AlgoID: a blockchain reliant Self-Sovereign Identity framework on Algorand

Andrea De Salve*, Damiano Di Francesco Maesa†, Fabio Federico†, Paolo Mori‡ and Laura Ricci†
*Institute of Applied Sciences and Intelligent Systems - National Research Council
Italy, Lecce, Email: andrea.desalve@isasi.cnr.it
† Department of Computer Science - University of Pisa
Italy, Pisa, Email: damiano.difrancesco@unipi.it - f.federico4@studenti.unipi.it - laura.ricci@unipi.it
‡Institute of Informatics and Telematics - National Research Council
Italy, Pisa, Email: paolo.mori@iit.cnr.it

***Abstract*—The Self-Sovereign Identity (SSI) is a novel paradigm aimed at giving back users sovereignty over their digital identities. Adopting the SSI approach prevents users to have a distinct identity for each service they use, instead, use a unique decentralised identity for all the services they need to access. However, to really benefit from the SSI advantages, an actual decentralised implementation is needed to fit the specific requirements and limits of decentralised architectures, such as blockchain. To this aim, this paper proposes Algorand Identity (AlgoID), a new SSI framework for the Algorand blockchain which differs from the already existing one, because it is fully blockchain based, i.e., it exploits Algorand itself for the storage of the data identity and as the registry location. The proposed framework has been completely implemented and validated through experiments, showing that the time required to execute the framework operations is acceptably low in realistic use cases.**

***Index Terms*—Digital Identity, Self Sovereign Identity, Blockchain, Algorand, Smart Contract**

I. Introduction

During the last years, online services have proliferated in ever increasing numbers and heterogeneity of applications, ranging from Online Social Networks to E-commerce portals to banking services. In order to exploit such new services, users are forced to create a new digital identity for each one, and share their personal data requested by the service providers. This model burdens users with the management of a large number of distinct digital identities and it forces them to share with service providers (possibly distinct subsets of) their personal data. Users so lose any further control on such data and do not have any guarantee beyond the trust they place in the service provider entity.

The SSI [1] paradigm revolutionises user identity management by solving these issues, as it gives back to users the sovereignty over their digital identities, guaranteeing them full control over their personal data. In this model, each user creates their own digital identity(ies) independently of the services on which they will be used, and they can later decide which subset of their personal data to disclose to each service provider [2]. SSI frameworks are typically built on top of a blockchain, which is used as a tamper-proof, always available, and decentralised registry of digital identifiers and their attributes. Since each blockchain has its own characteristics (such as the smart contract language, and the storage model), new SSI solutions have to be designed to meet the requirements and limitations of the specific blockchain. Furthermore, blockchain could affect the costs for the execution of some SSI operations, due to the need of writing information on the blockchain. For this reason, it is crucial to base SSI frameworks on blockchain protocols that make the SSI operations more effective and affordable in terms of costs, such as the Algorand one [3].

In light of the previous considerations, in this paper we present AlgoID a new SSI framework for the Algorand blockchain, implementing most of the functionalities declared in the W3C latest specifications[1]. The reason why we propose a new SSI framework for Algorand is to offer an alternative to the only existing one (called did-algo and described in Section III-B). Our proposal is based on different principles and technologies, as we exploit Algorand smart contracts as our main platform for attributes storage and for the registry location, thus benefiting from the blockchain advantages (e.g., tamper resistance and availability). To highlight such differences, a comparison of the features of AlgoID with did-algo, as well as ethr-did, based on Ethereum, and indy-did, based on Hyperledger, is provided. Moreover, to validate our framework, we evaluate the cost of each operation provided by the framework, and the cost incurred by the subjects of two distinct typical use cases of usage of SSI, one taken from the W3C use cases, and the other defined by us.

The paper is organised as follows. Section II provides the needed background, while Section III describes other existing SSI frameworks. Section IV presents the framework we propose, Section VI reports the costs of executing its operations, and Section VII compares it with other two SSI frameworks. Finally, Section VIII draws the conclusions.

This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

[1]W3C DID specification: https://www.w3.org/TR/did-core/

II. Background

A. Distributed Ledger Technology

As of today, *blockchain* is the most relevant and widespread example of Distributed Ledger Technology (DLT) [4]. It consists of a set of nodes communicating via a peer-to-peer (P2P) network in order to maintain a cryptographically linked concatenated list of blocks. A block consists of a set of records, named *transactions*, with each record contributing to update the global state represented by the chain.

Each participant in the network is anonymously identified by one or more accounts consisting of two cryptographic keys: a *private key*, which the user must use to *sign* transactions, and an *address* (the corresponding *public key* or an hashed version of it), which the user can distribute to others willing to create transactions interacting with them. The *consensus algorithm* is a protocol executed by voluntary participants of the P2P network, and it regulates how new blocks are verified and then appended to the blockchain.

In recent years, blockchain protocols have broadly extended their capabilities by allowing users not only to transfer value among themselves as originally intended but also to store and execute programs, dubbed *smart contracts*, using scripting languages. For example, Algorand is a second generation blockchain released in 2018 by Turing Award winner Silvio Micali [3] and it allows users to develop and execute smart contracts on the Algorand Virtual Machine (AVM). Smart contracts are written in Transaction Execution Approval Language (TEAL) [5], a low-level assembly-like language with several limitations in terms of space availability, number of arguments in function calls, and program length. One positive consequence of such limitations is that Algorand dictates a fixed fee of 0.001 ALGOs (Algorand's native currency) for each transaction, as it limits the number of instructions so that no smart contract computation can exceed such fee amount. Smart contracts on Algorand can be either *stateless* or *stateful*, with stateful contracts offering the possibility to allocate a small *local space* to each user (limited to 16 variables), in addition to the always present *global space* (limited to 64 variables). Unlike the majority of the other platforms, which only support immutable code, Algorand smart contracts can also be created as *mutable* programs: mutable smart contracts support *updating operations*, meaning that their code can be updated to accommodate future changes to the program specifications and/or to benefit from the evolution of the Algorand platform.

B. Self-Sovereign Identity

SSI [6] is a new approach to identity management, which has at its core the idea that every entity should be the unique owner and controller of their digital identity. To achieve this, SSI relies on two fundamental components: the Decentralized IDentifier (DID) [7] and the Verifiable Credential (VC) [8].

1) DID: they are unique alphanumeric identifiers paired with users. Each identifier is related to a *DID method*, i.e., a set of rules and tools defining how the creation, management, and maintenance of a DID is implemented. In the traditional workflow of a SSI system, a user *creates* a DID and *updates* it in order to add/remove additional attributes (such as cryptographic keys) to/from the DID. Each DID is paired with a *DID document*, i.e. a set of data specifying all relevant attribute values of the DID, which can be retrieved from a *verifiable registry* through the *resolution* operation.

2) VCs and Verifiable Presentations: VCs [9] are another W3C recommendation which can be used to represent an arbitrary piece of information about a particular subject. A VC is issued by an entity (*the issuer*) and it can be *signed* by the issuer using their DID. In this way, a *verifier* could then cryptographically verify the authenticity of that credential. VCs are not directly provided to a verifier, as one or multiple VCs can be aggregated in a *Verifiable Presentation (VP)*, which can then be presented by the user to a verifier.

Blockchain technology is widely used in SSI frameworks because of its immutability and decentralised nature. In particular, blockchain is adopted by several SSI frameworks for implementing *verifiable registries*, a component needed to support the resolution process by holding the correspondence between DIDs and their associated DID documents.

III. Related work

This section reports a set of W3C compliant DID Methods that are provided by popular blockchain platforms (i.e., Ethereum, Algorand, and Hyperledger) and for each of them studies the architectural style used to provide decentralisation and protection of the identities' information.

A. Ethereum

ethr-did[2] is a DID Method specification and a library for DIDs on the Ethereum blockchain. It leverages Ethereum's accounts and smart contracts both to implement the registry for DIDs resolution and to store a representation of each DID document in a persistent, decentralised, and always accessible storage medium. The ethr-did schema adopts `ethr` as the DID Method identifier and uses Ethereum addresses as the DID identifier. Since each Ethereum address is paired to a public key, the corresponding private key is used as verification method. Each DID document in the ethr-did method is not stored as a whole file: instead, it must be built from both read-calls and contract-events queries to the registry smart contract. Indeed, all of the attributes of a DID, except for the `id` and `controller` attribute, are stored as *contract events*, indexed by the `id` of the call's DID subject. The `id` and the `controller` attributes are stored as normal indexed variables instead. While ethr-did does not define any implementation-specific attributes, it makes all of the W3C core-attributes available to the user, meaning that anyone can *change the owner*, or `controller`, of a DID, add *aliases*, and add *temporary delegates* to transfer partial control of the DID to an externally managed key pair.

[2]ethr-did on GitHub: https://github.com/uport-project/ethr-did

B. Algorand

did-algo[3] is a SSI framework for the Algorand blockchain consisting of a *DID Method specification* with `algo` as the Method identifier, a *software client* supporting CRUD[4] operations and a *server side software* supporting DID resolution. In this framework the blockchain does not play a central role: DID documents are uploaded to the IPFS, which is the ultimate storage medium, Algorand addresses are not used as DID identifiers, as Universally Unique Identifiers (UUIDs) are used instead, and the verifiable registry is implemented as a proprietary system, hosted on private servers, which holds the correspondence between a given DID and the corresponding identifier to retrieve the associated DID document from the IPFS.
The system offers a few options to its users, such as the possibility of adding a number of cryptographic keys (of types Ed25519, secp256k1 and RSA) and specifying a service endpoint for the DID. DID creation, updating, resolution and deactivation operations can all be managed through the CLI software offered in the framework, which also provides an Algorand wallet that can be linked to a given DID and a user-guide for each of the available commands.

C. Hyperledger Indy

did-indy[5] offers an ecosystem of libraries, tools, and frameworks for SSI which are based on a permissioned blockchain. The set of nodes that composes the network is called *pool* and the blockchain consists of four different logical ledgers each storing a specific type of transaction (the domain ledger, the pool ledger, the config ledger, and the logical union of the previous ledgers). A DID in Indy consists of the `indy` method identifier, a namespace indicating the specific indy ledger, and a unique identifier in the given namespace. The unique identifier is derived from the initial part of the ED25519 key pair associated with the identifier, which is used to sign transactions and/or message. Creation of a DID, as well as the addition of attributes to a particular DID is performed by publishing a transaction. Since it is based on a permissioned blockchain, did-indy enables to assign a role to each DID in order to identify the set of capabilities and functionalities that the corresponding DID can access. Before issuing a verifiable credential, the attributes and the claims which compose the schema of a verifiable credential must be defined by a transaction and the issuer must create a transaction containing the unique identifier of the credential and the public key used by the issuer to sign credentials. A verifiable credential is issued after a negotiation phase implemented by using an off-chain credential offer/request mechanism. The verifier initiates the credential verification process by sending a proof request to the prover, who replies with a proof.

[3]did-algo on GitHub: https://github.com/algorandfoundation/did-algo
[4]Create, Read, Update, Delete
[5]did-indy on GitHub: https://hyperledger.github.io/indy-did-method/

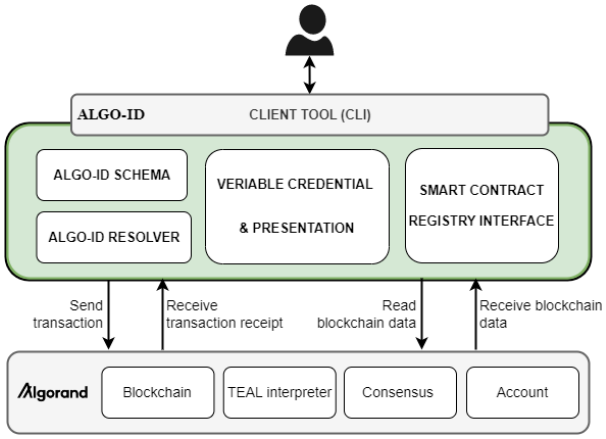

Fig. 1. General architecture of AlgoID

D. Further works

Other than those cited above, there have been numerous experiments with blockchain-based SSI frameworks (such as *uPort* [10]). Some of those are examined and evaluated, both in terms of methodology and technological choices, in other scientific publications, such as M. S. Ferdous *et Al.* [11] Eddine *et Al.* [12], O. Avellaneda *et Al.* [13], N. Naik and P. Jenkins [14].

IV. AlgoID: an SSI framework on Algorand

This section introduces the design of a novel SSI framework for identity management on the Algorand blockchain, named AlgoID, which complies with the W3C recommendations (see Sec. II-B). The AlgoID framework allows entities to create and manage their DIDs, sign and verify VCs and securely hold the cryptographic information associated to the DID.

A. Architectural overview

The proposed framework consists of a set of modules providing the functionalities for the management and maintenance of both DIDs and VCs on Algorand. Figure 1 shows the architecture of the AlgoID framework as well as the core modules used to support the service. A generic user that wants to create an identifier by exploiting the Algorand blockchain can interact with the AlgoID framework using the *client tool*, an interactive application software running on the user's device that allows to create, manage and delete DIDs, and to sign and verify VCs and VPs. The specific format and the syntax of the DID used by the AlgoID framework is provided by the *DID Schema* module, while the *Verifiable Credential & Presentation* module specifies the format of the VC and VP, and the corresponding signature mechanisms. The resolution of a DID in AlgoID is performed by the *AlgoID resolver* module, which interacts with the *Smart Contract Registry Interface* in order to retrieve from the verifiable registry updated information about the specified DID. The AlgoID framework exploits the Algorand blockchain functionalities to provide a secure data registry with augmented capabilities,

TABLE I
BASIC STRUCTURE OF A DID DOCUMENT IN ALGOID

Name	Description
context	explanation of the terms used in the document
id	Algorand address of the DID subject
controller	effective owner of the DID
verificationMethod	keys accepted verification purposes
assertionMethod	keys accepted to express *claims*, such as VCs
authentication	keys accepted for authentication purposes
service	set of services with the endpoints to reach them

such as perform computation using smart contracts and verifiable storage. For this reason, the AlgoID framework is able to send transaction signed with the private key of a DID to Algorand and read updated and authentic information from the Algorand blockchain.

B. DID Method

1) DID Schema: The AlgoID schema adopts `algoid` as Method identifier and uses plain **Algorand addresses** as DID identifiers. The public key associated with the Algorand address is also assumed to be a valid `verificationMethod`. This means that a given Algorand address is tied to one and only one DID, and that a given user could define as many DIDs as the number of Algorand accounts they own, in a *1-to-n* relationship between addresses and DIDs.

The AlgoID method also supports a name-spacing mechanism through the use of **tags** as a mean of grouping all DIDs having an arbitrary logical correlation amongst them; one possible application could be to use `test` as a tag to specify that the DID is being used on the testnet or the (reduced) name of an organisation, such as `acme`, to signify that the DID is to be used in its context. The following is an example of an AlgoID DID:

did:algoid:VCMJKWOY5P5P7SKMZFFOCEROPJCZOTIJMNIYNUCKH7LRO45JMJP6UYBIJA

The following is an example of an AlgoID DID with a tag:

did:algoid:acme:VCMJKWOY5P5P7SKMZFFOCEROPJCZOTIJMNIYNUCKH7LRO45JMJP6UYBIJA

2) DID Document: DID documents in AlgoID are compliant with the original W3C DID specification and support all attribute fields illustrated in table [I]. The table also provides a brief description of each one of the fields. We can see that the method allows a user to specify an extensive set of attributes for their DID; specifically, a user can add (and remove) to their DID one or more of the following:

- a **controller**, which will hold authority over modification of the DID's attributes;
- additional **cryptographic keys**, of types `ed25519` and `secp256k1`, both for authentication and for assertion purposes;
- a set of **service endpoints** for DIDs that are related to a particular service;
- a set of **aliases**, which allows an entity to link more DIDs to itself;
- a set of **delegate keys**, which can be distributed to trusted third-parties and are to be considered valid for the specified operations up to the specified expiration date.

The `controller` attribute is a *unique* property meant to support the notion of DID *ownership*, as seen in the attribute description in the W3C's specification. In particular, given a DID d, the DID e listed under the controller property of d is considered to be the owner of d and, as such, e has access and modification permissions upon d. As per the W3C specification, any DID document *must* always list exactly *one DID* under the `controller` attribute; at the moment of creation of a DID, the value of such property is the DID itself. Only the current owner of a DID can initiate a *change of ownership* with a specific call of the *AlgoID registry*, specifying the DID that should take its place.

3) AlgoID Registry: The AlgoID method uses an Algorand smart contract to implement a *verifiable registry*, the `algoid-registry`. The `algoid-registry` is used for DID documents storage and resolution: it contains a succinct representation of DIDs which allows resolver software to gather all the published information about a given DID and use it to construct the relative DID document. An ideal implementation would simply query the Algorand blockchain with the smart contract id corresponding to the registry contract and organise the data received in a human-readable format, with the help of the `algoid` method specification. Further details on how this can be achieved are provided in the sub section V-A.

4) Creation and Deactivation:

Creation: One of the reasons the AlgoID method uses Algorand addresses as DID identifiers, besides efficiency and convenience, is because in this way the creation of a new DID is free and can be safely conducted locally: since Algorand addresses are unique by definition, there is no risk of duplicate DIDs, and the private key associated to the newly generated DID can be securely stored in a credential wallet. Practically, the system assumes that there exists a unique DID for every possible Algorand account, identified by the account's address itself, and that the *public key* corresponding to that address is a valid *authenticationMethod*. This means that a user wanting to create a new DID on the AlgoID system doesn't need to register it in the `algoid-registry` and pay the associated fee. The payment of a fee is only requested when a user wants to *update* their DID by *adding* or *removing* attributes, or *changing* their values.

Whenever someone will try to resolve a DID that is not present in the `algoid-registry`, the related DID document is resolved to a *default representation*, which reports the address as both the *subject* and the *owner*, and the related public key as a valid *verificationMethod*.

Deactivation: Deactivation of a DID is not always necessary: if a user doesn't have any reason to disclose the information that their DID has been deactivated, it is enough to simply never use that DID again. If for a specific rea-

son, a user needs to publicly state that their DID has been deactivated, they need to make a specific *de-activation call* to the `algoid-registry` smart contract to register that information onto the DID. Such an operation is *not reversible*: a deactivated DID won't support any kind of update from that point on, and all of its related authentication and assertion methods will be considered no longer valid.

C. DID Management

Along with the DID Method specification, we present an Identity Management System (IMS) in the form of a client software that allows users to easily create, manage and delete DIDs and to easily create, sign and verify VCs and VPs in the form of JWTs [15].
The IMS is structured as an interactive command-line application that can be used both to fully manage AlgoID DIDs and to issue/verify VCs and VPs. The IMS also implements a credential wallet, developed according to the W3C's *Universal Wallet* specification[6], to store the user's cryptographic material (such as DIDs' private keys) and the VCs received from the issuers. In particular, the wallet is encrypted, where the symmetric encryption key is generated using the password-based key derivation function (PBKDF2) [16] with a randomly generated nonce.
Being a complete IMS, the proposed framework supports the following operations on DIDs:

- **Create:** the user can create a DID, specifying a *passphrase* for future update operations
- **Read:** the user can resolve a given DID document created under the `algoid` method
- **Update:** the user can update its DID by adding attributes, as seen in paragraph IV-B2
- **Delete:** the user can delete one of their DIDs, (*deactivation* has been described in Section IV-B4)

The IMS also supports creation, signature and verification operationsof VCs and VPs.
For both types of signature, the user can specify the DID with which they want to sign the data. Symmetrically, both types of verification require the user to specify the DID against which to check the signature; the system will then proceed with the resolution procedure, checking all of the DID's public keys listed under the `assertionMethod` property.

V. An implementation of AlgoID

This section describes the main implementation challenges that we have encountered during the development of the AlgoID framework, focusing on the computational and storage restrictions imposed by the underlying Algorand blockchain.

A. Structure of the AlgoID Registry Smart Contract

The AlgoID DID method, along with the attributes and operations it supports, is concretely represented by the *AlgoID verifiable registry*, implemented through an Algorand smart contract. The registry smart contract is an updatable, stateful smart contract in which *global storage* is used to store meta-data about the contract itself, while *local storage*, which is dedicated to each DID, is used to store the attribute values associated to a given DID. The contract's meta-data consists of the Algorand address of the *contract administrator*, who has permissions over updating operations to the contract's code, and the current version of the smart contract, each stored in its own variable. DID attributes instead adopt a specific encoding: since each attribute is made of multiple fields and local storage is limited to 16 variables, it would be too expensive, in terms of space, to store each field of an attribute in a dedicated variable; instead, AlgoID exploits TEAL's `bytes` data-type to store an aggregated binary-data representation, based on the JavaScript Object Notation (JSON) format, of all the fields of a given attribute in a single variable, so that one attribute occupies exactly one local storage variable. Additionally, we identified 3 attribute categories that are stored in different locations of the local storage in the registry contract: *i)* DID's meta-data are stored in a single local storage location; *ii)* information of the DID's controller is stored using a single local storage location; *iii)* additional attributes of the DID are stored on the remaining 14 local storage locations.

B. Implementation of the IMS

The AlgoID framework has been entirely developed in JavaScript, and several libraries have been exploited. In order to interact with the AlgoID registry smart contract, the proposed implementation makes RESTful requests to the Algorand blockchain by using the Algorand Software Development Kit[7] for JavaScript, provided by Algorand to simplify blockchain interaction and transaction creation.
The VC and VP are formatted according to the JSON Web Token (JWT) standard [15], which allows users to *sign* and *verify* VC and VP using several signature algorithms.
As part of the implementation, the proposed framework also provides a library (`algoid-did-resolver`) for DID resolution which contains the logic to communicate with the AlgoID registry smart contract to retrieve the DID document related to a specific Algorand address. This can be utilised by an independent developer to implement applications based on the AlgoID method.

VI. Experimental Evaluation

An instance of the AlgoID[8] framework previously described has been deployed on the *Algorand testnet* in order to profile its cost and performance.
As a first experimental evaluation, we focus on the cost required to deploy the AlgoID Registry[9] (*Contract deploy*) and to invoke its functions. Table II shows the execution time of each function of the AlgoID registry smart contract with the associated cost in ALGOS, also indicating whether it appends data on the blockchain.

[6]Universal Wallet specification: https://w3c-ccg.github.io/universal-wallet-interop-spec/

[7]algosdk on GitHub: https://algorand.github.io/js-algorand-sdk/modules.html

[8]AlgoID IMS on GitHub: https://github.com/Fred-ef/AlgoID-test-release

[9]Registry smart contract (testnet): https://testnet.algoexplorer.io/tx/FGR4NKBRILS2X4JF5FSJQPU52WWJXKBS5XILRDDNLCNUNZGXTLHQ

TABLE II
COST (IN ALGOS) AND EXECUTION TIME OF THE MAIN OPERATIONS

CALL	COST (ALGOS)	TIME	WRITES OPS ON BLOCKCHAIN
Contract deploy	0.001	6.569s	YES
Contract optin	0.001	4.412s	YES
Add key	0.001	4.373s	YES
Add service	0.001	4.584s	YES
Add alias	0.001	4.912s	YES
Add delegate	0.001	5.032s	YES
Remove attribute	0.001	5.443s	YES
DID creation	0	0.029s	NO
DID resolution	0	0.887s	NO

TABLE III
EXECUTION TIME OF VC OPERATIONS

CALL	TIME	VC/VP TYPE
Signing bytes	0.032s	University Degree
Verifying bytes	0.081s	
Signing VC	0.030s	Driver License
Verifying VC	0.122s	
Signing VP	0.037s	Vaccine Proof
Verifying VP	0.146s	

The figures reported in Table II have been determined by computing the average over hundreds of executions of the various operations on a Linux (Ubuntu) VM on a system with an i7 12650H CPU and 16GB of DDR5 RAM. The results reveal that operations that write on the blockchain incur in a *fixed fee* and take considerably more time to be executed than those that don't, as for a writing to take effect one has to wait for the *block confirmation*. That is why the creation of a DID and its resolution don't involve the payment of any fees, as the former doesn't require any interaction with the blockchain and the latter only requires to read from it.

In Table III, we evaluated the execution time for all operations related to VCs and VPs management, specifically *signing* and *verifying* operations. These operations don't involve any cost for the subject, as they do not write on the blockchain. Additionally, we can see that *signing* operations require considerably less time than *verification* ones. As a matter of fact, *signing* operations are executed locally, while *verification* operations take more time because of the latency due to the reading from the registry contract.

In order to give an idea of the total cost a subject could incur when using the AlgoID framework, we evaluate the following use-case, extracted from the W3C use-cases[10] page. Asako just got her driver's license, and would like to receive a credential on the AlgoID system that asserts that she has the right to drive a car. Therefore, Asako downloads the `AlgoID IMS` and creates a DID with no cost for her. Asako then requests a VC from the relevant certifying authority. The certifying authority uses the IMS to interactively create a VC and *sign* it in negligible time (0.030s), at no cost. The VC is sent to Asako's credential wallet. At that point, Asako is pulled-over by the police and the officer requests her driver's license. Asako uses the IMS to select her driver's license VC and, instantly (0.037s), generates a verifiable presentation including it to be sent to the officer, again at no cost. The officer then uses the IMS to promptly *verify* the received presentation (0.122s) without paying any fee.

Another realistic example to show the usefulness of an SSI framework supporting the reported functionalities and attributes is the following: James and his son Luke, who both possess an identity on the AlgoID system, go to the doctor for Luke's recurring vaccine dose. Before proceeding with the shot, the doctor asks James for Luke's ID, medical insurance and record of previous vaccines. These documents are represented by three distinct VCs. Being the *controller* of Luke's DID, James is able to use the `AlgoID IMS` to retrieve the previous VCs, construct and sign a VP for the three credentials in negligible time (0.037s) and present it to the doctor. In turn, the doctor can use the IMS to quickly verify James' VP (0.146s) and proceed with the shot. None of them has to pay any fees for any of the described operations.

[10]VC use cases: https://www.w3.org/TR/vc-use-cases

VII. COMPARISON

This section discusses the differences between the proposed AlgoID framework and the three main existing alternatives described in Section III. In particular, we consider as good candidates for the comparison the *did-algo*, *ethr-did* and *did-indy* frameworks because they are compliant with the SSI standards, based in well-established blockchain platforms, available to everyone, and widely used across several applications. In order to compare these frameworks, we identify interesting characteristics common to all of them and analyse how these characteristics are accommodated by each framework.

Table IV summarises for each candidate the platform for which the SSI framework is designed (*Platform*), the mechanism employed for storing DID documents (*Storage*), the type of storage support on which the DID method registry is located (*Registry location*), the type of identifier adopted by the DID method for DIDs generation (*Identifier type*), the cost of creating of a new DID (*DID creation cost*), the cost of adding/removing an attribute to/from the DID (*DID update cost*), whether the DID method supports additional cryptographic keys for DIDs (*Additional keys*), whether the DID method supports the addition of service endpoints to DIDs (*Service endpoints*), whether the DID method supports the use of aliases for DIDs (*Aliases*), whether the DID method supports delegate cryptographic keys for DIDs (*Delegate keys*), whether the DID method supports an ownership mechanism with change of ownership (*Ownership*).

As shown in Table IV, although AlgoID and did-algo are both intended for the Algorand blockchain, they show several differences related to the management of DIDs and the management of the storage.

Instead of relying on a private server for the resolution of DIDs, the proposed framework relies on a registry implemented through an Algorand smart contract, eliminating the

TABLE IV
COMPARISON BETWEEN ALGOID AND OTHER FRAMEWORKS

	algoid	did-algo	ethr-did	did-indy
Platform	Algorand	Algorand	Ethereum	Hyperledger
Storage	Blockchain	IPFS	Blockchain	Blockchain
Registry location	Blockchain	Server	Blockchain	Blockchain
Identifier type	Address	UUIDs	Address	Public key
DID creation cost	Free	Free	Free	Free
DID update cost	0.001 ALGOs	Free	Variable	Free
Additional keys	YES	YES	YES	YES
Service endpoints	YES	YES	YES	YES
Aliases	YES	NO	NO	YES
Delegate keys	YES	NO	YES	YES
Ownership	YES	NO	YES	YES

single-point-of-failure and further decentralising the system with a minimum transaction fee (of 0.001 Algos). Indeed, using the blockchain and its smart contracts instead of Inter-Planetary File System (IPFS) as the storage mechanism allows us to have a persistent medium (not always guaranteed by IPFS) which makes it easy to store and update DID attributes information (with smart contract calls).

Additionally, instead of using randomly generated identifiers as DIDs, the proposed framework utilises Algorand addresses, allowing users to exploit their existing Algorand accounts as DIDs, to use their account private keys to sign on behalf of their DIDs, and to create new DIDs free of charge. Finally, in contrast to *did-algo*, the proposed framework allows an entity to specify additional attributes (such as *aliases*, *delegates*, and *controllers*) and support changes of ownership. It is, however, important to note that *did-algo* does not impose any limit on the number of attributes a given DID may have, which may prove useful in some contexts, while, as said, AlgoID currently imposes a limit of 14 additional attributes (plus a *controller*).

The *ethr-did* and the *did-indy* frameworks exploit different blockchain platforms, but they share with AlgoID several common characteristics, such as the usage of the blockchain as verifiable data registry. In fact, just like AlgoID, also *ethr-did* and *did-indy* use a blockchain to model the registry. However, *ether-did* exploits smart contracts to manage DIDs while *did-indy* directly stores DID information on a blockchain purpose-built for decentralised identity.

Both *ethr-did* and *did-indy*, like AlgoID, use the public key of a user to derive a unique identifier for the DID and they make the DID method better integrated with the blockchain platform. As for example, transferring funds to a DID is natively supported without further operations.

VIII. CONCLUSIONS

This paper extensively covers the concepts of blockchains and SSI, and proposes AlgoID, an effective way to manage digital identities of Algorand users. Compared to the already existing framework proposed for Algorand, based on generic identifiers and IPFS as verifiable data registry, AlgoID is able to take full advantage of the underlying Algorand blockchain by allowing users to exploit Algorand addresses as DIDs and smart contracts for attributes storage and as verifiable data registry. Our experiments shown that the AlgoID framework is reasonably inexpensive and quick in producing and verifying all types of verifiable documents. Since AlgoID is also fully adherent to the W3C specifications, we believe it to be an optimal choice for identity management purposes.

We plan as future work to enhance the capabilities of the proposed framework by integrating other interesting characteristics and functionalities provided by Algorand, such as the rekeying feature, which enables an Algorand account to rotate the authoritative private keys while maintaining their public address fixed.

REFERENCES

[1] A. Mühle, A. Grüner, T. Gayvoronskaya, and C. Meinel, "A survey on essential components of a self-sovereign identity," *Computer Science Review*, vol. 30, pp. 80–86, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013718301217

[2] A. De Salve, A. Lisi, P. Mori, and L. Ricci, "Selective disclosure in self-sovereign identity based on hashed values," in *2022 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2022, pp. 1–8.

[3] S. M. Jing Chen, "Algorand," 2017. [Online]. Available: https://algorandcom.cdn.prismic.io/algorandcom%2Fece77f38-75b3-44de-bc7f-805f0e53a8d9_theoretical.pdf

[4] M. V. Gorbunova, P. Masek, M. M. Komarov, and A. Ometov, "Distributed ledger technology: State-of-the-art and current challenges," *Comput. Sci. Inf. Syst.*, vol. 19, no. 1, pp. 65–85, 2022. [Online]. Available: https://doi.org/10.2298/csis210215037g

[5] Algorand, "The algorand virtual machine (avm) and teal." [Online]. Available: https://developer.algorand.org/docs/get-details/dapps/avm/teal/specification/

[6] F. Schardong and R. Custódio, "Self-sovereign identity: A systematic review, mapping and taxonomy," *Sensors*, vol. 22, no. 15, p. 5641, 2022. [Online]. Available: https://doi.org/10.3390/s22155641

[7] D. Reed, M. Sporny, D. Longley, C. Allen, D. Longley, M. Sabadello, and O. Steele, "Decentralized identifiers (dids) v1.0: Core architecture, data model, and representations," 2022, working draft 2022-07-19. [Online]. Available: https://www.w3.org/TR/2022/REC-did-core-20220719/

[8] M. Sporny, D. Longley, and D. Chadwick, "Verifiable credentials data model v1.1," 2022, working draft 2022-03-03. [Online]. Available: https://www.w3.org/TR/2022/REC-vc-data-model-20220303/

[9] W3C, "Verifiable credentials data model v1.1," 2022. [Online]. Available: https://www.w3.org/TR/vc-data-model/

[10] N. Naik and P. Jenkins, "uport open-source identity management system: An assessment of self-sovereign identity and user-centric data platform built on blockchain," in *IEEE International Symposium on Systems Engineering, ISSE 2020, Vienna, Austria, October 12 - November 12, 2020*. IEEE, 2020, pp. 1–7. [Online]. Available: https://doi.org/10.1109/ISSE49799.2020.9272223

[11] M. S. Ferdous, F. Chowdhury, and M. O. Alassafi, "In search of self-sovereign identity leveraging blockchain technology," *IEEE Access*, vol. 7, pp. 103 059–103 079, 2019. [Online]. Available: https://doi.org/10.1109/ACCESS.2019.2931173

[12] B. N. Eddine, A. Ouaddah, and A. Mezrioui, "Exploring blockchain-based self sovereign identity systems: challenges and comparative analysis," pp. 21–22, 2021.

[13] O. Avellaneda, A. Bachmann, A. Barbir, J. Brenan, P. Dingle, K. H. Duffy, E. Maler, D. Reed, and M. Sporny, "Decentralized identity: Where did it come from and where is it going?" *IEEE Communications Standards Magazine*, vol. 3, no. 4, pp. 10–13, 2019.

[14] N. Naik and P. Jenkins, "Your identity is yours: Take back control of your identity using gdpr compatible self-sovereign identity," pp. 1–6, 2020.

[15] M. Jones, J. Bradley, and N. Sakimura, "Json web token (jwt)," Tech. Rep., 2015.

[16] A. Visconti, O. Mosnáček, M. Brož, and V. Matyáš, "Examining pbkdf2 security margin—case study of luks," *Journal of Information Security and Applications*, vol. 46, pp. 296–306, 2019.