Journal of Information Security and Applications 83 (2024) 103789

Contents lists available at ScienceDirect

Journal of Information Security and Applications

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

On cryptographic mechanisms for the selective disclosure of verifiable credentials

Andrea Flamini [a,*], Giada Sciarretta [b], Mario Scuro [a], Amir Sharif [b], Alessandro Tomasi [b], Silvio Ranise [a,b]

[a] Department of Mathematics, University of Trento, Trento, Italy
[b] Center for Cybersecurity, Fondazione Bruno Kessler, Trento, Italy

ARTICLE INFO

Keywords:
Selective disclosure
Verifiable credentials
Zero-knowledge proof
eIDAS 2
GDPR

ABSTRACT

Verifiable credentials are a digital analogue of physical credentials. Their authenticity and integrity are protected by means of cryptographic techniques, and they can be presented to verifiers to reveal attributes or even predicates about the attributes included in the credential. One way to preserve privacy during presentation consists in selectively disclosing the attributes in a credential. In this paper we present the most widespread cryptographic mechanisms used to enable selective disclosure of attributes identifying two categories: the ones based on hiding commitments - e.g., mDL ISO/IEC 18013-5 - and the ones based on non-interactive zero-knowledge proofs - e.g., BBS signatures. We also include a description of the cryptographic primitives used to design such cryptographic mechanisms.

We describe the design of the cryptographic mechanisms and compare them by performing an analysis on their standard maturity in terms of standardization, cryptographic agility and quantum safety, then we compare the features that they support with main focus on the unlinkability of presentations, the ability to create predicate proofs and support for threshold credential issuance.

Finally we perform an experimental evaluation based on the Rust open source implementations that we have considered most relevant. In particular we evaluate the size of credentials and presentations built using different cryptographic mechanisms and the time needed to generate and verify them. We also highlight some trade-offs that must be considered in the instantiation of the cryptographic mechanisms.

1. Introduction

As more services move online, increasing importance is given to an individual's digital identity as the foundation for secure and trusted online interactions, related to e-government, e-commerce, and e-health to name just a few.

A new paradigm for identity management based on digital identity wallets is emerging to empower data subjects to selectively disclose the user attributes within what is called verifiable credentials in a privacy-preserving and secure way. A verifiable Credential is a digital attestation or evidence of particular information about an individual that is intended to be cryptographically secure, and tamper-proof. The most prominent example of the aforementioned paradigm is the revised regulation eIDAS 2 [1], proposing a European Digital Identity (EUDI) wallet that can be used by the user to securely store the issued verifiable credentials and aims to improve cross-border interoperability. The privacy-enhancing aims of the EUDI wallet include offering data subjects the means to control who has access to which of their personally identifiable information, and making it possible to selectively disclose only some of the attributes in their verifiable credentials to trusted parties. When a service provider requests too many subject claims, it may dissuade users from utilizing the service. Furthermore, extensive data collection increases the risk of data breaches or misuse, and does not follow data minimization and privacy by design principles under the GDPR [2], a basic part of data protection.

In the design of their protocols and implementations, service providers must consider trade-offs between simplicity vs sophistication of protocol, implementation, and deployment issues including resource constraints.

Scenario. To exemplify selective disclosure, we consider the following simplified scenario: a subject wishes to purchase alcohol and prove that

* Corresponding author.
E-mail addresses: andrea.flamini@unitn.it (A. Flamini), g.sciarretta@fbk.eu (G. Sciarretta), mario.scuro@studenti.unitn.it (M. Scuro), asharif@fbk.eu (A. Sharif), altomasi@fbk.eu (A. Tomasi), ranise@fbk.eu (S. Ranise).

https://doi.org/10.1016/j.jisa.2024.103789

Available online 18 May 2024

(s)he is over the legal age limit in the jurisdiction, e.g., 18, without fully disclosing her entire mobile driving license (mDL).

In this example, the agency in charge of issuing mDL (Issuer) verifies the mDL Subject's age during the issuance process and includes it as an attribute in the mDL. The data Subject holding the mDL can select to disclose the single mDL attribute "age" to the liquor store employee (Verifier). The Verifier can check that the Subject is of age to buy alcohol without learning any other personal information.

This enhances privacy for the Subject while enabling the Verifier to check their age while complying with the data minimization principle.

Contributions. eIDAS 2 states that EUDI wallets "should technically enable the selective disclosure of attributes within verifiable credentials", and amendments to the proposal add "where attestation of attributes does not require the identification of the user, zero-knowledge attestation shall be performed" [3]. The EUDI Wallet Architecture and Reference Framework (ARF) [4], intended to provide more concrete technical guidelines and tools, states that "attestation MUST enable Selective Disclosure of attributes by using Selective Disclosure for JSON Web Tokens (SD-JWT) and Mobile Security Object (ISO/IEC 18013-5) scheme".

Both schemes cited in the ARF are based on hiding commitment mechanisms — generating a commitment to a value while keeping it hidden, with the ability to reveal the committed value later [5]. The ARF does not currently cover zero-knowledge proofs (ZKP) - e.g., repeatedly proving knowledge of a value without ever having to reveal it [6–8]. Given the complexity and range of available options, it is non-trivial to assess the pros and cons of each option.

Currently, the literature lacks a comprehensive study that analyses and compares different solutions based on their maturity and computational overhead in the context of digital identity wallets. There are research papers on the application of selective disclosure mechanisms to other specific use cases, e.g., education [9–11] or presenting a new proposed scheme for selective disclosure mechanisms [12–18]. We identified only one systematization of knowledge work in this field [19], but the focus is on a specific selective disclosure technique (Zero-Knowledge Range Proofs) applied to a different use case (cryptocurrencies).

To fill this gap and facilitate an informed choice, we provide cryptographic building blocks for credentials with selective disclosure capability based on hiding commitments and ZKP. In short, we extend our work in [20] and make the following main contributions:

- We summarize six cryptographic mechanisms (`cm`) for selective disclosure based on hiding commitment and ZKP, providing more detail than [20] and two new `cm`, BBS (Section 5.2) and PS (Section 5.4) signatures.
- We provide the structure of Verifiable Credentials and Presentations for the `cm`, together with the operation of entities that must be performed for their creation (issuing) and consumption (presentation).
- We compare the `cm` w.r.t. several features to assist in selecting the most appropriate for the use case of interest.
  Our analysis has been expanded over [20] by considering the following features: quantum safety of cryptographic algorithms, support for threshold credential issuance, and an analysis of trade-offs that lead to interesting implementation choices in the solutions we examined. While the first one has been considered to evaluate the maturity of cryptographic mechanisms w.r.t. quantum resistance, the rest have been considered to evaluate for each cryptographic mechanism how they support features that are relevant to the design and implementation of practical privacy preserving Verifiable Credentials.

Outline. Section 2 introduces the verifiable credential ecosystem, the formats of verifiable credentials that support the selective disclosure of attributes, and their lifecycle. Section 3 introduces the cryptographic primitives used to implement the cryptographic mechanisms described in Sections 4 and 5. In Section 6, we analyze the mechanisms and discuss how they support some privacy-enhancing features. In Section 7, we perform an experimental evaluation of the mechanisms described. We summarize the main results and discuss future work in Section 8.

For the interested reader, further cryptographic details on zero-knowledge proofs are provided in Appendix, where we moved this material to ease reading of the main text.

2. Verifiable credentials and selective disclosure

Following the Verifiable Credential data model [21], a credential can be defined as "a set of one or more claims [assertions about a Subject] made by an Issuer", and a Verifiable Credential (VC) as "a tamper-evident credential that has authorship that can be cryptographically verified". We consider the following entities and quote the descriptions from [22]:

Issuer: "a role an entity can perform by asserting claims about one or more subjects, creating a VC from these claims, and transmitting the VC to a holder".

Holder: "a role an entity might perform by possessing one or more VCs and generating presentations from them".

Subject: "the entity about which claims are made".

Verifier: "a role an entity performs by receiving one or more VCs, optionally inside a verifiable presentation" and verifies it "to make a decision regarding providing a service to the Subject".

We describe the general structure of VCs and Verifiable Presentations (VPs) regardless of the cryptographic mechanism used.

2.1. Verifiable credentials and presentations

A VC is composed of three sections (see Table 1): an *Issuer protected header*, containing general information about the credential, for instance the Issuer, the Subject and the credential type, an *Issuer payload* containing information about the credential attributes, and an *Issuer proof* which contains the cryptographic material which attests the authenticity of the credential.

A VP is composed of three sections (see Table 2): a *presentation protected header* with general information about the credential; a *presentation payload* with information related to the disclosed attributes; and a *presentation proof* with the cryptographic material that allows the Verifier to check the authenticity of the presentation.

The structure of the VC and VP we adopt is consistent, albeit simplified to focus on selective disclosure, with the structure of JSON Web Proof (JWP) [23], a proposal to standardize a JSON container which aims to describe the structure of VCs to allow the selective disclosure of attributes.

In a preliminary *set-up phase*, the Issuer must generate its private-public key pair $(\mathtt{sk}_{\mathtt{Iss}}, \mathtt{pk}_{\mathtt{Iss}})$ using the key generation function of the digital signature scheme used to sign the VCs, `keyGen()`. In the *issuing phase*, the Issuer generates an Issuer proof with the function `genIssuerProof(−)`. The Holder, upon reception of the VC created by the Issuer, verifies its validity computing the function `verIssuerProof(−)`. In the *presentation phase* the Holder can create a VP specifying the attributes it wants to disclose. In particular, the Holder creates the VP containing the *Holder-generated proof* by computing the function `genHolderProof(−)`. The Verifier, upon reception of the VP computes the function `verPresentProof(−)` to verify it and possibly accept the Holder's claims.

Table 1
A simplified representation of VC which allows for the selective disclosure of attributes.

VC	Hiding-commitment	Selective disclosure signature
Issuer protected header	Cryptographic mechanism: cm Issuer public key: $\mathtt{pk}_{\mathtt{Iss}}$	Cryptographic mechanism: cm Issuer public key: $\mathtt{pk}_{\mathtt{Iss}}$
Issuer payloads	Attributes and salts: $\mathtt{A} = (a_1, \ldots, a_m)$ $\mathtt{S} = (s_1, \ldots, s_m)$	Attributes: $\mathtt{A} = (a_1, \ldots, a_m)$
Issuer proof	Signed commitment: $\mathtt{genIssuerProof}(\mathtt{sk}_{\mathtt{Iss}}, \mathtt{A}, \mathtt{S})$ $= (\mathtt{CMT}, \sigma = \mathtt{genSig}(\mathtt{sk}_{\mathtt{Iss}}, \mathtt{CMT}))$	Selective disclosure signature: $\mathtt{genIssuerProof}(\mathtt{sk}_{\mathtt{Iss}}, \mathtt{A})$ $= \sigma = \mathtt{genSig}(\mathtt{sk}_{\mathtt{Iss}}, \mathtt{A})$

Table 2
The general structure of a VP derived from a VC as in Table 1.

VP	Hiding-commitment	Selective disclosure signature
Presentation protected header	Cryptographic mechanism: cm Issuer public key: $\mathtt{pk}_{\mathtt{Iss}}$	Cryptographic mechanism: cm Issuer public key: $\mathtt{pk}_{\mathtt{Iss}}$
Presentation payloads	Disclosed attributes and salts: $\mathtt{DA} = (a_{i_1}, \ldots, a_{i_d}) \subset \mathtt{A}$ $\mathtt{DS} = (s_{i_1}, \ldots, s_{i_d}) \subset \mathtt{S}$	Disclosed attributes: $\mathtt{DA} = (a_{i_1}, \ldots, a_{i_d}) \subset \mathtt{A}$
Presentation proof	Signed commitment: $(\mathtt{CMT}, \sigma)$ Holder-generated Proof: $P = \mathtt{genHolderProof}(\mathtt{DA}, \mathtt{DS}, \mathtt{A}, \mathtt{S})$	Holder-generated proof: $P = \mathtt{genHolderProof}(\mathtt{pk}_{\mathtt{Iss}}, \mathtt{DA}, \mathtt{A}, \sigma)$

2.2. Taxonomy of cryptographic techniques for VC selective disclosure

There are several methods that allow VCs to support selective disclosure. [24] identifies the following categories: *atomic credentials*, *hashed values* and *selective disclosure signatures* (which in literature are also referred to as *anonymous credentials* [6,15,16]). Atomic credentials contain only a single attribute, therefore the Issuer may provide a set of atomic credentials, then the Holder presents to a Verifier only those that it wants to show. Atomic credentials are unwieldy to manage, particularly to guarantee that a presentation contains a collection of atomic credentials that is valid as a whole, but do not introduce or require substantially different cryptographic techniques than the other two mechanisms; therefore, we do not discuss them further. Instead we focus on the other two categories of mechanisms:

Hashed values allow an Issuer to issue a single VC containing multiple claims. Each claim is hidden and committed to using hash functions, then the commitment is signed by the Issuer. Examples include hash lists (Section 4.1) and Merkle trees (Section 4.2).

Selective disclosure signatures are signatures schemes that natively support selective disclosure of VC claims by using non-interactive zero-knowledge proofs. Examples are *CL* (Section 5.1), *BBS* (Section 5.2), *BBS+* (Section 5.3) and *PS* (Section 5.4) signatures.

We provide noteworthy examples of cryptographic mechanisms based on hashed values, considered as an instance of hiding commitments, which are adopted in the standardized mobile Driving License [25] or discussed in [5] (Section 4). We also present examples of the most relevant selective disclosure signatures adopted in [6–8] (Section 5).

In Table 3 we report all the acronyms, functions and variables used in the paper.

3. Background on cryptographic building blocks

We provide the main cryptographic notions that are useful to understand the approaches for the creation of VCs supporting selective disclosure of attributes: digital signatures (Section 3.1), hashing and salting for the creation of hiding commitments (Section 3.2), and Non Interactive ZKP (NIZKP) to prove statements about undisclosed attributes in selective disclosure signatures (Section 3.3). We also highlight the threat models that are useful to understand the security properties satisfied by each cryptographic primitives.

Table 3
List of acronyms, functions and variables.

$\mathtt{keyGen}(-)$	Digital signature key generation algorithm
$\mathtt{genSig}(-)$	Signature generation algorithm
$\mathtt{verSig}(-)$	Signature verification algorithm
$\mathtt{genIssuerProof}(-)$	Issuer proof generation algorithm
$\mathtt{verIssuerProof}(-)$	Issuer proof verification algorithm
$\mathtt{genHolderProof}(-)$	Holder-generated proof generation algorithm
$\mathtt{verPresentProof}(-)$	Holder-generated proof verification algorithm
VC	Verifiable credential
VP	Verifiable presentation
cm	Cryptographic mechanism
HVZK	Honest verifier zero-knowledge
NIZKP	Non-interactive zero-knowledge proof
A	List of attributes included in the VC
S	List of salts included in the VC based on hiding commitments
DA	Disclosed attributes included in the VP
DS	Disclosed salts included in the VP
CMT	Commitment included in VC and VP based on hiding commitment
cmtList	List of hash and salt cryptographic mechanism
merTree	Merkle tree cryptographic mechanism
SDSig	Selective disclosure signature
σ	Output of any digital signature algorithm
$\mathcal{H}(-)$	Cryptographic hash function

3.1. Digital signatures

In cryptographic mechanisms based on hiding commitment or selective disclosure signature, the essential cryptographic tool for proving the authenticity of a VC, and the validity of the derived VPs, are the digital signature algorithms used by the Issuer to sign the VC when issuing it to the Holder.

Digital signature schemes are defined by the algorithms $\mathtt{setUp}(\lambda)$ to generate public parameters pp given a security level λ, $\mathtt{keyGen}(\mathtt{pp})$ to generate the public–private key pair $(\mathtt{pk}, \mathtt{sk})$, $\mathtt{genSig}(\mathtt{sk}, m)$ to sign a message m and generate a signature, and $\mathtt{verSig}(\mathtt{pk}, m, \sigma)$ to verify the signature σ.

While the digital signature schemes we use in hiding commitment-based cryptographic mechanisms (Section 4) may be any standardized digital signature algorithm, those used in selective disclosure signature-based cryptographic mechanisms (Section 5) are a special class of signatures designed to support ZKP, and may require more structured inputs, e.g., ordered lists of messages. We use the same notation for brevity, but we stress that signatures for selective disclosure support different features that additionally require the generation of public parameters that are specific to individual attributes in a credential. Depending on the algorithm and the trust model, these elements may be included in the list of public parameters, or be part of the public key. We summarize these distinctions in Section 5.5.

Threat model. According to [26], Section 13.2, given a public key pk generated by a signer S, a *forgery* is a valid signature σ (verifiable using pk) of a message m not previously signed by S, and a signature scheme is secure (or unforgeable) if an adversary is not able to produce a forgery. A signature scheme is said to be *unforgeable under a chosen message attack* if it is secure against an adversary with the power to ask S to provide the signature of many messages of its choice before producing a forgery. All the digital signatures that we describe in this paper are proven unforgeable under a chosen message attack.

To prove a signature secure, or more precisely unforgeable under a chosen message attack, it is necessary to prove that if an attacker with the above capability – querying S and creating a valid forgery – did exist, it could be used as a subroutine of an attacker who can win a different experiment $\mathcal{E}$ that is believed infeasible to win. In this case we will say that "experiment $\mathcal{E}$ is hard to win" is the assumption under which the digital signature is secure under a chosen message attack. As long as the assumption holds, no attacker should be able to forge a signature.

In the use case of our interest, breaking the unforgeability of the signature would give an attacker the ability to create new VCs that could be verified using the Issuer's public key.

3.2. Hiding commitments

Informally, a *commitment scheme* allows a party to commit to a value v by sending a commitment, and then to reveal v by *opening* the commitment later. The commitment scheme must satisfy the *binding* property, which is: a commitment to a value v cannot be opened to a value $v' \neq v$. A *hiding commitment scheme* must satisfy also the *hiding* property: from the commitment it must not be feasible to retrieve the committed value v.

Since our goal is to describe the design of VCs that allow the selective disclosure of attributes, we are interested in hiding commitment schemes that take as input an ordered list of values such that the opening algorithm can be performed on specific positions of the list. The hiding and binding properties must hold on the ordered list of commitments. They are adapted in the following way: it must be infeasible to retrieve the values in the positions of the list that do not get opened, and it must be infeasible to open a position of the list to a different value than the one used to create the commitment, a property often referred to as position binding [27].

The VC created using hiding commitment based cryptographic mechanisms instructs the Issuer to create a hiding commitment to the attributes it wants to include in the VC, then to sign it. By signing the commitment, the Issuer implicitly also signs the attribute used to create it. At a later point in time, when the Holder wants to present the VC, it shows the signature of the commitment to the Verifier, and thanks to the hiding property of the hiding commitment, this does not reveal any information about the attributes used to create it. Therefore, the attribute behind the commitment can be kept hidden if the Holder does not want to disclose it to a Verifier. On the other hand, if the Holder wants to reveal an attribute, it can open the signed commitment and show that the Issuer has certified it. Note that the Holder cannot open a commitment to a different message from the one used to create it, thanks to the position binding property of the commitment scheme.

Hash and salt technique. A widely adopted approach for the creation of hiding commitments is based on cryptographic hash functions. Cryptographic hash functions satisfy very important security properties such as the *preimage resistance* property that informally requires that given a digest y it is infeasible to find an input x whose digest $\mathcal{H}(x) = y$, and the *collision resistance* property which requires that it is infeasible to find x and y such that $\mathcal{H}(x) = \mathcal{H}(y)$.

The commitment scheme based on cryptographic hash functions is defined as follows: the commitment creation algorithm takes as input a value v to be committed to, and outputs $\mathcal{H}(v \parallel s)$, where s is chosen uniformly at random and is referred to as the salt of the commitment, where $v \parallel s$ is the concatenation of the bytes strings v and s.

Threat model. Similarly to digital signature schemes, the security of commitment schemes are defined using experiments that capture the security properties that a hiding commitment scheme must satisfy, namely the hiding property and the binding property.

According to [26], Section 6.6.5, the experiment used to prove that a commitment is hiding is the following: the attacker chooses two messages m_1 and m_2 and sends it to the challenger. The challenger chooses at random one of the two messages, creates a commitment to it and sends it to the attacker. The attacker must decide which of the two messages has been used to create the commitment. For what concerns the binding property, the experiment used to prove that a commitment is binding requires the attacker to give to the challenger a single commitment together with two distinct messages and associated opening material that allow to open the commitment to each of the two messages.

The hiding property of the commitment based on the hash and salt technique is derived from the preimage resistance property of the underlying hash function $\mathcal{H}$ and the binding property of the commitment is derived from the collision resistance property of $\mathcal{H}$.

If an attacker were capable of breaking the hiding property, it would be able to learn information about the attributes that the Holder wants to keep hidden during a verifiable presentation of a VC as the ones described in Section 4. If an attacker could break the binding property it would be able to open a commitment to two different values, therefore it would be able to present, in distinct VP, different values for the same attribute of the same VC as the ones described in Section 4.

3.3. Non-interactive zero-knowledge proofs

Non-interactive zero-knowledge proofs (NIZKP) for a relation $\mathcal{R} \subset W \times Y$ where W is the set of witnesses and Y the set of statements, allow an actor, called prover, to convince another actor, called verifier, that it knows a witness w for a statement y without revealing anything else to the verifier. The protocol is non-interactive, meaning that the prover generates a proof π and the verifier checks that π is valid without requiring additional interactions between prover and verifier.

Signature Proof of Knowledge (SPK). For the sake of brevity, along the paper we will adopt the notation introduced in [28] and we write

$$\pi \in SPK\{(w_1, \dots, w_n) : y = \prod_{i=1}^{n} g_i^{w_i}\}$$

to represent a NIZKP of knowledge of a witness $(w_1, \dots, w_n) \in W$ for the statement $y \in Y$ such that $y = \prod_{i=1}^{n} g_i^{w_i}$. The NIZKP used refers to the relation

$$\mathcal{R} = \{((w_1, \dots, w_n), y) | y = \prod_{i=1}^{n} g_i^{w_i}\} \subset W \times Y$$

and is referred to as NIZKP for linear relations which is a main building block for the cryptographic mechanisms presented in Section 5. In Appendix, Fig. A.5, we provide a description this algorithm.

In Section 5 we use NIZKP in combination with a special class of digital signatures, referred to as selective disclosure signatures. In particular, the Issuers create VCs by signing the attributes using this

kind of digital signature, and issue it by giving the signature and the attributes to the Holder. Later the Holder can prove knowledge of such signature on the set of attributes it wants to disclose to a Verifier in zero-knowledge using the NIZKP associated to each selective disclosure signature. The Verifier will only learn that the Holder knows a signature made by the Issuer over the disclosed attributes. It cannot learn any information about the signature and about the hidden attributes. We will describe four NIZKPs based on the NIZKP for linear relations: the first, in Section 5.1, is based on a variant of the sigma protocol for linear relation adapted to work having as set of statements Y a group of unknown order, whereas in Sections 5.2, 5.3 and 5.4 the set of statement Y will be a group of prime order p.

Threat model. According to [29] (Attack Game 20.3), the threat model considered for NIZKPs is the following. A challenger offers one of two games to an attacker, a "real world" game and a "simulated world" game, without revealing which one is being offered. The attacker must try to distinguish which of the two games it is playing, judging by the challenger's responses. The attacker sends the challenger a pair $(w, y) \in \mathcal{R}$, and asks for a proof about it. If the real world game is being played, the challenger creates a proof π as prescribed by the NIZKP using a real random oracle; if the simulated world game is being played, the challenger (also called simulator) creates a simulated proof without using the knowledge of w, as if it does not know w, by simulating also the random oracle[1] by programming it according to the queries it receives from the attacker as in [30], Lemma 3.5.

The protocol is a NIZKP if every attacker has a negligible advantage in distinguishing whether it is performing the real world experiment or the simulated world experiment. A more detailed discussion on the way the NIZKP are built is reported in Appendix and in [29] (Section 20.3.5).

An attacker who can distinguish the real world from the simulated world might be able to learn some information related to the witness known by the prover. In the case of selective disclosure signatures for VCs as in Section 5, this might imply the ability for a Verifier to gain information about the signature of the VC used by the Holder, or about the hidden attributes.

4. Hiding-commitment mechanisms

Instances of hiding commitment mechanisms can be obtained by using lists of hash-based hiding commitments (`cmtList`, see Section 4.1), or Merkle Trees (`merTree`, see Section 4.2), as suggested in [5].

The Issuer commits to a set of attributes, then digitally signs the commitment. The properties of hiding commitments allow the Issuer of a credential to sign the commitments, then a Holder, who knows the attribute values of a credential, can open only some of the committed values proving to a Verifier the truthfulness of its claims. The security of the schemes we describe below resides on the security of the digital signature used, as discussed in Section 3.1, and on the security of the hiding commitment schemes as discussed in Section 3.2.

Operations in the issuing phase. The Issuer can create a VC with the structure of Table 1 and issues it to the Holder. The VC is composed of the three parts already mentioned:

- the *Issuer protected header* containing the cryptographic mechanism identifier `cm`, – specifying primitives such as the chosen digital signature algorithm and cryptographic hash function – and the Issuer public key $\texttt{pk}_{\texttt{Iss}}$;
- the *Issuer payload* containing a list of attributes $\texttt{A} = (a_1, \ldots, a_m)$ certified by the Issuer who created the credential, together with a list of random salts, one for each attribute $\texttt{S} = (s_1, \ldots, s_m)$;
- the *Issuer Proof* containing the digital signature of the commitment `CMT` to the attributes `A`, constructed according to the chosen cryptographic mechanism and the list of attributes and salts, signed by the Issuer, obtaining $\sigma = \texttt{genSig}(\texttt{sk}_{\texttt{Iss}}, \texttt{CMT})$. These operations are performed executing the function $\texttt{genIssuerProof}(\texttt{sk}_{\texttt{Iss}}, \texttt{A}, \texttt{S})$.

Note that the choice of the digital signature scheme adopted by the Issuer to sign the `CMT` is not restricted to a specific primitive.

The Holder can verify the VC's validity by computing the function `verIssuerProof(VC)`, which consists in verifying that the commitment `CMT` is actually a commitment to the elements in `A` and `S`, and verifying the Issuer's digital signature.

Operations in the presentation phase. The Holder creates a VP to convince the Verifier that the attributes revealed are included in a credential issued by a trusted Issuer.

A VP in this context has the structure described in Table 2. It is composed by:

- a *presentation protected header* containing the name of the cryptographic mechanism `cm` adopted in the creation of the underlying credential and the Issuer public key;
- the *presentation payloads*, containing a subset $\texttt{DA} \subset \texttt{A}$ of attributes $(a_{i_1}, \ldots, a_{i_d})$ that the Holder wants to disclose together with $\texttt{DS} \subset \texttt{S}$, the list of associated salts $(s_{i_1}, \ldots, s_{i_d})$;
- a *presentation proof* generated by the Holder including the commitment `CMT` and its signature σ created by the Issuer associated to $\texttt{pk}_{\texttt{Iss}}$ and the Holder-generated proof obtained computing the function $\texttt{genHolderProof}(\texttt{DA}, \texttt{DS}, \texttt{A}, \texttt{S})$.

The Verifier verifies a VP received from the Holder by computing the function `verPresentProof(VP)`, which consists in (*i*) verifying the signature of the `CMT` created by the Issuer, and (*ii*) verifying the proof that the disclosed attributes in `DA` are a subset of the attributes committed to in `CMT`.

Once the commitment opening algorithm for the pairs (a_i, s_i) in $\texttt{DA} \times \texttt{DS} \subset \texttt{A} \times \texttt{S}$ is defined, the functions $\texttt{genHolderProof}(\texttt{DA}, \texttt{DS}, \texttt{A}, \texttt{S})$ and `verPresentProof(VP)` are well defined.

4.1. Commitment list mechanism

In the `cmtList` mechanism, credentials contain ordered lists of attribute-salt pairs; for each pair, the issuer creates a hiding commitment, then signs the list of commitments.

In $\texttt{genIssuerProof}(\texttt{sk}_{\texttt{Iss}}, \texttt{A}, \texttt{S})$, the Issuer generates a random salt s_i for each attribute a_i and computes the commitment list entries $L_i = \mathcal{H}(a_i \parallel s_i)$. Finally, $\texttt{CMT} = \left[L_i\right]_{i=1}^{\#A}$ is signed by the Issuer to create the Issuer proof.

Since the payload of a Holder-generated VP (Table 2, column 2) contains all the information needed to open the commitments to the disclosed attributes, the Presentation Proof only contains the signed commitment i.e. `genHolderProof` is the null function.

In `verPresentProof(VP)` the Verifier verifies the Issuer signature of `CMT` and compares $\mathcal{H}(a_{i_j} \parallel s_{i_j})$ with L_{i_j}, for each $(a_{i_j}, s_{i_j}) \in \texttt{DA} \times \texttt{DS}$. If the signature is verified and the digests $\mathcal{H}(a_{i_j} \parallel s_{i_j})$ match with L_{i_j}, the VP is accepted.

4.2. Merkle tree mechanism

The `merTree` mechanism uses Merkle trees to create commitments `CMT`.

$\texttt{genIssuerProof}(\texttt{sk}_{\texttt{Iss}}, \texttt{A}, \texttt{S})$: the Issuer generates one random salt s_i for each attribute a_i, then uses their ordered concatenated pairs as leaves of a Merkle tree. The Issuer sets the `CMT` equal to the Merkle tree

[1] This extra power that we give to the simulator is crucial: the protocol must be a proof of knowledge of the witness, i.e., a protocol whose output is proof that can be generated only by someone who knows the witness.

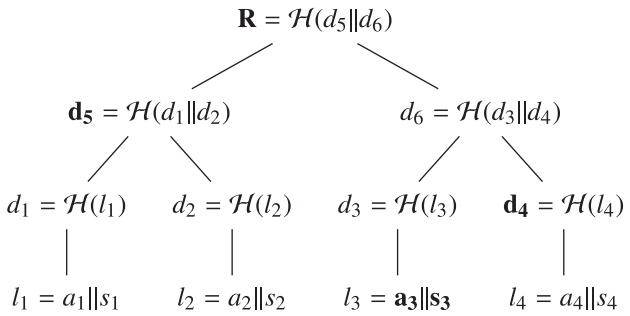

Fig. 1. Merkle tree constructed over 4 leaves. Disclosing $a_3 \parallel s_3$, their inclusion proof in R is $[3, d_4, d_5]$.

root

$$R = \texttt{getRoot}(a_1 \parallel s_1, a_2 \parallel s_2, \ldots, a_m \parallel s_m). \tag{1}$$

An example of Merkle tree is given in Fig. 1.

To create a VP, the Holder includes the presentation payload as in column 2 of Table 2. The presentation proof, together with the signed commitment, also requires the Holder-generated proof, which the Holder obtains by computing the inclusion paths of the attributes that the Holder wants to disclose.

The Verifier verifies the presentation computing `verPresentProof(VP)` verifying the signature of `CMT` and verifying that the inclusion paths in P let the Verifier reconstruct the signed root R, for each $(\mathtt{a}_{i_j}, \mathtt{s}_{i_j}) \in \mathtt{DA} \times \mathtt{DS}$.

For example, the inclusion path of the leaf l_3 in position 3 of the Merkle tree in Fig. 1, given the public root R, is $[3, d_4, d_5]$. In order to verify the inclusion of l_3, the Verifier computes $d_3 = \mathcal{H}(l_3)$, $d_6 = \mathcal{H}(d_3 \parallel d_4)$, and verifies that $\mathcal{H}(d_5 \parallel d_6) = R$.

5. Selective disclosure signature mechanism

Selective disclosure signatures, following the naming in [24], are a class of digital signature algorithms that enable (a) an Issuer to sign multiple attributes with a single signature, (b) a Holder to prove possession of a signature and some undisclosed attributes, generating fresh NIZKP without involving the Issuer - recall Section 3.3, and (c) a Verifier to verify the validity of a disclosed subset of attributes, given only the NIZKP of knowledge of the undisclosed attributes and of an associated signature. The NIZKP created by the Holder, in the literature are also referred to as *signatures of knowledge* [31] or *signatures proof of knowledge* [16].

Examples of selective disclosure signatures are `CL` (Section 5.1), `BBS` (Section 5.2), `BBS+` (Section 5.3) and `PS` (Section 5.4), which are signature algorithms for which an ordered list of messages is input to the signature generation $\texttt{genSig}(\texttt{sk}_{\texttt{Iss}}, (\texttt{a}_1, \ldots, \texttt{a}_\texttt{m})) = \sigma$ and signature verification $\texttt{verSig}(\texttt{pk}_{\texttt{Iss}}, (\texttt{a}_1, \ldots, \texttt{a}_\texttt{m}), \sigma) = \texttt{true/false}$.

The security of the schemes we describe below resides on the security of the selective disclosure digital signature used, as discussed in Section 3.1, and on the security of the NIZKP we will present, as discussed in Section 3.3 and more in detail in Appendix. In the next sections, before the description of each selective disclosure signature, we will mention the assumptions used to prove their security.

Operations in the issuing phase. The VC based on the use of selective disclosure signature algorithms as cryptographic mechanism is composed of three parts (see column 3 of Table 1):

- Issuer protected header, containing the name of the cryptographic mechanism `cm` i.e., the chosen selective disclosure signature scheme, and the Issuer public key $\texttt{pk}_{\texttt{Iss}}$;
- Issuer payloads, containing the list of attributes $\texttt{A} = (a_1, \ldots, a_m)$;
- Issuer proof, containing the *selective disclosure signature* (`SDSig`) of the attributes in `A`, $\sigma = \texttt{genSig}(\texttt{sk}_{\texttt{Iss}}, \texttt{A})$.

Therefore the function that allows the Issuer to create the Issuer proof is just $\texttt{genIssuerProof}(\texttt{sk}_{\texttt{Iss}}, \texttt{A}) = \texttt{genSig}(\texttt{sk}_{\texttt{Iss}}, \texttt{A})$, and the function that allows the Holder to verify it is $\texttt{verIssuerProof}(\texttt{VC}) = \texttt{verSig}(\texttt{pk}_{\texttt{Iss}}, \texttt{A}, \sigma)$.

Operations in the presentation phase. To selectively disclose some attributes of a VC to a Verifier, the Holder creates a VP (see column 3 of Table 2) composed of:

- presentation protected header, containing the name of the cryptographic mechanism `cm` and the Issuer public key;
- presentation payload, containing the list $\texttt{DA} = (a_{i_1}, \ldots, a_{i_d})$ of disclosed attributes;
- presentation proof P, generated by the Holder executing $\texttt{genHolderProof}(\texttt{pk}_{\texttt{Iss}}, \texttt{DA}, \texttt{A}, \sigma)$, a NIZKP of the signature σ, certifying the revealed attributes in `DA` and proving in zero-knowledge the knowledge of the hidden attributes in `A\DA`.

The Verifier verifies the NIZKP P by computing the function `verPresentProof(VP)`.

For `CL`, `BBS`, `BBS+` and `PS` we provide a high level description of $\texttt{genHolderProof}(\texttt{pk}_{\texttt{Iss}}, \texttt{DA}, \texttt{A}, \sigma)$ and `verPresentProof(VP)`, including references to computation details omitted for brevity.

`CMT` *vs.* `SDSig`. The purpose of `CMT` is to bind the attributes into an item that is subsequently signed by the Issuer. The Holder can perform selective disclosure by revealing `CMT`, the attributes to be disclosed, and a presentation proof. On the other hand, `SDSig` simultaneously binds the attributes into an item that is itself a digital signature, certifying the authorship of the VC. To create a presentation, the Holder must not reveal `SDSig`, but rather derive from `SDSig` a randomized proof that assures the Verifier about the claims. A detailed comparison between the cryptographic mechanisms that use `CMT` or `SDSig` is included in Sections 6 and 7.

5.1. CL signature

The CL signature scheme was presented by Camenish and Lysyanskaya and its security relies on the strong RSA assumption [15].

The CL digital signature algorithm is defined as follows [6]:

Key generation algorithm $\texttt{keyGen}()$. Let $n \leftarrow pq$ be an ℓ_n-bit *special RSA modulus*,[2] and choose uniformly at random quadratic residues $R_1, \ldots, R_m, S, Z$.[3]

Output the public key

$$\texttt{pk}_{\texttt{Iss}} = (n, R_1, \ldots, R_m, S, Z) \tag{2}$$

and the secret key

$$\texttt{sk}_{\texttt{Iss}} = (p). \tag{3}$$

Signing algorithm $\texttt{genSig}(\texttt{sk}_{\texttt{Iss}}, \texttt{A})$. On input the messages

$$\texttt{A} = \{a_1, \ldots, a_m\}, a_i \in \{0,1\}^{\ell_a}, \tag{4}$$

and a secret key (3) choose a random prime number $e \in \{0,1\}^{\ell_e}$, $\ell_e > \ell_a + 2$, $e > 2^{\ell_e - 1}$, and a random number $v \in \{0,1\}^{\ell_v}$, where

[2] $n = pq$ is a special RSA modulus if $p = 2p' + 1$ and $q = 2q' + 1$ with p', q' prime numbers.

[3] q is a quadratic residue modulo n if there exists $a \in \mathbb{Z}_n$ such that $q = a^2 \bmod n$. Note that these elements depend on the public key n.

$\ell_v = \ell_n + \ell_a + \ell_\emptyset$ with $\ell_\emptyset$ a security parameter (e.g. $\ell_\emptyset = 80$). Compute

$$A \leftarrow \left(\frac{Z}{R_1^{a_1} \dots R_m^{a_m} S^v} \right)^{\frac{1}{e}} \mod n \tag{5}$$

where $\frac{1}{e}$ is computed modulo $\phi(n) = (p-1)(q-1)$. The resulting output signature is

$$\sigma = (A, e, v). \tag{6}$$

Verification algorithm. $\texttt{verSig}(\texttt{pk}_{\texttt{Iss}}, \texttt{A}, \sigma)$**.** On input a public key (2), a set of messages (4) and a CL signature (6), check that the following holds:

$$Z = A^e R_1^{a_1} \dots R_m^{a_m} S^v \mod n \tag{7}$$

$$a_i \in \{0,1\}^{\ell_a} \tag{8}$$

$$e \in [2^{\ell_e - 1} + 1, 2^{\ell_e} - 1] \tag{9}$$

These functions completely define $\texttt{genIssuerProof}(\texttt{sk}_{\texttt{Iss}}, \texttt{A})$ which corresponds to $\texttt{SDSig} = \texttt{genSig}(\texttt{sk}_{\texttt{Iss}}, \texttt{A}) = \sigma = (A, e, v) \in \mathbb{Z}_n \times \{0,1\}^{\ell_e} \times \{0,1\}^{\ell_v}$ and $\texttt{verIssProof}(\texttt{VC})$.

VP creation. At every presentation, the Holder, who possesses (A, e, v) received from the Issuer, generates a new randomized signature (A', e, v') from a signature by generating a random integer r and computing $v' = v - re \in \mathbb{Z}$ and computing the NIZKP:

$$\begin{cases} A' = AS^r \mod n \\ \pi \in SPK\{(e, v', \{m_i \notin \texttt{DA}\}) : \\ \qquad \frac{Z}{\prod_{i \in \texttt{DA}} R_i^{a_i}} = A'^e S^{v'} \prod_{i \notin \texttt{DA}} R_i^{a_i} \} \end{cases} \tag{10}$$

according to the notation SPK introduced in Section 3.3. The Holder-generated proof presented above is a proof of knowledge of a signature from the Issuer and the attributes signed in it and has the following structure:

$$\begin{aligned} P &= \texttt{genHolderProof}(\texttt{DA}, \texttt{A}, \sigma, \texttt{pk}_{\texttt{Iss}}) \\ &= (A', \pi) = (A', c, \hat{e}, \hat{v}', \hat{a}_{i_1}, \dots, \hat{a}_{i_{(n-d)}}) \end{aligned} \tag{11}$$

with $\texttt{pk}_{\texttt{Iss}}$ from Eq. (2), and σ from Eq. (6); $c \in \{0,1\}^{256}$ is the challenge of the underlying NIZKP[4]; $A' \in \mathbb{Z}_n^*$ is a component of the randomized signature;

$$\hat{e} \in \{0,1\}^{\ell'_e + \ell_{\mathcal{H}} + \ell_\emptyset + 1} \tag{12}$$

$$\hat{v}' \in \{0,1\}^{\ell_v + \ell_{\mathcal{H}} + \ell_\emptyset + 1} \tag{13}$$

$$\hat{a}_{i_1}, \dots, \hat{a}_{i_{(n-d)}} \in \{0,1\}^{\ell_a + \ell_{\mathcal{H}} + \ell_\emptyset + 1} \tag{14}$$

are the response values of the underlying NIZKP for linear relations.

The protocol is described in detail in Section 6.2.4. of [6].

VP verification. The verification algorithm $\texttt{verPresentProof}(\texttt{VP})$ consists in (*i*) verifying the NIZKP for linear relations to prove the Holder knows a valid undisclosed signature, and (*ii*) verifying that the size of the received values $(\hat{e}, \hat{a}_{i_1}, \dots, \hat{a}_{i_{(n-d)}})$ lies in the expected integer interval [6] to ensure that the undisclosed attributes $a_{i_1}, \dots, a_{i_{(n-d)}}$ and parameter e used to build the NIZKP have the expected size.

5.2. BBS signature

BBS signatures are group signatures presented in [32] and later of readapted in [16,33] to obtain a selective disclosure signature BBS+ that we describe in Section 5.3. Recently Tessaro and Zhu [17] showed that the original BBS signature could be used to obtain a selective disclosure signature proving its security under the q-strong Diffie–Hellman assumption [17]. This signature algorithm is the object of a standardization effort from W3C and has led to an RFC draft by IRTF [34] which aims to standardize also the associated NIZKP.

The algorithms defining the BBS signature are:

Set-up. Let $\mathbb{G}_1 = \langle g_1 \rangle, \mathbb{G}_2 = \langle g_2 \rangle$ and $\mathbb{G}_T$ be groups of prime order p, $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a pairing[5] and $(h_1, \dots, h_m) \in \mathbb{G}_1^m$ a random vector. Set the public parameters

$$\texttt{pp} = (p, \mathbb{G}_1, g_1, \mathbb{G}_2, g_2, G_T, \mathbf{e}, h_1, \dots, h_m). \tag{15}$$

Key generation algorithm $\texttt{keyGen}(\texttt{pp})$**.** Take a random $x \in \mathbb{Z}_p^*$, set

$$\texttt{sk}_{\texttt{Iss}} = x \tag{16}$$

and set

$$\texttt{pk}_{\texttt{Iss}} = w = g_2^x. \tag{17}$$

Signing algorithm $\texttt{genSig}(\texttt{sk}_{\texttt{Iss}} = x, \texttt{A})$**.** On input the secret key (16) and the messages

$$\texttt{A} = (a_1, \dots, a_m) \in \mathbb{Z}_p^m, \tag{18}$$

randomly generate $e \in \mathbb{Z}_p$ and compute

$$C = (g_1 \prod_{i=1}^{m} h_i^{a_i}) \tag{19}$$

$$A = C^{\frac{1}{e+x}}. \tag{20}$$

Output the pair

$$\sigma = (A, e) \in \mathbb{G}_1 \times \mathbb{Z}_p. \tag{21}$$

Verification algorithm $\texttt{verSig}(\texttt{pk}_{\texttt{Iss}}, \texttt{A}, \sigma)$**.** On input the public key (17), the messages (18), and a signature (21), set $C = g_1 \prod_{i=1}^m h_i^{a_i}$ and check that

$$\mathbf{e}(A, wg_2^e) = \mathbf{e}(C, g_2).$$

VP creation. The Holder can generate a VP with $\texttt{genHolderProof}(\texttt{DA}, \texttt{A}, \sigma, \texttt{pk}_{\texttt{Iss}})$ whose output is obtained from the construction of a NIZKP of knowledge of the signature and the hidden attributes based on the NIZKP for linear relations. The Holder samples uniformly at random $r \in \mathbb{Z}_p$, computes:

$$C_D = g_1 \prod_{i \in \texttt{DA}} h_i^{a_i}, \tag{22}$$

and computes the NIZKP:

$$\begin{cases} \overline{A} = A^r \\ \overline{B} = C^r \overline{A}^{-e} \\ \pi \in SPK\{(r, e, \{a_i \notin \texttt{DA}\}) : \overline{B} = C_D^r \overline{A}^{-e} \prod_{i \notin \texttt{DA}} h_i^{ra_i} \} \end{cases} \tag{23}$$

The function returns

$$\begin{aligned} P &= \texttt{genHolderProof}(\texttt{DA}, \texttt{A}, \sigma, \texttt{pk}_{\texttt{Iss}}) \\ &= (\overline{A}, \overline{B}, \pi) = (\overline{A}, \overline{B}, T, \hat{r}, \hat{e}, \hat{a}_{i_1}, \dots, \hat{a}_{i_{m-d}}) \end{aligned} \tag{24}$$

where $\overline{A}, \overline{B}, T \in \mathbb{G}_1$, and all other elements lie in $\mathbb{Z}_p$. For a detailed description and the security proofs we refer to [17].

[4] Note that in this case the proof π contains the challenge c instead of the commitment T as described in Fig. A.5. This is an equivalent and more compact format for the NIZKP as we describe in Section 7.4.

[5] A pairing is a map satisfying *bilinearity*, i.e. $\mathbf{e}(g_1^x, g_2^y) = \mathbf{e}(g_1, g_2)^{xy}$, *non-degeneracy*, i.e. for each generator $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$, then $\mathbf{e}(g_1, g_2)$ generates $\mathbb{G}_T$, and *efficiency* which means that the map can be efficiently computed for any input.

VP verification. Having received a VP from a Holder, the Verifier computes the function `verPresentProof(VP)`, which consists in executing the verification steps of the underlying NIZKP for linear relations and verifying that the terms $\mathbf{e}(\overline{A}, w) = \mathbf{e}(\overline{B}, g_2)$.

An alternative VP construction. When creating a VP from multiple VCs, the separate randomization of each attribute in each VC may be a hindrance to proving predicates such as the equality of two hidden attributes.

In [34], the authors propose an alternative construction of the VP,[6] which allows the Holder not to store the variable $C_D = g_1 \prod_{i \in \mathtt{DA}} h_i^{a_i}$, which is used as an element of the representation of $\overline{B}$, when she computes a proof for a VP. Instead, since $\overline{B}$ must be sent to the verifier in any case, and since $\overline{B} = C^r \overline{A}^{-e} = C_D^r \prod_{i \notin DA} g_i^{ra_i} \overline{A}^{-e}$ holds, then the Holder proves knowledge of a representation of C_D as follows:

$$C_D = \overline{B}^{r^{-1}} \prod_{i \notin \mathtt{DA}} h_i^{-a_i} \overline{A}^{er^{-1}}. \tag{25}$$

Therefore the Holder computes:

$$\pi \in SPK\{(r, e, \{a_i \notin \mathtt{DA}\}) : C_D = \overline{B}^{r^{-1}} \prod_{i \notin \mathtt{DA}} h_i^{-a_i} \overline{A}^{er^{-1}}\}.$$

This alternative algorithm and another variant is described in the appendix of a recent update[7] of the paper presented at Eurocrypt 2023 by Tessaro and Zhu [17].

5.3. BBS+ signature

The BBS+ signature was presented by Au et al. [33] as a provably secure extension to BBS group signatures [32] and improved by Camenisch et al. [16]. Its security relies on the q-strong Diffie–Hellman assumption [16]. The digital signature BBS+ is defined by the following algorithms:

Set-up. Let $\mathbb{G}_1 = \langle g_1 \rangle, \mathbb{G}_2 = \langle g_2 \rangle$ and $\mathbb{G}_T$ be groups of prime order p, $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a pairing and $(h_0, \dots, h_m) \in \mathbb{G}_1^{m+1}$ a random vector. Set the public parameters

$$\mathtt{pp} = (p, \mathbb{G}_1, g_1, \mathbb{G}_2, g_2, G_T, \mathbf{e}, h_0, \dots, h_m). \tag{26}$$

Key generation algorithm `keyGen(pp)`. Sample uniformly at random a random $x \in \mathbb{Z}_p^*$, set

$$\mathtt{sk}_{\mathtt{Iss}} = x, \tag{27}$$

then set

$$\mathtt{pk}_{\mathtt{Iss}} = w = g_2^x. \tag{28}$$

Signing algorithm $\mathtt{genSig}(\mathtt{sk}_{\mathtt{Iss}} = x, \mathtt{A})$. On input the secret key (27) and the messages

$$\mathtt{A} = (a_1, \dots, a_m) \in \mathbb{Z}_p^m, \tag{29}$$

randomly generate $e, s \in \mathbb{Z}_p$, compute

$$C = g_1 h_0^s \prod_{i=1}^{m} h_i^{a_i} \tag{30}$$

$$A = C^{\frac{1}{e+x}}. \tag{31}$$

Output the triple

$$\sigma = (A, e, s). \tag{32}$$

Verification algorithm $\mathtt{verSig}(\mathtt{pk}_{\mathtt{Iss}}, \mathtt{A}, \sigma)$. On input the public key (28) messages (29), and a signature (32), check that the following holds:

$$\mathbf{e}(A, wg_2^e) = \mathbf{e}(g_1 h_0^s \prod_{i=1}^{m} h_i^{a_i}, g_2). \tag{33}$$

These algorithms define the functions $\mathtt{genIssuerProof}(\mathtt{sk}_{\mathtt{Iss}}, \mathtt{A})$ which corresponds to $\mathtt{SDSig} = \mathtt{genSig}(\mathtt{sk}_{\mathtt{Iss}}, \mathtt{A}) = \sigma = (A, e, s) \in \mathbb{G}_1 \times \mathbb{Z}_p^2$, and `verIssuerProof(VC)`.

We have presented the algorithms as described by Au et Al. [33]. In the paper from Camenish et Al. [16] the authors include also the elements $h_0, \dots, h_m$ as part of the public key, which they write as $(w = g_2^x, h_0, \dots, h_m) \in \mathbb{G}_2 \times \mathbb{G}_1^m$.

VP creation. The Holder can generate a VP proof with $\mathtt{genHolderProof}(\mathtt{DA}, \mathtt{A}, \sigma, \mathtt{pk}_{\mathtt{Iss}})$, whose output is obtained from the construction of a NIZKP of knowledge of the signature and the hidden attributes based on the NIZKP for linear relations. First, the Holder randomly generates $r_1 \in \mathbb{Z}_p^*$ and $r_2 \in \mathbb{Z}_p$. The Holder then sets

$$r_3 = \frac{1}{r_1} \mod p \tag{34}$$

$$s' = s - r_2 r_3 \mod p \tag{35}$$

and computes the NIZKP:

$$\begin{cases} A' = A^{r_1} \\ \overline{A} = A'^{-e} C^{r_1} (= A'^{x}) \\ d = C^{r_1} h_0^{-r_2} \\ \pi \in SPK\{(e, r_2, r_3, s', \{a_i \notin \mathtt{DA}\}) : \\ \qquad \frac{\overline{A}}{d} = A'^{-e} h_0^{r_2} \wedge \\ \qquad g_1 \prod_{i \in \mathtt{DA}} h_i^{a_i} = d^{r_3} h_0^{-s'} \prod_{i \notin \mathtt{DA}} h_i^{-a_i}\} \end{cases} \tag{36}$$

The proof is then computed as

$$P = \mathtt{genHolderProof}(\mathtt{DA}, \mathtt{A}, \sigma, \mathtt{pk}_{\mathtt{Iss}})$$

$$= (A', \overline{A}, d, \pi) \tag{37}$$

$$= (A', \overline{A}, d, T_1, T_2, \hat{e}, \hat{r}_2, \hat{r}_3, \hat{s}', \hat{a}_{i_1}, \dots, \hat{a}_{i_{m-d}}) \tag{38}$$

where $A', \overline{A}, d, T_1, T_2 \in \mathbb{G}_1$, and all other elements lie in $\mathbb{Z}_p$. For a detailed description we refer to [16].

VP verification. Having received a VP from a Holder, the Verifier computes the function `verPresentProof(VP)`, which consists in executing the verification steps of the underlying NIZKP for linear relations and verifying that the terms $A' \neq 1_{\mathbb{G}_1}$ computing $\mathbf{e}(A', w) = \mathbf{e}(\overline{A}, g_2)$.

Adapting the NIZKP for VC based on BBS to NIZKP for VC based on BBS+. In order to emphasize the differences between BBS and BBS+ signatures on messages $a_1, \dots, a_m$, we describe the BBS+ signature starting from the BBS signature.

- the public parameters pp of BBS+ (Eq. (26)) are the same as the ones of BBS (Eq. (15)) with an extra random element $h_0 \in \mathbb{G}_1$;
- the variable C computed to generate a BBS signatures is $C = g_1 \prod_{i=1}^m h_i^{a_i}$ (Eq. (20)), while for BBS+ signatures (we rename C as C' to distinguish it from the one used in BBS) the signer must generate at random $s \in \mathbb{Z}_p$ and compute $C' = g_1 h_0^s \prod_{i=1}^m h_i^{a_i} = C h_0^s$ (Eq. (30));
- the BBS signature is given by $(A, e) = (C^{\frac{1}{x+e}}, e)$ (Eq. (21)) while the BBS+ signature is given by $(A, e, s) = (C'^{\frac{1}{x+e}}, e, s)$.

Once highlighted these differences between the two, it is clear that a BBS+ signature (A, e, s) over messages $(a_1, \dots, a_m)$ w.r.t. the public parameters

$$\mathtt{pp} = (p, \mathbb{G}_1, g_1, \mathbb{G}_2, g_2, G_T, \mathbf{e}, h_0, \dots, h_m)$$

[6] Private communication with one of the authors of [34].

[7] https://eprint.iacr.org/2023/275 updated on 2023-12-09.

can be univocally turned into a BBS signature (A, e) over the messages $(s, a_1, \dots, a_m)$ w.r.t. exactly the same public parameters pp. Therefore, proving knowledge of a BBS+ signature (A, e, s) and of some hidden attributes $H = \{a_i \notin DA\}$, without revealing it, would be equivalent to prove knowledge of the univocally determined BBS signature (A, e) and of the same attributes H to which we will add s.

This means that the NIZKP used for BBS signatures (Eq. (23)) can be used also to prove knowledge of a BBS+ signature. The idea is the following: turn the BBS+ signature into the uniquely determined BBS signature as described above, then prove knowledge of the derived BBS signature and of the hidden attributes considering that hiding s is mandatory.

5.4. PS signature

The Pointcheval-Sanders (PS) signature is secure under the LRSW assumption [14]. The PS signature is defined by the following algorithms.

Set-up Let $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$, and $\mathbb{G}_T$ be groups of prime order p, and $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a pairing. Set the public parameters

$$\mathtt{pp} = (p, \mathbb{G}_1, g_1, \mathbb{G}_2, g_2, G_T, \mathbf{e}). \tag{39}$$

Key generation algorithm $\mathtt{keyGen(pp)}$**.** Take a random vector

$$\mathtt{sk}_{\mathtt{Iss}} = (x, y_1, \dots, y_m) \in \mathbb{Z}_p^{m+1}, \tag{40}$$

then set

$$\mathtt{pk}_{\mathtt{Iss}} = (X, Y_1, \dots, Y_m) \tag{41}$$

$$= (g_1^x, g_1^{y_1}, \dots, g_1^{y_m}) \in \mathbb{G}_1^{m+1}, \tag{42}$$

Signing algorithm $\mathtt{genSig}(\mathtt{sk}_{\mathtt{Iss}}, \mathtt{A})$**.** On input the secret key (40) and the messages

$$\mathtt{A} = (a_1, \dots, a_m) \in \mathbb{Z}_p^m, \tag{43}$$

randomly generate $h \in \mathbb{G}_2^*$ and compute

$$\sigma = (\sigma_1, \sigma_2) = (h, h^{x + \sum_{j=1}^m y_j a_j}) \in \mathbb{G}_2^2 \tag{44}$$

Verification algorithm $\mathtt{verSig}(\mathtt{pk}_{\mathtt{Iss}}, \mathtt{A}, \sigma)$**.** On input a public key (41), messages (43), and a signature (44), check that both (45) and (46) hold:

$$h \neq 1_{\mathbb{G}_1} \tag{45}$$

$$\mathbf{e}(g_1, \sigma_2) = \mathbf{e}(X \prod_{i=1}^m Y_i^{a_i}, \sigma_1). \tag{46}$$

VP creation. The Holder can generate a VP with $\mathtt{genHolderProof}(\mathtt{DA}, \mathtt{A}, \sigma, \mathtt{pk}_{\mathtt{Iss}})$ whose output is obtained from the construction of a NIZKP of knowledge of the signature and the hidden attributes based on the NIZKP for linear relations applied to a randomized signature. In particular, the Holder computes a signature of the messages $(a_1, \dots, a_m, t)$, where $t \in \mathbb{Z}_p$ is a random message associated to the dummy public key $Y_{m+1} = g_1$, i.e. $(h, h^{x + (\sum_{j=1}^m y_j a_j) + t}) \in \mathbb{G}_2^2$, then randomizes it by picking a random $r \in \mathbb{Z}_p$ and computing:

$$\begin{cases} \sigma' = (\sigma'_1, \sigma'_2) = (h^r, h^{r(x + (\sum_{j=1}^m y_j a_j) + t)}) \\ \pi \in SPK \left\{ (t, \{m_i \notin \mathtt{DA}\}) : \mathbf{e}(g_1, \sigma'_1)^t \prod_{i \notin \mathtt{DA}} \mathbf{e}(Y_i, \sigma'_1)^{m_i} \right. \\ \quad \left. = \mathbf{e}(g_1, \sigma'_2) \left(\mathbf{e}(X, \sigma'_1) \prod_{i \in \mathtt{DA}} \mathbf{e}(Y_i, \sigma'_1)^{m_i} \right)^{-1} \right\}. \end{cases} \tag{47}$$

and returns to the Verifier the tuple

$$P = \mathtt{genHolderProof}(\mathtt{DA}, \mathtt{A}, \sigma, \mathtt{pk}_{\mathtt{Iss}})$$

$$= (\sigma'_1, \sigma'_2, \pi) = (\sigma'_1, \sigma'_2, T, \hat{t}, \hat{a}_{i_1}, \dots, \hat{a}_{i_{m-d}}) \tag{48}$$

where $\sigma'_1, \sigma'_2 \in \mathbb{G}_2$, $T \in \mathbb{G}_T$ and the other elements are in $\mathbb{Z}_p$ [35].

VP verification. Having received a VP from a Holder, the Verifier computes the function $\mathtt{verPresentProof(VP)}$, which consists in executing the verification steps of the underlying NIZKP for linear relation.

As we will show in the next paragraph it is possible to avoid to perform computations in $\mathbb{G}_T$ and avoid to compute so many pairings as one would expect by looking at the SPK described above.

A more practical VP construction. The algorithm presented above for the creation of the Holder-generated proof requires the Holder to perform computations in $\mathbb{G}_T$, the codomain of the pairing $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. However, this can be avoided according to the implementation proposed in Ursa.[8] In fact, it is possible to observe that, by the bilinearity of $\mathbf{e}$, it holds that

$$\mathbf{e}(g_1, \sigma'_1)^t \mathbf{e}(X, \sigma'_1) \prod \mathbf{e}(Y_i, \sigma'_1)^{a_i} = \mathbf{e}(g_1^t X \prod Y_i^{a_i}, \sigma'_1),$$

so the Holder can send to the Verifier

$$\begin{cases} \sigma' = (\sigma'_1, \sigma'_2) = (h^r, h^{r(x + (\sum_{j=1}^m y_j a_j) + t)}) \\ J = g_1^t \prod_{i \notin \mathtt{DA}} Y_i^{a_i} \\ \pi = SPK \left\{ (t, \{a_i \notin \mathtt{DA}\}) : J = g_1^t \prod_{i \notin \mathtt{DA}} Y_i^{a_i} \right\} \\ \quad = (T, \hat{t}, \hat{a}_{i_1}, \dots, \hat{a}_{i_{m-d}}) \in \mathbb{G}_1 \times \mathbb{Z}_p^{m-d+1} \end{cases} \tag{49}$$

where $r, t \in \mathbb{Z}_p$ are the random elements used to randomize the signature, d is the number of disclosed attributes and m the number of attributes in the VC. In this way, the Holder-generated proof has the following form:

$$P = \mathtt{genHolderProof}(\mathtt{DA}, \mathtt{A}, \sigma, \mathtt{pk}_{\mathtt{Iss}})$$

$$= (\sigma'_1, \sigma'_2, J, \pi) \tag{50}$$

$$= (\sigma'_1, \sigma'_2, J, T, \hat{t}, \hat{a}_{i_1}, \dots, \hat{a}_{i_{m-d}}) \in \mathbb{G}_2^2 \times \mathbb{G}_1^2 \times \mathbb{Z}_p^{m-d+1} \tag{51}$$

To verify π, the Verifier can compute $J' = J X \prod_{i \in \mathtt{DA}} Y_i^{a_i}$ and check that $\mathbf{e}(\sigma'_1, J') \mathbf{e}(\sigma'^{-1}_2, g_1) = 1_{\mathbb{G}_T}$.

In this way all the computations are performed in $\mathbb{G}_1$ which is the group in which computations are more efficient among the ones involved in the pairing definition. Also the Holder does not have to compute any pairing and the number of pairing computations performed by the Verifier is reduced to two.

5.5. Efficiency and trust on issuer set-up domain parameters

For BBS and BBS+ signatures, the public parameters output by $\mathtt{setUp}(\lambda)$ that give structure to a VC and are used to generate and verify VCs and VPs may be generated from a seed in a manner that is not confidential. It is possible to reduce the size of data required for verification by requiring the Verifier to reconstruct the public parameters from the seed. This is not true for CL and PS digital signatures, where parameters must be generated by the Issuer in the key generation algorithm and are part of the Issuer's public key.

In addition, BBS and BBS+ public parameters may be provided by a trusted third party. This may enable the re-use of the same set of public parameters by multiple Issuers, e.g., for the same kind of VC.

If the dimension of the public key is not a concern or it is preferable to require each Issuer to perform its own setup, the public parameters of Eqs. (15) and (26) can be included in the public key; indeed, this is how they appear in [8,16,36].

- CL: the quadratic residues $R_1, \dots, R_m, S, Z \in \mathbb{Z}_n$ are needed to generate the A component of the signature according to Eq. (5). These elements are quadratic residues modulo $n = pq$, therefore must be computed according to the secret key (Eq. (3)) and must necessarily be part of the Issuer public key (Eq. (2)).

[8] https://docs.rs/ursa/.

- BBS (BBS+): $h_1, \ldots, h_m$ ($h_0, \ldots, h_m$) are needed to generate the A component of the signature according to Eq. (20) (Eq. (30)). These are random elements of the group $\mathbb{G}_1$ and do not depend on the Issuer secret key in Eq. (16) (Eq. (27)). As proposed in [17,34], these parameters may be generated either by the Issuer or by a trusted third party by using a hash-to-curve function that maps a seed to a set of random elements in $\mathbb{G}_1$ of the required cardinality. For security reasons it must be infeasible to compute the discrete logarithm of h_i, therefore these random elements cannot be generated by picking random scalars $z_1, \ldots, z_m$ and computing $h_i = g_1^{z_i}$.
- PS: $(X, Y_1, \ldots, Y_m)$ are needed to give structure to the element of $\mathbb{G}_1$ in the right-hand side of Eq. (46), whose validity is part of the signature verification process. In contrast with the approach proposed by BBS and BBS+, the signer must know the discrete logarithm $x, y_1, \ldots, y_m$ of $X, Y_1, \ldots, Y_m$ w.r.t. g_1 to compute their analogue w.r.t. any basis $h \in \mathbb{G}_2$ during the signing process. In fact, the PS signature (Eq. (44)) can be rewritten as $(h, h^x \prod_{i=1}^{m} (h^{y_i})^{a_i})$. For this reason $(X, Y_1, \ldots, Y_m)$ are part of the PS public key (Eq. (41)) and their discrete logarithms w.r.t. g_1 are the secret key (Eq. (40)).

6. Solution design analysis

To assess the maturity of options, we consider their standardization (Section 6.1.1), cryptographic agility (Section 6.1.2) and quantum safety (Section 6.1.3).

6.1. Standard maturity

Standardization is important for cryptographic protocols to ensure expert vetting of correctness, security, and other properties claimed, as well as to promote interoperability as encouraged e.g., by the proposed Interoperable Europe Act [37]. Cryptographic agility [38] "is achieved when a protocol can easily migrate from one algorithm suite to another more desirable one, over time" [39]. The need to transition between cryptographic algorithms and key lengths has been steadily gaining importance, e.g., replacing older versions of the Secure Hash Algorithm, and preparing for quantum computing [40].

We observe how each mechanism supports privacy and offline features with regards to presentation unlinkability (Section 6.2.1), and briefly discuss the advantages of predicate proofs (Section 6.2.2). We compare the computation cost of each function described in Section 2, and the size of presentation elements of each mechanism (Section 7); we also note some trade-offs made by implementations to balance performance between these measures (Section 7.4). Finally, we describe how it is possible to perform threshold issuance of the VCs based on the cryptographic mechanisms we have described (Section 6.2.3). Our assessment is summarized in Section 7.5.

6.1.1. Standardization

`cmtList` is the only mechanism featured in official standards: it is enabled by design in ISO 18013-5 [25], and it is the basis for the IETF draft SD-JWT [41]. Both are considered mandatory for the European digital identity wallet [4] developed in the context of the revised eIDAS regulation [1].

`merTree` has been proposed in [5] as a possible mechanism for JSON Web Proof (JWP) [23] - a proposed container format for VCs and VPs that aims to be agnostic to the proof mechanism, currently an IETF draft on the Standards Track. `merTree` also appears in the experimental Certificate Transparency 2.0 proposal [42].

The BBS (previously BBS+) specification [34] is an IRTF draft. PS and CL signatures are not specified independently, but CL appear as part of the Identity Mixer [6] and Hyperledger Ursa [7] anonymous credentials protocols.

ETSI Technical Report 119 476 [43] gives recommendations on issuing, storage, and presentation of attestations under eIDAS2 in the form of ISO mDL and/or SD-JWT, with a view towards selective disclosure and unlinkability.

6.1.2. Cryptographic agility

`cmtList` and `merTree` offer the greatest agility: any cryptographic hash function can be used to construct them, and any digital signature can be chosen to sign the hash list or tree root.

BBS, BBS+, and PS signatures can in theory be based on any pairing-friendly curve, of which several have been identified [44] up to 256-bit security, and any correspondingly secure cryptographic hash function. Cipher suites have been drafted [34].

The idemix specification [6] for anonymous credentials with CL signatures contains a default value for 14 parameters and 7 "constraints which parameter choices must satisfy to ensure security and soundness" (Tables 2 and 3 therein), and it is left to the reader to adjust these as required. The default RSA modulus is 2048 bits, which corresponds to only 112 bits of security.[9] Other than increasing the prime factor length, it is non-trivial to establish how parameters should change to increase the security level of the scheme as a whole.

The Ursa library also defaults to 2048-bit modulus; the Ursa specification [7] lists individual parameter values scattered throughout, including 1536-bit RSA factors, but it is left to the reader to gather the information, to modify the source code, and to assume that all other parameters have been set to meet the same level of security.

6.1.3. Quantum safety

The security of cryptographic algorithms is based on the assumption that a given problem is hard to solve — for example, the factorization of a number that is the product of two prime numbers, or the discrete logarithm problems; as long as that problem is difficult to solve, the cryptographic algorithm has a solid basis on which to claim a certain level of security.

The cryptographic techniques discussed in this paper to create VCs and to enable selective disclosure of attributes are digital signatures, commitment schemes, and NIZKPs. Sections 3.1, 3.2, and 3.3 outline the security properties that must be met in the respective threat model paragraphs. These security features hold as long as the assumptions on which they are based are valid.

The introduction of quantum computers may rewrite the list of assumptions that can be regarded as reliable, because many conventional assumptions may fail to hold against adversaries with sufficiently powerful quantum computers, and the cryptographic techniques relying on these assumptions will become insecure.

As a result, one must be aware of the issues that may develop when quantum computers will be powerful enough to solve the compromised cryptographic problems. The consequences that this would have on VC protocols include:

- the use of an insecure digital signature scheme would allow an adversary to create VCs without the Issuer's involvement;
- the use of insecure hiding commitment schemes or NIZKPs would cause the ability of a Verifier to learn the value of the attributes that the Holder wants to keep hidden;

Some of the solutions described in this paper are considered quantum resistant.

As noted in [43], cryptographic mechanisms based on hiding commitment can be instantiated using one of the post-quantum digital signature algorithms selected for standardization by NIST: CRYSTALS-Dilithium [45], FALCON [46], or SPHINCS$^+$ [47]. The use of post-quantum signatures makes these cryptographic mechanisms quantum resistant as well, as long as the hiding commitment scheme also is quantum resistant — in particular, for those in Section 4, as long as the cryptographic hash function used to create the list of commitments or the Merkle tree is quantum resistant.

It should be noted that the three above algorithms were selected for standardization in 2022, but at the time of writing the standardization

[9] See www.keylength.com and references therein.

process is still in the draft stage, and some changes have been proposed in the current draft FIPS 204 and 205 [48,49].

For what concerns the quantum resistance of selective disclosure signatures, the algorithms described in Section 5 rely on assumptions that do not hold in a post-quantum setting. Lattice-based cryptography schemes have been proposed very recently [50–53], but there are no complete libraries available yet to make a full comparison with other schemes described here.

6.2. Supported features

We evaluate for each cryptographic mechanism how they support features that are relevant to the design of practical privacy preserving VCs, namely the ability to create unlinkable VPs (Section 6.2.1), the ability to include predicate proofs in the VPs (Section 6.2.2) and the support for threshold credential Issuance that allow multiple Issuers to issue a single VC to a Holder (Section 6.2.3).

6.2.1. Presentation unlinkability

Unlinkability can be defined as ensuring that "no correlatable data are used in a digitally-signed payload" [54]. Sources of correlation include the signature itself and long-term identifiers, such as the credential subject, a credential identifier, revocation status information etc. Guaranteeing this property goes beyond selective disclosure only; here we focus on signature-based correlation.

Hiding commitment. Since the Presentation Proof of a VP contains the issuer-signed commitment included in the associated VC, this identifier links each VP uniquely to one VC, and therefore to its Holder. This means that the Holder should use always different VCs to generate new VPs, therefore in the issuing phase the Issuer must provide the Holder with several distinct versions of the same VC where a distinct version of VC is built including the same set of attributes hidden using different salts.

When all the distinct versions of the same VC have been used, the Issuer must produce and send new ones to the Holder. There must therefore be an available channel between the Issuer and the Holder device storing the VCs that guarantees ready access to brand new VCs that can be used to create unlinkable presentations.

Selective disclosure signatures. As summarized in Table 2, the presentation proof contains only the Holder-generated proof, which is a randomized element.

Given a VC, the Holder can create a new presentation proof each time that is indistinguishable from random, and therefore cannot be correlated to other VPs. The Holder can use the same VC multiple times; therefore, interaction with the Issuer is required only when requesting a new credential or renewing an expired one.

6.2.2. Predicate proofs

In some use cases, there may be an interest in asking a question ("predicate") about an attribute, without disclosing the attribute itself. For instance, a Verifier may need to know whether an mDL subject's age is over some threshold `NN`, or in some range, without needing to know their full date of birth. This feature would enhance privacy and follow the data minimization principle. We discuss how it is possible to create predicate proofs for hiding commitment based cryptographic mechanisms and for selective disclosure signature cryptographic mechanisms.

Hiding commitments. The `cmtList` mechanism used in mDL allows the Issuer to create range proofs only by treating them as individual attributes; for instance, in the AAMVA mDL implementation guidelines [55] Issuers must identify every likely threshold value in their jurisdiction and encode a separate attribute `age_ over_NN=True` or `False` for each `NN`.

The disadvantages of implementing this feature with this mechanism are: (a) an increased size of every VC and VP, (b) requiring the Issuer to keep track of when each threshold is crossed to issue a new VC, (c) interoperability issues - a Verifier may not find all the same thresholds represented every separate jurisdiction for the same VC type (e.g., age above 18, 21, 65, etc.), (d) mistakes are easy to make and hard to spot, e.g., in a long list of individual and unrelated entries it would be possible to enter e.g., `age_over_18=False` and `age_over _21=True`. All hiding commitment based `cm` including `merTree` suffer the same disadvantages.

It is possible to create range proofs using hash functions using a protocol called HashWire [56]. HashWire is an optimization of the technique introduced by Rivest and Shamir in PayWord [57]. To create a commitment to an integer k, the Issuer generates a random string r and computes the commitment $c = \mathcal{H}^k(r)$, namely k repeated iterations of the hash function $\mathcal{H}$. The Issuer reveals the random string r and the integer k to the Holder, who can prove to a Verifier that $k \geq t$, for a given threshold t, by sending the proof $\pi = \mathcal{H}^{k-t}(r)$. The Verifier considers the proof π valid if $\mathcal{H}^t(\pi) = c$. This method is both more compact and less error-prone than a list of unrelated statements.

Selective disclosure signatures. By contrast, selective disclosure signatures enable the Holder to build NIZKP of predicates about the attributes included in the VC without prior involvement of the Issuer. For example, *range proofs* and *set membership proof*s [58] allow the Prover to prove that an attribute a lies within a range $v < a < u$, or in a given set of values $\mathcal{V}$, i.e. $a \in \mathcal{V}$, respectively. Examples of predicate proofs for the `CL` mechanism can be found in Section 6 of [6].

6.2.3. Support for threshold credential issuance

VC ecosystems are initially designed for individual Issuers issuing VCs to Holders; however, this requires all trust to be placed on individual Issuers, which constitute a single point of failure. This problem can be mitigated by having the secret key shared among multiple Issuers, and by designing threshold digital signatures so that they may agree in order to create VCs.

An (n, t)-threshold signature scheme allows a group of n signers to create a digital signature only if t members of the group agree to sign the message.

Threshold signature schemes are often designed generalizing standard digital signature schemes. In fact, a desirable property of some threshold signature schemes is that the resulting signature has exactly the same structure of the signature they generalize, so that the verification algorithm remains unchanged.

For the threshold issuance of hiding commitment based credentials, any threshold digital signature scheme may be used since the signed commitment is always revealed. Examples of threshold signature schemes are the threshold version of EdDSA [59], ECDSA [60], and Schnorr signature [61].

Threshold versions have been proposed for the PS signature [62] and for the BBS+ signature [63], which also applies to the BBS signature with some simple modifications. Since the signatures produced by the Issuers have the same structure as the one they generalize, the protocols for the creation and verification of VP also remain unchanged. To the best of our knowledge, no threshold version of CL has been proposed.

7. Experimental evaluation

Our use case of interest is a proximity flow for EUDI Wallets, in which the Holder and Verifier are physically close and the attestation exchange and disclosure occurs using proximity protocols - e.g., NFC, Bluetooth, QR-Codes. The Holder, Verifier, or both may also be offline. A concrete example involves checking a mobile driving license (mDL), as considered in Section 1. In this scenario, the Holder device may be resource-constrained in both computational capability and presentation exchange bandwidth; we therefore measure the cost of computation, particularly `genHolderProof`, in Section 7.2 and the size of VP elements for each `cm` in Section 7.3. Based on ISO/IEC 18013-5 [25], in which an mDL consists of 11 mandatory and 22 optional attributes, we use credentials with $n_a \leq 33$ total attributes.

7.1. Experimental set-up

Security level. To ensure a fair comparison, we aim for an equivalent level of security of 128 bits in all tested mechanisms — see SP 800-57 [64], Table 2. Enforcing this common security level is non-trivial, as mentioned in Section 6.1.2. This level is achieved by BBS, BBS+, and PS over BLS12-381, CL with 3072-bit RSA modulus, EdDSA over ed25519, and the post-quantum signature parameter sets Falcon-512, Dilithium2, and SPHINCS$^+$-SHA2-128f. Several SPHINCS$^+$ sets meet the same security level, but the chosen one is optimized for computation cost, to the detriment of size.

Libraries. We use the Hyperledger Ursa[10] library for PS and CL signature performance. As noted in Section 6.1.2, we had to modify the CL implementation to achieve the required security level. For BBS and BBS+, we test docknetwork[11] since they implement both, so differences in measurements can be attributed with greater confidence to the algorithm rather than implementation differences. They are also the recently most used BBS crate[12] after Ursa at the time of writing, have an implementation of the threshold versions presented in [63], and are an active project.

For all proofs in `merTree` mechanisms we use SHA-256 and rs_merkle.[13] For `merTree` digital signatures we test EdDSA over ed25519 using the popular rust crate ed25519-dalek,[14] and three PQ signature standardization candidates implemented in Open Quantum Safe (OQS).[15]

Processors. In order to test performance on both desktop PCs and constrained devices with ARM CPUs more closely resembling mobile phones – our use case for Holder devices – experiments are run on AMD Ryzen 7 5800X, raspberry pi 3B+ 1 GB RAM, and pi 4B 4 GB RAM.

7.2. Computational cost

We measure the computational cost of key generation, and signature and presentation proof generation and verification — see Fig. 2. The presentation phase is of particular interest since it is expected to occur frequently on constrained devices; we show results in Table 4. There is approximately an order of magnitude difference in performance between a modern desktop CPU (Ryzen 7 5800X) and an ARM raspberry pi 4B, and another between the pi 4B and pi 3B+. `merTree` results are very close for each algorithm except SPHINCS$^+$, which is reported separately; similarly BBS and BBS+ are very close, so only the former is reported.

[10] https://docs.rs/ursa/.
[11] https://github.com/docknetwork/crypto.
[12] https://crates.io/search?q=bbs%20signature&sort=recent-downloads.
[13] https://docs.rs/rs_merkle/.
[14] https://docs.rs/ed25519-dalek/.
[15] https://github.com/open-quantum-safe/liboqs.

Table 4
Presentation Proof generation and verification times by CPU with $n_A = 33$. Times in ms are median over $n_D \leq n_A$. Falcon and Dilithium results are not significantly different from EdDSA; BBS+results are not significantly different from BBS.

`cm: genHolderProof`	5800X	pi 4B	pi 3B+
`merTree` – $EdDSA$	0.0007	0.0044	0.0143
`merTree` – $SPHINCS^+$	0.0007	0.0044	0.0174
BBS	1.6508	11.2507	92.9144
PS	9.3314	34.6971	303.4390
CL	55.270	394.9524	1475.9164
`cm: verPresentProof`	5800X	pi 4B	pi 3B+
`merTree` – $EdDSA$	0.040	0.2911	1.0715
`merTree` – $SPHINCS^+$	0.4176	7.0385	19.3736
BBS	2.1360	15.7105	117.4054
PS	3.5201	19.4249	210.9906
CL	64.969	456.0057	1687.3662

The computational cost of hashing and of generating Merkle inclusion paths is negligible compared with generating and verifying the digital signature in `cmtList` and `merTree`; we therefore only report the results of `merTree` tests.

For SDSig over elliptic curves – BBS, BBS+, and PS – we also provide a comparison of the number of operations in Table 5. Scalar multiplication (M) is nQ for a curve point Q and a scalar n, and multi-scalar multiplication of n points, $MSM(n)$, is an operation designed to be more efficient than the sum of $(n-1)$ separate scalar multiplications.

The relation between the costs of the operations over the curve BLS12-381 are the following: $P > MSM > M \gg A \gg$ field operations, which we do not include. Roughly speaking, $P \approx 2\ MSM(30)$, $MSM(30) \approx 3\ M$, $M \approx 200\ A$, $A \approx 100$ field operations, and operations over $\mathbb{G}_2$ cost approximately 2 to 3 times as operations over $\mathbb{G}_1$. These ratios consider multiplications M and multi-scalar multiplications by random scalars.

7.3. Presentation size

Presentation proof. We compare VP size contributions for each `cm`, with trends summarized in Fig. 3.

Attribute size is arbitrary, so `DA` is not included. For commitment based mechanisms, one disclosed salt `DS` per disclosed attribute must be included; therefore, VP size tends to grow with n_D for `cmtList` and `merTree`, while it decreases for `CL`, `PS`, `BBS` and `BBS+` due to one zero-knowledge proof per undisclosed attribute.

Presentation proof size, for the cases in which multiple constructions are available, namely BBS and PS, is calculated according to the libraries we have tested.

The presentation proof size for each cryptographic mechanism is computed as follows:

- `cmtList`: one digest per attribute in the credential, a signature of the list of digests, one disclosed salt per disclosed attribute:

$$|\mathtt{CMT}| + |\sigma| + |\mathtt{DS}| = dn_A + |\sigma| + sn_D \tag{52}$$

- `merTree`: one tree root (of digest size), a signature of the tree root, one disclosed salt per disclosed attribute, an inclusion proof for disclosed attributes. The size of an inclusion proof for a single attribute is equal to the tree height $\lceil \log_2(n_A) \rceil$ times the digest size; a simple implementation may return a separate proof per disclosed attribute, an optimized implementation will be more compact. An upper bound is therefore:

$$\begin{aligned} &|\mathtt{CMT}| + |\sigma| + |\mathtt{DS}| + |P| \\ &\qquad = d + |\sigma| + dn_D + \lceil \log_2(n_A) \rceil dn_D \end{aligned} \tag{53}$$

- `CL`: in order to make a fair comparison between the algorithms, we consider a modulus of $|n| = 3072$ bits to have a security level of 128 bits. Therefore, a NIZKP of knowledge of a signature and of the undisclosed attributes (Eq. (11)) is given by:

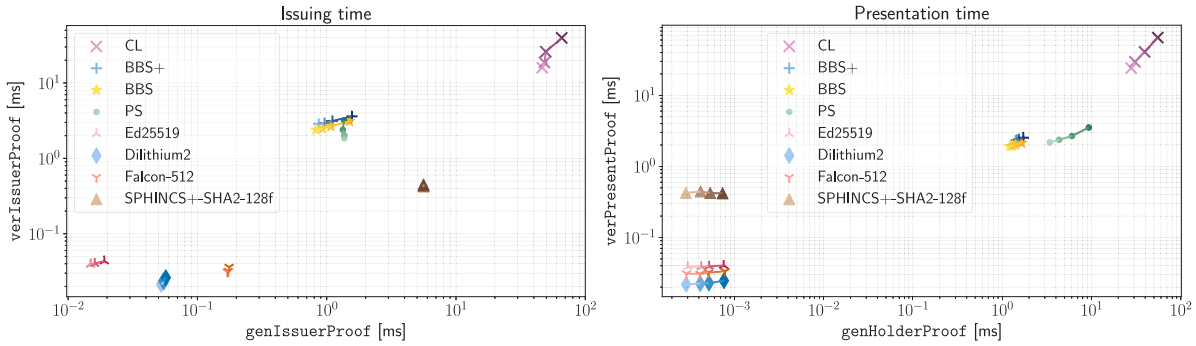

Fig. 2. Scatter plot of issuing and presentation performance test results on Ryzen 7 5800X for all algorithms. Lower values are better (shorter run time), to the bottom left. Points are median values over all possible disclosed number of attributes in the range $n_D \in \{1, n_A\}$ with $n_A = [4, 8, 16, 33]$. Darker colors correspond to higher n_A. `merTree` algorithms are faster in both generation (x axis) and verification (y axis) of signatures and presentations. Quantum-Safe algorithms are very competitive with EdDSA, except SPHINCS$^+$.

Table 5
Number of multiplication (M), multi-scalar multiplication (MSM), addition (A), pairing (P), and random sampling (R) operations over elliptic curves for BBS, BBS+, and PS, with number of disclosed (n_D), undisclosed (n_U), and total number of attributes (n_A). All operations are in the group $\mathbb{G}_1$ unless otherwise subscripted with $\mathbb{G}_2$.

SDSig	genSig	verSig	genHolderProof	verPresentProof
BBS	$MSM(n_A) + M + A$	$MSM(n_A) + A + A_{\mathbb{G}_2} + M_{\mathbb{G}_2} + 2P$	$MSM(n_A) + 3M + 2A + MSM(n_D) + MSM(2 + n_U)$	$MSM(n_D) + MSM(3 + n_U) + 2A + 2P$
BBS+	$MSM(1 + n_A) + M + A$	$MSM(1 + n_A) + A + A_{\mathbb{G}_2} + M_{\mathbb{G}_2} + 2P$	$MSM(1 + n_A) + 3M + 2A + MSM(2 + n_U) + MSM(2)$	$MSM(n_D) + MSM(3 + n_U) + MSM(3) + 3A + 2P$
PS	$M_{\mathbb{G}_2} + R_{\mathbb{G}_2}$	$MSM(n_A) + A + 2P$	$3M_{\mathbb{G}_2} + A_{\mathbb{G}_2} + MSM(1 + n_U)$	$MSM(2 + n_U) + A + MSM(n_D) + 2P$

- a digest $c \in \{0,1\}^{256}$ (32 bytes);
- an element $A' \in \mathbb{Z}_n$ (384 bytes), an element $\hat{e} \in \{0,1\}^{457}$ (58 bytes), and $\hat{v}' \in \{0,1\}^{3744}$ (468 bytes)
- an element $\hat{a}_i \in \{0,1\}^{593}$ (75 bytes) for each undisclosed attribute.

Therefore, the presentation proof size is, in bytes:

$$|c| + |A'| + |\hat{e}| + |\hat{v}'| + |\hat{a}_i|(n_A - n_D)$$
$$= 32 + 384 + 58 + 468 + 75(n_A - n_D). \quad (54)$$

- `BBS` : a NIZKP of knowledge of a signature and of the undisclosed values (Eq. (24)) is given by:
  - three elements $\overline{A}, \overline{B}, T \in \mathbb{G}_1$;
  - two elements $\hat{r}, \hat{e} \in \mathbb{Z}_p$;
  - one $\hat{a}_i \in \mathbb{Z}_p$ for each undisclosed attribute.

  `BBS` can be implemented using the pairing-friendly elliptic curve BLS12-381, with the prime order of the subgroup of $\mathbb{G}_1$ being $p \in \{0,1\}^{256}$. Therefore, the elements in $\mathbb{G}_1$ - i.e., $\overline{A}, \overline{B}$ and T - can be represented as 48-byte strings and the integer elements as 32-byte strings. Therefore, the presentation proof size is, in bytes:

$$\left|\overline{A}\right| + \left|\overline{B}\right| + |T| + |\hat{r}| + |\hat{e}'| + |\hat{a}_i|(n_A - n_D)$$
$$= 3 \cdot 48 + 32(2 + n_A - n_D). \quad (55)$$

- `BBS+`: a NIZKP of knowledge of a signature and of the undisclosed values (Eq. (38)) is given by:
  - five elements $A', \overline{A}, d, T_1, T_2 \in \mathbb{G}_1$;
  - four elements $\hat{e}, \hat{r}_2, \hat{r}_3, \hat{s}' \in \mathbb{Z}_p$;
  - one $\hat{a}_i \in \mathbb{Z}_p$ for each undisclosed attribute.

  As with `BBS`, `BBS+` can be implemented using the pairing-friendly elliptic curve BLS12-381, with the prime order of the subgroup of $\mathbb{G}_1$ being $p \in \{0,1\}^{256}$. Therefore, the elements in $\mathbb{G}_1$ - i.e., $A', \overline{A}, d$ - can be represented as 48-byte strings and the integer elements as 32-byte strings. Therefore, the presentation proof size is, in bytes:

$$|A'| + \left|\overline{A}\right| + |d| + |T_1| + |T_2| + |\hat{e}| + |\hat{r}_2| + |\hat{r}_3| +$$
$$|\hat{s}'| + |\hat{a}_i|(n_A - n_D) = 5 \cdot 48 + 32(4 + n_A - n_D). \quad (56)$$

- `PS`: a NIZKP of knowledge of a signature and of the undisclosed values (Eq. (51)) is given by:
  - two elements $\sigma_1', \sigma_2' \in \mathbb{G}_2$;
  - two elements $J, T \in \mathbb{G}_1$;
  - an element $\hat{t} \in \mathbb{Z}_p$;
  - one $\hat{a}_i \in \mathbb{Z}_p$ for each undisclosed attribute.

  Also PS can be implemented using the pairing-friendly elliptic curve BLS12-381 with the prime order of the subgroup of $\mathbb{G}_1$ being $p \in \{0,1\}^{256}$. Therefore, the elements in $\mathbb{G}_2$ - i.e., σ_1', σ_2' - can be represented as 96-byte strings, the elements in $\mathbb{G}_1$ - i.e., J, T - can be represented as 48-byte strings and the integer elements as 32-byte strings. Therefore, the presentation proof size is, in bytes:

$$\left|\sigma_1'\right| + \left|\sigma_2'\right| + |J| + |T| + \left|\hat{t}\right| + |\hat{a}_i|(n_A - n_D)$$
$$= 2 \cdot 96 + 2 \cdot 48 + 32(1 + n_A - n_D). \quad (57)$$

Issuer public keys ($\texttt{pk}_{\texttt{Iss}}$). A VP header may contain either $\texttt{pk}_{\texttt{Iss}}$ or a reference to it. For instance, a JWS [65] header may contain $\texttt{pk}_{\texttt{Iss}}$ as JWK or X.509 certificate, or a url as JKU, or a certificate thumbprint, etc. Note that in standardized digital signatures, public parameters are assumed to be known and common to all issuers and verifiers, typically written in source code of implementing libraries, and need not be fetched repeatedly. For instance, the Digital Signature Standard [66] (DSS) specifies methods for signature generation and verification, while specifications for the generation of domain parameters e.g., for ECDSA and EdDSA are included in SP 800-186 [67]. We do not describe these algorithms in detail [66,67].

$\texttt{pk}_{\texttt{Iss}}$ size may be calculated as follows.

- `merTree`, `cmtList`: the public key of an EdDSA digital signature is a 32-byte point of the curve ed25519. A SPHINCS$^+$-128 [47]

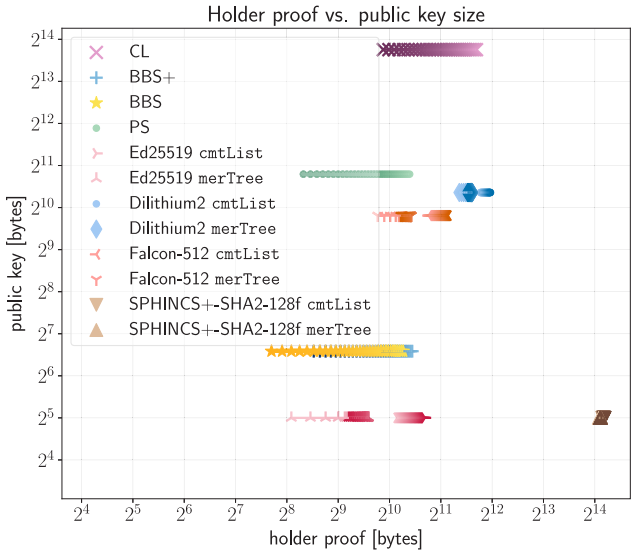

Fig. 3. VP proof size - Eqs. (52) to (57) - vs. public key size - Eqs. (2), (17), (28), (41) for SDSig. Both are required by the Verifier, but the public key may be cached over several presentations, and the Holder-Verifier channel is more likely to have bandwidth constraints than the Issuer-Verifier channel in a digital wallet scenario, so a smaller proof size is more significant than a smaller public key size. Lighter hues are more disclosed attributes (higher n_D). SDSig have smaller holder proofs for higher n_D as fewer ZKP need to be generated for undisclosed attributes; `cmtList` mechanisms follow the opposite trend, as more salts need to be disclosed. Common values used for comparison, in bytes: salt size $s = 16$; digest size $d = 32$; number of attributes $n_A = 33$.

public key is 32 bytes: two 128-bit numbers, a seed and a tree root. A Dilithium2 [45] public key is 1312 bytes: a 32-byte seed and a vector of polynomials. A Falcon-512 [46] public key is 897 bytes: 512 14-bit integers (polynomial coefficients), and 1 header byte.

- `CL`: A CL public key (Eq. (2)) is $(n, R_1, \dots, R_m, S, Z) \in \mathbb{Z}_n^{m+3}$ of size $384(m+3)$ bytes.
- `BBS`: A BBS public key (Eq. (17)) is $w = g_2^x \in \mathbb{G}_2$ of size 96 bytes, where $\mathbb{G}_2$ is obtained using curve BLS12-381.
- `BBS+`: A BBS+ public key (Eq. (28)) is $w = g_2^x$ of size 96 bytes, where $\mathbb{G}_2$ is obtained using curve BLS12-381.
- `PS`: A PS public key (Eq. (41)) is $(X, Y_1, \dots, Y_m) \in \mathbb{G}_1^{m+1}$ of size $48(m+2) = 96 + 48m$ bytes if PS is instantiated using curve BLS12-381.

7.4. Trade-offs

Switching $\mathbb{G}_1$ and $\mathbb{G}_2$. When considering the groups derived from the elliptic curve BLS12-381, which is used in the implementations we have tested, performing computations over the group $\mathbb{G}_1$ is more efficient than performing the same computations in $\mathbb{G}_2$. Also the size of elements in $\mathbb{G}_1$ (48 bytes) is smaller than the size of elements in $\mathbb{G}_2$ (96 bytes). However, the bilinearity of the pairing operation allows to define the BBS, BBS+ and PS signature algorithms (and the associated NIZKP) inverting the roles of $\mathbb{G}_1$ and $\mathbb{G}_2$, and this leads to the identification of some trade-offs that it is worth to consider when instantiating these schemes.

Note that if PS is implemented inverting the role of $\mathbb{G}_1$ and $\mathbb{G}_2$, then the signatures are $(\sigma_1, \sigma_2) \in \mathbb{G}_1^2$ and the public keys are $(X, Y_1, \dots, Y_m) \in \mathbb{G}_2^{m+1}$. Then the size of the signature is reduced to 96 bytes, the public key size is increased to $96 + 96m$ bytes and the presentation proof dimension is unchanged. Since the computations are more efficient when computed in $\mathbb{G}_1$, the signature generation (VC issuance) will be faster, while the presentation proof generation will be slower.

For what concerns BBS and BBS+ signatures, switching the two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ reduces the size of the public key to 48 bytes, increases the size of both the signature and the presentation proofs, and slows down the computations for the generation of both the signatures and the NIZKPs, since they would be performed in $\mathbb{G}_2$.

Structure of NIZKP. In Section 5 we describe the structure of the NIZKPs corresponding to each selective disclosures signature. As it is possible to note from Eq. (11), the NIZKP of CL has an element c in $\mathbb{Z}_p$, while the NIZKP of BBS (Eq. (24)) and PS (Eq. (51)) have an element T in $\mathbb{G}_1$ and BBS+ has two elements $T_1, T_2 \in \mathbb{G}_1$ (Eq. (38)).[16] The NIZKP derived from the sigma protocols for linear relations described in Appendix (see Fig. A.5) instructs the prover to compute a random commitment $T \in \mathbb{G}$, derive deterministically a challenge $c = \mathcal{H}(\mathrm{pp}, T) \in \mathbb{Z}_p$, where pp are public parameters known both to the prover and to the verifier, and from these compute the response $(r_1, \dots, r_n) \in \mathbb{Z}_p^n$. Finally the prover can build its proof π in two equivalently secure ways:

1. if the prover sends $T, (r_1, \dots, r_n)$, the verifier can compute c and verify the validity of the proof π;
2. if the prover sends $c, (r_1, \dots, r_n)$, the verifier can retrieve T and verify the validity of the proof π. Note that this does not require inverting $\mathcal{H}$ - see Fig. A.5.

When the representation of an element of the group $\mathbb{G}$ is bigger in size than an element of $\mathbb{Z}_p$, by choosing the second approach the proof is smaller.

However, it might be preferable to choose the first approach in a context in which (*i*) the Holders can present multiple VCs all at once (by proving multiple statements), for example to prove predicates which relate different VCs, and (*ii*) the Verifiers, in case of failure, want to identify the statements whose proof was incorrect.[17] This approach improves error checking.

In this context, a single challenge $c = \mathcal{H}(\mathrm{pp}, T1, \dots, T_n)$ is generated according to the commitments associated to each of the n relations to be proved. Then, according to $c, T_1, \dots, T_n$, the Holder generates the responses included in the VP. If the Verifier receives the commitments $T_1, \dots, T_n$ and the responses, it can compute the challenge and verify the correctness of each proof individually, identifying which proofs have failed, if any. Instead, if we use the second approach by sending to the Verifier the challenge c and the responses, if any of the relations have not been proven correctly, it would be impossible to identify which statements caused the failure. From the responses and the challenge c, the Verifier can reconstruct the commitments $T_1, \dots, T_n$ obtaining $\mathcal{H}(\mathrm{pp}, T_1, \dots, T_n) = c' \neq c$; this means that one or more statements have not been proven, but the Verifier cannot identify which ones.

7.5. Assessment summary

We find that `cmtList` and `merTree` are very fast to compute, cryptographically agile and with quantum-safe options, easier to implement than `SDSig` but more cumbersome for the Issuer to manage. `merTree` is reliably smaller in size, but widely adopted in standards and RFCs. Predicates must be defined by the issuer, and unlinkability requires the issuer to provide a supply of single-use credentials with new attribute salts and signatures in advance. While `CL` is particularly computationally expensive and large in size, `BBS` is computationally feasible and compact; in both cases, predicate proofs can be provided by the Holder, and the randomness for unlinkability is also generated by the Holder based on a single selective disclosure signature. Our assessment is summarized qualitatively in Table 6.

[16] The NIZKP that we have described in Eq. (38) is obtained by proving two relations as in Fig. A.5. Therefore the prover must generate two commitments T_1, T_2, one for each relation, but it can use the same challenge $c = \mathcal{H}(\mathrm{pp}, T_1, T_2)$.

[17] Private communication with one of the developers of the Docknetwork Library.

Table 6
cm assessment summary.

Feature	cmtList	merTree	CL	BBS(+)	PS
Standard	+	±	−	±	−
Agile	+++	+++	−−	+	+
Unlinkable	±	±	+	+	+
Predicates	±	±	+	+	+
Efficient	+++	+++	−	±	±
Compact	−	+	−	+	+
Quantum-safe	+	+	−	−	−

8. Conclusion

As the digital landscape continues to expand, the significance of an individual's digital identity cannot be overstated, particularly in the realms of e-government and e-commerce. The emergence of digital identity wallets represents a pivotal shift in identity management, enabling data subjects to exercise control over the information they disclose in a secure and privacy-preserving manner. This paradigm shift is exemplified by the eIDAS2 regulation, which proposes the EUDI wallet to enhance cross-border interoperability. Thus, there is a need for service providers to strike a balance between protocol sophistication, implementation intricacy, and resource constraints. The ARF document published by the EU Commission underscores the importance of cryptographic mechanisms that enable selective disclosure of verifiable credentials, which is a crucial component of providing privacy-preserving solutions.

To fill this gap, this paper provides an overview of cryptographic mechanisms to enable selective disclosure of verifiable credentials, which serve as digital counterparts to physical credentials and are safeguarded by cryptographic techniques. We analyzed a total of six mechanisms: hash-based hiding commitments `cmtList`, `merTree`, as well as CL, BBS/BBS+, and PS signatures. For each mechanism, we defined the credential and presentation structures, and summarized the operations to be performed to issue VCs and provide VPs. In order to assist stakeholders with the knowledge needed to make informed decisions regarding the selection and implementation of cryptographic mechanisms based on their specific use cases and system requirements, we compared the cryptographic mechanisms w.r.t. several features such as standardization, cryptographic agility, performance, and quantum safety.

In summary, our solution analysis indicates that `cmtList` and `merTree` are highly efficient in terms of computation cost and offer cryptographic agility, along with quantum-safe options. These mechanisms are relatively straightforward to implement, although they may pose greater management challenges for the Issuer. On the other hand, CL exhibits high computational expense and size, whereas BBS presents a more feasible and compact alternative. In both cases, the Holder has the capability to provide predicate proofs, and randomness for achieving unlinkability based on a single selective disclosure signature. We stress that currently, to develop quantum resistant VCs, one must employ hiding commitment based cryptographic mechanisms, foregoing the handy ability to create numerous unlinkable VPs (Section 6.2.1) or predicate proofs (Section 6.2.2) beginning with the same VC given by selective disclosure signatures.

Future work. The most relevant currently missing piece is given by the absence of implementations of practical selective disclosure signatures that are quantum resistant. Solutions based on lattices have been proposed [50–53] that would be interesting to test once libraries implementing them become available. The design of alternative schemes based on other quantum resistant problems would also be of interest, such as the code-based cryptography used in fourth-round standardization candidate key encapsulation mechanisms BIKE [68], HQC [69], and McEliece [70].

Another topic to investigate is an improvement of the cryptographic mechanisms based on hiding commitments. In fact, as we mentioned in Section 6.2.2, since the outputs of hash functions lose any algebraic structure related to the input, the commitment based on hash functions do not allow to prove predicates different from range proofs that can be performed e.g., by using the HashWire [56] technique. Therefore, it would be interesting to consider post quantum hiding commitment schemes that would allow the Holder to prove in zero-knowledge predicates such as equality proofs or set membership proofs about the committed values without affecting the quantum resistance of the mechanism. A candidate commitment scheme for such application could be the non-interactive commitment scheme from cryptographic group actions proposed in [71].

CRediT authorship contribution statement

Andrea Flamini: Writing – review & editing, Writing – original draft, Supervision, Conceptualization. **Giada Sciarretta:** Writing – review & editing, Writing – original draft, Supervision, Conceptualization. **Mario Scuro:** Writing – review & editing, Writing – original draft, Supervision, Conceptualization. **Amir Sharif:** Writing – review & editing, Writing – original draft, Supervision, Conceptualization. **Alessandro Tomasi:** Writing – review & editing, Writing – original draft, Supervision, Conceptualization. **Silvio Ranise:** Writing – review & editing, Writing – original draft, Supervision, Conceptualization.

Declaration of competing interest

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

Data availability

No data was used for the research described in the article.

Acknowledgments

The first author acknowledges support from Eustema S.p.A. through the PhD scholarship and is a member of GNSAGA of INdAM.

This work has been partially supported by a joint laboratory between FBK, Italy and the Italian Government Printing Office and Mint, Italy.

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

The authors would like to thank Vasilis Kalos and Lovesh Harchandani for the insightful discussions on BBS specification and implementations.

Appendix. NIZKP from sigma protocols via Fiat-Shamir transform

The NIZKP that we use in this paper are derived from three-step interactive protocols called *sigma protocols*. Sigma protocols allow a prover to prove a statement to a verifier based on the knowledge of a secret (e.g. "given $h, g \in \mathbb{G}$, I know $x \in \mathbb{Z}_p$ such that $h = g^x$"). In order to do that, it sends a commitment T to the verifier who returns a random challenge c. Finally, according to T, c and the statement to be proven, the prover sends a response r to the verifier who checks the validity of the transcript (T, c, r) for that specific statement and accept or rejects the interactive proof.

A sigma protocol can be turned into a non-interactive protocol by applying the *Fiat-Shamir transform*. Informally, the Fiat-Shamir transform prescribes to replace the generation of the challenge by the verifier with a computation of a digest of T and other public data via a cryptographic hash function $\mathcal{H}$ computed by the prover. Below we describe more in detail what is a sigma protocol and what is the Fiat-Shamir transform.

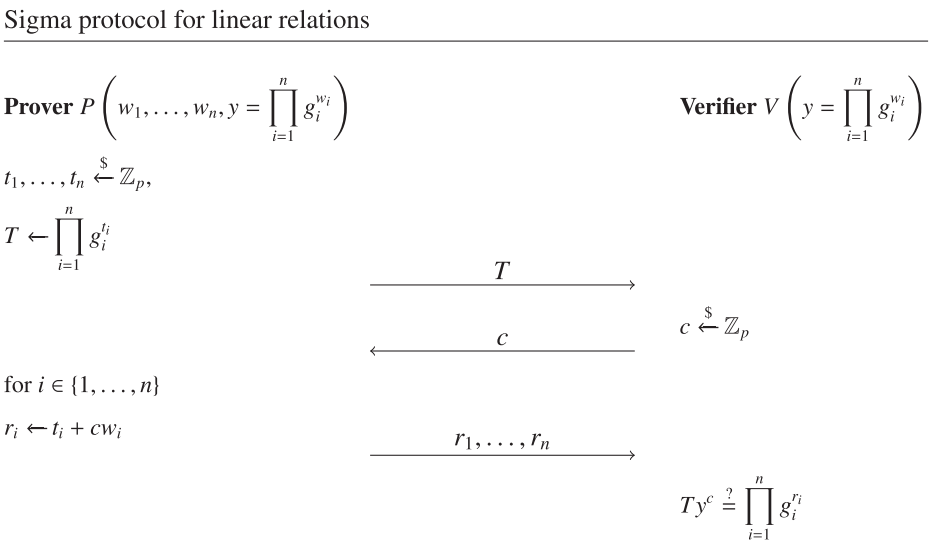

Fig. A.4. Sigma protocol for linear relations. The simulator used to prove the zero-knowledge property is defined as follows: it generates uniformly at random $s_1,\dots,s_n, c \in \mathbb{Z}_p$ and sets $T = y^{-c}\prod_{i=1}^{n} g_i^{s_i}$. The transcript $(T, c, s_1,\dots,s_n)$ is indistinguishable from a real transcript since c is random and T is random as well since it is univocally determined by $s_1,\dots,s_n$ which are chosen uniformly at random. The transcripts verify and have been created without knowing $w_1,\dots,w_n$.

Sigma protocols. A sigma protocol (P, V) for the relation $\mathcal{R}$ is defined by

- the relation $\mathcal{R} \subset W \times Y$, where Y is called the *set of statements* and W the *set of witnesses*;
- two algorithms describing the behavior of the actors involved: the prover P and the verifier V.

We say that $(w, y) \in \mathcal{R}$ if and only if w is a *witness* for the *statement* y. A sigma protocol for a relation $\mathcal{R}$ allows the prover to convince a verifier about the knowledge of a secret witness w for a public statement y.

The sigma protocols are three steps protocols with the following structure: the prover P computes a message T called *commitment* and sends it to the verifier V. Once V has received the commitment, it chooses a random *challenge* c and sends it to the prover P. Then, P computes the response r and sends it to V. Finally V outputs 1 (accept) or 0 (reject) which must be computed according to statement and the *transcript* (T, c, r) generated by the interaction. A secure sigma protocol must satisfy the following properties:

- *completeness*: when a prover knows a witness w for a statement y, the verifier will output `accept` at the end of the protocol.
- *knowledge soundness*: if the verifier outputs `accept`, it is assured that the prover actually knows a witness w for the public statement y;
- *honest-verifier zero-knowledge*: the interaction with an honest verifier in a sigma protocol execution does not leak any information about the witness known by the prover.

An example of secure sigma protocol, which is the building block of the NIZKPs mentioned in Section 5, is the *sigma protocol for linear relations* [29], a generalization of the well known Schnorr sigma protocol [72].

Given $n \in \mathbb{N}$ and $g_1,\dots,g_n \in \mathbb{G}$, where $\mathbb{G}$ is a group of order p, $g_i \in \mathbb{G}$ are public parameters, the sigma protocol for linear relations is defined by $\mathcal{R} = \{((w_1,\dots,w_n), y) \in \mathbb{Z}_p^n \times \mathbb{G} \mid y = \prod_{i=1}^{n} g_i^{w_i}\}$ and by the algorithms (P, V) presented in Fig. A.4.

Fiat-Shamir transform. In 1986 A. Fiat and A. Shamir introduced in [73] a technique to convert *identification schemes*, used to identify a user according to a secret only she knows,[18] into digital signature schemes. However, the same technique can be applied to secure sigma protocols – satisfying the completeness, knowledge soundness, and HVZK properties above – to obtain non-interactive zero-knowledge proofs [29].

The Fiat-Shamir transform substitutes the verifier with a random oracle during the second step of the sigma protocol, in which the verifier generates a random challenge; this operation can be performed by a random oracle, considered as a trusted third party that can be impersonated by a cryptographic hash function. The prover, instead of sending the commitment to the verifier, computes a cryptographic hash of the commitment, together with other public data `pp` that identify the protocol execution; the output becomes the sigma protocol challenge. The prover then computes the response to the challenge and sends the transcript to the verifier, who can compute using the same hash function the same random challenge and verify that the prover does know a witness.

Applying the Fiat-Shamir transform to the secure sigma protocol in Fig. A.4 (i.e. which satisfies the completeness, knowledge soundness and honest-verifier zero-knowledge properties) yields the NIZKP in Fig. A.5, secure according to the threat model described in Section 3.3.

Below we provide a more formal description of what it means for a sigma protocol to be HVZK compared to the one given above.

Honest-verifier zero-knowledge. The term HVZK refers to protocols in which a prover proves to an honest verifier, i.e. a verifier that follows the protocol instructions, the knowledge of some secret information w without disclosing any other information. This concept is formalized starting from a very reasonable observation: the only way an eavesdropper, who observes their interaction, can try to extract some information about w is by observing real executions of the protocol performed by the prover and the verifier. Then, based on the information exchanged between the two parties (a transcript of the protocol), the eavesdropper tries to learn something about the secret known by the prover.

However, if there exists an efficient algorithm, referred to as *Simulator*, which does not take w in input and is capable to create transcripts that are indistinguishable from the transcripts of real protocol executions, then we say that the protocol is *honest-verifier zero-knowledge*. The reason is the following: observing the prover interacting with the verifier, i.e. a real transcript, is indistinguishable from a transcript

[18] From sigma protocols is possible to derive identification schemes. In identification schemes, the statements of the sigma protocols are the public keys of users, and the user who knows the witness for such statement proves its identity.

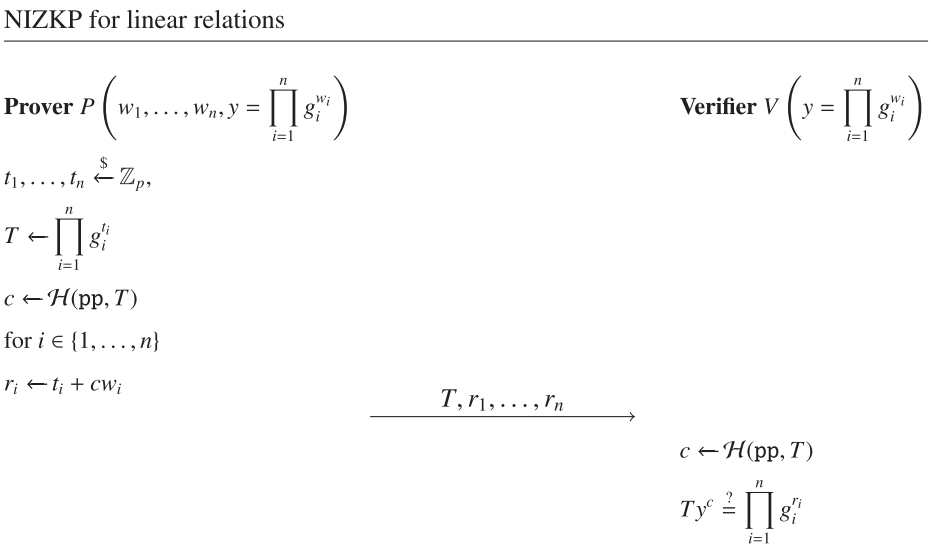

Fig. A.5. NIZKP for linear relations. The term pp is given by the public parameters such as $g_1,\dots,g_n,y$. It would be possible for the prover to create a NIZK proof by sending $(c,r_1,\dots,r_n)$. In this case the verifier must compute $T \leftarrow \prod_{i=1}^{n} g_i^{r_i} y^{-c}$ and check that $c \stackrel{?}{=} \mathcal{H}(\text{pp}, T)$.

generated by the Simulator, i.e. a simulated transcript. This means that it is possible to extract the same amount of information from the two. However, the simulator does not know the secret w, therefore it is impossible, based on the transcripts it generates, to learn some information about w. This means that, for an eavesdropper, eavesdropping the conversation between the prover and the verifier and generating transcripts on its own by executing the Simulator on her laptop, gives her the same advantage in learning information about w.

It is not trivial to determine whether a protocol satisfies the HVZK property, or indeed whether it even admits a simulator or not. The algorithm must be capable of producing transcripts indistinguishable from the ones generated in real protocol executions.

An example of simulator for the sigma protocol for linear relations is provided in the caption of Fig. A.4.

References

[1] Proposal for a Regulation of the European Parliament and of the Council amending Regulation (EU) No 910/2014 as regards establishing a framework for a European Digital Identity. 2021.

[2] Consolidated text: Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance). 2016.

[3] Amendments by the European Parliament to the Commission Proposal for a Regulation of the European Parliament and of the Council amending Regulation (EU) No 910/2014 as regards establishing a framework for a European Digital Identity. 2023.

[4] The European digital identity wallet architecture and reference framework, version 1.0.0. DG CONNECT; 2024.

[5] Steele O, Prorock M. JSON web proof for binary Merkle trees. 2021.

[6] Specification of the identity mixer cryptographic library version 2.3.0. Security Team, Computer Science Dept., IBM Research Zurich; 2010.

[7] Khovratovich D, Lodder M, Parra C. Anonymous credentials with type-3 revocation, version 0.6. Hyperledger Ursa; 2022.

[8] Lodder M, Zundel B, Khovratovich D. Pairings-based anonymous credentials with circuit-based revocation and permission policies, version 0.7. Hyperledger Ursa; 2019.

[9] Xu X. Zero-knowledge proofs in education: a pathway to disability inclusion and equitable learning opportunities. Smart Learn Environ 2024;11(7).

[10] Mashima D, Roy A. Privacy preserving disclosure of authenticated energy usage data. In: 2014 IEEE international conference on smart grid communications. SmartGridComm, 2014, p. 866–71.

[11] Ermolaev E, Abellán Álvarez I, Sedlmeir J, Fridgen G. z-Commerce: Designing a data-minimizing one-click checkout solution. In: Design science research for a new society: society 5.0. DESRIST 2023. 2023, p. 3–17.

[12] Sonnino A, Al-Bassam M, Bano S, Meiklejohn S, Danezis G. Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. In: Network and distributed systems security (NDSS) symposium 2019. 2019.

[13] Babel M, Sedlmeir J. Bringing data minimization to digital wallets at scale with general-purpose zero-knowledge proofs. 2023.

[14] Pointcheval D, Sanders O. Short randomizable signatures. In: Topics in cryptology - CT-RSA 2016. Springer; 2016, p. 111–26.

[15] Camenisch J, Lysyanskaya A. A signature scheme with efficient protocols. In: SCN 2002. LNCS, vol. 2576, 2002, p. 268–89.

[16] Camenisch J, Drijvers M, Lehmann A. Anonymous attestation using the strong Diffie Hellman assumption revisited. In: Trust 2016. LNCS, vol. 9824, 2016, p. 1–20.

[17] Tessaro S, Zhu C. Revisiting BBS signatures. In: Advances in cryptology – EUROCRYPT 2023. LNCS, vol. 14008, Springer; 2023, p. 691–721.

[18] UL LLC. Verifiable credentials and ISO/IEC 18013-5 based credentials. 2021.

[19] Christ M, Baldimtsi F, Chalkias KK, Maram D, Roy A, Wang J. SoK: Zero-knowledge range proofs. Cryptol ePrint Arch 2024. Paper 2024/430.

[20] Flamini A, Ranise S, Sciarretta G, Scuro M, Sharif A, Tomasi A. A first appraisal of cryptographic mechanisms for the selective disclosure of verifiable credentials. In: Proceedings of the 20th international conference on security and cryptography - SECRYPT. INSTICC, SciTePress; 2023, p. 123–34.

[21] Sporny M, Longley D, Chadwick D. Verifiable credentials data model. W3C; 2022.

[22] Lodderstedt T, Yasuda K, Looker T. OpenID for verifiable credential issuance. 2023.

[23] Miller J, Waite D, Jones MB. JSON web proof. 2023.

[24] Sporny M, Longley D, Chadwick D, Terbu O, Zagidulin D, Zundel B. Verifiable credentials implementation guidelines 1.0. W3C; 2019.

[25] ISO/IEC 18013-5 Personal identification - ISO-compliant driving licence - Part 5: Mobile driving licence (mDL) application. ISO; 2021.

[26] Katz J, Lindell Y. Introduction to modern cryptography: principles and protocols. Chapman and hall/CRC; 2007.

[27] Catalano D, Fiore D. Vector commitments and their applications. In: PKC 2013. LNCS, vol. 7778, Springer; 2013, p. 55–72.

[28] Camenisch J, Stadler M. Efficient group signature schemes for large groups. In: Annual international cryptology conference. 1997, p. 410–24.

[29] Boneh D, Shoup V. A graduate course in applied cryptography. 2023, https://toc.cryptobook.us/.

[30] Abdalla M, An JH, Bellare M, Namprempre C. From identification to signatures via the Fiat-Shamir transform: Minimizing assumptions for security and forward-security. In: EUROCRYPT 2002. 2002, p. 418–33.

[31] Chase M, Lysyanskaya A. On signatures of knowledge. In: Advances in cryptology - CRYPTO 2006: 26th annual international cryptology conference, Santa Barbara, California, USA, August 20-24, 2006. proceedings 26. Springer; 2006, p. 78–96.

[32] Boneh D, Boyen X, Shacham H. Short group signatures. In: CRYPTO 2004. LNCS, vol. 3152, 2004, p. 41–55.

[33] Au MH, Susilo W, Mu Y. Constant-size dynamic k-TAA. In: SCN 2006. LNCS, vol. 4116, 2006, p. 111–25.

[34] Looker T, Kalos V, Whitehead A, Lodder M. The BBS signature scheme. Internet-Draft draft-irtf-cfrg-bbs-signatures-01, Internet Engineering Task Force; 2022.

[35] Pointcheval D, Sanders O. Reassessing security of randomizable signatures. In: Topics in cryptology - CT-RSA 2018. LNCS, vol. 10808, 2018, p. 319–38.

[36] Hesse J, Singh N, Sorniotti A. How to bind anonymous credentials to humans. In: 32nd USENIX security symposium. USENIX security 23, Anaheim, CA: USENIX Association; 2023, p. 3047–64.
[37] Proposal for a Regulation of the European Parliament and of the Council laying down measures for a high level of public sector interoperability across the Union (Interoperable Europe Act). 2022.
[38] Sullivan B. Cryptographic agility. In: Black hat USA. 2010, p. 0740–7459.
[39] Housley R. Guidelines for cryptographic algorithm agility and selecting mandatory-to-implement algorithms. 2015.
[40] Barker E, Roginsky A. NIST SP 800-131A transitioning the use of cryptographic algorithms and key lengths. NIST; 2019.
[41] Fett D, Yasuda K, Campbell B. Selective disclosure for JWTs (SD-JWT). IETF; 2023.
[42] Laurie B, Messeri E, Stradling R. Certificate transparency version 2.0. 2021.
[43] ETSI TR 119 476: Electronic Signatures and Infrastructures (ESI); Analysis of selective disclosure and zero-knowledge proofs applied to Electronic Attestation of Attributes. 2023.
[44] Sakemi Y, Kobayashi T, Saito T, Wahby RS. Pairing-friendly curves. IRTF; 2022.
[45] Lyubashevsky V, Ducas L, Kiltz E, Lepoint T, Schwabe P, Seiler G, Stehlé D, Bai S. Crystals-dilithium. Algorithm Specif Support Doc 2020.
[46] Fouque P-A, Hoffstein J, Kirchner P, Lyubashevsky V, Pornin T, Prest T, Ricosset T, Seiler G, Whyte W, Zhang Z, et al. Falcon: Fast-Fourier lattice-based compact signatures over NTRU. 2018, p. 1–75, Submission to the NIST's post-quantum cryptography standardization process, 36(5).
[47] Bernstein DJ, Hülsing A, Kölbl S, Niederhagen R, Rijneveld J, Schwabe P. The SPHINCS$^+$ signature framework. In: Proceedings of the 2019 ACM SIGSAC conference on computer and communications security. 2019, p. 2129–46.
[48] NIST. FIPS 204 (initial public draft) module-lattice-based digital signature standard. 2023.
[49] NIST. FIPS 205 (initial public draft) stateless hash-based digital signature standard. 2023.
[50] Boschini C, Camenisch J, Neven G. Relaxed lattice-based signatures with short zero-knowledge proofs. In: Chen L, Manulis M, Schneider S, editors. Information security (ISC) 2018. LNCS, vol. 11060, Cham: Springer International Publishing; 2018, p. 3–22.
[51] Jeudy C, Roux-Langlois A, Sanders O. Lattice-based signature with efficient protocols, revisited. In: CRYPTO 2023. LNCS, vol. 14082, 2023, p. 351–83.
[52] Bootle J, Lyubashevsky V, Nguyen NK, Sorniotti A. A framework for practical anonymous credentials from lattices. In: CRYPTO 2023. LNCS, vol. 14082, 2023, p. 384–417.
[53] Blazy O, Chevalier C, Renaut G, Ricosset T, Sageloli E, Senet H. Efficient implementation of a post-quantum anonymous credential protocol. In: ARES '23. 2023, p. 1–11.
[54] Sporny M, Longley D. Verifiable credentials data integrity 1.0. W3C; 2022.
[55] Mobile Driver's License (mDL) implementation guidelines, version 1.2. The American Association of Motor Vehicle Administrators (AAMVA); 2023.
[56] Chalkias K, Cohen S, Lewi K, Moezinia F, Romailler Y. HashWires: Hyperefficient credential-based range proofs. In: Proceedings on privacy enhancing technologies. PoPETS, 2021.
[57] Rivest RL, Shamir A. PayWord and MicroMint: Two simple micropayment schemes. In: International workshop on security protocols. Springer; 1996, p. 69–87.
[58] Camenisch J, Chaabouni R, Shelat A. Efficient protocols for set membership and range proofs. In: ASIACRYPT. LNCS, vol. 5350, 2008, p. 234–52.
[59] Battagliola M, Longo R, Meneghetti A, Sala M. Provably unforgeable threshold EdDSA with an offline participant and trustless setup. Mediterr J Math 2023;20(5):253.
[60] Gennaro R, Goldfeder S, Narayanan A. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In: Applied cryptography and network security: 14th international conference, ACNS 2016, Guildford, UK, June 19-22, 2016. proceedings 14. Springer; 2016, p. 156–74.
[61] Crites E, Komlo C, Maller M. Fully adaptive schnorr threshold signatures. In: Handschuh H, Lysyanskaya A, editors. Advances in cryptology – CRYPTO 2023. Cham: Springer Nature Switzerland; 2023, p. 678–709.
[62] Camenisch J, Drijvers M, Lehmann A, Neven G, Towa P. Short threshold dynamic group signatures. In: International conference on security and cryptography for networks. SCN, LNCS, vol. 12238, Springer; 2020, p. 401–23.
[63] Doerner J, Kondi Y, Lee E, Shelat A, Tyner L. Threshold BBS+ signatures for distributed anonymous credential issuance. In: 2023 IEEE symposium on security and privacy. SP, IEEE; 2023, p. 773–89.
[64] Barker E. NIST SP 800-57r5 recommendation for key management, Part 1: General. NIST; 2020.
[65] Jones MB, Bradley J, Sakimura N. JSON Web Signature (JWS). 2015.
[66] NIST. FIPS 186-5 Digital Signature Standard (DSS). 2023.
[67] Barker E, Chen L, Moody D, Randall K, Regenscheid A, Robinson A. NIST SP 800-186 recommendations for discrete logarithm-based cryptography: Elliptic curve domain parameters. NIST; 2023.
[68] Aragon N, Barreto PSLM, Bettaieb S, Bidoux L, Blazy O, Deneuville J-C, Gaborit P, Ghosh S, Gueron S, Güneysu T, Aguilar-Melchor C, Misoczki R, Persichetti E, Richter-Brockmann J, Sendrier N, Tillich J-P, Vasseur V, Zémor G. BIKE: Bit flipping key encapsulation. Round 4 submission. 2022.
[69] Aguilar-Melchor C, Aragon N, Bettaieb S, Bidoux L, Blazy O, Bos J, Deneuville J-C, Dion A, Gaborit P, Lacan J, Persichetti E, Robert J-M, Véron P, Zémor G. Hamming quasi-cyclic (HQC). Fourth round version. 2023.
[70] Bernstein DJ, Chou T, Cid C, Gilcher J, Lange T, Maram V, von Maurich I, Misoczki R, Niederhagen R, Persichetti E, Peters C, Sendrier N, Szefer J, Tjhai CJ, Tomlinson M, Wang W. Classic McEliece: conservative code-based cryptography: cryptosystem specification. 2022.
[71] Gilchrist V, Marco L, Petit C, Tang G. Solving the Tensor Isomorphism Problem for special orbits with low rank points: Cryptanalysis and repair of an Asiacrypt 2023 commitment scheme. Cryptol ePrint Arch 2024. Paper 2024/337.
[72] Schnorr C-P. Efficient signature generation by smart cards. J Cryptology 1991;4:161–74.
[73] Fiat A, Shamir A. How to prove yourself: Practical solutions to identification and signature problems. In: CRYPTO '86. 1986, p. 186–94.