal{P}$ associated with v_f, then it retrieves the symmetric key k_f and, through $i_{IPFS}(c_f)$, c_f from $IPFS$. Finally, SP accesses to $VC(f) = D_{k_f}(c_f)$.

The service request is summarized in the sequence diagram in Fig. 13.

We want to highlight the cryptographic proofs stored on the blockchain. They are (1) the signature of the issuers (as in the standard SSI-approach) and (2) the digests of some information. Specifically, concerning (2), as described in the protocol, these digests are as follows:

- $H(s_f||i_{IPFS}(c_f)||l_i)$, where s_f is an ABE encryption of a symmetric key, $i_{IPFS(c_f)}$ is an IPFS index, and l_i is a file label;
- $H(l_i||RK_{l_i})$, where l_i is a file label and RK_{l_i} is a re-encryption key;
- $H(W)$, where W is a set of pairs, containing the ABE re-encryption of a symmetric key and an IPFS index.

As a final remark, we want to highlight that the data stored on the blockchain do not introduce privacy leakage. Indeed, concerning the traditional issuing of SSI credentials (Section 7.4), only cryptographic proofs (i.e., the signature of the issuers) are included in the blockchain. On the other hand, as better highlighted in Section 9, the additional steps (with respect to standard SSI interactions) introduced by our approach, require the storage on the smart contract of some cryptographic digests of some potentially sensitive information. However, these digests are not reversible without the collaboration of II and CP, which is just the accountability requirement we want to achieve.

To conclude this section, in Fig. 14, we show that the standard SSI paradigm (represented in Fig. 1) is extended by the introduction of our proposal.

8. Prototype and experiment

In this section, we describe a prototype (developed in Java) of the solution proposed in Section 7. Furthermore, we measure its performance. Finally, we discuss the motivation leading to the choice of a public Ethereum blockchain.

8.1. Prototype

The source code of the prototype is available at https://github.com/vincenzodeangelisrc/Self-Sovereign-Identity-Solution-UNIRC.

Our implementation consists of 10 modules.

The `LiangScheme` module includes our Java implementation of the scheme [15]. We implemented it from scratch by relying on the JPBC library [73] for the pairing function [43].

The `Ethereum` and `IPFS` modules are two wrappers that leverage the Infura [74] APIs, enabling connectivity with the Ethereum and IPFS networks, respectively.

The `Issuer`, `Identity Issuer`, and `Attribute Issuers` modules implement the functions of an issuer in the standard SSI approach. They are implemented as web applications reachable through HTTP requests. Our implementation relies on the Trinsic Ecosystems [75], implementing the Sovrin Framework [1]. This is a reference technol-

Fig. 15. Identity credential released by the `Identity Issuer`.

Fig. 16. Private Key Generator (PKG) interface for generating a private key for a Service Provider SP.

ogy adhering to the main SSI standards in terms of the format of the verifiable credentials (compliant with W3C specifications [12,13]) and ToIP stack [76]. Each of these three modules releases a verifiable credential in JSON-LD format. An example of a credential released by the `Identity Issuer` module is reported in Fig. 15.

It includes the real identity of the user and their Ethereum address as attributes.

The main modules of our application are `PKG`, `User`, `CP`, and `SP`. We describe them in detail.

The `PKG` module is composed of a stand-alone application (with GUI) to implement the Setup function of the scheme [15] and generates PK and MSK. Furthermore, it includes two web interfaces allowing users and service providers to retrieve private keys.

For example, in Fig. 16, we can see the web interface by which a service provider declares its attributes and provides its verifiable credential (containing all the attributes) to retrieve the private key from the PKG. The verification of the credential is performed through the Trinsic API.

The `User` module is composed of two applications (with GUI). The first implements the user-side component of the VC Storage operation presented in Section 7.6. Its GUI is displayed in Fig. 17.

In detail, in the yellow panel, the user selects the VC to store and the label to be associated with this VC. By pressing the button **Store**, this application generates an AES128 symmetric key, encrypts the VC, and stores it in the IPFS. Then, the symmetric key is encrypted with CP-ABPRE. At the end of the procedure, the application generates the **StorageFile.txt**, which contains all the information to be sent to CP

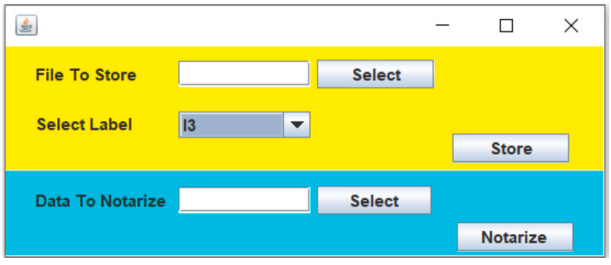


Fig. 17. User GUI to store a Verifiable Credential (VC).

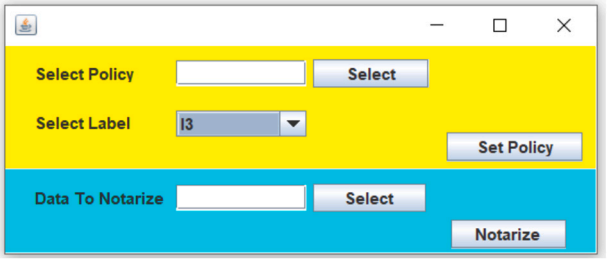


Fig. 18. User GUI to set a policy for a Verifiable Credential (VC).

(see the next module). In particular, it includes (s_f, i_{IPFS}, l_i), as described in Section 7.6. Furthermore, such a file will be given as input to the user application (light blue panel in Fig. 17) to be notarized on the blockchain.

The other user-side application is similar, and it allows the users to set a policy for the VCs with a given label. It outputs the **PolicyFile.txt**, including (rk_{l_i}, l_i), as explained in Section 7.7. Its GUI is displayed in Fig. 18.

The core module of our proposal is `CP`. It includes an application (with GUI) for deploying smart contracts and four web interfaces.

Two web interfaces are exploited by the user to upload the **StorageFile.txt** and **PolicyFile.txt** in order to trigger the CP-side part of the VC Storage and Policy Setting/Change operations, respectively. Fur-

Fig. 19. Ciphertext-Policy (CP) interface for the Verifiable Credential (VC) Storage operation.

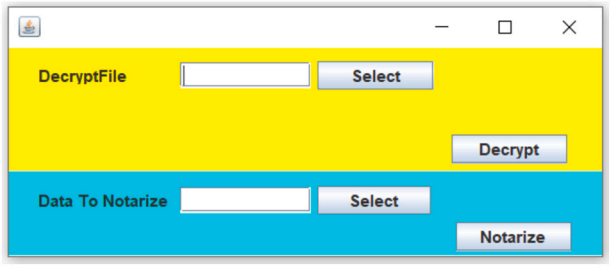


Fig. 20. Service Provider (SP) GUI to retrieve the Verifiable Credentials (VCs).

thermore, the user also provides their Ethereum address for notarizing the operation and its credential $VC(u)$. For example, the web interface for the VC Storage operation is reported in Fig. 19.

When CP receives the **StorageFile.txt**, it performs all the operations described in Section 7.6, including the verification of the credential (through the Trinsic API) and the notarization of the received information. This notarization has to be confirmed by the user, as reported in the confirmation message of Fig. 19, by uploading the **StorageFile.txt** in the light blue panel of Fig. 17.

The same happens for the Policy Set/Change web interface.

The other two web interfaces provided by the `CP` module are used by service providers and users to implement the Service Request operation. At the end of the procedure, the service provider receives the **DecryptFile.txt** representing the set W of Section 7.8, containing all the information to access the VCs of the users.

Finally, the `SP` module is composed of an application (with GUI displayed in Fig. 20) allowing a service provider to retrieve the VCs. In particular, it uploads the **DecryptFile.txt** on the yellow panel and the application decrypts the CP-ABPRE encrypted symmetric keys, retrieves the encrypted VCs from the IPFS, and decrypts them. Then, through the Trinsic API, the VCs are verified. As for the User applications, the light blue panel allows the notarization of the **DecryptFile.txt**.

8.2. Experiments

In this section, we evaluate the performance of our prototype with respect to temporal and economic considerations.

We measured the time needed to perform the operations required by our protocol. To perform such measurements, we use a personal computer equipped with a 1.8 GHz Intel i7-8850 CPU and 16 GB of RAM. We did not consider the user-dependent times (e.g., the time to select a VC or a policy).

Furthermore, we measured the time needed to deploy the smart contract and invoke its functions.

These results are reported in Table 2. We also report, in Fig. 21, a timeline that includes all actors (and the blockchain) to complete the execution of our protocol. For graphical reasons, the timeline is not to scale.

Finally, we report the costs in terms of ETH and US dollars (in May 2023) for the deployment of the smart contract and invocation of the relative functions in Table 3. Since they depend on the state of the Ethereum network, we used *Ropsten* [77] as a testnet since it is the most similar to the real Ethereum network.

Table 2
Time to perform operations.

Operation	Time (seconds)
PKG Setup	0.7–0.8
PKG User Keys Generation (20 labels)	45–65
PKG SP Key Generation (10 attributes)	10–12
Smart Contract Deploy	15–25
StorageFile.txt creation (AES Encryption, IPFS Storage, ABE encryption)	4–5
VC Storage CP-side (User Association Table and VC Table update)	0.5–1
VC Storage Notarization CP-side (`startNotarization (type1)`)	15–25
VC Storage Notarization User-side (`confirmNotarization (type1)`)	15–25
PolicyFile.txt creation ABE Re-Encryption Key generation)	5–6
Policy Set/Change CP-side (Policy Table Update)	0,5–1
Policy Set/Change Notarization CP-side (`startNotarization (type2)`)	15–25
Policy Set/Change Notarization User-side (`confirmNotarization (type2)`)	15–25
Service Request CP-side (Re-encryption of a single symmetric key)	2–3
VC Access SP-side (ABE-Decryption, IPFS download, AES decryption)	4–6
Service Request Notarization CP-side (`confirmNotarization (type3)`)	15–25
Service Request Notarization SP-side (`confirmNotarization (type3)`)	15–25

Note: PKG—Private Key Generator, SP—Service Provider, AES—Advanced Encryption Standard, IPFS—InterPlanetary File System, ABE—Attribute-Based Encryption, VC—Verifiable Credential, CP—Ciphertext-Policy.

Table 3
Cost in terms of ETH and USD (in May 2023) to perform the operations. We applied the default gas-price.

Operation	Cost (ETH / US dollars)
Smart Contract Deployment	0.0038728149/ 7.13
VC Storage Notarization CP-side (`startNotarization (type1)`)	0.0002839865/0.53
VC Storage Notarization User-side (`confirmNotarization (type1)`)	0.0002219084/0.41
Policy Set/Change Notarization CP-side (`startNotarization (type2)`)	0.0002840931/0.53
Policy Set/Change Notarization User-side (`confirmNotarization (type2)`)	0.0002219904/0.41
Service Request Notarization CP-side (`confirmNotarization (type3)`)	0.0002841874/0.53
Service Request Notarization SP-side (`confirmNotarization (type3)`)	0.0002221954/0.41

Note: VC—Verifiable Credential, CP—Ciphertext-Policy, SP—Service Provider.

We observe that all operations, apart from smart contract deployment, require less than 0.55 USD. Moreover, `startNotarization` operations require essentially the same cost regardless of the type (i.e., 0.41 USD). Similarly, the `confirmNotarization` operations also require the same cost regardless of the type (i.e., 0.53 USD). The deployment of the smart contract requires 7.13 US dollars, but it is performed just once.

Actually, these prices may vary over time according to two (correlated) factors:

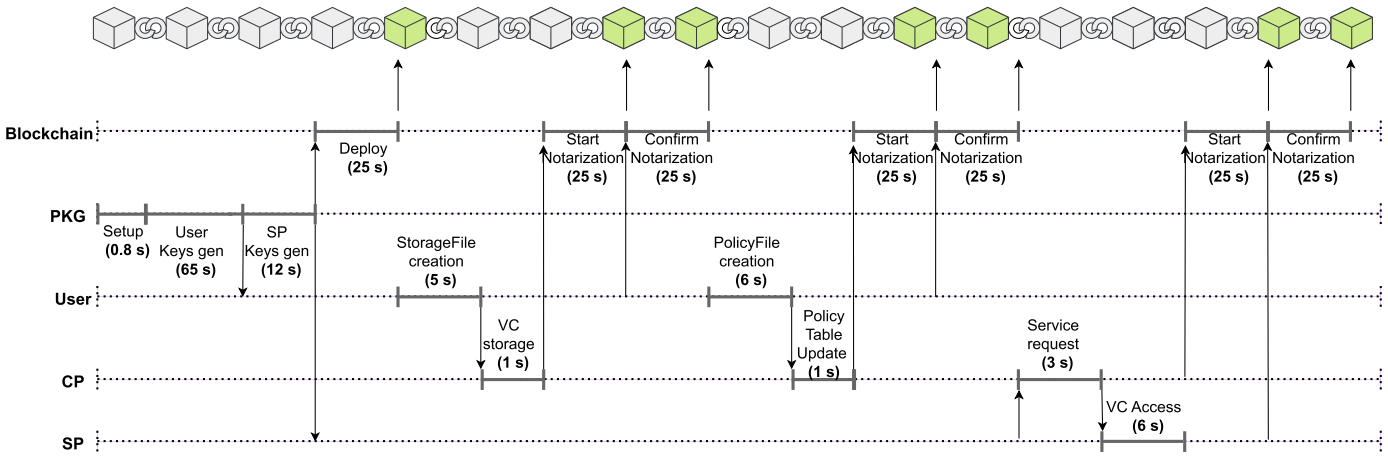


Fig. 21. Timeline of execution of our protocol.

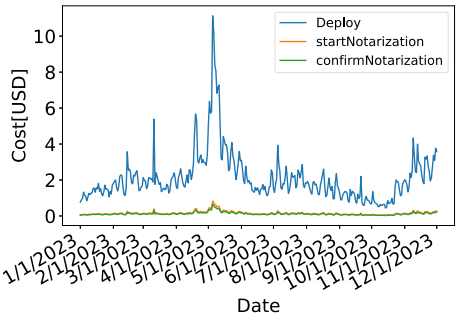


Fig. 22. USD cost over time for smart contract management.

1. The ETH to USD exchange rate;
2. The current gas price to invoke the functions and deploy the smart contract.

In Fig. 22, we present the cost trends from January 1, 2023, to December 1, 2023, indicating the deployment cost of the smart contract and the costs associated with the `startNotarization` and `confirmNotarization` operations.

We observe that no appreciable difference exists for `startNotarization` and `confirmNotarization` operations. The deployment cost reaches a peak of about 10 USD, but it is below 6 USD for the majority of the time.

Based on these observations, we can conclude that our solution is easily implementable, fast to execute, and cheap.

8.3. Choice of Ethereum blockchain

In this paper, we based our solution on the public Ethereum blockchain. In general, when dealing with public blockchains, one drawback is associated with the cost. It might be worthwhile to consider implementing our solution on a permissioned blockchain, such as Hyperledger Fabric (HLF), to achieve cost savings.

However, for research purposes, we decided to place ourselves at the greatest disadvantage (public blockchain) to prove that, even in this case, the costs are affordable (we analyzed this in Section 8.2) and that we do not suffer from confidentiality problems (see Section 9) inherent to the public nature of Ethereum (in other words, we proved that it is possible to find suitable cryptographic countermeasures to work in a public environment).

Our design can be easily moved to HLF (with proper adaptations), but we argue that having demonstrated that the solution is feasible even in the context of public blockchains is an added value of our research. Moreover, in a practical adoption of our solution, the choice of Ethereum instead of HLF (public vs. permissioned) may be required, due to the higher level of assurance in terms of immutability given by public blockchains. We recall that, indeed, that in our solution, the role of lawful notarization is central. It is widely accepted that this function is better accomplished with public blockchains instead of permissioned ones. Moreover, in a practical application of our solutions, it is not clear which parties should play the "super" role of the orderers and endorsers nodes required by HLF in such a way that collusion is improbable and there is sufficient independence with respect to Trusted Third Party (TTP).

Based on the above considerations, we can claim that Ethereum represents a valid option in terms of security and efficiency for practical adoption.

9. Security analysis

In this section, we provide a security analysis of the protocol presented in Section 7.

We start by introducing the following assumptions.

A1: The cryptographic primitives used in the protocol are secure.
A2: PKG is a TTP.
A3: II and AIs are *honest-but-curious*, i.e., they perform honestly the steps of the protocol, but they try to steal information about users and service providers. In addition, they can collude with each other and with other entities to gather information (but without deviating from protocol specifications).
A4: CP is *honest-but-curious* as defined in **A3**, but no collusion is allowed for CP.

The other actors (users and service providers) are considered malicious.

We discuss the plausibility of the assumptions.

A1 is a basic assumption. Specifically, it simply means that the CP-ABPRE scheme, the cryptographic hash function H, the symmetric encryption scheme (E, D), and the asymmetric encryption/signature schemes used in the Ethereum and the IPFS are robust. Obviously, these primitives are adopted in several real-life contexts, and their security is well-documented in the literature.

Assumption **A2** is a standard assumption done in any existing CP-ABPRE scheme. The presence of this TTP is necessary for the delivery of private keys. However, it cannot forge any credentials. We argue that in the scenario that we are considering in our paper, in which the identity of citizens is managed and government parties can be involved, it is realistic to assume that some parties can play a similar role.

Concerning **A3**, observe that II and AIs are entities already included in the SSI architecture as TTPs. Since in real-life applications,

II and AIs may (also partially) coincide, we assume that they collude *passively* (by exchanging information) between them. Furthermore, we allow them to collude passively with other entities.

However, **A3** also requires that II and AIs do not collude *actively* (by deviating from the protocol) with other entities. For example, we exclude the case in which an AI releases a fake attribute to a service provider to satisfy a policy. Obviously, in traditional SSI, an issuer releasing a fake VC allows the user to access a service even though they are not authorized. Therefore, some level of trust is normally required of issuers.

Finally, regarding **A4**, again, it is the standard assumption of all the CP-ABPRE solutions. As discussed in Section 4, the presence of a proxy is a price to pay to achieve our security requirements.

Now, we discuss the Security Properties guaranteed by our solution.

From now on, we use the term *Operation* to refer to one of the following operations: VC Storage, Policy Setting/Change, or Service Request. Furthermore, we use the term *Information* (associated with the Operation) to refer to the content stored (in hashed form) on the blockchain during a given Operation. Specifically, the Information associated with the VC Storage Operation is $(s_f||i_{IPFS}(c_f)||l_i)$, the Information associated with the Policy Setting/Change Operation is $(l_i||RK_{l_i})$, and the Information associated with the Service Request Operation is W.

We guarantee the following Security Properties.

SP1: No entity, except for a service provider satisfying the policy chosen by a user, can access the content of the VCs stored by the user. (**Confidentiality**).

SP2: No entity, except for CP, is able to know if a user or a service provider performs an Operation(**Privacy**).

SP3: The collaboration between CP and IP allows us to prove that an Operation (with the associated Information) has been performed by a user or a service provider. On the other hand, a user or a service provider can prove that an Operation (with the associated Information) is acknowledged by CP (**Accountability**).

SP4: No Operation not actually performed can be attributed to a user, a service provider, or CP (**No Fake Attribution**).

We discuss the meaning of Security Properties.

SP1 is about the confidentiality of the VCs. Only an authorized service provider can access the VCs of a user.

SP2 is about privacy. It is desirable that no entity can know if a user or a service provider performs a given Operation. This property is guaranteed for all entities except for CP.

This limitation is compensated by the benefits in terms of reduction of the computational and storage resources required client-side and the accountability guarantees that the introduction of CP provides (see Section 1).

SP3 is important when it is necessary to prove to an agent authorized by the law that a given Operation (with the associated Information) is performed by a user or a service provider. In this case, the collaboration between II and CP can disclose the identity of the user. **SP3** also regards the possibility of proving to the agent that CP has taken charge of a given Operation.

Finally, **SP4** is related to **SP3** and regards the fact that an Operation cannot be attributed to a user who did not perform it. For example, CP cannot invent that a service provider has the required access to some users' VCs.

At this point, we show that the above properties are guaranteed in our solution.

We start from **SP1**. Due to Assumption **A2**, we do not consider PKG as an adversary.

Consider a $VC(f)$ stored by the user u with policy $\mathcal{P}$. Due to Assumption **A1**, the symmetric encryption scheme cannot be broken; thus, the only way for the adversary to access $VC(f)$ is to recover the symmetric key k_f. Such a key is encrypted by u with CP-ABPRE under the Basic Policy $\mathcal{B}$ obtaining s_f that is sent to CP. $\mathcal{B}$ requires the identity of u to be satisfied and since the CP-ABPRE scheme is secure by **A1**, only u can decrypt s_f. Therefore, there is no way for the adversary to recover k_f from s_f. Due to Assumption **A4**, CP performs legally the steps of the protocol, then s_f is re-encrypted under the policy $\mathcal{P}$, obtaining v_f that is sent to the service provider SP. Again, due to **A1**, v_f can be decrypted by SP only if it owns a private key associated with attributes that satisfy $\mathcal{P}$. Due to Assumption **A3**, the attribute providers do not collude actively with SP by certificating fake attributes. Therefore, such a private key can be obtained only if SP really owns such attributes. Thus, **SP1** holds.

Now, we consider **SP2.** It is easy to see that any attacker, except CP, learns nothing about the Operations performed by users and service providers. Indeed, the VC storage and Policy Setting/Change Operations performed by a user only involve CP. Similarly, the Service Request Operation performed by the service provider (on the request of a user) only involves CP too. However, the Information associated with each Operation is stored (in hashed form) on the blockchain. Therefore, an external observer (different from CP) can only see the DID (Ethereum address) of the user or the service provider performing an Operation but it does not know the mapping between such a DID and the real identity. Furthermore, since the CP-ABPRE scheme is probabilistic, the elements generated by this scheme and stored on the blockchain in hashed form (specifically, s_f, RK_{l_i}, and the first element v_f of each pair $w_f \in W$) are not guessable by any attacker (different from CP). This shows that **SP2** holds.

Concerning **SP3**, when the user/service provider performs an Operation, it generates a transaction from its Ethereum address to the smart contract, including the Information associated with this Operation in hashed form. CP can disclose this Information to an agent authorized by the law, and II can confirm the mapping between the Ethereum address and the real identity of the user/service provider. Specifically, consider the VC Storage Operation. u generates the transaction T_2 from Eth_u, including $H(s_f||i_{IPFS}(c_f)||l_i)$. CP can disclose $(s_f||i_{IPFS}(c_f)||l_i)$ and this proves that, since the signature scheme of Ethereum is secure by Assumption **A1**, the user owning Eth_u performs the above Operation. Finally, II confirms the real identity associated with Eth_u. The same reasoning applies to the other two Operations (Policy Setting/Change and Service Request). On the other hand, since CP confirms the Operations by storing the associated Information on the blockchain, the user/service provider can disclose such Information to prove that CP has acknowledged it. This proves **SP3**.

Finally, we consider **SP4**. Due to **A1**, no attacker can generate a transaction starting from the Ethereum address of another entity. Then, we consider the case in which the attacker uses their own Ethereum address to attribute an Operation to a user, a service provider, or CP. Since the Ethereum address of CP is public, no fake attribution can be done on CP. Regarding users and service providers, when II is involved in confirming the real identity of the user/service provider associated with a given Ethereum address, due to Assumption **A3**, it does not disclose a fake mapping. Therefore, **SP4** also holds.

This concludes the security analysis.

10. Conclusion

In this paper, we extend the SSI paradigm by including a CP-ABPRE mechanism enabling the control that the verifier (i.e., the service provider) has the right to access the claims/credentials submitted by the holder. This is a missed point in the current SSI solutions, also regarding what the international (European) standards include. The contribution of the paper is to highlight the importance of covering this gap and an effective way to achieve this result. We obtain a more complex SSI architecture that gives the user an increased power of control over their information. The fact that our solution requires an intermediary (i.e., the consensus provider) should not be viewed as something that diverges from the standard disintermediated SSI paradigm. Indeed, it has not a conceptual role in the credential-issuing/presenting process, but

only a role allowing the user to be relieved by storage and computation. The consensus provider does not keep any personal data of the user (only some metadata) so that the fundamentals of the SSI paradigms basically remain. In contrast, we have an advantage that leverages the blockchain that the SSI architecture generally includes per se, which is a certain level of accountability, and is very useful in real-life contexts in which possible legal disputes can arise. This advantage would not have been reached without the presence of the consensus provider.

More importantly, to test the applicability of our solution in real-life contexts, we provide a prototype of the solution and evaluate its performance in terms of computational time and costs to interact with the blockchain.

CRediT authorship contribution statement

Francesco Buccafurri: Conceptualization, Formal analysis, Investigation, Methodology, Project administration, Supervision, Validation, Writing – original draft, Writing – review & editing. **Vincenzo De Angelis:** Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Resources, Software, Validation, Writing – original draft, Writing – review & editing. **Roberto Nardone:** Conceptualization, Formal analysis, Investigation, Methodology, Validation.

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.

Acknowledgements

This work was partially supported by the project STRIDE included in the Spoke 5 (Cryptography and Distributed Systems Security) of the Research and Innovation Program PE00000014-H73C22000880001, “SEcurity and RIghts in the CyberSpace (SERICS)”, under the National Recovery and Resilience Plan, funded by the European Union, NextGenerationEU.

References

[1] A. Tobin, D. Reed, The inevitable rise of self-sovereign identity, https://sovrin.org/wp-content/uploads/2018/03/The-Inevitable-Rise-of-Self-Sovereign-Identity.pdf, 2016. (Accessed 2 April 2024).

[2] N. Naik, P. Jenkins, Uport open-source identity management system: an assessment of self-sovereign identity and user-centric data platform built on blockchain, in: Proceedings of the 2020 IEEE International Symposium on Systems Engineering (ISSE), IEEE, 2020, pp. 1–7, https://doi.org/10.1109/ISSE49799.2020.9272223.

[3] M.P. Bhattacharya, P. Zavarsky, S. Butakov, Enhancing the security and privacy of self-sovereign identities on hyperledger indy blockchain, in: Proceedings of the 2020 International Symposium on Networks, Computers and Communications (ISNCC), IEEE, 2020, pp. 1–7, https://doi.org/10.1109/ISNCC49221.2020.9297357.

[4] Q. Stokkink, J. Pouwelse, Deployment of a blockchain-based self-sovereign identity, in: Proceedings of the 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, 2018, pp. 1336–1342, https://doi.org/10.1109/Cybermatics_2018.2018.00230.

[5] D. van Bokkem, R. Hageman, G. Koning, et al., Self-sovereign identity solutions: the necessity of blockchain technology, arXiv, 2019, preprint, arXiv:1904.12816.

[6] V.C. Hu, D. Ferraiolo, R. Kuhn, et al., Guide to attribute based access control (ABAC) definition and considerations (draft), NIST Spec. Publ. 800 (162) (2013) 1–54, https://doi.org/10.6028/NIST.SP.800-162.

[7] J. Bethencourt, A. Sahai, B. Waters, Ciphertext-policy attribute-based encryption, in: Proceedings of the 2007 IEEE Symposium on Security and Privacy (SP ’07), IEEE, 2007, pp. 321–334, https://doi.org/10.1109/SP.2007.11.

[8] X. Liang, Z. Cao, H. Lin, et al., Attribute based proxy re-encryption with delegating capabilities, in: Proceedings of the 4th International Symposium on Information, Computer, and Communications Security, ACM, 2009, pp. 276–286, https://doi.org/10.1145/1533057.1533094.

[9] F. Buccafurri, V. De Angelis, S. Lazzaro, A blockchain-based framework to enhance anonymous services with accountability guarantees, Future Internet 14 (8) (2022) 243, https://doi.org/10.3390/fi14080243.

[10] F. Buccafurri, V. De Angelis, G. Lax, et al., An attribute-based privacy-preserving ethereum solution for service delivery with accountability requirements, in: Proceedings of the 14th International Conference on Availability, Reliability and Security (ARES ’19), ACM, 2019, pp. 1–6, https://doi.org/10.1145/3339252.3339279.

[11] G. Wood, Ethereum: a secure decentralised generalised transaction ledger, https://ethereum.github.io/yellowpaper/paper.pdf, 2014. (Accessed 2 April 2024).

[12] Decentralized identifiers (DIDs) v1.0, https://www.w3.org/TR/did-core. (Accessed 2 April 2024).

[13] Verifiable credentials data model v1.1, https://www.w3.org/TR/vc-data-model. (Accessed 2 April 2024).

[14] F. Buccafurri, V. De Angelis, Self-sovereign management of privacy consensus using blockchain, in: Proceedings of the 15th International Conference on Web Information Systems and Technologies - WEBIST, INSTICC, SciTePress, 2019, pp. 426–431, https://doi.org/10.5220/0008493804260431.

[15] K. Liang, L. Fang, W. Susilo, et al., A ciphertext-policy attribute-based proxy re-encryption with chosen-ciphertext security, in: Proceedings of the 2013 5th International Conference on Intelligent Networking and Collaborative Systems, IEEE, 2013, pp. 552–559, https://doi.org/10.1109/INCoS.2013.103.

[16] P. Voigt, A. von dem Bussche, The Eu General Data Protection Regulation (GDPR), A Practical Guide, 1st ed., Springer, Cham, 2017.

[17] N. Naik, P. Jenkins, Your identity is yours: take back control of your identity using gdpr compatible self-sovereign identity, in: Proceedings of the 2020 7th International Conference on Behavioural and Social Computing (BESC), IEEE, 2020, pp. 1–6, https://doi.org/10.1109/BESC51023.2020.9348298.

[18] G. Kondova, J. Erbguth, Self-sovereign identity on public blockchains and the GDPR, in: Proceedings of the 35th Annual ACM Symposium on Applied Computing, ACM, 2020, pp. 342–345, https://doi.org/10.1145/3341105.3374066.

[19] M. Shuaib, S. Alam, M.S. Alam, et al., Self-sovereign identity for healthcare using blockchain, Mater. Today Proc. 81 (2023) 203–207, https://doi.org/10.1016/j.matpr.2021.03.083.

[20] B. Houtan, A.S. Hafid, D. Makrakis, A survey on blockchain-based self-sovereign patient identity in healthcare, IEEE Access. 8 (2020) 90478–90494, https://doi.org/10.1109/ACCESS.2020.2994090.

[21] P.C. Bartolomeu, E. Vieira, S.M. Hosseini, et al., Self-sovereign identity: use-cases, technologies, and challenges for industrial IoT, in: Proceedings of the 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), IEEE, 2019, pp. 1173–1180, https://doi.org/10.1109/ETFA.2019.8869262.

[22] A. Mühle, A. Grüner, T. Gayvoronskaya, et al., A survey on essential components of a self-sovereign identity, Comput. Sci. Rev. 30 (2018) 80–86, https://doi.org/10.1016/j.cosrev.2018.10.002.

[23] Q. Stokkink, D. Epema, J. Pouwelse, A truly self-sovereign identity system, arXiv, 2020, preprint, arXiv:2007.00415.

[24] A. Grüner, A. Mühle, C. Meinel, An integration architecture to enable service providers for self-sovereign identity, in: 2019 IEEE 18th International Symposium on Network Computing and Applications (NCA), IEEE, 2019, pp. 1–5, https://doi.org/10.1109/NCA.2019.8935015.

[25] L. Stockburger, G. Kokosioulis, A. Mukkamala, et al., Blockchain-enabled decentralized identity management: the case of self-sovereign identity in public transportation, Blockchain Res. Appl. 2 (2) (2021) 100014, https://doi.org/10.1016/j.bcra.2021.100014.

[26] S. Nakamoto, Bitcoin: a peer-to-peer electronic cash system, http://www.bitcoin.org/bitcoin.pdf, 2008. (Accessed 2 April 2024).

[27] Z. Zheng, S. Xie, H. Dai, et al., Blockchain challenges and opportunities: a survey, Int. J. Web Grid Serv. 14 (4) (2018) 352–375, https://doi.org/10.1504/ijwgs.2018.10016848.

[28] Performant and modular apis for verifiable data and ssi, https://veramo.io/docs/basics/introduction. (Accessed 2 April 2024).

[29] Ethr-did library, https://github.com/uport-project/ethr-did. (Accessed 2 April 2024).

[30] Self-sovereign and decentralised identity by design, https://github.com/jolocom/jolocom-lib/wiki/Jolocom-Whitepaper, 2018. (Accessed 2 April 2024).

[31] S.K. Radha, I. Taylor, J. Nabrzyski, et al., Verifiable badging system for scientific data reproducibility, Blockchain Res. Appl. 2 (2) (2021) 100015, https://doi.org/10.1016/j.bcra.2021.100015.

[32] D. Reed, J. Law, D. Hardman, The technical foundations of sovrin, https://sovrin.org/wp-content/uploads/2018/03/The-Technical-Foundations-of-Sovrin.pdf, 2016. (Accessed 2 April 2024).

[33] E. Bandara, X. Liang, S. Shetty, et al., Octopus: privacy preserving peer-to-peer transactions system with interplanetary file system (IPFS), Int. J. Inf. Secur. 22 (2023) 591–609, https://doi.org/10.1007/s10207-022-00650-2.

[34] G. Zyskind, O. Nathan, A.S. Pentland, Decentralizing privacy: using blockchain to protect personal data, in: Proceedings of the 2015 IEEE Security and Privacy Workshops, IEEE, 2015, pp. 180–184, https://doi.org/10.1109/SPW.2015.27.

[35] K. Fan, S. Wang, Y. Ren, et al., Medblock: efficient and secure medical data sharing via blockchain, J. Med. Syst. 42 (8) (2018) 1–11, https://doi.org/10.1007/s10916-018-0993-7.

[36] A. De Salve, D.D.F. Maesa, P. Mori, et al., A multi-layer trust framework for self sovereign identity on blockchain, Online Soc. Netw. Media 37 (2023) 100265, https://doi.org/10.1016/j.osnem.2023.100265.

[37] A. Sahai, B. Waters, Fuzzy identity-based encryption, in: R. Cramer (Eds.), Advances in Cryptology—EUROCRYPT 2005, Springer, Berlin, 2005, pp. 457–473, https://doi.org/10.1007/11426639_27.
[38] V. Goyal, O. Pandey, A. Sahai, et al., Attribute-based encryption for fine-grained access control of encrypted data, in: Proceedings of the 13th ACM Conference on Computer and Communications Security, ACM, 2006, pp. 89–98, https://doi.org/10.1145/1180405.1180418.
[39] L. Cheung, C. Newport, Provably secure ciphertext policy ABE, in: Proceedings of the 14th ACM Conference on Computer and Communications Security, ACM, 2007, pp. 456–465, https://doi.org/10.1145/1315245.1315302.
[40] K. Emura, A. Miyaji, A. Nomura, et al., A ciphertext-policy attribute-based encryption scheme with constant ciphertext length, in: Proceedings of the International Conference on Information Security Practice and Experience, Springer, 2009, pp. 13–23, https://doi.org/10.1007/978-3-642-00843-6_2.
[41] T. Nishide, K. Yoneyama, K. Ohta, Attribute-based encryption with partially hidden encryptor-specified access structures, in: Proceedings of the International Conference on Applied Cryptography and Network Security, Springer, 2008, pp. 111–129, https://doi.org/10.1007/978-3-540-68914-0_7.
[42] C. Wang, J. Luo, An efficient key-policy attribute-based encryption scheme with constant ciphertext length, Math. Probl. Eng. (2013) 810969, https://doi.org/10.1155/2013/810969.
[43] N. Koblitz, A. Menezes, Pairing-based cryptography at high security levels, in: N.P. Smart (Eds.), Cryptography and Coding, Springer, Berlin, 2005, pp. 13–36, https://doi.org/10.1007/11586821_2.
[44] S.D. Galbraith, K.G. Paterson, N.P. Smart, Pairings for cryptographers, Discrete Appl. Math. 156 (16) (2008) 3113–3121, https://doi.org/10.1016/j.dam.2007.12.010.
[45] X. Yao, Z. Chen, Y. Tian, A lightweight attribute-based encryption scheme for the Internet of things, Future Gener. Comput. Syst. 49 (2015) 104–112, https://doi.org/10.1016/j.future.2014.10.010.
[46] S. Ding, C. Li, H. Li, A novel efficient pairing-free cp-abe based on elliptic curve cryptography for iot, IEEE Access. 6 (2018) 27336–27345, https://doi.org/10.1109/ACCESS.2018.2836350.
[47] M. Blaze, G. Bleumer, M. Strauss, Divertible protocols and atomic proxy cryptography, in: K. Nyberg (Eds.), Advances in Cryptology—EUROCRYPT'98, Springer, Berlin, 1998, pp. 127–144, https://doi.org/10.1007/BFb0054122.
[48] G. Ateniese, K. Fu, M. Green, et al., Improved proxy re-encryption schemes with applications to secure distributed storage, ACM Trans. Inf. Syst. Secur. 9 (1) (2006) 1–30, https://doi.org/10.1145/1127345.1127346.
[49] A.-A. Ivan, Y. Dodis, Proxy cryptography revisited, NDSS (2003).
[50] S.S. Chow, J. Weng, Y. Yang, et al., Efficient unidirectional proxy re-encryption, in: D.J. Bernstein, T. Lange (Eds.), Progress in Cryptology—AFRICACRYPT 2010, Springer, Berlin, 2010, pp. 316–332, https://doi.org/10.1007/978-3-642-12678-9_19.
[51] P.-S. Chung, C.-W. Liu, M.-S. Hwang, A study of attribute-based proxy re-encryption scheme in cloud environments, Int. J. Netw. Secur. 16 (1) (2014) 1–13.
[52] S. Luo, J. Hu, Z. Chen, Ciphertext policy attribute-based proxy re-encryption, in: M. Soriano, S. Qing, J. López (Eds.), Information and Communications Security, Springer, Berlin, 2010, pp. 401–415, https://doi.org/10.1007/978-3-642-17650-0_28.
[53] K. Liang, M.H. Au, J.K. Liu, et al., A secure and efficient ciphertext-policy attribute-based proxy re-encryption for cloud data sharing, Future Gener. Comput. Syst. 52 (2015) 95–108, https://doi.org/10.1016/j.future.2014.11.016.
[54] S. Sicari, A. Rizzardi, G. Dini, et al., Attribute-based encryption and sticky policies for data access control in a smart home scenario: a comparison on networked smart object middleware, Int. J. Inf. Secur. 20 (5) (2021) 695–713, https://doi.org/10.1007/s10207-020-00526-3.
[55] M. Rasori, P. Perazzo, G. Dini, ABE-cities: an attribute-based encryption system for smart cities, in: Proceedings of the 2018 IEEE International Conference on Smart Computing (SMARTCOMP), IEEE Computer Society, 2018, pp. 65–72, https://doi.org/10.1109/SMARTCOMP.2018.00075.
[56] S. Fugkeaw, H. Sato, An extended cp-abe based access control model for data outsourced in the cloud, in: Proceedings of the 2015 IEEE 39th Annual Computer Software and Applications Conference, IEEE, 2015, pp. 73–78, https://doi.org/10.1109/COMPSAC.2015.216.
[57] S. Banerjee, B. Bera, A.K. Das, et al., Private blockchain-envisioned multi-authority cp-abe-based user access control scheme in IIoT, Comput. Commun. 169 (2021) 99–113, https://doi.org/10.1016/j.comcom.2021.01.023.
[58] T. Feng, J. Guo, A new access control system based on cp-abe in named data networking, Int. J. Netw. Secur. 20 (4) (2018) 710–720, https://doi.org/10.6633/IJNS.201807_20(4).13.
[59] W. Li, K. Xue, Y. Xue, et al., TMACS: a robust and verifiable threshold multi-authority access control system in public cloud storage, IEEE Trans. Parallel Distrib. Syst. 27 (5) (2016) 1484–1496, https://doi.org/10.1109/TPDS.2015.2448095.
[60] D. Hardman, Aries RFC 0005: DID communication, https://github.com/hyperledger/aries-rfcs/blob/main/concepts/0005-didcomm/README.md. (Accessed 2 April 2024).
[61] Eidas supported self-sovereign identity, https://ec.europa.eu/futurium/en/system/files/ged/eidas_supported_ssi_may_2019_0.pdf, 2019. (Accessed 2 April 2024).
[62] A. Beimel, Secure schemes for secret sharing and key distribution, https://www.cs.bgu.ac.il/~beimel/Papers/thesis.pdf, 1996. (Accessed 2 April 2024).
[63] B. Waters, Ciphertext-policy attribute-based encryption: an expressive, efficient, and provably secure realization, in: D. Catalano, N. Fazio, R. Gennaro, et al. (Eds.), Public Key Cryptography—PKC 2011, Springer, Berlin, 2011, pp. 53–70, https://doi.org/10.1007/978-3-642-19379-8_4.
[64] Q. Liu, G. Wang, J. Wu, Time-based proxy re-encryption scheme for secure data sharing in a cloud environment, Inf. Sci. 258 (2014) 355–370, https://doi.org/10.1016/j.ins.2012.09.034.
[65] H. Deng, Z. Qin, Q. Wu, et al., Flexible attribute-based proxy re-encryption for efficient data sharing, Inf. Sci. 511 (2020) 94–113, https://doi.org/10.1016/j.ins.2019.09.052.
[66] F. Luo, S. Al-Kuwari, Revocable attribute-based proxy re-encryption, J. Math. Cryptol. 15 (1) (2021) 465–482, https://doi.org/10.1515/jmc-2020-0039.
[67] J. Benet, IPFS-content addressed, versioned, P2P file system, arXiv, 2014, preprint, arXiv:1407.3561.
[68] MetaVerse Chat—Web3 and SSI in action, https://bitbucket.org/netis/veramo-aceblock-didcomm-over-libp2p-react/src/master/. (Accessed 2 April 2024).
[69] K.A.M. Ahmed, S.F. Saraya, J.F. Wanis, et al., A blockchain self-sovereign identity for open banking secured by the customer's banking cards, Future Internet 15 (2023) 208, https://doi.org/10.3390/fi15060208.
[70] A. Preukschat, D. Reed, Self-Sovereign Identity, Manning Publications, 2021.
[71] Solidity, Solidity 0.8.3 documentation, https://solidity.readthedocs.io/en/v0.8.3, 2021. (Accessed 2 April 2024).
[72] F. Buccafurri, V. De Angelis, M.F. Idone, et al., Achieving sender anonymity in tor against the global passive adversary, Appl. Sci. 12 (2022) 137, https://doi.org/10.3390/app12010137.
[73] A. De Caro, V. Iovino, jPBC: Java pairing based cryptography, in: Proceedings of the 2011 IEEE Symposium on Computers and Communications (ISCC), IEEE, 2011, pp. 850–855, https://doi.org/10.1109/ISCC.2011.5983948.
[74] Infura, Secure and scalable access to Ethereum apis and IPFS gateways, https://infura.io/docs, 2021. (Accessed 2 April 2024).
[75] Trinsic, https://github.com/trinsic-id. (Accessed 2 April 2024).
[76] M. Davie, D. Gisolfi, D. Hardman, et al., The trust over IP stack, IEEE Commun. Stand. Mag. 3 (4) (2019) 46–51, https://doi.org/10.1109/MCOMSTD.001.1900029.
[77] Ropsten, Ropsten testnet explorer, https://ropsten.etherscan.io, 2021. (Accessed 2 April 2024).