Combining Decentralized IDentifiers with Proof of Membership to Enable Trust in IoT Networks

Alessandro Pino, Davide Margaria, Andrea Vesco
LINKS Foundation - Cybersecurity Research Group
Torino 10138, Italy

***Abstract*—The Self-Sovereign Identity (SSI) is a decentralized paradigm enabling full control over the data used to build and prove the identity. In Internet of Things networks with security requirements, the Self-Sovereign Identity can play a key role and bring benefits with respect to centralized identity solutions. The challenge is to make the SSI compatible with resource-constraint IoT networks. In line with this objective, the paper proposes and discusses an alternative (mutual) authentication process for IoT nodes under the same administration domain. The main idea is to combine the Decentralized IDentifier (DID)-based verification of private key ownership with the verification of a proof that the DID belongs to an evolving trusted set. The solution is built around the proof of membership notion. The paper analyzes two membership solutions, a novel solution designed by the Authors based on Merkle trees and a second one based on the adaptation of Boneh, Boyen and Shacham (BBS) group signature scheme. The paper concludes with a performance estimation and a comparative analysis.**

***Index Terms*—Self-Sovereign Identity, Decentralized IDentifiers, Proof of Membership, Group Signatures, Merkle Trees, Trust, Internet of Things.**

I. Introduction

The Self-Sovereign Identity (SSI) [1] is a decentralized digital identity paradigm that gives a peer full control over the data it uses to build and to prove its identity. The overall SSI stack, depicted in Fig. 1, enables a new model for trusted digital interactions.

The Layer 1 is implemented by means of any Distributed Ledger Technology (DLT) acting as the Root-of-Trust (RoT) for identity data. In fact, DLTs are distributed and immutable means of storage by design [2]. A Decentralized IDentifier (DID) [3] is the new type of globally unique identifier designed to verify a peer. The DID is a Uniform Resource Identifier (URI) of the following form:

did:method-name:method-specific-id

where *method-name* is the name of the DID Method used to interact with the DLT and *method-specific-id* is the pointer to the DID Document stored on the DLT, denoted as *index* in this paper for simplicity.

Thus, DIDs associate a peer with a DID Document [3] to enable trustable interactions with it. The DID Method [3], [4] is the software implementation used by a peer to interact with the DLT. In accordance with W3C recommendation [3], a DID Method must provide the primitives to:

- *create* a DID, that is, generate an identity key pair (sk_{id}, pk_{id}) for authentication purposes, the corresponding DID Document containing the public key of the pair pk_{id} and store the DID Document into the distributed ledger at the *index* pointed by the DID,
- *resolve* a DID, that is, retrieve the DID Document from the *index* on the ledger pointed to by the DID,
- *update* a DID, that is, generate a new key pair (sk'_{id}, pk'_{id}) and store a new DID Document at the same *index* or at a new *index* if the subject requires changing the DID, and
- *revoke* a DID, that is, provide an immutable evidence on the ledger that a DID has been revoked by the owner.

The DID Method implementation is ledger-specific and it makes the upper layers independent of the DLT of choice.

The Layer 2 makes use of DIDs and DID Documents to establish a secure channel between two peers. In principle, both peers prove the ownership of their private key sk_{id} bound to the public key pk_{id} in their DID Document that is stored on the distributed ledger. While the Layer 2 leverages DID technology (i.e. the security foundation of the SSI stack) to begin the authentication procedure, the Layer 3 finalizes it and deals with authorization to services and resources with

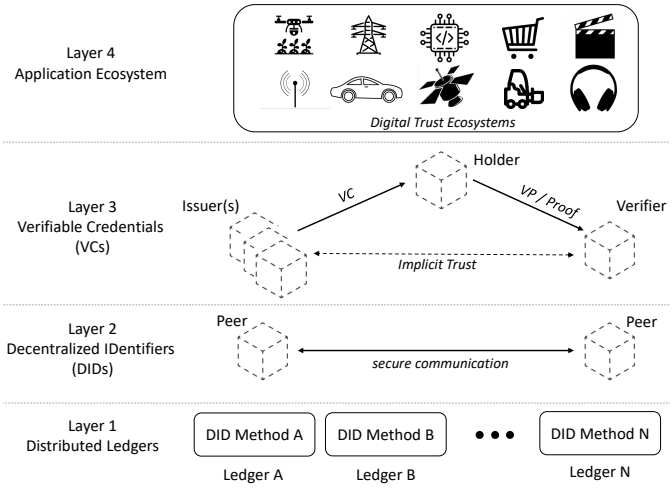

Fig. 1. The Self-Sovereign Identity stack.

This work has been developed within the MASTERMINE project (European Mining in the Green and Digital Era, https://www.mastermine-project.eu/), funded by the European Union under the Horizon Europe framework programme [GA 101091895].

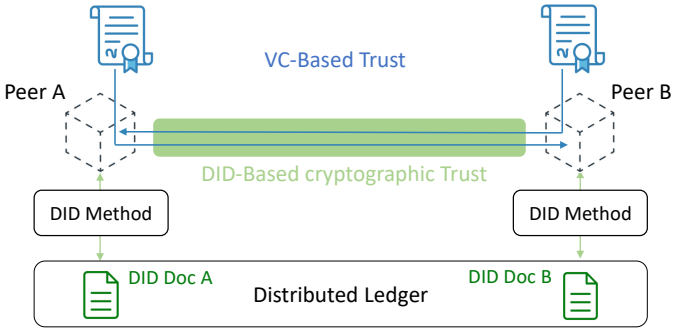

Fig. 2. Mutual authentication between two peers in the SSI framework.

Verifiable Credentials (VCs) [5].

A VC is an unforgeable, secure, and machine verifiable digital credential that contains further characteristics of the digital identity of a peer than its key pair (sk_{id}, pk_{id}), the DID and the related DID Document.

The combination of the key pair (sk_{id}, pk_{id}), the DID, the corresponding DID Document and at least one VC forms the digital identity in the SSI framework. This composition of the digital identity reflects the decentralized nature of SSI. There is no authority that provides all the components of the identity to a peer, and no authority is able to revoke completely the identity of a peer. Moreover, a peer can enrich its identity with multiple VCs issued by different Issuers.

The Layer 3 works in accordance with the Triangle-of-Trust depicted in Fig. 1. Three different roles coexist:

- **Holder** is the peer that possesses one or more VCs and that generates a Verifiable Presentation (VP) to request a service or a resource from a Verifier;
- **Issuer** is the peer that asserts claims about a subject, creates a VC from these claims, and issues the VC to the Holders.
- **Verifier** is the peer that receives a VP from the Holder and verifies the two signatures made by the Issuer on the VC and by the Holder on the VP before granting the access to a service or a resource based on the claims.

The VC contains the metadata to describe properties of the credential (e.g. context, ID, type, Issuer of the VC, issuance and expiration dates) and most importantly, the DID and the claims about the identity of the peer in the `credentialSubject` field.

The Issuer signs the VC to make it an unforgeable and verifiable digital document. The Holder requests access to services and resources from the Verifier by presenting a VP. A VP is built as an envelope of the VC. The VC is issued by an Issuer and a signature is made by the Holder with his sk_{id}. Issuers are also responsible for VCs revocation for cryptographic integrity and for status change purposes [5].

On top of these three layers, it is possible to build any ecosystem of trustable interactions among peers. The authentication process at the core of Trust between two SSI-aware peers is depicted in Fig. 2.

In principle, the peers use an (ephemeral) Diffie–Hellman key exchange to build up a confidential channel. Then, the peers exchange their respective DIDs and prove the possession of the sk_{id} associated to the pk_{id} stored in their corresponding DID Documents. This verification, in case of success, ends up with a cryptographic trust between the two peers while preventing the passive Man-in-the-Middle (MitM) attack. However, in a permissionless distributed ledger, anyone is entitled to create its own DID, therefore the procedure is still vulnerable to active MitM attack. In fact, the (mutual) authentication takes place only after the peers exchange and verify the respective VCs. At that point, the peers establish a secure communication channel.

There are Internet of Things (IoT) use cases in which networks of nodes support or make themselves digital infrastructures with security requirements, such as (mutual) authentication, confidentiality, and integrity. The SSI framework can play a key role and bring benefits with respect to centralized identity solutions [6]. The challenge is to make these solutions compatible with resource constraints [7].

With the aim of pursuing this objective, the paper proposes and discusses an alternative (mutual) authentication process for IoT nodes under the same administration domain. Finalizing the authentication by means of VC verification at Layer 3 is the most demanding operation of the authentication procedure depicted in Fig. 2 due to the complex data model of VCs [5]. The proposed alternative is to complement the DID-based verification of the sk_{id} ownership with the verification of a proof that a DID belongs to an evolving trusted set of DIDs (i.e. the DID has not been created by an adversarial node). In other words, the idea is to complement the DID-based verification of the sk_{id} ownership with the verification of a *proof of membership* and avoid the use of VCs. For the sake of clarity, the membership concept refers to DIDs and not to nodes.

From an implementation point of view, a node combines the DID with the proof of membership and forwards them to the counterpart node that proceeds with the verification of the proof of membership and of the DID ownership (i.e. sk_{id} ownership) to complete the authentication procedure.

This paper proposes and analyzes two membership solutions for the purpose of implementing the new authentication procedure: a novel solution based on Merkle trees [8] designed by the Authors of this paper and a second solution built as an adaptation of a well-known and largely used group signature scheme proposed by Boneh, Boyen and Shacham and conventionally referred to as BBS [9].

The paper presents and critically reviews the two proposed solutions in four typical operational phases of an IoT network, namely:

1) **Provisioning**: corresponding to the initial setup of the IoT nodes in the network;
2) **Operation**: corresponding to the operation of the IoT nodes when deployed on the field;
3) **Secret rotation**: corresponding to the update of node identity keys (sk_{id}, pk_{id}) and other relevant secret keys;

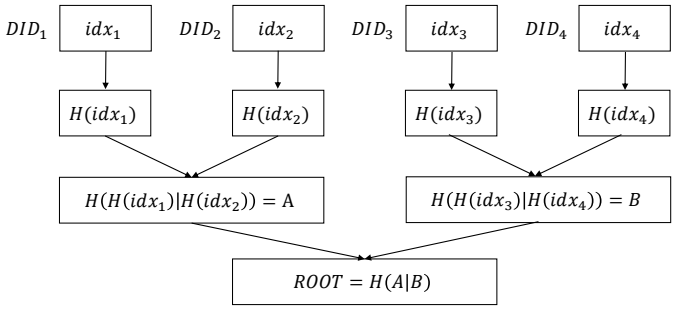

Fig. 3. Example of Merkle tree with four Decentralized IDentifiers.

4) **Network update**: corresponding to the action of either adding or removing an IoT node to/from the network.

II. Membership through Merkle trees

A Merkle tree, also known as a Hash tree, is a data structure that is used to efficiently verify the integrity of large amounts of data. It is named after its inventor, Ralph C. Merkle, who first introduced the concept in a patent filed in 1979 [8].

The Merkle tree is here used to solve the membership problem presented in Section I. In principle, the DIDs selected by a node must be part of a Merkle tree whose root is considered trusted.

A. Provisioning

The provisioning phase consists of the typical configuration procedure, in a secure environment, of each node before their deployment on the field.

Upon configuration, each node selects a first set of DIDs that will use during the operation phase (i.e. defines the indexes idx_i of these DIDs). Then, the node generates autonomously its own Merkle tree, as depicted in Fig. 3.

Basically, the node uses the selected indexes idx_i as inputs for a Hash function $H(\cdot)$: the outputs are the leaves of the Merkle tree (e.g. $leaf_1 = H(idx_1)$). Reminding that every element that is not a leaf is the digest of its child elements (e.g. $parent = H(child_1|child_2)$), the construction consists in hashing the previous two values until only a value remains; this value is called $ROOT$.

In a possible design, all the leaves of the Merkle tree can be calculated starting from a single master secret S. A HMAC-based Extract-and-Expand Key Derivation Function [10] (HKDF) can generate a number of seeds s_i, to be used as inputs for deriving the indexes of the DIDs. That way, the node is required to store securely only S and can regenerate the DIDs on the fly when needed during the operation phase, thus avoiding the secure storage of the entire set of DIDs.

After the construction of the Merkle tree, each node interacts with a Trusted Party (TP). The identity key pair of the TP is (sk_{TP}, pk_{TP}). The TP provides the public key pk_{TP} to the node and the node shares the $ROOT$ of its Merkle tree with the TP.

Once the TP has collected all the $ROOT$ values from the N nodes, it builds and publishes on the distributed ledger, at a given well-know predefined index idx_{list}, the list of trusted roots in the form:

$$\{ROOT_1, \ldots, ROOT_N, ROOT_{TP}; TS; Signature_{sk_{TP}}\}$$

where TS is a timestamp that provides the date and time of list generation, and $Signature_{sk_{TP}}$ is the signature of the TP made with its private key sk_{TP}.

All nodes have simple access to this list of trusted roots by querying the DLT. The nodes verify $Signature_{sk_{TP}}$ with the pk_{TP} before consuming the list during the operation phase.

B. Operation

The nodes enter into digital interaction with the other nodes after (mutual) authentication. Upon the selection of a DID from the tree, a node generates the proof of membership that it will use during the authentication procedure with another node, as discussed in Section I. The proof of membership coincides with the value of the Siblings of the corresponding leaf. For example, if the node n_1 selects DID_1, the proof is:

$$(Sib_1, Sib_2) \doteq (H(idx_2), H(H(idx_3)|H(idx_4))).$$

When an interaction between node n_1 and a node n_2 takes place, first n_2 sends a nonce to n_1, meant to avoid replay attacks. Then n_1 sends DID_1, the proof of membership and a signature with its sk_{id} on the message $H((Sib_1, Sib_2)|nonce)$ (i.e. $Sig_{sk_{id}}(H((Sib_1, Sib_2)|nonce))$.

At that point, n_2 verifies the signature with the pk_{id} retrieved from the DID Document pointed by DID_1, then it recalculates the root of the Merkle tree of node n_1 as:

$$ROOT_1 = H(H(H(idx_1)|Sib_1)|Sib_2).$$

The authentication succeeds if $ROOT_1$ is in the list of trusted roots, otherwise n_2 closes the communication. In case of mutual authentication, the same procedure takes place in both directions.

C. Secret rotation

Secret rotation is the process of replacing cryptographic secrets with new ones periodically or in response to a critical event. Key rotation is an example of this practice.

In case the TP needs to update its keys (sk_{TP}, pk_{TP}), the TP is required to provide to all nodes in the network its new public key pk'_{TP} and, then, to sign and publish again the list of trusted roots on the distributed ledger at idx_{list}.

Differently, when a node needs to update its identity keys (sk_{id}, pk_{id}), it selects a new DID in its Merkle tree, revokes the previous one, and generates and stores the new DID Document on the distributed ledger at the index pointed by the new DID. The proof of membership it will use during the authentication with another node changes accordingly, but since the $ROOT$ value does not change, the node does not need to interact with the TP. In other words, it updates the DID in full compliance with the SSI paradigm. Note that, identity key rotation does not necessarily imply the selection of a new

DID, because W3C DID recommendation [3] allows a node to update its DID Document without changing the DID.

A special case occurs when a node needs to update its identity keys (sk_{id}, pk_{id}) and it is using the last DID in the Merkle tree (e.g. idx_4 in Fig. 3). The node autonomously generates a new Merkle tree and, then, shares the new $ROOT'$ value with the TP upon (mutual) authentication. At that point, the node updates the DID, selecting the DID from the new tree, and the TP updates and publishes the new list of trusted roots. This key rotation process can continue indefinitely in normal operation phase in full compliance with the SSI paradigm.

D. Network update

A new node can be deployed on the network without disrupting the operation of the other nodes. When the provisioning of the new node is concluded, the TP updates the list of the trusted roots and publishes it on the distributed ledger to inform the other nodes. The same happen when a node is removed from the network for any reason; the TP updates the list of trusted roots (i.e. removes the corresponding $ROOT$ value) and publishes it on the distributed ledger.

E. Critical analysis

The following aspects are worth to be remarked:

- the solution is compliant with the SSI principles, since it does not affect the autonomy of a node to control its identity data (sk_{id}, DID, DID document);
- the decentralized nature is respected; after the provisioning phase, a node follows the SSI paradigm without requiring major interactions with the TP;
- however, the role of the TP makes it a point of attack; the TP private key sk_{TP} must be properly protected, because an adversary capable to gain access to sk_{TP} can manipulate the list of trusted roots;
- the solution scales with the number of nodes and it seems to be appropriate also for networks with high update frequency. Adding or removing a node only affects the size of the list of trusted roots (i.e. proportional to the number of active nodes) but it does not imply interactions with the other nodes in the network;
- a trade-off exists between the size of the Merkle tree, the size of the proof of membership (i.e. the number of Siblings) and the number of interactions with the TP. The larger the tree size, the larger the proof, but the lower the number of interactions with the TP to send the root of a new Merkle tree;
- the security of the membership solution relies on the preimage resistance property of the hash function (i.e. it is hard to invert) used to build the tree and on the HKDF used as a source of randomness to build the seeds; in this sense, it is reasonable to consider it quantum safe [11];
- both the Merkle tree and HKDF are mature technologies. However, their combined use in the proposed solution may need further security validation.

III. Membership through BBS Group Signature

The BBS group signature scheme [9] has been developed to allow a member of the group to secretly sign a message on the group's behalf with its group private key gsk_i; the signature of any member can be verified with the unique group public key gpk. Moreover, the scheme allows a TP to revoke the private key of any member, triggering the update of the private keys of all the other members and of the group public key.

The BBS scheme is here adapted to solve the membership problem presented in Section I. In principle, a node proves that its DID belongs to a trusted set by means of a BBS signature. The paper here adopts the notation from [9] and, when appropriate, directly refers to specific BBS algorithms, namely $KeyGen$, $Sign$, $Verify$, $Update$, $Open$, $Join$, and $Revoke$.

A. Provisioning

The provisioning phase consists of the common configuration procedure in a secure environment of each node before deployment on the field.

A TP supervises this phase and begins the provisioning by performing the $KeyGen$ algorithm. This step consists in the generation of the TP key pair (sk_{TP}, pk_{TP}), the group public key gpk, the TP group private key $tpsk$, and the private keys gsk_i for all nodes in the network.

The TP provides any node with its group private key gsk_i, the group public key gpk, the $index_{RL}$ on the distributed ledger where to retrieve a Revocation List, and the public key pk_{TP} to verify the TP signature on such list, as will be explained in detail in Section III-B.

According to the original $Join$ algorithm in [9], the TP generates and gives a group private key gsk_i to each node; this protocol implies that the TP knows all the group private keys, making it a single point of attack. An alternative $Join$ algorithm, proposed in [12], introduces the property of *Strong Exculpability* (SE) and, for clarity, will be denoted as $Join_{SE}$ in the following discussion. The SE concept is an evolution of the exculpability concept that was first introduced by [13]. In accordance with its definition in [12], [14], SE ensures that no member of the group and not even the entity that issues the private keys can forge a signature on behalf of another group member.

The authors of BBS in [9] suggested acquiring the SE property by generating each gsk_i via a procedure in which the TP only learns a share of gsk_i. However, to the best of our knowledge, beside this suggestion, no practical implementation of the $Join_{SE}$ algorithm for BBS has been published. Therefore, we here propose for the first time an implementation tailored to the BBS group signature.

Firstly, the $KeyGen$ algorithm must be modified to add one more base to the original $gpk = (g_1, g_2, h, u, v, w)$ where $g_1, h, u, v \in \mathbb{G}_1$ and $g_2, w \in \mathbb{G}_2$ with $\mathbb{G}_1, \mathbb{G}_2$ multiplicative cyclic groups of prime order. Accordingly, the TP selects at random $h_1 \xleftarrow{R} \mathbb{G}_1$ and adds it to the new $gpk = (g_1, g_2, h, h_1, u, v, w)$.

Then, the $Join_{SE}$ algorithm can be constructed as follows:

1) the node n_i selects at random $y_i \xleftarrow{R} \mathbb{Z}_p^*$ and sends $Y = h_1^{-y_i}$ to the TP;
2) given γ a TP's random secret value defined as $\gamma \in \mathbb{Z}_p^*$, the TP selects $x_i \xleftarrow{R} \mathbb{Z}_p^*$, computes $A_i = (g_1 Y)^{\frac{1}{\gamma + x_i}}$ and $H_i = h_1^{\frac{1}{\gamma + x_i}}$, and sends H_i to n_i;
3) n_i sends $B_i = H_i^{-y_i}$ back to the TP;
4) the TP computes $A_i' = B_i(g_1^{\frac{1}{\gamma + x_i}})$ and check if $A_i' = A_i$ to convince itself that n_i knows y_i;
5) if and only if previous step 4) succeeds, the TP sends (A_i, x_i) to n_i;
6) finally, n_i builds its entire private key $gsk_i = (A_i, x_i, y_i)$.

It is worth noting that only n_i knows y_i. The Discrete Logarithm's Problem (DLP) protects y_i from being discovered by the TP that, in fact, only knows $Y = h_1^{-y_i}$ and (A_i, x_i). In our solution n_i proves the knowledge of the private key share y_i in Zero-Knowledge.

The $Sign$, $Verify$, $Update$, $Open$, $Join$, and $Revoke$ algorithms in [9] must be accordingly adapted to the new definition of $gsk_i = (A_i, x_i, y_i)$. The adaptation consists in adding the Zero-Knowledge proof of knowledge of the entire group private key gsk_i, following the same approach used in constructing the $Join_{SE}$. These adaptations are here omitted for conciseness.

B. Operation

A node generates the DID and enters into interaction with other nodes of the network after proper (mutual) authentication. A node n_1 computes the proof of membership (i.e. the BBS signature) running the $Sign$ algorithm using its gsk_1 on a digest computed as $H(DID_{n_1}|nonce)$, where the $nonce$ is generated by the counterpart node n_2 to avoid replay attacks. The node n_2 can verify the proof of membership with the $Verify$ algorithm using the group public key gpk and, then, the ownership of sk_{id} of n_1. In case of mutual authentication, the same procedure takes place in the two directions.

In any case, each node must maintain its own group private key and the group public key up to date. After the revocation of a key gsk_r, all nodes must update their own private key gsk_i and gpk with the $Update$ algorithm (see Sect. 7 in [9]). The $Update$ algorithm requires some knowledge of the revoked private keys. For this reason, the TP publishes a list of such knowledge, i.e. a Revocation List (RL), on the distributed ledger at a well-know $index_{RL}$. All the nodes in the network can easily access this list by querying the distributed ledger.

According to the new $Join_{SE}$ algorithm, the RL contains a processed version of the share of the revoked private keys $(gsk_r^*, \ldots, gsk_s^*)$ known by the TP. The RL has the form:

$$\{gsk_r^*, \ldots, gsk_s^*; TS; Signature_{sk_{TP}}\}$$

where TS is a timestamp that provides the date and time of the list, and $Signature_{sk_{TP}}$ is the signature of the TP. The nodes whose group private key gsk_r is in the RL, cannot update their own gsk_r by design of the $Update$ algorithm [9], hence they are no more able to generate a valid proof of membership.

C. Secret rotation

A node is able to update its identity keys (sk_{id}, pk_{id}) for key rotation purpose, and the respective DID and DID Document, in full compliance with SSI paradigm, without the need to update its gsk_i. The node must only generate the new proof when starting the authentication procedure with another node of the network.

Differently, if the TP needs to update its keys (sk_{TP}, pk_{TP}), the TP has to share the new pk'_{TP} with all the nodes, then sign with sk'_{TP} and publish again the RL.

Moreover, if the TP needs to update its group secret key $tpsk$ for rotation purpose, the TP has to start a new provisioning phase to provide all the nodes with new group keys (i.e. gsk'_i, gpk').

D. Network update

A new node can be deployed on the network without disrupting the operation of the other nodes. The TP concludes the $Join_{SE}$ procedure with the new node and, then, shares the group public key gpk, $index_{RL}$, and pk_{TP} with it.

On the contrary, when a node is removed from the network for any reason, the TP performs the $Revoke$ algorithm and publishes the new RL. This revocation action causes all the other nodes to update their group keys.

It is worth noting that the BBS group signature scheme provides a specific algorithm, named $Open$, that can be used to trace a signature to a signer (i.e. retrieve a share of gsk_r of the signer from a signature). This tools can be useful to detect a misbehaving node (e.g. a compromised node) and revoke its group private key gsk_r.

E. Critical Analysis

The following aspects are worth to be remarked:

- the solution is compliant with the SSI principles, since it does not affect the autonomy of a node to control its identity data (sk_{id}, DID, DID document);
- beside the provisioning phase where the TP provides the group keys to every node, the solution respects the decentralized nature of SSI;
- the revocation of a group private key implies some operations to be executed by all the other nodes (i.e. they check the latest RL to update their group private keys gsk_i and public key gpk with the $Update$ algorithm);
- the TP can be identified as a single point of attack. Both private keys sk_{TP} and $tpsk$ must be properly protected. An adversary gaining access to those secrets can add a malicious node to the network and revoke the capability of a legitimate node to make valid proofs of membership. The $Join_{SE}$ protocol offers a protection against the adversaries willing to generate a valid proof of membership on behalf of another node, since the TP does not know the full gsk_i;
- the solution scales as the number of nodes increases. However, each revocation triggers the update of the group keys in all the other nodes. Notably, the size of the RL could grow with the number of revocations;

- the solution ensures total flexibility for the node to deal with its DIDs. In fact, once a node is provisioned with a valid gsk_i, it can freely create and update its DIDs, proving that they are in a trusted set by means of the BBS signature. Notably, the signature has a constant size and there is not a trade-off between the dimension of the proof and other parameters of the solution;
- the security of the BBS group signature scheme relies on the Linear assumption and on the Strong Diffie-Hellman assumption. As a consequence, it can be considered vulnerable to attack by quantum computers [15];
- finally, this solution is based on an already well-established and mature construction (i.e. the BBS scheme), that can be used with minor modifications.

IV. Performance estimation

The feasibility of the two solutions is here addressed by estimating and comparing their computational load and expected performances on a target IoT node (i.e. Raspberry Pi® 4 Model B, 4 GB RAM, 1.5 GHz processor [16]).

This work adopts the same methodology applied in [17] to estimate and to compare the execution time of the cryptographic operations in the four different operational phases.

First of all we have measured on the selected IoT node the execution time of the specific cryptographic algorithms heavily used as elemental building blocks in the two solutions under evaluation (i.e. hash computation, scalar multiplication, exponentiation, and pairing). Table I shows the results of measurements assuming a 128-bit security level.

The initial benchmark shows that the `sha256` hash computation lasts 4 µs and it is the less expensive cryptographic algorithm, whereas the paring computation lasts 50.4 ms and it is the most expensive one, as expected. As an additional remark, the results in Table I are consistent with the values reported in [17], taking into account the different processor clock speed of the target nodes (i.e. 1.5 GHz versus 1.2 GHz).

These results are the basis for estimating and comparing the execution time of the two proposed solutions, as reported in the following subsections.

A. Results for Merkle tree-based solution

The Merkle tree-based solution implies a computational load proportional to the size of the tree that, according to the structure in Fig. 3, depends on the number of leaves on the tree (i.e. DIDs).

Let us denote the number of leaves with k. This value is the key parameter to estimate the execution time in the four operational phases. In fact, it corresponds to the number of seeds s_i to be generated by the HKDF and to the number of inputs to the hash algorithm to derive the indexes of the DIDs (i.e. leaves of the Merkle tree). In addition, the number of leaves k has an impact on the computational load to generate the Merkle tree, to create a proof of membership using the proper siblings and to verify a proof of membership given the siblings.

TABLE I
Benchmark results for specific cryptographic algorithms

Cryptographic Algorithm	Notation	Time (ms)
Hash computation (`sha256`)	$\mathbf{h}$	0.004
Scalar multiplication in $\mathbb{G}_1$	$\mathbf{m}$	4.6
Exponentiation in $\mathbb{G}_T$	$\mathbf{e}$	33.6
Ate pairing e	$\mathbf{P}$	50.4

TABLE II
Estimated performance for Merkle tree-based solution

Operational Phase	Estimated Computations	Time (ms)
Provisioning	$\mathbf{h}(4k+1)$	0.516
Operation (Proof)	$\mathbf{h}(4k+1)$	0.516
Operation (Verify)	$\mathbf{h}(\log_2(k)+1)$	0.024
Secret Rotation	none or $\mathbf{h}(4k+1)$	$\leq$ 0.516
Network Update	none	0

TABLE III
Estimated performance for BBS-based solution

Operational Phase	Estimated Computations	Time (ms)
Provisioning	$2\mathbf{m}$	9.2
Operation (Proof)	$\mathbf{h}+5\mathbf{m}+2(2\star\mathbf{m})+3\star\mathbf{e}$	75.7
Operation (Verify)	$\mathbf{h}+4(2\star\mathbf{m})+4\star\mathbf{e}+\mathbf{P}$	115.3
Secret Rotation	none or $2\mathbf{m}$	$\leq$9.2
Network Update	none or $2\star\mathbf{m}$	$\leq$5.4

Table II reports the number of required computations and the estimated execution times for the selected IoT node, assuming a Merkle tree with 32 leaves (i.e. $k = 32$). These results neglect the operations executed by the TP and other operations with a limited impact on the computational load of the node (e.g. random number generations). The rows of Table II represents the operational phases; it must be noted that the *Operation* phase is split between the generation of a proof of membership (i.e. *Proof*) and its verification (i.e. *Verify*), since they can be executed by two distinct nodes (i.e. n_1 and n_2, as explained in Section II-B).

The generation of k seeds s_i with HKDF requires, according to [10], the following computations:

- $2\mathbf{h}$ for the initial `HKDF-Extract` function, and
- $2\mathbf{h}k$ to generate k seeds with `HKDF-Expand` function.

Moreover, the generation of a Merkle tree from k seeds requires $2k-1$ hash computations and, thus, a time equal to $\mathbf{h}(2k-1)$.

From these remarks, it is possible to state that the *Provisioning* phase consists in the generation of k seeds with HKDF plus the construction of a Merkle tree and, thus, it requires $2\mathbf{h}+2\mathbf{h}k+\mathbf{h}(2k-1)=\mathbf{h}(4k+1)$.

On the other hand, the verification of a proof of membership implies $\mathbf{h}(\log_2(k)+1)$ to compare the siblings against the $ROOT$ value.

Assuming that each node stores only a single master secret S and regenerates the seeds s_i and the Merkle tree on the fly when needed, thus avoiding the secure storage of the entire set of DIDs, the number of computations, hence the execution times, can be derived in the same way.

B. Results for BBS-based solution

The computation load for the BBS-based solutions has been estimated by considering the analytical results in [17]. Table III shows the number of required computations and the estimated execution times for the selected IoT node.

This work considers all the optimizations suggested in [17], especially the general low level optimizations proposed in [18], the optimal Ate pairing implementation in [19], and the suggestions in Section 6 of [9] to eliminate all the pairings in the computation of a BSS signature and to perform only one pairing to verify a signature.

Moreover, the BBS scheme requires several multi-scalar multiplication and exponentiation operations, with a remarkable computational load. For this reason, Table III denotes a multi-scalar multiplication in $\mathbb{G}_1$ with $\ell \star \mathbf{m}$, while a multi-scalar exponentiation in $\mathbb{G}_T$ is denoted as $\ell \star \mathbf{e}$, where ℓ is the total number of multiplication or exponentiation operations to be executed. For example, the second row of Table III reports a double scalar multiplication as $2 \star \mathbf{m}$, while $3 \star \mathbf{e}$ denotes a triple exponentiation.

This work considers also an optimization of these multi-scalar multiplication and exponentiation operations using a generalization of the Shamir's trick [20] that, according to [17], allows accelerating these computations by a factor equal to $\frac{2^{\ell+1}-1}{3\times 2^{\ell-1}}$.

It must be noted that the results about the Operation phase (both Proof and verify) include a `sha256` digest computation that is executed before running the BBS $Sign$ and $Verify$ algorithms, respectively, according to Section III-B.

V. Comparative analysis

The two proposed approaches show some similarities. Both solutions take advantage of mature building blocks (i.e. Merkle tree, HKDF, and BBS algorithms) and both comply with SSI principles (i.e. they do not interfere with the decision of a node to create, update or revoke a DID, just add the mechanism to prove that the DID belongs to an evolving trusted set). Moreover, both solutions respect the decentralized nature of SSI, because, apart from the initial provisioning phase, they do not strictly require other direct interactions between the TP and the nodes.

The TP is a single point of attack in both solutions, but with a difference. In the Merkle tree-based solution, an adversary capable to gain access to sk_{TP} can arbitrarily compromise the list of trusted roots; in the BBS-based solution the adversary must gain access also to $tpsk$ to be able to add a malicious node to the network, or to revoke the capability of a legitimate node to make valid proofs of membership. In any case, a compromised TP has not direct access to the critical secrets of the nodes, especially their identity private keys sk_{id}.

The main difference between the two solutions resides in the provisioning phase. In the Merkle tree-based solution a node builds on its own the knowledge to generate the proof of membership (i.e. the Merkle tree). In the BBS-based solution the TP generates and provides that knowledge to the node (i.e. gsk_i). The $tpsk$ is the secret underpinning the group scheme. When the TP needs to update $tpsk$, for rotation purpose, the TP must start a new provisioning phase with all single nodes. In the former solution the nodes cyclically refresh their secrets (i.e. the Merkle trees) and share the $ROOT$ values with the TP during the operation phase without interrupting their operation. Moreover, the adoption of a group signature scheme implies a less efficient revocation procedure, because it requires all nodes to update their group keys gsk_i and gpk every time the TP revokes a group private key. However, apart from these disadvantages, the BBS-based solution provides full flexibility in DID creation, ensures a constant size for the proof that a DID belongs to a trusted set and does not impose any design constraint on the DID update. In fact, a node can potentially generate on the fly an unlimited number of DIDs, without the need to find a trade-off between the Merkle tree size and the number of interactions with the TP.

As far as the performance of the two solutions is concerned, the Merkle tree-based solution clearly outperforms the BBS-based solution in all the considered operational phases, especially in the Operation (Verify) and Network Update phases. It mainly relies only on fast hash computations and it does not require pairing computation at every verification or specific update operations at every revocation.

For these reasons, the Merkle tree-based approach can be considered the most appropriate solution for IoT networks. On the other hand, the BBS-based solution could be of interest for possible use cases that require a constant/small size for the proof of membership in order to minimise the data exchange between nodes.

VI. Conclusions and future works

This paper has proposed an alternative (mutual) authentication process for a network of IoT nodes leveraging SSI. The main idea is to complement the DID-based verification of the identity private key ownership with the verification of a proof of membership during the (mutual) authentication process. The paper has analyzed two membership solutions, a novel solution designed by the Authors based on Merkle trees and a second solution built as an adaptation of the BBS group signature scheme. The performance evaluation has provided an estimate of the computational load on an IoT node for each method, while the comparative analysis has highlighted the advantages and drawbacks of both solutions.

Future works will focus on (*i*) the adoption of threshold signature schemes to reduce the impact of a possible attack to the TP, and (*ii*) the adoption of dynamic accumulators and their properties to build another possible alternative.

Acknowledgment

The Authors would like to thank Alberto Carelli for his technical support to the performance estimation of the two proposed solutions.

References

[1] A. Preukschat and D. Reed, *Self-Sovereign Identity – Decentralized digital identity and verifiable credentials*. Shelter Island, NY: Manning, 2021. [Online]. Available: https://www.manning.com/books/self-sovereign-identity

[2] N. Kannengießer, S. Lins, T. Dehling, and A. Sunyaev, "Trade-offs between distributed ledger technology characteristics," *ACM Computing Surveys*, vol. 53, no. 2, pp. 1–37, 2020.

[3] W3C, "Decentralized Identifiers (DIDs) v1.0. Core architecture, data model, and representations. W3C Recommendation," 2022. [Online]. Available: https://www.w3.org/TR/did-core/

[4] ——, "DID Specification Registries. The interoperability registry for Decentralized Identifiers. W3C Group Note," 2023. [Online]. Available: https://www.w3.org/TR/did-spec-registries/

[5] ——, "Verifiable Credentials Data Model v1.1. W3C Recommendation," 2022. [Online]. Available: https://www.w3.org/TR/vc-data-model/

[6] J. Won, A. Singla, E. Bertino, and G. Bollella, "Decentralized public key infrastructure for internet-of-things," in *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*, 2018, pp. 907–913.

[7] O. Hahm, E. Baccelli, H. Petersen, and N. Tsiftes, "Operating systems for low-end devices in the internet of things: A survey," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 720–734, 2016.

[8] R. C. Merkle, "Method of providing digital signatures," US Patent 4309569, filed on Sep. 5, 1979. [Online]. Available: https://patents.google.com/patent/US4309569A/en?oq=US4309569A

[9] D. Boneh, X. Boyen, and H. Shacham, "Short group signatures," in *Advances in Cryptology – CRYPTO 2004*, M. Franklin, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 41–55.

[10] H. Krawczyk and P. Eronen, "HMAC-based extract-and-expand key derivation function (HKDF)," May 2010. [Online]. Available: https://datatracker.ietf.org/doc/pdf/rfc5869.pdf

[11] J. P. Mattsson, B. Smeets, and E. Thormarker, "Quantum-resistant cryptography," arXiv, 2021. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/2112/2112.00399.pdf

[12] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik, "A practical and provably secure coalition-resistant group signature scheme," in *Advances in Cryptology — CRYPTO 2000*, M. Bellare, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 255–270.

[13] G. Ateniese and G. Tsudik, "Some open issues and new directions in group signatures," in *Financial Cryptography*, M. Franklin, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 196–211.

[14] M. Bellare, D. Micciancio, and B. Warinschi, "Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions," in *Advances in Cryptology — EUROCRYPT 2003*, E. Biham, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 614–629.

[15] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1484–1509, oct 1997. [Online]. Available: https://doi.org/10.1137%2Fs0097539795293172

[16] Raspberry Pi® Trading Ltd, "Raspberry Pi® 4 Computer Model B, Product brief," January 2021. [Online]. Available: https://datasheets.raspberrypi.org/rpi4/raspberry-pi-4-product-brief.pdf

[17] S. Canard, N. Desmoulins, J. Devigne, and J. Traoré, "On the implementation of a pairing-based cryptographic protocol in a constrained device," in *Pairing-Based Cryptography – Pairing 2012*, M. Abdalla and T. Lange, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 210–217.

[18] Z. Cheng and M. Nistazakis, "Implementing pairing-based cryptosystems," *Proceedings of IWWST*, 2005.

[19] J.-L. Beuchat, J. E. González-Díaz, S. Mitsunari, E. Okamoto, F. Rodríguez-Henríquez, and T. Teruya, "High-speed software implementation of the optimal ate pairing over barreto–naehrig curves," in *Pairing-Based Cryptography - Pairing 2010*, M. Joye, A. Miyaji, and A. Otsuka, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 21–39.

[20] B. Möller, "Algorithms for multi-exponentiation," in *Selected Areas in Cryptography*, S. Vaudenay and A. M. Youssef, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 165–180.