Efficient Issuer-Hiding Authentication, Application to Anonymous Credential

Olivier Sanders[1] and Jacques Traoré[2]

[1] Orange Innovation, Applied Crypto Group, Cesson-Sévigné, France
[2] Orange Innovation, Applied Crypto Group, Caen, France

Abstract. Anonymous credentials are cryptographic mechanisms enabling users to authenticate themselves with a fine-grained control on the information they leak in the process. They have been the topic of countless papers which have improved the performance of such mechanisms or proposed new schemes able to prove ever-more complex statements about the attributes certified by those credentials. However, whereas these papers have studied in depth the problem of the information leaked by the credential and/or the attributes, almost all of them have surprisingly overlooked the information one may infer from the knowledge of the credential issuer.
In this paper we address this problem by showing how one can efficiently hide the actual issuer of a credential within a set of potential issuers. The novelty of our work is that we do not resort to zero-knowledge proofs but instead we show how one can tweak Pointcheval-Sanders signatures to achieve this issuer-hiding property at a very low cost. This results in an efficient anonymous credential system that indeed provide a complete control of the information leaked in the authentication process. Our construction is moreover modular and can then fit a wide spectrum of applications, notably for Self-Sovereign Identity (SSI) systems.

1 Introduction

Authentication in the digital world often consists in presenting certificates delivered by issuers to verifiers that can check them with the sole knowledge of the issuers' public keys. For example, in the context of electronic passport, the issuers are governmental agencies (or private companies mandated by governments) that deliver certificates (embedded in the passport chip) that can be verified worldwide. Similarly, the European Union digital Covid certificate was issued by governmental health agencies, mostly to attest to the vaccination status of European citizens. Here again, the certificate was widely verifiable using, e.g., a dedicated application. More generally, this approach is exactly the one used by the W3C for its Verifiable Credentials [22], which represent the core of Self-Sovereign Identity (SSI) systems [13], and is envisioned by the future European Digital Identity (EUDI) wallet [11] where certificates will be issued by a set of (qualified or non-qualified) *Electronic Attestation of Attributes Providers* and then verified by *Relying Parties*.

This approach intrinsically relies on cryptographic digital signatures that support the main requested features of those use-cases: (1) centralised issuance, (2) public verification and (3) strong unforgeability assurances. However, digital signatures also come with features that may be undesirable in some situations. For example, each presentation of the certificate requires to transmit the underlying digital signature, which allows to trace users. Verification of the signature additionally requires the knowledge of all the signed data which is typically a problem in the context of digital identity where certificates usually attest to several attributes such as the name, date of birth, address, etc. It indeed means that controlling the user's age requires the knowledge of all the other attributes which are totally irrelevant. Far from being contrived, these problems are exactly the ones faced by governments trying to enforce age control to access adult-only websites, as is currently the case in France [10] or United Kingdom [16].

These limitations of standard digital signatures led cryptographers to introduce *anonymous credential*, a mechanism replicating the previous approach but in a privacy-preserving way. Concretely, users of such systems still get certificates (called *credentials*) from issuers that can then be presented while limiting the information revealed in the process. Of course, this generic goal of "limiting the information" encompasses different situations with different privacy requirements. For example, one may require different shows of the same credential to be unlinkable [7, 8, 14, 17, 19] whereas, in some cases, one simply seeks to break the link between credential issuance and credential presentation [5]. Similarly, some systems [6, 14] offer a binary control over the attributes in the sense that the attributes can be either disclosed or hidden whereas others [7, 8, 17, 19] allow to prove statements about the attributes without revealing them. This diversity of features explains the absence of widely accepted definitions and models, such as the ones [1, 2] existing for groups signatures, despite 40 years of existence [9].

In all cases, it is important to note that anonymous credentials are designed to replace digital signatures as smoothly as possible. They indeed consider the same trust model and achieve the same security property (unforgeability) than digital signatures but add this additional layer of privacy protection. For example, in the context of digital identity (*e.g.* passport) the issuer would issue an anonymous credential instead of a digital signature. Thanks to this credential, the user could reveal the requested information, for example that he is over 18 years old, without leaking anything else. Just after that, he could use the same credential to disclose his city of residence (*e.g.* to have access to a preferential rate for public transport) while being untraceable. The possibilities are actually endless, hence the diversity of constructions mentioned above.

However, a very recent line of works [3, 4, 12] has pointed out the blindspot of such mechanisms, namely the inability of hiding the credential issuer. Indeed, all the mechanisms mentioned above have devised intricate ways of concealing the credential and/or some attributes but all of them assume public knowledge of the public key of the issuer who generated the credential. This is obviously a natural assumption when only one issuer is authorised in a system but this does not reflect usual situations where credentials are generally issued by differ-

ent entities. Typically, passports and Covid certificates are issued by different countries. The architecture of the (EUDI) wallet mentioned above considered an even broader set of issuers which can be local authorities, educational entities (*e.g.* universities), private companies (*e.g.* banks), etc. In this context, revealing the credential issuer might lead to leak the very information we try to hide by using anonymous credentials. For example, in the case of age control, one may want to hide information about irrelevant attributes such as his address and this is exactly what anonymous credentials enable to do. However, doing that by using a credential issued by some local authority (e.g. a town hall) would be contradictory as it would at least reveal the area of residence.

More generally, one can infer information on the user from the sole knowledge of the credential issuer, which makes the anonymous credential approach incomplete. To fill this gap, the authors of [3,4,12] propose solutions to hide the credential issuer among a set of authorised issuers, which is also called a *policy*. The goal of the user then becomes proving possession of a valid credential from one of these issuers without telling which one.

The approach in [12] is conceptually the simplest one as it does not require any action from the verifier. Concretely, the user will raise each element of the issuer public key to some random power and then generate an OR-proof that the resulting elements are indeed a re-randomisation of one the keys in the policy. Thanks to the properties of zero-knowledge proofs, one is ensured that nothing leaks beyond that but it requires to send a number of elements linear in the number of keys in the policy, which can quickly become cumbersome.

The solution described in the PETS 2022 paper [4] is very different and relies on an intermediate primitive called aggregator. The purpose of the latter resembles the one of accumulators but the concrete instantiation consists in using a secret value sk to generate witness elements $W = g^{\frac{\mathsf{sk}}{x_i}}$ for each element $g^{\frac{1}{x_i}}$ in the authorised set $\mathcal{S}$. This is unfortunately very malleable as someone knowing x_i could derive a new witness for any x_j without knowing sk. This probably explains why the authors hash the attributes in the credential as it somehow breaks the malleability of the aggregator. In all cases, this forces the user to reveal the full set of certified attributes $\{m_i\}$ (since classical zero-knolwedge proofs do not interact smoothly with hash functions), which is something we usually try avoid in anonymous credential system. In other words, [4] trades attributes privacy for issuers privacy and so does not provide a fully complete answer to our problem.

In [3], the authors propose an elegant solution to the problem of issuer-hiding anonymous credential. Their core idea is that the verifier of the credential does not only select the policy it wants to enforce but it also signs each public key contained in the policy. This makes the user's work much easier as he can now prove that his credential is valid for one of the authorised public keys by only proving knowledge of a signature on his issuer's public key. This leads to a constant size proof, contrarily to [12]. However, this makes the whole statement to prove rather complex. Indeed one must prove knowledge of a credential and a signature S on potentially hidden attributes issued under an hidden public

key that is certified by S. The complexity of the statement is obviously reflected by the size of the proof of knowledge despite the clever instantiation described in [3]. Concretely, showing a credential in their case requires the user to send a *presentation token* of 688 Bytes (see Section 5), which is still high.

1.1 Our Contribution.

In this paper, we aim at designing an issuer-hiding anonymous credential with shorter presentation tokens without relaxing any privacy requirements. Instead of introducing a new generic framework that we would then try to instantiate as in [3, 4, 12], we start from one of the most common tools used to design standard anonymous credentials, namely PS signatures [17, 18], and then adapt it to conceal all the information about the actual issuer. The advantages of this approach is that it does not fundamentally change the underlying credential and that it avoids zero-knowledge proofs and thus the incompressible costs they entail.

Starting Point: PS signatures. PS signatures [17, 18] are pairing-based digital signatures with two important properties that make them amenable for anonymous credentials. First, they can be re-randomized, meaning that raising each element of the signature σ to the same random power leads to a new signature σ' which is still valid on the same attributes. This enables users to derive as many variants of their credentials as necessary, which is the key to untraceability: if one of the attributes is hidden (which can easily be enforced) no one can tell if σ and σ' are related. Second, they have a rather simple algebraic structure that interacts smoothly with the celebrated Schnorr zero-knowledge protocol [21].

In practice, building an anonymous credential system based on PS signature is rather straightforward. A credential on a set of n attributes $\{\mathsf{m}_i\}_{i=1}^n$ is simply a PS signature on those attributes, that is, a pair of elements σ_1 and σ_2 in a pairing-friendly group $\mathbb{G}_1$ verifying $e(\sigma_1, \widetilde{X} \prod_{i=1}^n \widetilde{Y}_i^{\mathsf{m}_i}) = e(\sigma_2, \widetilde{g})$, where $(\widetilde{g}, \widetilde{X})$ are some public parameters and $\{\widetilde{Y}_i\}_{i=1}^n$ constitute the issuer's public key. Presenting a credential then consists in re-randomising (σ_1, σ_2) and then proving knowledge of the attributes m_i one wants to conceal while revealing (or proving more complex statements about) the others. The facts that the elements m_i are involved as exponents of the previous equation makes such proofs very easy to produce using [21], as explained above. However, when it comes to hiding the issuer's identity, the problem gets much more complicated as one must now hide the elements $\widetilde{Y}_i$ that are specific to the issuer while proving that they indeed belong to the set of authorised public keys defined by the policy. As in [3], we could resort to zero-knolwedge proofs for that but we would end up with the same problem, namely a rather large proof to transmit when showing a credential.

Making PS signatures issuer-hiding. In our construction, we follow a very different approach which consists in forcing the verification equation to depend on *all* the public keys in the policy *in the same way*. Concretely, if the policy defines a set of $|\mathcal{J}|$ authorised public keys, then we replace $\prod_{i=1}^n \widetilde{Y}_i^{\mathsf{m}_i}$, which is specific to

an issuer, by $\prod_{i=1}^{n} \prod_{j \in \mathcal{J}} \widetilde{Y}_{j,i}^{\mathsf{m}_i}$ which puts all issuers on the same level. Obviously, this is just a first step as adding those elements naturally unbalance the equation. We then show that one can compensate for all those elements by adding a *single* group element $\widetilde{\sigma}$ to the signature without jeopardising security. This is easier said than done as one must ensure that $\widetilde{\sigma}$ is only used to remove the terms we artificially added in the process. In particular, one must ensure that $\widetilde{\sigma}$ is not used to cancel elements that are crucial for the security of PS signatures such as $\widetilde{X}$ or $\widetilde{Y}_i$. Of course, all of this must be done without using any zero-knowledge proof, otherwise it would bring us back to the situation of [3].

To this end, we will extend the policy by adding some elements, generated by the verifiers, that the users will be forced to use to compute $\widetilde{\sigma}$. This way, we will limit the possibilities offered by the latter element to what is strictly necessary for honest users and in particular prevent malicious use of this element as shown in our security analysis. We note that extending the policy with elements generated by the verifier is already done in [3, 4] and this is actually one of the key steps of their protocol. The same holds true in our case although the nature of those elements is extremely different. In [3] and [4], these are respectively signatures and aggregators. In our case, these additional elements are essentially re-randomised version of the public keys but with a few tweaks to prevent malleability but also to keep track of the number of such elements used to generate $\widetilde{\sigma}$. As we shall explain in Section 3, this will be crucial for security. There are also a few remaining challenges as, for example, (σ_1, σ_2) still provides information on the issuer. Fortunately, we can solve them without adding new elements by slightly adapting the protocol.

In the end, this means that our issuer-hiding variant of PS signatures only consists in two elements σ_1 and σ_2, which are essentially the ones from the original PS signature, and $\widetilde{\sigma}$. In other words, hiding issuer of PS signatures only adds one element to the credential, which seems optimal. As we believe that this variant of PS signature could be of independent interest we chose to introduce a new primitive, called Issuer-Hiding Authentication (IHA), which can be seen as a lightweight version of Issuer-Hiding Anonymous Credential (IHAC) in the sense that it only seeks to hide the issuer, not the attributes. As we shall see, upgrading our IHA construction to get an IHAC system is rather straightforward, also there are some subtleties to address. In other words, IHA can also be seen as an intermediate primitive towards full-fledged IHAC which allows to better understand the actual impact of each property on the construction. Beyond the sole efficiency of our construction, we believe that it provides new insights on the problem of hiding the issuer of a credential. In particular , it shows that one does not necessarily need to resort to the full arsenal of zero-knowledge proofs to achieve this goal, which may lead to new contributions in this area.

1.2 Organisation

In Section 2, we recall the definition of bilinear groups and a few facts about PS signatures. In Section 3, we introduce IHA that which will be the cornerstone of

the IHAC system presented the next Section. Finally, we evaluate the complexity of our construction in Section 5.

2 Preliminaries

Bilinear Groups. Our construction requires bilinear groups whose definition is recalled below.

Definition 1. *Bilinear groups are a set of three groups* $\mathbb{G}_1$, $\mathbb{G}_2$, *and* $\mathbb{G}_T$ *of order* p *along with a map, called pairing,* $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ *that is*

1. *bilinear: for any* $g \in \mathbb{G}_1, \widetilde{g} \in \mathbb{G}_2$, *and* $a, b \in \mathbb{Z}_p$, $e(g^a, \widetilde{g}^b) = e(g, \widetilde{g})^{ab}$;
2. *non-degenerate: for any* $g \in \mathbb{G}_1^*$ *and* $\widetilde{g} \in \mathbb{G}_2^*$, $e(g, \widetilde{g}) \neq 1_{\mathbb{G}_T}$;
3. *efficient: for any* $g \in \mathbb{G}_1$ *and* $\widetilde{g} \in \mathbb{G}_2$, $e(g, \widetilde{g})$ *can be efficiently computed.*

In [15], the authors introduced a classification of bilinear groups according to the existence of efficiently computable homomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$. The most interesting groups are arguably those belonging to the "type-3" family as they are, by far, the most efficient ones while supporting a broader set of computational assumptions. In this paper, we will only consider this type of bilinear groups which is anyway required by PS signatures [17].

PS Signature. PS signatures [17] are a popular tool for building privacy-preserving systems. They consist in two group elements of $\mathbb{G}_1$, no matter the size n of the signed vector $(\mathsf{m}_1, \ldots, \mathsf{m}_n)$, and can be re-randomised by raising each element to the same random power. They also benefit from a relatively simple algebraic structure which interacts smoothly with Schnorr's zero-knowledge proof systems [21]. The original paper introduced two versions of these signatures: the classical one, without any setup assumption, and the one supporting *sequential aggregation* where a pair $(X, \widetilde{X}) = (g^x, \widetilde{g}^x)$ is made public in the system parameter. This pair can for example be jointly generated by a set of issuers (or any other parties) who would each compute $(g^{x_i}, \widetilde{g}^{x_i})$, for some random x_i, and prove knowledge of the latter. If the proofs verify, the resulting pair $(X, \widetilde{X})$ would then be set as $(\prod_i g^{x_i}, \prod_i \widetilde{g}^{x_i})$. We will use this second version in our construction but will still refer to it as "PS signatures" for sake of simplicity.

- $\mathtt{Setup}(1^\lambda, n)$. This algorithm outputs the parameters pp containing the description of type-3 bilinear groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ of order p along with a set of generators $(g, \widetilde{g}) \in \mathbb{G}_1 \times \mathbb{G}_2$ and a pair $(X, \widetilde{X}) = (g^x, \widetilde{g}^x) \in \mathbb{G}_1 \times \mathbb{G}_2$ for some $x \in \mathbb{Z}_p$.
- $\mathtt{Keygen}(pp)$. This algorithm generates n random scalars $(y_1, \ldots, y_n)$ and sets sk as $(y_1, \ldots, y_n)$ and pk as $(\widetilde{Y}_1, \ldots, \widetilde{Y}_n) = (\widetilde{g}^{y_1}, \ldots, \widetilde{g}^{y_n})$.
- $\mathtt{Sign}(\mathsf{sk}, (\mathsf{m}_1, \ldots, \mathsf{m}_n))$. On input a set of n messages $(\mathsf{m}_1, \ldots, \mathsf{m}_n)$, it generates a signature $(\sigma_1, \sigma_2) \leftarrow (g^r, X^r \cdot g^{r(\sum_{i=1}^n y_i \cdot \mathsf{m}_i)})$ for some random $r \in \mathbb{Z}_p$.
- $\mathtt{Verify}(\mathsf{pk}, (\mathsf{m}_1, \ldots, \mathsf{m}_n), (\sigma_1, \sigma_2))$. This algorithm returns 1 (accept) if (1) $\mathsf{m}_i \neq 0\ \forall i$ and (2) $e(\sigma_1, \widetilde{X} \cdot \prod_{i=1}^n \widetilde{Y}_i^{\mathsf{m}_i}) = e(\sigma_2, \widetilde{g})$ and 0 otherwise.

3 Issuer-Hiding Authentication

As a first step towards an Issuer-Hiding Anonymous Credential System, we introduce an intermediate primitive called issuer-hiding authentication. It is essentially a signature scheme but verification is done through an interactive protocol [$\texttt{Show}, \texttt{Verify}$], as in an anonymous credential (AC) systems [14], and depends on a policy $\mathcal{P}$ which is essentially a set of acceptable public keys. This protocol allows the verifier to check that the user it is interacting with owns a valid signature for one of these public keys, without being able to tell which one. We however stress that this primitive only focuses on hiding the issuer of a signature and thus do not try to conceal some of the signed messages. This latter goal will be considered in Section 4.

3.1 Security Model

Syntax. An issuer-hiding authentication mechanism scheme is defined by sets of users $\mathcal{U}$, issuers $\mathcal{I}$ and verifiers $\mathcal{V}$ along with the following algorithms and protocols.

- $\texttt{Setup}(1^\lambda, n)$. This algorithm outputs the public parameters enabling to sign vectors of n messages.
- $\texttt{IKeygen}(pp)$. This algorithm is run by an issuer to generate its secret key sk_I and a public key pk_I consisting of n elements $\mathsf{pk}_{I,i}$, for $i \in [1, n]$.
- $\texttt{Sign}(\mathsf{sk}_I, (\mathsf{m}_1, \ldots, \mathsf{m}_n))$. On input a set of n messages $(\mathsf{m}_1, \ldots, \mathsf{m}_n)$, this algorithm run by the issuer owning sk_I generates a signature σ.
- $\texttt{VerifSign}(\mathsf{pk}_I, (\mathsf{m}_1, \ldots, \mathsf{m}_n), \sigma)$. This algorithm enables to check the validity of the signatures generated by the previous algorithm. It outputs 1 (valid) or 0 (invalid).
- $\texttt{SetPolicy}(\{\mathsf{pk}_{j,1}\}_{j \in \mathcal{J}_1}, \ldots, \{\mathsf{pk}_{j,n}\}_{j \in \mathcal{J}_n})$. On input a policy $\mathcal{P}$ consisting in n sets of public key elements, the algorithm generates a public component $\mathsf{pk}_\mathcal{P}$ of the policy along with a secret component $\mathsf{sk}_\mathcal{P}$.
- $\texttt{AuditPolicy}(\mathsf{pk}_\mathcal{P})$. On input $\mathsf{pk}_\mathcal{P}$, this algorithms returns either 1 or 0.
- $[\texttt{Show}((\mathsf{m}_1, \ldots, \mathsf{m}_n), \sigma, \mathsf{pk}_I, \mathsf{pk}_\mathcal{P}), \texttt{Verify}((\mathsf{m}_1, \ldots, \mathsf{m}_n), \mathsf{sk}_\mathcal{P})]$. This interactive protocol between a user and a verifier is initiated by the former who owns some signature σ issued under public key pk_I on some vector $(\mathsf{m}_1, \ldots, \mathsf{m}_n)$. At the end of the interaction, the verifier returns either 1 or 0.

Remark. We chose to keep our definitions as general as possible by considering different sets $\mathcal{J}_1, \ldots, \mathcal{J}_n$ in the policy. This in particular reflects the case where a policy would contain several public keys from the same issuer that would coincide in some indices. For those indices, our construction allows to include only one element in the policy, regardless of the number of such keys. However, in practice, we believe that the most general case will be the one where the policy will contain a set of complete public keys, each with one component per position, leading to $\mathcal{J}_1 = \ldots = \mathcal{J}_n$.

Correctness. Let pk_I be $(\mathsf{pk}_{I,1}, \ldots, \mathsf{pk}_{I,n})$. Correctness requires that the [`Show`, `Verify`] protocol between honest users and verifiers outputs 1 with probability 1 if the following two conditions are satisfied:

1. σ has been output by $\mathtt{Sign}(\mathsf{sk}_I, (\mathsf{m}_1, \ldots, \mathsf{m}_n))$
2. $\exists (j_1, \ldots, j_n) \in \mathcal{J}_1 \times \ldots \times \mathcal{J}_n$ such that $\mathsf{pk}_{I,i} = \mathsf{pk}_{j_i,i}$.

In other words, every honestly generated signature satisfying a policy $\mathcal{P}$ should be accepted by the verifier.

Remark. In our system, each public key element $\mathsf{pk}_{I,i}$ is unambiguously associated with a given position $i \in [1, n]$. This is done to retain a strong definition of unforgeability where a valid signature on $(\mathsf{m}_{\pi(1)}, \ldots, \mathsf{m}_{\pi(n)})$ -for any permutation π of $\{1, \ldots, n\}$- is considered as a valid forgery, even if a signature on $(\mathsf{m}_1, \ldots, \mathsf{m}_n)$ was already issued. It is thus assumed that all sets $\{\mathsf{pk}_{j,i}\}_{j \in \mathcal{J}_i}$ contained public key elements corresponding to the position i. The verifier, who is honest during the unforgeability game, can easily enforce this when it defines the policy $\mathcal{P}$.

Policy Audit. As we explain above, each verifier selects a policy $\mathcal{P}$ and generates a corresponding data $\mathsf{pk}_{\mathcal{P}}$ through `SetPolicy`. As we shall see, the latter specifies the authorised issuers' public keys but can also provide some elements accelerating users' computations during signature presentation. For completeness, we nevertheless need to consider the case where the verifier generating $\mathsf{pk}_{\mathcal{P}}$ is malicious. In such a case, this entity could depart from the specifications of the `SetPolicy` algorithm and generate $\mathsf{pk}_{\mathcal{P}}$ so as to leak some information on the public keys related to the signature presented by the user. This is not a specificity of our scheme as [3,4] face the same problem and therefore provide a way to check the validity of the policy. In [3], the policy contains one signature per authorised public key. Checking the policy thus consists in running the verification algorithm for every signature. In [4], the policy[3] contains a zero-knowledge proof of well-formedness that can be verified to check the policy. Finally, we note that [12] is also theoretically vulnerable to this kind of attacks as one could place invalid public keys in the policy. However, in their case, checking the validity of the policy only consists in checking whether the public keys are valid, which should not require cryptographic computations.

In all cases, the `AuditPolicy` algorithm allows anyone to detect a malicious policy and then denounce the verifier enforcing it. A verifier publishing an ill-formed $\mathsf{pk}_{\mathcal{P}}$ would then take a considerable risk, all the more so as we expect policy modifications to remain rare in most scenarios: the verifier could go unpunished only if no one runs `AuditPolicy` over the whole lifespan of the policy. In situations where this policy check cannot be done once and for all by some entity on behalf of all users, we could program the users' devices to run `AuditPolicy` with some probability ϵ *after* the [`Show`, `Verify`] protocol. This way, policy verification does not impact the performance of the signature presentation and, even

[3] [4] does not use this specific terminology but their *aggregator* notion essentially plays this role.

with a very low ϵ, one could ensure that at least one user will detect an invalid policy with overwhelming probability.

Finally, we note that the `AuditPolicy` algorithm is unnecessary if one only considers honest-but-curious adversaries.

Unforgeability. The first property we expect from an Issuer-Hiding Authentication mechanism is unforgeability. It requires that no user should be able to falsely claim possession of a signature on $(\mathsf{m}_1, \ldots, \mathsf{m}_n)$ satisfying a policy $\mathcal{P}$. This must hold true even if the messages $\mathsf{m}_1, \ldots, \mathsf{m}_n$ have been individually signed.

As in [3], we assume that the public keys accepted by the policy belong to honest issuers. This looks like a natural restriction given the issuer-hiding property we aim to achieve. We indeed recall that the very purpose of the verification algorithm in our context is to check that the user has received a signature from one of the issuers accepted by the policy $\mathcal{P}$. If one of these issuers is malicious, then the adversary can generate valid signatures (with respect to $\mathcal{P}$) on any messages of its choice and the unforgeability property becomes moot.

The formal definition of unforgeability is provided in Figure 1. It makes use of the following oracles.

- $\mathcal{O}\mathtt{Add}_I()$: this oracle creates a new issuer key pair $(\mathsf{sk}_I, \mathsf{pk}_I)$ and returns pk_I. The set of all created issuers' public key is $\mathcal{K}$.
- $\mathcal{O}\mathtt{Sign}(\mathsf{pk}_I, (\mathsf{m}_1, \ldots, \mathsf{m}_n))$: on input a vector of n messages, this oracle returns $\mathtt{Sign}(\mathsf{sk}_I, (\mathsf{m}_1, \ldots, \mathsf{m}_n))$.
- $\mathcal{O}\mathtt{SetPolicy}(\{\mathsf{pk}_{j,1}\}_{j\in\mathcal{J}_1}, \ldots, \{\mathsf{pk}_{j,n}\}_{j\in\mathcal{J}_n})$: this oracle is used to generate $(\mathsf{sk}_\mathcal{P}, \mathsf{pk}_\mathcal{P}$ for a policy $\mathcal{P} = \{\mathsf{pk}_{j,1}\}_{j\in\mathcal{J}_1}, \ldots, \{\mathsf{pk}_{j,n}\}_{j\in\mathcal{J}_n})$. The adversary then receives $\mathsf{pk}_\mathcal{P}$ while $\mathsf{sk}_\mathcal{P}$ remains secret.
- $\mathcal{O}\mathtt{Show}((\mathsf{m}_1, \ldots, \mathsf{m}_n), \mathsf{pk}_\mathcal{P})$: this oracle is queried by the adversary playing the role of the user. If no $\mathsf{pk}_\mathcal{P}$ was created through a query to $\mathcal{O}\mathtt{SetPolicy}$, then the oracle aborts. Else, it runs the `Verify` part of the protocol on $((\mathsf{m}_1, \ldots, \mathsf{m}_n), \mathsf{sk}_\mathcal{P})$ and returns the corresponding bit.

Issuer-Hiding. The second property we expect from our primitive is the issuer-hiding one which requires that the verifier, even colluding with all issuers, is unable to identify under which public key the presented signature has been signed. The adversary thus controls all entities of our system except the user that runs the `Show` protocol.

At this stage, we nevertheless need to make a few comments on some inherent limitations of the issuer-hiding property. The latter is indeed limited by the information leaked by the revealed messages as they may give some hints on the potential issuers. Typically, a proof of possession of a valid driving licence should only involve public keys of entities authorised to issue this kind of licences. Adding the public key of an university in the policy would for example be pointless as one could trivially exclude it from the list of potential issuers of the driving licence.

Unforgeability
$\texttt{Exp}^{uf}_{\mathcal{A}}(1^\lambda, n)$

1. $pp \leftarrow \texttt{Setup}(1^\lambda, n)$
2. $\mathcal{P} = (\{\mathsf{pk}_{j,1}\}_{j\in\mathcal{J}_1}, \ldots, \{\mathsf{pk}_{j,n}\}_{j\in\mathcal{J}_n}) \leftarrow \mathcal{A}^{\mathcal{O}}(pp)$
3. If $\exists i \in [1,n]$ and $j_i \in \mathcal{J}_i$ such that $\mathsf{pk}_{j_i,i} \neq \mathsf{pk}_i$, for all $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n) \in \mathcal{P}$, return 0
4. $(\mathsf{sk}_{\mathcal{P}}, \mathsf{pk}_{\mathcal{P}}) \leftarrow \texttt{SetPolicy}(\mathcal{P})$
5. $(\mathsf{m}_1, \ldots, \mathsf{m}_n) \leftarrow \mathcal{A}^{\mathcal{O}}(pp, \mathsf{pk}_{\mathcal{P}})$
6. If $\mathcal{O}\texttt{Sign}(\mathsf{pk}_I, (\mathsf{m}_1, \ldots, \mathsf{m}_n))$ was queried for some $\mathsf{pk}_I \in \mathcal{I}$ satisfying $\mathcal{P}$, return 0
7. return $b \leftarrow [\mathcal{A}(), \texttt{Verify}((\mathsf{m}_1, \ldots, \mathsf{m}_n), \mathsf{sk}_{\mathcal{P}})]$

Issuer-Hiding $\texttt{Exp}^{ih-b}_{\mathcal{A}}(1^\lambda, n)$

1. $pp \leftarrow \texttt{Setup}(1^\lambda, n)$
2. $(\mathsf{m}_1, \ldots, \mathsf{m}_n, \mathsf{pk}_{\mathcal{P}}, \sigma^{(0)}, \sigma^{(1)}, \mathsf{pk}_I^{(0)}, \mathsf{pk}_I^{(1)}) \leftarrow \mathcal{A}^{\mathcal{O}}(pp)$
3. If $\texttt{AuditPolicy}(\mathsf{pk}_{\mathcal{P}}) = 0$, return 0
4. If $\texttt{VerifSign}(\mathsf{pk}_I^{(b')}, (\mathsf{m}_1^{(b')}, \ldots, \mathsf{m}_n^{(b')}), \sigma) = 0$ for some $b' \in \{0,1\}$, return 0
5. $b^* \leftarrow [\texttt{Show}((\mathsf{m}_1, \ldots, \mathsf{m}_n), \sigma^{(b)}, \mathsf{pk}_I^{(b)}, \mathsf{pk}_{\mathcal{P}})\mathcal{A}()]$
6. Return $(b^* = b)$.

Fig. 1. Security Notions for Issuer-Hiding Authentication Mechanisms

We therefore stress that the issuer-hiding property is meaningful only if there are at least two issuers authorised for the presented attributes in the policy selected by the verifier. This is an obvious requirement of all papers targeting this property. In practice, this calls for a quick check of the verifier's policy which is completely transparent by design in our case. The way it would be done concretely obviously depends on the use case so we will not discuss it further in this paper.

In our formal definition of this property, this will be modelled by the fact that the adversary outputs $(\mathsf{m}_1, \ldots, \mathsf{m}_n)$ along with a policy $\mathcal{P} = \{\mathsf{pk}_{b,1}\}_{b\in\{0,1\}}, \ldots, \{\mathsf{pk}_{b,n}\}_{b\in\{0,1\}}$ where $(\mathsf{pk}_{b,1}, \ldots, \mathsf{pk}_{b,n})$ is the public key of some issuer I_b, for $b \in \{0,1\}$. The adversary additionally outputs the corresponding signatures $\sigma^{(0)}$ and $\sigma^{(1)}$ along with the elements in $\mathsf{pk}_{\mathcal{P}}$ that will be used in the `Show` protocol. In other words, our adversary has a total control over all the elements used in the latter protocol, which makes our model very strong. In particular, it can maliciously generate the signatures and $\mathsf{pk}_{\mathcal{P}}$, provided that they pass the sanity checks constituted by `AuditPolicy` and `VerifSign`.

In the issuer-hiding experiment of Figure 1, the adversary has access to the following oracle:

– $\mathcal{O}\texttt{Show}((\mathsf{m}_1, \ldots, \mathsf{m}_n), \sigma, \mathsf{pk}_I, \mathsf{pk}_{\mathcal{P}})$: this oracle plays the role of the user in the $[\texttt{Show}, \texttt{Verify}]$ protocol. The adversary has a full control of its inputs as long as $\texttt{AuditPolicy}(\mathsf{pk}_{\mathcal{P}}) = 1$.

Let $\mathcal{A}$ be a probabilistic polynomial adversary. An IHA mechanism is

– unforgeable if $\texttt{Adv}^{uf}(\mathcal{A}) = |\Pr[\texttt{Exp}^{uf}_{\mathcal{A}}(1^\lambda, n) = 1]|$ is negligible for any $\mathcal{A}$.

– anonymous if $\mathtt{Adv}^{ih} = |\Pr[\mathtt{Exp}_{\mathcal{A}}^{ih-1}(1^\lambda, n) = 1]$
$- \Pr[\mathtt{Exp}_{\mathcal{A}}^{ih-0}(1^\lambda, n) = 1]|$ is negligible for any $\mathcal{A}$.

3.2 Construction

Our first construction is designed as an Issuer-Hiding variant of PS signatures. The system indeed works as the PS signatures we recall in Section 2 except during the verification stage. Concretely, a signature issued on $(\mathsf{m}_1, \ldots, \mathsf{m}_n)$ by an issuer with public key $(\widetilde{Y}_{j_1,1}, \ldots, \widetilde{Y}_{j_n,n})$ is still a pair $(\sigma_1, \sigma_2) = (g^r, X^r \cdot g^{r(\sum_{i=1}^n y_{j_i,i} \cdot \mathsf{m}_i)})$. What does change is the presentation of the signature that we describe as an interactive protocol, prefiguring the `Show` protocol of an anonymous credential system.

In classical digital signature or anonymous credentials systems, verification takes as input a unique public key which acts as a very basic policy. The presented signature or anonymous credential is either valid for this public key or it is rejected. In our system, the verifier selects n sets $\{\widetilde{Y}_{j,1}\}_{j \in \mathcal{J}_1}, \ldots, \{\widetilde{Y}_{j,n}\}_{j \in \mathcal{J}_n}$ of allowed public keys that constitute the policy for the current authentication. The goal of the user is then to prove that it owns a PS signature on $(\mathsf{m}_1, \ldots, \mathsf{m}_n)$ valid for a vector of n public key elements $(\widetilde{Y}_{j_1,1}, \ldots, \widetilde{Y}_{j_n,n})$ without revealing anything on these elements beyond that $\widetilde{Y}_{j_k,k} \in \{\widetilde{Y}_{j,k}\}_{j \in \mathcal{J}_k}$, for all $1 \leq k \leq n$. In other words, it proves that $(\mathsf{m}_1, \ldots, \mathsf{m}_n)$ has been certified by one of the authorised issuers without revealing which one.

To achieve this goal, we need to move from the current verification equation of PS signatures, namely $e(\sigma_1, \widetilde{X} \cdot \prod_{i=1}^n \widetilde{Y}_{j_i,i}^{\mathsf{m}_i}) = e(\sigma_2, \widetilde{g})$, which inherently reveals the issuer, to an equation involving in the same way all issuers' public keys from the policy. More precisely, we would like to replace $\prod_{i=1}^n \widetilde{Y}_{j_i,i}^{\mathsf{m}_i}$ by something akin to $\prod_{i=1}^n [\prod_{j \in \mathcal{J}_i} \widetilde{Y}_{j,i}]^{\mathsf{m}_i}$. Clearly, this does not work with the original signature σ_1 and σ_2. We then add a new element $\widetilde{\sigma} = \prod_{i=1}^n [\prod_{j \in \mathcal{J}_i \setminus \{j_i\}} \widetilde{Y}_{j,i}]^{\mathsf{m}_i}$ to compensate for all the public keys in the product above. By construction, $\widetilde{\sigma}$ satisfies

$$e(\sigma_1, \widetilde{X} \cdot \widetilde{\sigma}^{-1} \prod_{i=1}^n [\prod_{j \in \mathcal{J}_i} \widetilde{Y}_{j,i}]^{\mathsf{m}_i}) = e(\sigma_2, \widetilde{g}).$$

We thus have our issuer-hiding verification equation but the triplet $(\sigma_1, \sigma_2, \widetilde{\sigma})$ would be a very poor solution to our problem. First, because one can still infer information on the issuer from σ_2 and $\widetilde{\sigma}$. Second, because the resulting signature scheme could be trivially broken. For example, by setting $\widetilde{\sigma} = \prod_{i=1}^n [\prod_{j \in \mathcal{J}_i} \widetilde{Y}_{j,i}]^{\mathsf{m}_i}$, one removes all the public keys from the verification equation. One can thus claim a signature on any vector $(\mathsf{m}_1, \ldots, \mathsf{m}_n)$ by simply setting $\sigma_1 = g$ and $\sigma_2 = X$.

If we focus on the second problem we can see that it stems from the inability to bound the number of public keys used to compute $\widetilde{\sigma}$. Intuitively, we use $\widetilde{\sigma}$ to compensate for the public keys that we artificially added to the equation. For each position i, we indeed added $|\mathcal{J}_i| - 1$ elements $\widetilde{Y}_{j,i}^{\mathsf{m}_i}$ and they are exactly the ones we cancel with $\widetilde{\sigma}$. The latter element can thus legitimately be built from

$|\mathcal{J}_i| - 1$ public keys. If one exceeds this threshold then one can remove *all* the public keys from $\mathcal{J}_i$ and the verification becomes moot: this is exactly the idea behind the forgery we have just described.

To enforce this threshold, the verifier will provide a set of elements $\widetilde{T}_{j,i} = (\widetilde{Y}_{j,i} \cdot \widetilde{g}^{b_i})^a$, for all public keys $\widetilde{Y}_{j,i}$ in his policy, and will force the prover to use them to build $\widetilde{\sigma}$ by raising the latter to the power $\frac{1}{a}$ in the equation. Concretely, $\widetilde{\sigma}$ will now be computed as $\prod_{i=1}^{n}[\prod_{j\in\mathcal{J}_i\setminus\{j_i\}} \widetilde{T}_{j,i}]^{\mathsf{m}_i}$ and our verification equation becomes

$$e(\sigma_1, \widetilde{X}\widetilde{\sigma}^{-\frac{1}{a}} \prod_{i=1}^{n}[\widetilde{g}^{b_i(|\mathcal{J}_i|-1)} \prod_{j\in\mathcal{J}_i} \widetilde{Y}_{j,i}]^{\mathsf{m}_i}) = e(\sigma_2, \widetilde{g}).$$

An honestly computed $\widetilde{\sigma}$ contains a component $\widetilde{g}^{b_i\cdot\mathsf{m}_i(|\mathcal{J}_i|-1)}$ which cancels the one in the equation above. Conversely, building $\widetilde{\sigma}$ from $|\mathcal{J}_i|$ elements or more leads to residual elements that invalidate the equation with overwhelming probability.

Our security analysis will formalise this, showing that satisfying this equation without a valid PS signature on $(\mathsf{m}_1, \ldots, \mathsf{m}_n)$ is not possible but we provide here the intuition behind the proof. Indeed, let us consider a malicious prover trying to forge a valid $(\sigma_1, \sigma_2, \widetilde{\sigma})$ for $(\mathsf{m}_1, \ldots, \mathsf{m}_n)$ without a valid signature. It can compute $\widetilde{\sigma}$ as $\prod_{i=1}^{n}[\prod_{j\in\mathcal{J}_i} \widetilde{T}_{j,i}^{\alpha_j^{(i)}}]$, for any tuples $(\alpha_1^{(i)}, \ldots, \alpha_{\mathcal{J}_i}^{(i)})$ but the condition we introduced means that $\sum \alpha_j^{(i)} = \mathsf{m}_i(|\mathcal{J}_i| - 1)$, for all $i \in [1, n]$. Therefore, if we consider the left member of the verification equation, we have

$$e(\sigma_1, \widetilde{X}\widetilde{\sigma}^{-\frac{1}{a}} \prod_{i=1}^{n}[\widetilde{g}^{b_i(|\mathcal{J}_i|-1)} \prod_{j\in\mathcal{J}_i} \widetilde{Y}_{j,i}]^{\mathsf{m}_i}) = e(\sigma_1, \widetilde{X} \prod_{i=1}^{n} \prod_{j\in\mathcal{J}_i} \widetilde{Y}_{j,i}^{\mathsf{m}_i - \alpha_j^{(i)}})$$

which can be satisfied only if $\sigma_2 = \sigma_1^{x+\sum_{i=1}^{n}\sum_{j\in\mathcal{J}_i} y_{j,i}(\mathsf{m}_i - \alpha_j^{(i)})}$. Since $\sum_{j\in\mathcal{J}_i} \alpha_j^{(i)} = \mathsf{m}_i(|\mathcal{J}_i| - 1)$, there is no i such that $\alpha_j^{(i)} = \mathsf{m}_i \ \forall j \in \mathcal{J}_i$. For all i, there is thus at least one $\widetilde{Y}_{j,i}$ that has a non-zero exponent $\mathsf{m}_i - \alpha_j^{(i)}$. If, for all i, there is only one $j_i \in \mathcal{J}_i$ such that $(\mathsf{m}_i - \alpha_j^{(i)}) \neq 0$, then we must have $\alpha_{j_i}^{(i)} = 0$ and $\alpha_j^{(i)} = 0$ for all $j \neq j_i$. If we simplify σ_2 accordingly, we get that (σ_1, σ_2) is exactly PS signature on $(\mathsf{m}_1, \ldots, \mathsf{m}_n)$, which would contradict the EUF-CMA security of this signature scheme. Now, if there is some i with at least two non-zero elements $(\mathsf{m}_i - \alpha_{j_b}^{(i)})$ for $b \in \{0, 1\}$, then (σ_1, σ_2) is essentially an aggregate PS signature under the public keys $\widetilde{Y}_{j_b,i}$. However PS signatures can only be sequentially aggregated, meaning that aggregation is not possible without the help of the second signer. As no issuer generates aggregated signatures in our protocol, (σ_1, σ_2) are necessarily forgeries, which, here again, would contradict the security of PS signatures.

We have thus solved the forgeability problem mentioned above but it remains to address the one regarding the information leaked by σ_2 and $\widetilde{\sigma}$. To this end

we resort to a technique similar to the one used in [19, 20]. We indeed add σ_1^{-t} to σ_2 and $\widetilde{g}^{a \cdot t}$ to $\widetilde{\sigma}$, for some random t. Taken individually, the resulting σ_2 and $\widetilde{\sigma}$ are perfectly uniform element that no longer leak anything on the issuer. One must then combine these two elements using a pairing equation to remove the masks σ_1^{-t} and $\widetilde{g}^{a \cdot t}$ but then one essentially falls back on the verification equation. Actually, our security proof will essentially show that our system is *unconditionally* issuer-hiding. While this is obviously a good point from the privacy standpoint, this has a side-effect on the unforgeability proof. As the underlying PS signature is now perfectly hidden, one can no longer reduce unforgeability to the one of PS signatures, as sketched above. This is the problem already experienced in [19,20] which adapted the original proof of PS signatures (in the Generic Group Model) to their variants. We resort to the same solution here and will then provide a tailored proof in the GGM while emphasizing that it relies on the same arguments as the original PS proof.

The scheme. As explained above, the scheme only differs from PS signatures during the [`Show`, `Verify`] step. To match the syntax of anonymous credentials we nevertheless rename `Keygen` as `IKeygen` to emphasize that it is run by an issuer.

- $\mathtt{Setup}(1^\lambda, n)$. This algorithm outputs the parameters pp containing the description of type-3 bilinear groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ of order p along with a set of generators $(g, \widetilde{g}) \in \mathbb{G}_1 \times \mathbb{G}_2$ and a pair $(X, \widetilde{X}) = (g^x, \widetilde{g}^x) \in \mathbb{G}_1 \times \mathbb{G}_2$ for some $x \in \mathbb{Z}_p$.
- $\mathtt{IKeygen}(pp)$. This algorithm generates n random scalars $(y_1, \ldots, y_n)$ and sets sk_I as $(y_1, \ldots, y_n)$ and pk_I as $(\widetilde{Y}_1, \ldots, \widetilde{Y}_n) = (\widetilde{g}^{y_1}, \ldots, \widetilde{g}^{y_n})$.
- $\mathtt{Sign}(\mathsf{sk}_I, (\mathsf{m}_1, \ldots, \mathsf{m}_n))$. On input a set of n messages $(\mathsf{m}_1, \ldots, \mathsf{m}_n)$, this algorithm generates a signature $(\sigma_1, \sigma_2) \leftarrow (g^r, X^r \cdot g^{r(\sum_{i=1}^n y_i \cdot \mathsf{m}_i)})$ for some random $r \in \mathbb{Z}_p$.
- $\mathtt{VerifSign}(\mathsf{pk}_I, (\mathsf{m}_1, \ldots, \mathsf{m}_n), \sigma)$. This algorithm checks the validity of $\sigma = (\sigma_1, \sigma_2)$ on $(\mathsf{m}_1, \ldots, \mathsf{m}_n)$ by parsing pk_I as $(\widetilde{Y}_1, \widetilde{Y}_n)$ and testing whether the following equation holds:

$$e(\sigma_1, \widetilde{X} \cdot \prod_{i=1}^{n} \widetilde{Y}_i^{\mathsf{m}_i}) = e(\sigma_2, \widetilde{g}).$$

  Note that this is exactly the verification algorithm of PS signatures.
- $\mathtt{SetPolicy}(\{\widetilde{Y}_{j,1}\}_{j \in \mathcal{J}_1}, \ldots, \{\widetilde{Y}_{j,n}\}_{j \in \mathcal{J}_n})$. On input n sets of $\mathbb{G}_2$ group elements such that $\widetilde{Y}_{j_1,i_1} \neq \widetilde{Y}_{j_2,i_2}$ for all pairs $(i_1, j_1) \neq (i_2, j_2)$, the algorithm generates $n+1$ random scalars $a, b_1, \ldots, b_n$ and computes $\widetilde{S} = \widetilde{g}^a$ along with $\widetilde{T}_{j,i} \leftarrow (\widetilde{Y}_{j,i} \cdot \widetilde{g}^{b_i})^a$ for all $i \in [1, n]$ and $j \in \mathcal{J}_i$. It then produces a zero-knowledge proof π that the elements $\widetilde{T}_{j,i}$ are well formed. An example of such a zero-knowledge proof is provided below. It then outputs the public part of the policy $\mathsf{pk}_{\mathcal{P}} = [\pi, \widetilde{S}, \{(\widetilde{Y}_{j,1}, \widetilde{T}_{j,1})\}_{j \in \mathcal{J}_1}, \ldots, \{(\widetilde{Y}_{j,n}, \widetilde{T}_{j,n})\}_{j \in \mathcal{J}_n}]$ along with a secret part $\mathsf{sk}_{\mathcal{P}} = [a, b_1, \ldots, b_n]$.

– $\texttt{AuditPolicy}(\mathsf{pk}_{\mathcal{P}})$. This algorithm runs the verification algorithm of the zero-knowledge proof π and returns 1 if π is correct and 0 otherwise.
– $[\texttt{Show}((\mathsf{m}_1,\ldots,\mathsf{m}_n),(\sigma_1,\sigma_2),\mathsf{pk}_I,\mathsf{pk}_{\mathcal{P}}),\texttt{Verify}((\mathsf{m}_1,\ldots,\mathsf{m}_n),\mathsf{sk}_{\mathcal{P}})]$. This interactive protocol is initiated by the user who first checks if the public key $\mathsf{pk}_I = (\widetilde{Y}_1,\ldots,\widetilde{Y}_n)$ associated with his signature (σ_1,σ_2) is compatible with the policy $\mathcal{P}$. This concretely means that $\exists(j_1,\ldots,j_n) \in \mathcal{J}_1 \times \ldots \times \mathcal{J}_n$ such that $\widetilde{Y}_i = \widetilde{Y}_{j_i,i}$. It then selects two random scalars r and t and computes $\sigma_1' \leftarrow \sigma_1^r$, $\sigma_2' \leftarrow \sigma_2^r \cdot (\sigma_1')^{-t}$ and $\widetilde{\sigma} \leftarrow \widetilde{S}^t \prod_{i=1}^n [\prod_{j\in\mathcal{J}_i\setminus\{j_i\}} \widetilde{T}_{j,i}]^{\mathsf{m}_i}$. The elements $(\sigma_1',\sigma_2',\widetilde{\sigma})$ are then sent to the verifier who returns 1 if (1) $\sigma_1' \neq 1$, (2) $\exists i \in [1,n] : \mathsf{m}_i \neq 0$ and (3) the following equation holds:

$$e(\sigma_1', \widetilde{X}\widetilde{\sigma}^{-\frac{1}{a}} \prod_{i=1}^{n} [\widetilde{g}^{b_i(|\mathcal{J}_i|-1)} \prod_{j\in\mathcal{J}_i} \widetilde{Y}_{j,i}]^{\mathsf{m}_i}) = e(\sigma_2', \widetilde{g}).$$

Else, it returns 0.

Remark. We explicitly include the elements $\widetilde{Y}_{j,i}$ in $\mathsf{pk}_{\mathcal{P}}$ so as to allow the prover to easily check if its certificate satisfies the policy. However, any (hopefully shorter) element unambiguously identifying $\widetilde{Y}_{j,i}$ would be sufficient.

Correctness. Let (σ_1,σ_2) be a signature under public keys $(\widetilde{Y}_{j_1,1},\ldots,\widetilde{Y}_{j_n,n})$ on $(\mathsf{m}_1,\ldots,\mathsf{m}_n)$ satisfying the verifier's policy $\mathcal{P}$ and $(\sigma_1',\sigma_2',\widetilde{\sigma})$ be the elements generated by the user during the $\texttt{Show}$ protocol. We have:

$$\begin{aligned}
&e(\sigma_1', \widetilde{X}\widetilde{\sigma}^{-\frac{1}{a}} \prod_{i=1}^{n} [\widetilde{g}^{b_i(|\mathcal{J}_i|-1)} \prod_{j\in\mathcal{J}_i} \widetilde{Y}_{j,i}]^{\mathsf{m}_i}) \\
=&e(\sigma_1', \widetilde{X}(\widetilde{S}^t \prod_{i=1}^{n} [\prod_{j\in\mathcal{J}_i\setminus\{j_i\}} \widetilde{T}_{j,i}]^{\mathsf{m}_i})^{-\frac{1}{a}} \prod_{i=1}^{n} [\widetilde{g}^{b_i(|\mathcal{J}_i|-1)} \prod_{j\in\mathcal{J}_i} \widetilde{Y}_{j,i}]^{\mathsf{m}_i}) \\
=&e(\sigma_1', \widetilde{X}(\widetilde{g}^t \prod_{i=1}^{n} [\prod_{j\in\mathcal{J}_i\setminus\{j_i\}} (\widetilde{Y}_{j,i}\widetilde{g}^{b_i})]^{\mathsf{m}_i})^{-1} \prod_{i=1}^{n} [\widetilde{g}^{b_i(|\mathcal{J}_i|-1)} \prod_{j\in\mathcal{J}_i} \widetilde{Y}_{j,i}]^{\mathsf{m}_i}) \\
=&e(\sigma_1', \widetilde{X} \cdot \widetilde{g}^{-t} \prod_{i=1}^{n} \widetilde{Y}_{j_i,i}^{\mathsf{m}_i}) = e((\sigma_1')^{x-t+\sum_{i=1}^n y_{j_i,i}\mathsf{m}_i}, \widetilde{g}) = e(\sigma_2', \widetilde{g}).
\end{aligned}$$

Policy Audit. As discussed in Section 3.1, we include in $\mathsf{pk}_{\mathcal{P}}$ a zero-knowledge proof π attesting that it has been honestly generated. Our protocol does not prescribe any particular instantiation but we note that Schnorr's proofs [21] are very well suited to our case. It indeed results in a proof π containing only $n+2$ scalars, regardless of the number of public keys in $\mathcal{P}$.

Concretely, let $H : \{0,1\}^* \to \mathbb{Z}_p$ be a hash function. The entity running $\texttt{SetPolicy}$ produces π by selecting $n+1$ random scalars $r_0,\ldots,r_n$ and computes, for all $i \in [1,n]$ and $j \in \mathcal{J}_i$, $\widetilde{K}_{j,i} \leftarrow \widetilde{T}_{j,i}^{r_0} \cdot \widetilde{g}^{r_i}$ along with $\widetilde{K} = \widetilde{S}^{r_0}$. It then computes

$c \leftarrow H(\mathsf{pk}_{\mathcal{P}}, \widetilde{K}, \{\widetilde{K}_{j,1}\}_{j\in\mathcal{J}_1}, \ldots, \{\widetilde{K}_{j,n}\}_{j\in\mathcal{J}_n})$, $z_0 = r_0 + c \cdot a^{-1}$ and $z_i = r_i - b_i \cdot c$, $\forall i \in [1, n]$. It finally outputs $\pi \leftarrow (c, z_0, \ldots, z_n)$.

The algorithm `AuditPolicy` then consists in computing $c' \leftarrow H(\mathsf{pk}_{\mathcal{P}}, \widetilde{S}^{z_0} \cdot \widetilde{g}^{-c}, \{\widetilde{T}_{j,1}^{z_0} \cdot \widetilde{g}^{z_i} \cdot \widetilde{Y}_{j,1}^{-c}\}_{j\in\mathcal{J}_1}, \ldots, \{\widetilde{T}_{j,n}^{z_0} \cdot \widetilde{g}^{z_i} \cdot \widetilde{Y}_{j,n}^{-c}\}_{j\in\mathcal{J}_n})$ and checking whether $c = c'$.

3.3 Security Analysis

The theorem below formalises the security claims we made in Section 3.2.

Theorem 2. – *Our construction is unforgeable in the generic group model if π is a zero-knowledge proof system.*
– *Our construction is issuer-hiding if π is a sound proof system.*

The proof π affects the security of our system in the sense that (1) an ill-formed $\mathsf{pk}_{\mathcal{P}}$ could leak information on the issuer's identity and (2) a proof π that would not be zero-knowledge could leak information on the scalars a or b_i, enabling the adversary to circumvent the countermeasures embedded in the elements $\widetilde{T}_{j,i}$. However, as we explain above, the proof π is only necessary to deal with active adversaries that do not shy away from tampering with $\mathsf{pk}_{\mathcal{P}}$. In situations where one would only expect honest-but-curious adversaries, one could remove it altogether. In this case, the issuer-hiding property would be ensured unconditionally.

Proof of the Issuer-Hiding Property. We prove here that the elements $(\sigma'_1, \sigma'_2, \widetilde{\sigma})$ are distributed independently of those depending on the bit b in the experiment of Figure 1, namely $(\sigma_1^{(b)}, \sigma_2^{(b)})$ and the associated public key $(\widetilde{Y}_{b,1}, \ldots, \widetilde{Y}_{b,n})$, provided that the elements in $\mathsf{pk}_{\mathcal{P}}$ are well-formed. Thanks to π, we can assume the latter point. Otherwise, one would get a direct attack against the soundness of the proof system used to generate π.

By construction $(\sigma_1^{(b)}, \sigma_2^{(b)})$ is exactly $(g^{r_b}, g^{r_b(x+\sum_i y_{b,i}\mathsf{m}_i)})$ for some r_b. This can be checked by running the verification algorithm of PS signatures. Let r and t be the scalars used to generate $(\sigma'_1, \sigma'_2, \widetilde{\sigma})$ during the `Show` protocol and let us write them $r = \frac{r'}{r_b}$ and $t = u + \sum_i y_{b,i} \cdot \mathsf{m}_i$ for some r' and u. Let $\overline{b}$ be $1 - b$. We then have

1. $\sigma'_1 = g^{r'}$
2. $\sigma'_2 = (\sigma_2^{(b)})^r \cdot (\sigma'_1)^{-t}) = g^{r'(x-u)}$
3. $\widetilde{\sigma} = \widetilde{g}^{a\cdot t} \prod_{i=1}^{n} [(\widetilde{g}^{b_i} \widetilde{Y}_{\overline{b},i})^a]^{\mathsf{m}_i}$
   $= (\widetilde{g}^u)^a \prod_{i=1}^{n} [(\widetilde{g}^{b_i} \prod_{j\in\{0,1\}} \widetilde{Y}_{j,i})^a]^{\mathsf{m}_i}$

Since r and t are uniformly random, r' and u are also uniformly random and in particular do not leak any information on b. The elements (σ'_1, σ'_2) are thus totally independent of b. The element $\widetilde{\sigma}$ involves both $\widetilde{Y}_{b,i}$ and $\widetilde{Y}_{\overline{b}}$, in the same way. One thus cannot infer information on b from it, which concludes the proof.

Proof of Unforgeability. As we explain in Section 3.2, it seems impossible to directly rely on the unforgeability of PS signatures. We will therefore adapt the arguments underlying the security of PS signatures to our system, leading to the following proof in the Generic Group Model. But first we need to deal with the zero-knowledge proof π, which is done by introducing an intermediary game.
Game 0. This is exactly the unforgeability game described in Section 1.
Game 1. This is the same game as Game 0 except at Step 4 where the challenger runs `SetPolicy` as usual but now simulates the proof π. This game is then indistinguishable from the previous one if π has indeed be generated using a zero-knowledge proof system.

Lemma 3. *In the generic group model, no adversary can succeed in Game 1 with probability greater than* $2(5 + 2(\sum_{i=1}^{n} |\mathcal{J}_i|) + 3q_S + q_G)^2/p$, *where* q_G *is a bound on the number of group oracle queries and* q_S *is a bound on the number of* $\mathcal{O}$`Sign` *queries.*

Proof of Lemma 3. Any adversary against our system has access to the following elements:

- (g, X) and $(\widetilde{g}, \widetilde{X})$ from the public parameters;
- $\{\widetilde{Y}_{j,1}\}_{j\in\mathcal{J}_1}, \ldots, \{\widetilde{Y}_{j,n}\}_{j\in\mathcal{J}_n}$ from the public keys;
- $\widetilde{S} = \widetilde{g}^a$ and $\{\widetilde{T}_{j,1}\}_{j\in\mathcal{J}_1}, \ldots, \{\widetilde{T}_{j,n}\}_{j\in\mathcal{J}_n}$ from the policy;
- $(\sigma_{k,1}, \sigma_{k,2})$ by running $\mathcal{O}$`Sign` on $(\mathsf{m}_{k,1}, \ldots, \mathsf{m}_{k,n})$ for some public key $\widetilde{Y}_{j_k,1}, \ldots, \widetilde{Y}_{j_k,n}$.

Each of these group elements is associated with a formal polynomial representing its exponent in base g (for the elements in $\mathbb{G}_1$) and $\widetilde{g}$ (for those in $\mathbb{G}_2$) whose variables are the scalars unknown to the adversary, namely x, a, $\{b_i\}_{i=1}^n$, $\{y_{j,1}\}_{j\in\mathcal{J}_1}, \ldots, \{y_{j,n}\}_{j\in\mathcal{J}_n}$, along with r_k such that $\sigma_{k,1} = g^{r_k}$.

The first step of a proof in the Generic Group Model is to show that an adversary is unable to symbolically produce a tuple $(\sigma_1^*, \sigma_2^*, \widetilde{\sigma}^*)$ passing the verification test for the policy $\mathcal{P}$ and a message $(\mathsf{m}_1, \ldots, \mathsf{m}_n) \neq (\mathsf{m}_{k,1}, \ldots, \mathsf{m}_{k,n})$, $\forall k \in [1, q_S]$.

In the GGM, the adversary can only obtain new group elements by querying the group oracle. In $\mathbb{G}_1$, this means that there are known scalars (α, α'), (β, β'), $\{(\gamma_k, \gamma'_k)\}_{k=1}^{q_s}$, $\{(\delta_k, \delta'_k)\}_{k=1}^{q_s}$ such that the polynomials $[\sigma_1^*]$ and $[\sigma_2^*]$ are respectively:

$$[\sigma_1^*] = \alpha + \beta \cdot x + \sum_k \gamma_k r_k + \sum_k \delta_k \cdot r_k (x + \sum_{i=1}^{n} y_{j_k,i} \cdot \mathsf{m}_{k,i})$$

$$[\sigma_2^*] = \alpha' + \beta' \cdot x + \sum_k \gamma'_k r_k + \sum_k \delta'_k \cdot r_k (x + \sum_{i=1}^{n} y_{j_k,i} \cdot \mathsf{m}_{k,i})$$

Similarly, in $\mathbb{G}_2$, there must be known α", β", $\{\gamma"_{j,1}, \delta"_{j,1}\}_{j\in\mathcal{J}_1}, \ldots, \{\gamma"_{j,n}, \delta"_{j,n}\}_{j\in\mathcal{J}_n}$ and ϵ such that

$$[\widetilde{\sigma}^*] = \alpha" + \beta" \cdot x + \epsilon" \cdot a + \sum_i \sum_{j \in \mathcal{J}_i} [\gamma"_{j,i} \cdot y_{j,i} + a \cdot \delta"_{j,i} \cdot (b_i + y_{j,i})]$$

Since we know that $(\sigma_1^*, \sigma_2^*, \widetilde{\sigma}^*)$ passes the verification test for the policy $\mathcal{P}$ and message $(\mathsf{m}_1^*, \ldots, \mathsf{m}_n^*)$, we have:

$$[\sigma_1^*] \cdot [x - \frac{1}{a}[\widetilde{\sigma}^*] + \sum_i \mathsf{m}_i(b_i(|\mathcal{J}_i| - 1) + \sum_{j \in \mathcal{J}_i} y_{j,i})] = [\sigma_2^*] \qquad (1)$$

Before developing this formula we note that all the monomials in $[\widetilde{\sigma}^*]$ that are not a multiple of a will end up with a factor $\frac{1}{a}$ that cannot be cancelled by any of the other elements from the equations. This means that the equations cannot be satisfied unless if those monomials are 0, which concretely means that $\alpha" = \beta" = \gamma"_{j,i} = 0$, $\forall i$ and $j \in \mathcal{J}_i$. We thus get $[\sigma_2^*] =$

$$[\sigma_1^*] \cdot [x - \epsilon" - \sum_i \sum_{j \in \mathcal{J}_i} \delta"_{j,i}(b_i + y_{j,i}) + \sum_i \mathsf{m}_i(b_i(|\mathcal{J}_i| - 1) + \sum_{j \in \mathcal{J}_i} y_{j,i})].$$

Now, we can notice that every monomial in $[\sigma_1^*]$ will be multiplied by x in the right member, resulting in some cases in monomials of degree 2 in x. However, there are no monomials of such a degree in the left member of the equation. This means that we necessarily have $\beta = \delta_k = 0$, $\forall k$.

At this stage, we therefore know that:

$$[\sigma_1^*] = \alpha + \sum_k \gamma_k r_k$$

$$[\sigma_2^*] = \alpha' + \beta' \cdot x + \sum_k \gamma'_k r_k + \sum_k \delta'_k \cdot r_k(x + \sum_{i=1}^{n} y_{j_k,i} \cdot \mathsf{m}_{k,i})$$

$$[\widetilde{\sigma}^*] = \epsilon" \cdot a + \sum_i \sum_{j \in \mathcal{J}_i} [a \cdot \delta"_{j,i} \cdot (b_i + y_{j,i})]$$

and we can thus rewrite the verification equation: $[\sigma_2^*] =$

$$[\alpha + \sum_k \gamma_k r_k] \cdot [x - \epsilon" + \sum_i \mathsf{m}_i \cdot b_i(|\mathcal{J}_i| - 1) + \sum_{j \in \mathcal{J}_i} (\mathsf{m}_i \cdot y_{j,i} - \delta"_{j,i}(b_i + y_{j,i}))]$$

If we consider the monomials of degree 0 in r_k, we get the following relations:

$$\begin{aligned} &\alpha' = -\alpha \cdot \epsilon"; \\ &\alpha = \beta'; \\ &0 = \alpha \sum_i (\mathsf{m}_i \cdot b_i(|\mathcal{J}_i| - 1) + \sum_{j \in \mathcal{J}_i} (\mathsf{m}_i \cdot y_{j,i} - \delta"_{j,i}(b_i + y_{j,i}))). \end{aligned}$$

The last equation implies that either $\alpha = 0$ or we have, for all $i \in [1, n]$:

$$\mathsf{m}_i(|\mathcal{J}_i| - 1) = \sum_{j \in \mathcal{J}_i} \delta"_{j,i} \tag{1}$$

$$0 = \mathsf{m}_i - \delta"_{j,i}, \forall j \in \mathcal{J}_i \tag{2}$$

However the two equations cannot be simultaneously satisfied unless $\mathsf{m}_i = 0$ for all $i \in [1, n]$, which would make the whole forgery invalid. We can then conclude that $\alpha = 0$.

Now, if we focus, for each k, on the monomials in r_k, we have

$$\begin{aligned}&\gamma'_k + \delta'_k(x + \sum_{i=1}^{n} y_{j_k,i} \cdot \mathsf{m}_{k,i}) \\ &= \gamma_k[x - \epsilon" + \sum_i \mathsf{m}_i \cdot b_i(|\mathcal{J}_i| - 1) + \sum_{j \in \mathcal{J}_i} (\mathsf{m}_i \cdot y_{j,i} - \delta"_{j,i}(b_i + y_{j,i}))]\end{aligned}$$

which means that $\delta'_k = \gamma_k$ and $\gamma'_k = -\gamma_k \cdot \epsilon"$. If $\gamma_k = 0\ \forall k \in [1, q_S]$, then $[\sigma_1^*] = 0$ and $(\sigma_1^*, \sigma_2^*, \widetilde{\sigma}^*)$ would then not be a valid forgery. We can then assume that there is at least one $k \in [1, q_S]$ such that $\gamma_k \neq 0$. For this k, we can note that there is no monomial in b_i in the left member of the equation. We must then have, for all $i \in [1, n]$:

$$\mathsf{m}_i(|\mathcal{J}_i| - 1) = \sum_{j \in \mathcal{J}_i} \delta"_{j,i}\ (1)$$

Similarly, considering the monomials in $y_{j,i}$ leads to the following relations, $\forall i \in [1, n]$:

$$\begin{aligned}\delta'_k \cdot \mathsf{m}_{k,i} &= \gamma_k(\mathsf{m}_i - \delta"_{j_k,i})\ (2) \\ \text{and } 0 &= \mathsf{m}_i - \delta"_{j_k,i}, \forall j \in \mathcal{J}_i \setminus \{j_k\}\ (3)\end{aligned}$$

Therefore, $\delta"_{j_k,i} = \mathsf{m}_i\ \forall j \in \mathcal{J}_i \setminus \{j_k\}$. Plugging this result in (1) means that $\delta"_{j_k,i} = 0$ and so (2) becomes $\delta'_k \cdot \mathsf{m}_{k,i} = \gamma_k \cdot \mathsf{m}_i$. As we have shown that $\delta'_k = \gamma_k$, this means that $\mathsf{m}_{k,i} = \mathsf{m}_i\ \forall i \in [1, n]$ and so that $(\sigma_1^*, \sigma_2^*, \widetilde{\sigma}^*)$ is not a valid forgery, which concludes this part of the proof.

In practice, the variables of these formal polynomials will be replaced by random scalars which could lead to accidental equalities, where two elements associated with different polynomials would be equal. This would mean that the corresponding polynomials would evaluate to the same value, leading the simulation in the GGM to fail. Thanks to the Schwartz-Zippel lemma, we can bound the probability of such an event. We indeed note that the polynomials in $\mathbb{G}_1$ and $\mathbb{G}_2$ are of degree at most 2, leading to polynomials of degree at most 4 in $\mathbb{G}_T$. In parallel, the adversary has access to $(5 + 2(\sum_{i=1}^{n} |\mathcal{J}_i|) + 3q_S + q_G)$ polynomials. The probability of an accidental validity is then at most $2(5 + 2(\sum_{i=1}^{n} |\mathcal{J}_i|) + 3q_S + q_G)^2/p$, which remains negligible for a polynomial number of queries to the group and signing oracles.

4 Issuer-Hiding Anonymous Credentials

In this section, we extend our IHA construction to build an Issuer-Hiding Anonymous Credentials (IHAC) system. Actually, we have done the bulk of the work in our previous section as we already dealt with the unforgeability and issuer-hiding properties while retaining essentially the same syntax as the one of IHAC. What remains to be done is then to allow the user to control the information it leaks on the signed messages $\mathsf{m}_1, \ldots, \mathsf{m}_n$. This will essentially be achieved thanks to an appropriate zero-knowledge proof during the [$\mathtt{Show}, \mathtt{Verify}$] protocol. Note that this proof can easily be implemented in our case. Our verification equation is indeed

$$e(\sigma'_1, \widetilde{X}\widetilde{\sigma}^{-\frac{1}{a}} \prod_{i=1}^{n} [\widetilde{g}^{b_i(|\mathcal{J}_i|-1)} \prod_{j \in \mathcal{J}_i} \widetilde{Y}_{j,i}]^{\mathsf{m}_i}) = e(\sigma'_2, \widetilde{g})$$

where the messages $\mathsf{m}_1, \ldots, \mathsf{m}_n$ to be hidden are involved as exponents. Although the elements $\widetilde{g}^{b_i}$ are not public in our IHA construction, we will show that they can be added to $\mathsf{pk}_{\mathcal{P}}$ without jeopardising security. Schnorr-like proofs [21] are thus very well suited to our case, either to perform selective disclosure of the messages, or to prove more complex statements if necessary.

4.1 Security Model

We here essentially follow the model of [3] that we slightly adapt to match the syntax of IHA. The main difference is that we do not consider a generic policy ϕ for the messages but instead allow the user to selectively disclose a subset $\{\mathsf{m}_i\}_{i \in \mathcal{I}}$ of the messages he wants to reveal, where $\mathcal{I} \subset [1, n]$. This makes the whole description simpler and better matches the usual syntax of classical anonymous credential system [14]. We nevertheless stress that proving more complex statements about $\mathsf{m}_1, \ldots, \mathsf{m}_n$ would be rather simple with our concrete construction given the form of the verification equation.

Syntax. An IHAC system involves a set of issuers, users and verifiers running some of the following algorithms. In this section, the messages m_i (resp. the signature) will be called "attributes" (resp. "credential") to comply with the usual terminology of anonymous credential systems.

- $\mathtt{Setup}(1^\lambda, n)$. This algorithm outputs the public parameters enabling to sign vectors of n attributes.
- $\mathtt{IKeygen}(pp)$. This algorithm is run by an issuer to generate his secret key sk_I and a public key pk_I consisting of n elements $\mathsf{pk}_{I,i}$, for $i \in [1, n]$.
- $\mathtt{Sign}(\mathsf{sk}_I, (\mathsf{m}_1, \ldots, \mathsf{m}_n))$. On input a set of n attributes $(\mathsf{m}_1, \ldots, \mathsf{m}_n)$, this algorithm run by the issuer owning sk_I generates a credential σ.
- $\mathtt{VerifSign}(\mathsf{pk}_I, (\mathsf{m}_1, \ldots, \mathsf{m}_n), \sigma)$. This algorithm enables to check the validity of the certificates generated by the previous algorithm. It outputs 1 (valid) or 0 (invalid).

Unforgeability
$\mathtt{Exp}^{uf-ac}_{\mathcal{A}}(1^\lambda, n)$

1. $pp \leftarrow \mathtt{Setup}(1^\lambda, n)$
2. $\mathcal{P} = (\{\mathsf{pk}_{j,1}\}_{j\in\mathcal{J}_1}, \ldots, \{\mathsf{pk}_{j,n}\}_{j\in\mathcal{J}_n}) \leftarrow \mathcal{A}^{\mathcal{O}}(pp)$
3. If $\exists i \in [1,n]$ and $j_i \in \mathcal{J}_i$ such that $\mathsf{pk}_{j_i,i} \neq \mathsf{pk}_i$, for all $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n) \in \mathcal{P}$, return 0
4. $(\mathsf{sk}_{\mathcal{P}}, \mathsf{pk}_{\mathcal{P}}) \leftarrow \mathtt{SetPolicy}(\mathcal{P})$
5. $\{\mathsf{m}_i\}_{i\in\mathcal{I}\subset[1,n]} \leftarrow \mathcal{A}^{\mathcal{O}}(pp, \mathsf{pk}_{\mathcal{P}})$
6. If $\mathcal{O}\mathtt{Sign}(\mathsf{pk}_I, (\mathsf{m}_1^*, \ldots, \mathsf{m}_n^*))$ was queried for some $\mathsf{pk}_I \in \mathcal{K}$ satisfying $\mathcal{P}$, with $\mathsf{m}_i^* = \mathsf{m}_i\ \forall i \in \mathcal{I}$, return 0
7. return $b \leftarrow [\mathcal{A}(), \mathtt{Verify}(\{\mathsf{m}_i\}_{i\in\mathcal{I}}, \mathsf{sk}_{\mathcal{P}})]$

Anonymity $\mathtt{Exp}^{ano-b}_{\mathcal{A}}(1^\lambda, n)$

1. $pp \leftarrow \mathtt{Setup}(1^\lambda, n)$
2. $(\mathcal{I}, (\mathsf{m}_1^{(0)}, \ldots, \mathsf{m}_n^{(0)}), (\mathsf{m}_1^{(1)}, \ldots, \mathsf{m}_n^{(1)}), \mathsf{pk}_{\mathcal{P}}, \sigma^{(0)}, \sigma^{(1)}, \mathsf{pk}_I^{(0)}, \mathsf{pk}_I^{(1)}) \leftarrow \mathcal{A}^{\mathcal{O}}(pp)$
3. If $\exists i^* \in \mathcal{I}$: $\mathsf{m}_{i*}^{(0)} \neq \mathsf{m}_{i*}^{(1)}$, return 0
4. If $\mathtt{AuditPolicy}(\mathsf{pk}_{\mathcal{P}}) = 0$, return 0
5. If $\exists b' \in \{0,1\}$: $\mathtt{VerifSign}(\mathsf{pk}_I^{(b')}, (\mathsf{m}_1^{(b')}, \ldots, \mathsf{m}_n^{(b')}), \sigma) = 0$, return 0
6. $b^* \leftarrow [\mathtt{Show}((\mathsf{m}_1^{(b)}, \ldots, \mathsf{m}_n^{(b)}), \sigma^{(b)}, \mathsf{pk}_I^{(b)}, \mathsf{pk}_{\mathcal{P}})\mathcal{A}()]$
7. Return $(b^* = b)$.

Fig. 2. Security Notions for Issuer-Hiding Anonymous Credentials

- $\mathtt{SetPolicy}(\{\mathsf{pk}_{j,1}\}_{j\in\mathcal{J}_1}, \ldots, \{\mathsf{pk}_{j,n}\}_{j\in\mathcal{J}_n})$. On input a policy $\mathcal{P}$ consisting in n sets of public key elements, the algorithm generates a public component $\mathsf{pk}_{\mathcal{P}}$ of the policy along with a secret component $\mathsf{sk}_{\mathcal{P}}$.
- $\mathtt{AuditPolicy}(\mathsf{pk}_{\mathcal{P}})$: on input $\mathsf{pk}_{\mathcal{P}}$, this algorithms returns either 1 or 0.
- $[\mathtt{Show}((\mathsf{m}_1, \ldots, \mathsf{m}_n), \mathcal{I}, \sigma, \mathsf{pk}_I, \mathsf{pk}_{\mathcal{P}}), \mathtt{Verify}(\{\mathsf{m}_i\}_{i\in\mathcal{I}}, \mathsf{sk}_{\mathcal{P}})]$. This interactive protocol between a user and a verifier is initiated by the former who wants to show that the presented attributes $\{\mathsf{m}_i\}_{i\in\mathcal{I}}$ have been certified under some public key pk_I accepted by the policy $\mathcal{P}$. At the end of the interaction, the verifier returns either 1 or 0.

Security Properties. Correctness can be defined as in Section 3.1. The unforgeability property is also very similar to the one defined in Figure 1 - and uses the same oracles- although we have to slightly adapt the success condition as the adversary can choose to only perform selective disclosure. The resulting experiment can be found in Figure 2. The anonymity experiment described in the same figure encompasses the issuer-hiding property and the privacy of unrevealed messages. The novelty compared to the issuer-hiding property in Figure 1 is that the challenge signatures $\sigma^{(0)}$ and $\sigma^{(1)}$ may now be associated to two different message vectors $(\mathsf{m}_1^{(0)}, \ldots, \mathsf{m}_n^{(0)})$ and $(\mathsf{m}_1^{(1)}, \ldots, \mathsf{m}_n^{(1)})$, provided that they coincide in all indices $i \in \mathcal{I}$.

Let $\mathcal{A}$ be a probabilistic polynomial adversary. An IHAC mechanism is

- unforgeable if $\mathtt{Adv}^{uf}(\mathcal{A}) = |\Pr[\mathtt{Exp}^{uf}_{\mathcal{A}}(1^\lambda, n) = 1]|$ is negligible for any $\mathcal{A}$.

– anonymous if $\mathtt{Adv}^{ano} = |\Pr[\mathtt{Exp}_{\mathcal{A}}^{ano-1}(1^\lambda, n) = 1] - \Pr[\mathtt{Exp}_{\mathcal{A}}^{ano-0}(1^\lambda, n) = 1]|$ is negligible for any $\mathcal{A}$.

4.2 Construction

Let Σ be the issuer-hiding authentication mechanism presented in Section 3.2. We construct our IHAC system as follows.

– $\mathtt{Setup}(1^\lambda, n)$. This algorithm runs $\Sigma.\mathtt{Setup}(1^\lambda, n)$ and then returns the resulting public parameters, namely the description of type-3 bilinear groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ of order p along with a set of generators $(g, \widetilde{g}) \in \mathbb{G}_1 \times \mathbb{G}_2$ and a pair $(X, \widetilde{X}) = (g^x, \widetilde{g}^x) \in \mathbb{G}_1 \times \mathbb{G}_2$ for some $x \in \mathbb{Z}_p$.
– $\mathtt{IKeygen}(pp)$. This algorithm runs $\Sigma.\mathtt{IKeygen}(pp)$ and returns the resulting keys $\mathsf{sk}_I = (y_1, \ldots, y_n)$ and $\mathsf{pk}_I = (\widetilde{Y}_1, \ldots, \widetilde{Y}_n) = (\widetilde{g}^{y_1}, \ldots, \widetilde{g}^{y_n})$.
– $\mathtt{Sign}(\mathsf{sk}_I, (\mathsf{m}_1, \ldots, \mathsf{m}_n))$. This algorithm runs $\Sigma.\mathtt{Sign}(\mathsf{sk}_I, (\mathsf{m}_1, \ldots, \mathsf{m}_n))$ to output a certificate $(\sigma_1, \sigma_2) \leftarrow (g^r, X^r \cdot g^{r(\sum_{i=1}^n y_i \cdot \mathsf{m}_i)})$ for some random $r \in \mathbb{Z}_p$.
– $\mathtt{VerifSign}(\mathsf{pk}_I, (\mathsf{m}_1, \ldots, \mathsf{m}_n), \sigma)$. This algorithm returns $\Sigma.\mathtt{VerifSign}(\mathsf{pk}_I, (\mathsf{m}_1, \ldots, \mathsf{m}_n), \sigma)$.
– $\mathtt{SetPolicy}(\{\mathsf{pk}_{j,1}\}_{j \in \mathcal{J}_1}, \ldots, \{\mathsf{pk}_{j,n}\}_{j \in \mathcal{J}_n})$. This algorithm generates $n+1$ random scalars $a, b_1, \ldots, b_n$ and computes $\widetilde{S} = \widetilde{g}^a, \widetilde{g}^{b_i(|\mathcal{J}_i|-1)}$, for $i = 1, \ldots, n$, along with $\widetilde{T}_{j,i} \leftarrow (\widetilde{Y}_{j,i} \cdot \widetilde{g}^{b_i})^a$ for all $i \in [1, n]$ and $j \in \mathcal{J}_i$. It then produces a zero-knowledge proof π that these elements are well formed by adapting the solution described in Section 3.2. It then outputs the public part of the policy $\mathsf{pk}_{\mathcal{P}} = [\pi, \widetilde{S}, \{\widetilde{g}^{b_i(|\mathcal{J}_i|-1)}\}_{i=1}^n, \{(\widetilde{Y}_{j,1}, \widetilde{T}_{j,1})\}_{j \in \mathcal{J}_1}, \ldots, \{(\widetilde{Y}_{j,n}, \widetilde{T}_{j,n})\}_{j \in \mathcal{J}_n}]$ along with the secret part $\mathsf{sk}_{\mathcal{P}} = [a, b_1, \ldots, b_n]$.
– $\mathtt{AuditPolicy}(\mathsf{pk}_{\mathcal{P}})$. This algorithm runs the verification algorithm of the zero-knowledge proof on π and returns the corresponding bit.
– $[\mathtt{Show}((\mathsf{m}_1, \ldots, \mathsf{m}_n), \mathcal{I}, \sigma, \mathsf{pk}_I, \mathsf{pk}_{\mathcal{P}}), \mathtt{Verify}(\{\mathsf{m}_i\}_{i \in \mathcal{I}}, \mathsf{sk}_{\mathcal{P}})]$. As in $\Sigma.\mathtt{Show}$, the user is expected to own a certificate $\sigma = (\sigma_1, \sigma_2)$ generated under some public key $\mathsf{pk}_I = (\widetilde{Y}_1, \ldots, \widetilde{Y}_n)$ such that $\exists (j_1, \ldots, j_n) \in \mathcal{J}_1 \times \ldots \times \mathcal{J}_n$ with $\widetilde{Y}_i = \widetilde{Y}_{j_i,i}$. It then selects two random scalars r and t and computes $\sigma_1' \leftarrow \sigma_1^r$, $\sigma_2' \leftarrow \sigma_2^r \cdot (\sigma_1')^{-t}$ and $\widetilde{\sigma} \leftarrow \widetilde{S}^t \prod_{i=1}^n [\prod_{j \in \mathcal{J}_i \setminus \{j_i\}} \widetilde{T}_{j,i}]^{\mathsf{m}_i}$. As it can no longer reveal anything about $\{\mathsf{m}_i\}_{i \in \overline{\mathcal{I}}}$, where $\overline{\mathcal{I}} = [1, n] \setminus \mathcal{I}$, it will generate a proof of knowledge π' of these values which satisfy the following equation:

$$\prod_{i \in \overline{\mathcal{I}}} e(\sigma_1', \widetilde{g}^{b_i(|\mathcal{J}_i|-1)} \prod_{j \in \mathcal{J}_i} \widetilde{Y}_{j,i})^{\mathsf{m}_i}$$
$$= e(\sigma_2', \widetilde{g}) \cdot e(\sigma_1', \widetilde{X}^{-1} \widetilde{\sigma}^{\frac{1}{a}} \prod_{i \in \mathcal{I}} [\widetilde{g}^{b_i(|\mathcal{J}_i|-1)} \prod_{j \in \mathcal{J}_i} \widetilde{Y}_{j,i}]^{\mathsf{m}_i}).$$

To this end, it can follow the Schnorr's protocol [21] by generating $|\overline{\mathcal{I}}|$ scalars $\{k_i\}_{i \in \overline{\mathcal{I}}}$ and compute $K = \prod_{i \in \overline{\mathcal{I}}} e(\sigma_1', \widetilde{g}^{b_i(|\mathcal{J}_i|-1)} \prod_{j \in \mathcal{J}_i} \widetilde{Y}_{j,i})^{k_i}$. The latter is taken as input, along with $(\sigma_1, \sigma_2, \widetilde{\sigma})$ and $\mathsf{pk}_{\mathcal{P}}$ by a hash function H

to produce a challenge c which is used to compute and $z_i = k_i + c \cdot \mathsf{m}_i$, for all $i \in \overline{\mathcal{I}}$. Verification of $(c, z_1, \ldots, z_n)$ can then be done by computing $T = e(\sigma'_2, \widetilde{g}) \cdot e(\sigma'_1, \widetilde{X}^{-1} \widetilde{\sigma}^{\frac{1}{a}} \prod_{i \in \mathcal{I}} [\widetilde{g}^{b_i(|\mathcal{J}_i|-1)} \prod_{j \in \mathcal{J}_i} \widetilde{Y}_{j,i}]^{\mathsf{m}_i})$ along with $K' = [\prod_{i \in \overline{\mathcal{I}}} e(\sigma'_1, \widetilde{g}^{b_i(|\mathcal{J}_i|-1)} \prod_{j \in \mathcal{J}_i} \widetilde{Y}_{j,i})^{z_i}] \cdot T^{-c}$ and then testing whether it leads to the same value c.

Remark. In alternative models for anonymous credentials, such as the one from [14], the users generate some key pair $(\mathsf{usk}, \mathsf{upk})$ that is used to bind the user and his credential. Concretely, this means that the credential cannot be showed without the knowledge of usk. As we use the model from [3], our users do not generate such a key pair and we thus do not consider this property. We nevertheless note that it could readily be added to our construction through a minor tweak. One of the attributes (let us say m_1) would have to be defined as the user's secret key usk. This would slightly change the signing procedure as m_1 could no longer be revealed but PS signature can easily be generated on committed values as shown in the original paper [17]. Now, since m_1 will never be revealed (that is, $\{1\} \subset \overline{\mathcal{I}}$), the user will prove knowledge of it during each authentication, which clearly leads to the requested property. An adversary able to illicitly use a given credential would have to prove knowledge of the associated secret key which, in our case, would imply an attack against the discrete logarithm assumption.

	pp	pk_I	$\mathtt{cred}$	$\mathsf{pk}_{\mathcal{P}}$	$\mathtt{Show}$
[3]	$(2+n)\mathbb{G}_1 + 2\mathbb{G}_2$	$1\mathbb{G}_2$	$2\mathbb{G}_1 + 1\mathbb{G}_2$	$(1+\lvert\mathcal{J}\rvert)\mathbb{G}_1 + 3\lvert\mathcal{J}\rvert\mathbb{G}_2$	$3\mathbb{G}_1 + 4\mathbb{G}_2 + 5\mathbb{Z}_p$
Ours	$2\mathbb{G}_1 + 2\mathbb{G}_2$	$n\mathbb{G}_2$	$2\mathbb{G}_1$	$(1+n+2n\lvert\mathcal{J}\rvert)\mathbb{G}_2 + (n+2)\mathbb{Z}_p$	$2\mathbb{G}_1 + 1\mathbb{G}_2 + 1\mathbb{Z}_p$

Table 1. Size complexity.

	pk_I	$\mathtt{cred}$	$\mathsf{pk}_{\mathcal{P}}$	$\mathtt{Show}$	$\mathtt{Verify}$
[3]	$1\mathsf{e}_2$	$4\mathsf{e}_1 + 1\mathsf{e}_2$	$(1+\lvert\mathcal{J}\rvert)\mathsf{e}_1 + 4\lvert\mathcal{J}\rvert\mathsf{e}_2$	$(11+\lvert\overline{\mathcal{I}}\rvert)\mathsf{e}_1 + 5\mathsf{e}_2 + 7\mathsf{p}$	$(8+n)\mathsf{e}_1 + 13\mathsf{p}$
Ours	$n\mathsf{e}_2$	$3\mathsf{e}_1$	$(1+2n+4n\lvert\mathcal{J}\rvert)\mathsf{e}_2$	$(1+n+\lvert\overline{\mathcal{I}}\rvert)\mathsf{e}_2 + 3\mathsf{e}_1 + \mathsf{p}$	$(1+n)\mathsf{e}_2 + 1\mathsf{e}_T + 3\mathsf{p}$

Table 2. Computational complexity.

4.3 Security Analysis

The security analysis directly follows from the one of our IHA system and is stated by the theorem below.

Theorem 4. – *Our construction is unforgeable in the generic group model if π is a zero-knowledge proof system and π' is an extractable proof system.*
– *Our construction is anonymous if π is a sound proof system and π' is a zero-knowledge proof system.*

Proof of the Issuer-Hiding Property. The proof is very similar to the one of Section 3.3. Thanks to the soundness of π we can assume well-formedness of the element in $\mathsf{pk}_{\mathcal{P}}$. We also know that $(\sigma_1^{(b)}, \sigma_2^{(b)}) = (g^{r_b}, g^{r_b(x+\sum_i y_{b,i}\mathsf{m}_i)})$ for some scalar r_b. If we write the scalars r and t generated in the `Show` protocol as $r = \frac{r'}{r_b}$ and $t = u + \sum_{i\in\overline{\mathcal{I}}} y_{b,i}\mathsf{m}_i - \sum_{i\in\mathcal{I}} y_{\overline{b},i}\mathsf{m}_i$, we then have:

1. $\sigma'_1 = g^{r'}$
2. $\sigma'_2 = (\sigma_2^{(b)})^r \cdot (\sigma'_1)^{-t}) = g^{r'(x-u+\sum_{j\in\{0,1\}}\sum_{i\in\mathcal{I}} y_{j,i}\mathsf{m}_i)}$
3. $\widetilde{\sigma} = \widetilde{g}^{a\cdot t}\prod_{i=1}^{n}[(\widetilde{g}^{b_i}\widetilde{Y}_{\overline{b},i})^a]^{\mathsf{m}_i} =$
   $(\widetilde{g}^u)^a\prod_{i\in\mathcal{I}}[(\widetilde{g}^{b_i}\prod_{j\in\{0,1\}}\widetilde{Y}_{j,i})^a]^{\mathsf{m}_i}$

It then only remains to simulate the zero-knowledge proof π' of the [`Show`, `Verify`] protocol. The elements r' and u are distributed as r and t (which are random) and so do not leak information on either b or $\{\mathsf{m}_i\}_{i\in\overline{\mathcal{I}}}$. Regarding σ'_2 and $\widetilde{\sigma}$, we note that they do not depend on $\{\mathsf{m}_i\}_{i\in\overline{\mathcal{I}}}$ and that they involve $y_{b,i}$ and $y_{\overline{b},i}$ in the same way for all $i \in \mathcal{I}$. They thus do not leak any information on b, which concludes the proof.

Proof of the Unforgeability Property. The proof relies on the same arguments as the one of the IHA mechanism but we need to slightly adapt the latter to take into account the modifications we introduced for anonymous credentials.

We first note that, when producing its forgery, the adversary no longer reveals the attributes m_i for $i \in \overline{\mathcal{I}}$. However, we can easily revert to the previous situation by running the extraction algorithm for π'. Our proof in the generic group model can thus work exactly as the one in Section 3.3 except that the adversary has now access to a few additional elements in $\mathsf{pk}_{\mathcal{P}}$, namely the ones in $\{\widetilde{g}^{b_i(|\mathcal{J}_i|-1)}\}_{i=1}^{n}$. As they all belong to $\mathbb{G}_2$, they could only be used to construct the element $\widetilde{\sigma}^*$ of the forgery. However, the equation (1) of the unforgeability proof of Section 3.3 implies that all these elements will end up with a factor $\frac{1}{a}$ in their exponent and so can only have null coefficients to satisfy the verification equation. In other words, these additional elements are useless to the adversary and so do not change[4] the security analysis.

5 Performance

In this section, we compare the performance of our IHAC scheme with the one of the instantiation proposed in [3]. We indeed recall that the two other contributions [4, 12] from the state-of-the-art do not allow like for like comparisons. Indeed, [12] only provides a generic solution to the problem of hiding the issuer of a credential which consists in an OR-proof that the issuer key is one of those from the verifier's list. This single proof is of size linear in $|\mathcal{J}|$, the number of

[4] Technically, providing new elements in a generic group model analysis changes the bound on the success probability of the adversary but the difference is insignificant.

public keys in the verifier's list, and so cannot compete with the sizes of the presentation tokens from [3] and our construction. The situation of [4] is different as it only considers the problem of hiding the issuer, not the one of leaking as few information as possible on the attributes $\mathsf{m}_1, \ldots, \mathsf{m}_n$. In that sense, it is closer to an IHA mechanism than to an anonymous credential system. However, even in this context, we note that the user has to send 960 Bytes in [4] whereas he can only send 192 Bytes with our construction of Section 3, when using BLS12-381 curves in both cases.

We recall that we have considered the most general situation where $|\mathcal{J}_i|$ may be different from $|\mathcal{J}_j|$ for some $i \neq j$. For a proper comparison with [3], we will nevertheless assume that $|\mathcal{J}_1| = \ldots = |\mathcal{J}_n| = |\mathcal{J}|$, which slightly disadvantages our construction.

For both [3] and our construction, the `Show` complexity does not include the n attributes that are either revealed or hidden in a proof of knowledge. Note that, in all cases, this leads to n elements of $\mathbb{Z}_p$ when using the Schnorr's proof of knowledge.

Table 1 shows that our public keys, and hence our policies, are larger than those in [3] but that our `Show` protocol requires to send less elements. Concretely, when instantiating the bilinear groups with the BLS12-381 curve, the `Show` protocol requires to send 688 Bytes in the case of [3] and 224 Bytes in the case of our construction, hence a threefold improvement. We believe that this is an interesting feature of our construction because the `Show` protocol is usually the one subject to the most stringent requirements but we note that [3] offers an interesting tradeoff in the case where public elements (public keys or policies) would frequently change.

Table 2 evaluate the computational complexity of [3] and our construction in the worst case, namely the one where all attributes would be hidden. It only considers costly operations, namely exponentiations (denoted e_i) and pairings (denoted p). In the case of our construction, we note that the element K can be computed as

$$K = e(\sigma'_1, \prod_{i \in \overline{\mathcal{I}}} [\widetilde{g}^{b_i(|\mathcal{J}_i|-1)} \prod_{j \in \mathcal{J}_i} \widetilde{Y}_{j,i}]^{k_i})$$

so as to minimise the number of pairings. We have rewritten similarly the elements in [3] for a fair comparison. We also recall that the elements $\widetilde{g}^{b_i(|\mathcal{J}_i|-1)}$ are provided in $\mathsf{pk}_{\mathcal{P}}$ and so do need to be recomputed. This table illustrates the differences of the approach in [3] and ours. In our case, the user's computations in the `Show` protocol involve more exponentiations but less pairings than in [3]. The number of exponentiations is essentially the same during verification but our protocol require to evaluate less pairings (3) than in [3] (13).

Swapping groups. We instantiate our construction so as to optimise the sizes of the credential and of the elements sent by the user during `Show` as these are usually the main metrics for anonymous credentials. Concretely, we generate our credentials in $\mathbb{G}_1$ to minimise the number of elements in $\mathbb{G}_2$. This nevertheless implies that most computations will be performed in $\mathbb{G}_2$, which is not ideal if

one favours computational performance. In the latter case, we recall that the roles of $\mathbb{G}_1$ and $\mathbb{G}_2$ are interchangeable and so that one can straightforwardly derive a variant of our scheme by simply swapping groups $\mathbb{G}_1$ and $\mathbb{G}_2$. In that case, almost all computations will be performed in $\mathbb{G}_1$ at the cost of twice larger credentials (192B instead of 96B) and slightly larger (+48B) `Show` transcript.

6 Conclusion

In this paper, we have proposed a new approach to hide all information about the issuer when presenting a credential, which was so far an important source of leakage in classical anonymous credentials systems. Our approach does not require heavy machinery such as complex zero-knowledge proofs but instead simply extends PS signatures with one additional element. It leads to a `Show` transcript that is only slightly larger than the one of classical systems, proving that the issuer-hiding feature is not necessarily costly to add. In the process, we have introduced an intermediate primitive, Issuer-Hiding Authentication, whose purpose is orthogonal, but complementary, to the one of anonymous credentials. We believe it constitutes a worthwhile addition to the arsenal of privacy-preserving mechanisms as it efficiently addresses a problem that was, so far, mostly overlooked.

Acknowledgments

The authors are grateful for the support of the ANR through project ANR-18-CE-39-0019-02 MobiS5.

References

1. Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 614–629. Springer, Heidelberg, May 2003.
2. Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 136–153. Springer, Heidelberg, February 2005.
3. Jan Bobolz, Fabian Eidens, Stephan Krenn, Sebastian Ramacher, and Kai Samelin. Issuer-hiding attribute-based credentials. In Mauro Conti, Marc Stevens, and Stephan Krenn, editors, *CANS 21*, volume 13099 of *LNCS*, pages 158–178. Springer, Heidelberg, December 2021.
4. Daniel Bosk, Davide Frey, Mathieu Gestin, and Guillaume Piolle. Hidden issuer anonymous credential. *Proc. Priv. Enhancing Technol.*, 2022.
5. Stefan Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy.* 01 2000.

6. Jan Camenisch, Maria Dubovitskaya, Kristiyan Haralambiev, and Markulf Kohlweiss. Composable and modular anonymous credentials: Definitions and practical constructions. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 262–288. Springer, Heidelberg, November / December 2015.
7. Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN 02*, volume 2576 of *LNCS*, pages 268–289. Springer, Heidelberg, September 2003.
8. Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72. Springer, Heidelberg, August 2004.
9. David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 1981.
10. CNIL. Online age verification: balancing privacy and the protection of minors. https://www.cnil.fr/en/online-age-verification-balancing-privacy-and-protection-minors, 2022.
11. European Comission. The european digital identity wallet architecture and reference framework. https://digital-strategy.ec.europa.eu/en/library/european-digital-identity-wallet-architecture-and-reference-framework, 2023.
12. Aisling Connolly, Pascal Lafourcade, and Octavio Perez-Kempner. Improved constructions of anonymous credentials from structure-preserving signatures on equivalence classes. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022, Part I*, volume 13177 of *LNCS*, pages 409–438. Springer, Heidelberg, March 2022.
13. Sovrin Foundation. Principles of ssi v3. https://sovrin.org/principles-of-ssi/, 2022.
14. Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. *Journal of Cryptology*, 32(2):498–546, April 2019.
15. Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.
16. Open Rights Group. Uk online safety bill will mandate dangerous age verification for much of the web. https://www.openrightsgroup.org/publications/uk-online-safety-bill-will-mandate-dangerous-age-verification-for-much-of-the-web/, 2023.
17. David Pointcheval and Olivier Sanders. Short randomizable signatures. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 111–126. Springer, Heidelberg, February / March 2016.
18. David Pointcheval and Olivier Sanders. Reassessing security of randomizable signatures. In Nigel P. Smart, editor, *CT-RSA 2018*, volume 10808 of *LNCS*, pages 319–338. Springer, Heidelberg, April 2018.
19. Olivier Sanders. Efficient redactable signature and application to anonymous credentials. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 628–656. Springer, Heidelberg, May 2020.
20. Olivier Sanders. Improving revocation for group signature with redactable signature. In Juan Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 301–330. Springer, Heidelberg, May 2021.

21. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, August 1990.
22. World Wide Web Consortium (W3C). Verifiable credentials data model v2.0. https://www.w3.org/TR/vc-data-model-2.0/, 2023.