Attribute-Based Signatures for Supporting Anonymous Certification

Nesrine Kaaniche(✉) and Maryline Laurent

SAMOVAR, Telecom SudParis, CNRS, University Paris-Saclay, Evry, France
{nesrine.kaaniche,maryline.laurent}@telecom-sudparis.eu

Abstract. This paper presents an anonymous certification (AC) scheme, built over an attribute based signature (ABS). After identifying properties and core building blocks of anonymous certification schemes, we identify ABS limitations to fulfill AC properties, and we propose a new system model along with a concrete mathematical construction based on standard assumptions and the random oracle model. Our solution has several advantages. First, it provides a data minimization cryptographic scheme, permitting the user to reveal only required information to any service provider. Second, it ensures unlinkability between the different authentication sessions, while preserving the anonymity of the user. Third, the derivation of certified attributes by the issuing authority relies on a non interactive protocol which provides an interesting communication overhead.

Keywords: User privacy · Anonymous certification · Attribute-based signatures

1 Introduction

Anonymous Credentials (AC) were first introduced by David Chaum [9] in 1982, and fully formalized by Camenisch and Lysyanskaya [4] in 2001. These schemes are considered to be an important building block in privacy-preserving identity management systems, as they permit users to prove ownership of credentials to service providers while not being traced in the system. That is, after he gets credentials over some of his attributes from some trusted issuing authorities, the user can derive proofs for successive presentations to service providers. The AC properties include that the service providers are not able to link a single received proof to another or to any information relative to the owner, even in case of collusion between providers and with the credential issuer.

Up to now, two main AC solutions emerged from industry, the Idemix scheme [15] from IBM based on the Camenish-Lysyanskaya (CL) signatures [4,5], which is a close variant of group signatures, and the U-Prove scheme [20] from Microsoft which relies on the Brands' blind signature [2].

N. Kaaniche and M. Laurent—Member of the Chair Values and Policies of Personal Information.

I. Askoxylakis et al. (Eds.): ESORICS 2016, Part I, LNCS 9878, pp. 279–300, 2016.
DOI: 10.1007/978-3-319-45744-4_14

Interest for AC comes from their ability to strictly support the data minimization principle [17], which expects that data collection should be proportional and not excessive compared to the purpose of the collection. This interest is today magnified as this principle is at the core of the future European General Data Protection Regulation [11] and also the U.S. National Strategy for Trusted Identities in Cyberspace (NSTIC) [14].

This paper proposes a new AC scheme based on the Attribute Based Signatures (ABS). Originally, the ABS is designed for the user to sign a message with fine grained control over identifying information, and it does not support the properties required for AC. As such, after a clear identification of missing properties, an abstract scheme $\mathcal{HABS}$ is presented followed by a concrete construction detailing how these access-policy based signatures can efficiently serve AC objectives. Our scheme has several advantages over industrial AC solutions. First, the issuance procedure is much more efficient, as there is no need for a heavy interactive protocol between the user and the issuer. The issuer can generate a credential based on the user's public key only, while Idemix and U-Prove schemes require the user to introduce a random part of the secret key each time a new credential is certified as they rely on group and blind signatures. Second, our scheme supports a flexible selective disclosure mechanism at no extra computation cost, which is inherited from the expressiveness of ABS for defining access policies.

Paper Organization – Section 2 introduces Anonymous Credential systems (AC) along with the actors, procedures and security requirements. Section 3 defines the Attribute based Signatures (ABS), and provides a generic analysis of ABS properties thus highlighting the missing properties for the ABS to align to the AC requirements. Section 5 presents a concrete construction of our novel AC system, and Sect. 6 gives a detailed security analysis with an extension of the scheme to support multiple issuers. Finally, theoretical comparisons with existing systems are discussed in Sect. 7 and conclusions are given in Sect. 8.

2 Anonymous Credentials

Anonymous Credentials (AC) also known as privacy preserving attribute credentials refer to some well identified entities and procedures and need to achieve some security requirements.

Entities – An anonymous credential system involves several entities. Some entities, such as the *user*, the *verifier* and *issuer* are mandatory, while other entities, such as the *revocation authority* and the *inspector* are optional [3].

The *user* is the central entity, whose interest is to get privacy-preserving access to services, offered by service providers, known as *verifiers*. Each verifier enforces an access control policy to its resources and services based on the credentials owned by the users and the information selected and included in *presentation tokens*. For this purpose, each user has first to obtain credentials from the *issuer(s)*. Then, he selects the appropriate information from the credentials and shows the selected information to the requesting verifier, under a

presentation token. Note that the verifier access control policy is referred to as *presentation policy*. Both the user and the verifier have to obtain the most recent revocation information from the *revocation authority* to generate, respectively verify, presentation tokens. The *revocation authority* has to revoke issued credentials and maintain the list of valid credentials in the system. When revoked, a credential can no longer be used to derive presentation tokens. The *inspector* is a trusted entity, which has the technical capabilities to remove the anonymity of a user when needed.

PROCEDURES – An AC system is defined by the following algorithms:

- SETUP: this algorithm takes as input a security parameter ξ (security level) and outputs the public parameters $params$ and the public-private key pair of the issuer (pk_{is}, sk_{is}).
- USERKG: this algorithm takes as input $j \in \mathbb{N}$ and outputs the key pair (pk_u, sk_u) of the user j.
- OBTAIN $\leftrightarrow$ ISSUE: the OBTAIN $\leftrightarrow$ ISSUE presents the issuance procedure. The ISSUE algorithm performed by the issuer takes as input the public parameters $params$, the private key of the issuer pk_{is}, the public key of the user sk_u and the set of attributes $\{a_i\}_{i=1}^{N}$, where N is the number of attributes. The OBTAIN algorithm executed by the user takes as input the secret key of the user sk_u and the public key of the issuer pk_{is}. At the end of this phase, the user receives from the issuer a credential C.
- SHOW $\leftrightarrow$ VERIFY: the SHOW $\leftrightarrow$ VERIFY is the presentation procedure between the user and the verifier. With respect to the presentation policy, the SHOW algorithm takes as input the user's secret key sk_u, the issuer's public key pk_{is}, the set of required attributes $\{a_i\}_{i=1}^{N'}$ and a credential C, and it outputs a presentation token. VERIFY is a public algorithm performed by the verifier; it takes as input the public key of the issuer pk_{is}, the set of attributes $\{a_i\}_{i=1}^{N'}$, and the presentation token. At the end of this presentation phase, the VERIFY outputs a bit $b \in \{0, 1\}$ for success of failure of the verification.

SECURITY REQUIREMENTS – Anonymous credential systems have to fulfill the following security properties:

correctness – a honest user must always succeed in proving validity of proofs to the verifier in an anonymous way.
anonymity – the user must remain anonymous among a set of users during the presentation procedure to the verifier.
unforgeability – a user not owning an appropriate legitimate credential is not able to generate a valid presentation token.
unlinkability – this property is essential for user privacy support and is closely related to the anonymity property. Unlinkability is divided into two properties *issue-show unlinkability* and *multi-show unlinkability* as follows: (i) the issue-show unlinkability ensures that any information gathered during credential issuing cannot be used later to link the presentation token to the original credential, (ii) the multi-show unlinkability guarantees that several presentation tokens derived from the same credential and transmitted over several sessions can not be linked by the verifier.

Additionally, privacy preserving attribute based credentials have to ensure several functional features, namely revocation, inspection and *selective disclosure*. The selective disclosure property refers to the ability provided to the user to present to the verifier partial information extracted or derived from his credential, for instance, to prove he is older than 18 to purchase liquors, while not revealing his birth date.

3 Attribute Based Signatures for Anonymous Credentials

This section introduces Attribute based Signature schemes (ABS) with their associated algorithms and their security properties. Then, an analysis shows that ABS is missing some properties to serve as a building block for AC support.

3.1 Attribute-Based Signatures (ABS)

Attribute-based Signatures (ABS) [19] is a flexible primitive that enables a user to sign a message with fine grained control over identifying information. In ABS, the user possesses a set of attributes and one secret signing key per attribute which is obtained from a trusted authority. The user can sign a message with respect to a predicate satisfied by his attributes. In commonly known settings, the different parties include a Signature Trustee (ST), the Attribute Authority (AA), and potentially several signers and verifiers. The ST acts as a global entity that generates authentic global systems parameters, while the AA issues the signing keys for the set of attributes of the users (signers). The role of ST and AA can be merged into the same entity. ABS supports the following property which is fundamental for support of privacy. AA, although knowing the signing keys and the attributes of the users, is unable to identify which attributes have been used in a given valid signature, and thus he is unable to assign the signature to his originating user and/or to link several signatures as originating from the same user. In the last few years, multiple ABS schemes emerged in the cryptographic literature, considering different design directions. In a nutshell, (i) the attribute value can be a binary-bit string [13,18,19,21,22], or has a particular data structure [23], (ii) access structures may support threshold policies [13,18,22], monotonic policies [19,23] or non-monotonic policies [21], and (iii) the capacity of attributes' private keys issuance can be provided by a single authority [19,22,23], or a group of authorities [19,21].

Let us explain the general ABS signing procedure in the simple case with one single AA authority. First, the AA derives the private keys $\{sk_1, \cdots, sk_N\}$, with respect to the attribute set identifying the requesting signer, denoted by $\mathcal{S} = \{a_1, \cdots, a_N\}$, where N is the number of attributes. The private keys' generation procedure is performed using the AA's master key MK and some related public parameters, both generated during the setup phase. Then, for signing a message m sent by the verifier along with a signing predicate Υ, the user needs his private keys and a set of attributes satisfying the predicate Υ. Finally, the verifier is able to verify that some user who holds a set of attributes satisfying the signing

predicate has signed the message. An ABS scheme is defined by the following algorithms:

- $\mathcal{ABS}$.`setup` – this algorithm is performed by (ST). It takes as input the security parameter ξ and outputs the global public parameters $params$, considered as an auxiliary input to all the following algorithms, and the master key MK of AA.
- $\mathcal{ABS}$.`keygen` – this algorithm executed by AA takes as input his master key MK and a set of attributes $\mathcal{S} \subset \mathbb{S}$ (where $\mathcal{S} = \{a_i\}_{i=1}^{N}$, N is the number of attributes and $\mathbb{S}$ is the attribute universe). It outputs a signing key $sk_{\mathcal{S}}$[1].
- $\mathcal{ABS}$.`sign` – this algorithm takes as input the private key $sk_{\mathcal{S}}$, a message m and a signing predicate Υ, such as $\Upsilon(\mathcal{S}) = 1$ ($\mathcal{S}$ satisfies Υ). This algorithm outputs a signature σ (or an error message $\perp$).
- $\mathcal{ABS}$.`verif` – this algorithm takes as input the received signature σ, the signing predicate Υ and the message m. It outputs a bit $b \in \{0, 1\}$, where 1 denotes *accept*; i.e., the verifier successfully checks the signature, with respect to the signing predicate. Otherwise, 0 means *reject*.

3.2 Security Properties of Attribute Based Signatures

First, an ABS scheme has to satisfy the correctness property (Definition 1)

Definition 1. ***Correctness*** – *An ABS scheme is correct, if for all* $(params, MK) \leftarrow \mathcal{ABS}.\texttt{setup}(\xi)$*, all messages* m*, all attribute sets* $\mathcal{S}$*, all signing keys* $sk_{\mathcal{S}} \leftarrow \mathcal{ABS}.\texttt{keygen}(\mathcal{S}, MK)$*, all claiming predicates* Υ *such as* $\Upsilon(\mathcal{S}) = 1$ *and all signatures* $\sigma \leftarrow \mathcal{ABS}.\texttt{sign}(sk_{\mathcal{S}}, m, \Upsilon)$*, we have* $\mathcal{ABS}.\texttt{verif}(\sigma, m, \Upsilon) = 1$.

In addition, based on Maji et al. work [19], we provide the two following formal definitions that capture security properties of ABS schemes.

Definition 2. ***Perfect Privacy*** – *An ABS scheme is perfectly private, if for all* $(params, MK) \leftarrow \mathcal{ABS}.\texttt{setup}(\xi)$*, all attribute sets* $\mathcal{S}_1$*,* $\mathcal{S}_2$*, all secret signing keys* $sk_1 \leftarrow \mathcal{ABS}.\texttt{keygen}(\mathcal{S}_1, MK)$*,* $sk_2 \leftarrow \mathcal{ABS}.\texttt{keygen}(\mathcal{S}_2, MK)$*, all messages* m *and all claiming predicates* Υ *such as* $\Upsilon(\mathcal{S}_1) = \Upsilon(\mathcal{S}_2) = 1$*, the distributions* $\mathcal{ABS}.\texttt{sign}(sk_1, m, \Upsilon)$ *and* $\mathcal{ABS}.\texttt{sign}(sk_2, m, \Upsilon)$ *are indistiguishable.*

In a nutshell, if the perfect privacy property holds, then a signature does not leak which set of attributes or private signing key were originally used.

Definition 3. ***Unforgeability*** – *An ABS scheme is unforgeable if the adversary* $\mathcal{A}$ *can not win the following game:*

- *setup phase: the challenger* $\mathcal{C}$ *chooses a large security parameter* ξ *and runs* `setup`*.* $\mathcal{C}$ *keeps secret the master key* MK *and sends params generated from* $\mathcal{ABS}.\texttt{setup}$ *to the adversary* $\mathcal{A}$*.*

[1] For ease of presentation, we denote the signing key as a monolithic entity, but, in many existing schemes, the signing key consists of separate elements for each single attribute in $\mathcal{S}$.

- *query phase: the adversary $\mathcal{A}$ can perform a polynomially bounded number of queries on $\mathcal{S}$ and (m, Υ) to first the private key generation oracle and second the signing oracle.*
- *forgery phase: $\mathcal{A}$ outputs a signature σ^* on messages m^* with respect to Υ^*.*

The adversary $\mathcal{A}$ wins the game if σ^* is a valid signature on messages m^* for a predicate Υ^*, the couple (m^*, Υ^*) has not been queried to the signing oracle and no attribute set $\mathcal{S}^*$ satisfying Υ^* has been submitted to the private key generation oracle.

This unforgeability property also includes the collusion among users trying to override their rights by combining their complementary attributes to generate a signature satisfying a given predicate Υ. It also covers the non-frameability case when a user also aims to override his rights but on his own.

3.3 Bridging the Gap Between ABS and AC

As far as we know, ABS is still considered as being incompatible with AC purpose of anonymity [21], mostly because ABS assumes that *AAs* are fully trusted authorities as they know the secret keys of each user. Moreover, in case of multiple *AAs*, as needed in AC systems, the issued credentials can be linked by the *AAs* as they are all based on the same public key.

Let us give a simple example to illustrate how ABS could be adapted to AC purpose. A student (acting as user) obtains a certified credential (i.e. student card) by the University (which plays the role of the issuer) over the set of his attributes $\mathcal{S} = \{a_1 := \texttt{Name}; a_2 := \texttt{Bob}, a_3 := \texttt{City}, a_4 := \texttt{Paris}, a_5 := \texttt{Studies}, a_6 := \texttt{InformationSecurity}\}$. The whole set of attributes is committed to a single value using the public key of the user, and it is signed with the private key of the issuer, to generate the resulting credential, denoted by C.

Later, the student can, for example, prove that he is student living in Paris, without revealing his name nor his studies' major. For this purpose, we consider the signing predicate $\Upsilon = (\texttt{Studies} \vee \texttt{Teaches}) \wedge (\texttt{City} \wedge (\texttt{Paris} \vee \texttt{Lille}))$. The user whose attributes satisfy the predicate can use his credential C to successfully extract the appropriate keys relative to the requested attributes a_3, a_4 and a_5. The student thus remains anonymous among the group of students living in Paris, and is able to prove the requested features because the signature of the University over the student's attributes is valid. This example brings first elements for adaptation of ABS to AC purpose, but additional work is necessary.

Additional Requirements For ABS – Let us analyse first the formal security model proposed in the litterature for ABS to satisfy the required AC properties of anonymity and unforgeability (Sect. 2). The first model is proposed by Shahandashti and Safavi-Naini [22], and gives main procedures and basic security properties, such as correctness, unforgeability and signer-attribute privacy. Later, Maji et al. [19] and El Kaafarani et al. [10] introduce and formalize the *perfect privacy* property which requires that a signature reveals neither the identity of the user nor the set of attributes used for the signing procedure. These models

do not entirely match our needs for the design of secure AC scheme. More precisely, the following requirements need to be addressed:

- Traceability of signatures: by essence, an ABS scheme supports the anonymity of the user. As a consequence, there is a need to introduce a new procedure Inspec to remove anonymity, and identify the user originating an ABS signature. To prevent issuers to trace users, this algorithm should be carried out by a tracing authority, equipped with a secret key and referred to as inspector. Such a feature is important in settings where accountability and abuse prevention are required.
- Unlinkability between issuers: in ABS schemes, when a user requests multiple authorities to issue credentials with respect to his attributes, these authorities can link issued credentials to one user through its public key. To satisfy the unlinkability property of AC schemes, a novel ABS issuance procedure has to be designed.
- Replaying sessions: to counteract ABS signature replay attacks, the verifier has to generate for each authentication session, a new message which can might depend on the session data, such as the verifier's identity and the current time.

4 Our New Anonymous Certification Scheme

This section gives a high-level presentation of our new AC scheme based on ABS with an overview of the procedures and algorithms. Then the considered security model with formalized security properties are defined.

4.1 System Model

Our new privacy-preserving attribute based signature $\mathcal{HABS}$ relies on three procedures based on the following seven algorithms that might involve several users (i.e.; signers). The verification and inspection procedures involve only public data. In the following, we denote by $\mathcal{HABS}$ our new AC scheme and by $\mathcal{ABS}$ the ABS basic functions as defined in Sect. 3.

$\mathcal{HABS}$.Setup – this algorithm runs the $\mathcal{ABS}$.`setup` algorithm. It takes as input the security parameter ξ and outputs the global public parameters $params$. This algorithm also derives a pair of public and private keys (pk_{ins}, sk_{ins}) for the tracing authority referred to as the inspector. In the following, public parameters $params$ are assumed to include the public key of the inspector, and all the algorithms have default input $params$.

$\mathcal{HABS}$.KeyGen – this algorithm takes as input the global parameters $params$ and outputs the pair of public and private keys either for users and for the issuer. The public and private keys are noted respectively $(pk_u, sk_u)_j$ for user j and (pk_{is}, sk_{is}) for the issuer.

$\mathcal{HABS}$.Obtain $\leftrightarrow$ $\mathcal{HABS}$.Issue – the credential issuing procedure corresponds to the $\mathcal{ABS}$.`keygen` algorithm. The $\mathcal{HABS}$.Issue algorithm executed by the issuer takes as input the public key of the user pk_u, a set of attributes $\mathcal{S} \subset \mathbb{S}$

(where $\mathcal{S} = \{a_i\}_{i=1}^{N}$, N is the number of attributes and $\mathbb{S}$ is referred to as the attribute universe), the private key of the issuer sk_{is} and the public key of the inspector pk_{ins}. It outputs a signed commitment C over the set of attributes $\mathcal{S}$.

The $\mathcal{HABS}$.Obtain algorithm is executed by the user and corresponds to the collection of the certified credentials from the issuer. This is up to the user to verify the correctness of the received signed commitment over his attributes. In case of verification, the $\mathcal{HABS}$.Obtain algorithm takes as input the signed commitment C, the private key of the user sk_u, the public key of the issuer pk_{is} and eventually the public key of the inspector pk_{ins}. It outputs a bit $b \in \{0, 1\}$.

$\mathcal{HABS}$.Show $\leftrightarrow$ $\mathcal{HABS}$.Verify – the presentation procedure includes the $\mathcal{ABS}$.`sign` and $\mathcal{ABS}$.`verif` algorithms of the ABS signature. This procedure enables the verifier to check that a user has previously obtained credentials on some attributes from a certified (i.e.; authentic) issuer and that he is authorized to access a service with respect to some access policy. As such, the verifier has first to send a random value m (which corresponds to the message m in $\mathcal{ABS}$.`sign`) to the user. To counteract replay attacks (Sect. 3.3), each authentication session is personalized with this random value which can be for instance the verifier's identity concatenated with his clock value. Second, the user signs the received random value, based on his credential. In a nutshell, the user first selects the sub-set of his attributes that satisfies the signing predicate Υ ($\Upsilon(\mathcal{S}') = 1$) and he signs the received value m. Note that an attribute based signature can generally be considered as a non-interactive proof of knowledge based on the Fiat-Shamir heuristic [12]. That is, instead of sending his attributes to the verifier, the user only has to prove he gets from a certified issuer some attributes satisfying the access policy. The user finally sends his signature Σ to the verifier who checks the resulting signature by verifying whether $\mathcal{ABS}.\texttt{verif}(pk_{is}, \Sigma, \Upsilon, m) = 1$.

The $\mathcal{HABS}$.Show algorithm takes as input the randomized message m, a signing predicate Υ, the private key of the user sk_u, the credential C and a sub-set of his attributes $\mathcal{S}'$, such as $\Upsilon(\mathcal{S}') = 1$. This algorithm outputs a signature Σ (or an error message $\perp$).

The $\mathcal{HABS}$.Verify algorithm takes as input the received signature Σ, the public key of the issuer(s) pk_i, the signing predicate Υ and the message m. It outputs a bit $b \in \{0, 1\}$, where 1 denotes *accept* for a successful verification of the signature, and 0 means *reject*.

$\mathcal{HABS}$.Inspec – our scheme supports the inspection procedure performed by a separate and trusted entity referred to as the inspector. It relies on two algorithms namely $\mathcal{HABS}$.`trace` and $\mathcal{HABS}$.`judge` needed to identify the user and give a proof of judgment.

The $\mathcal{HABS}$.`trace` algorithm takes as input the secret key of the inspector sk_{ins}, the issuer(s) public key(s) pk_{is} and the signature Σ. It outputs the index j of the user that has signed the message m with respect to the predicate Υ. It also outputs a proof ϖ.

The $\mathcal{HABS}$.`judge` algorithm takes as input the public key(s) of the issuer(s) pk_{is}, the signature Σ, the user index j and the proof ϖ. It outputs $b \in \{0, 1\}$, where 1 means that ϖ is a valid proof proving that user j originating the signature Σ.

4.2 Security Model

We consider two realistic threat models for proving security and privacy properties of our attribute based credential construction. We first point out the case of *honest but curious* verifiers and issuers. That is, both the verifiers and issuers are honest as they provide proper inputs or outputs, at each step of the protocol, properly performing any calculations expected from them, but they are curious in the sense that they attempt to gain extra information from the protocol. As such, we consider the honest but curious threat model against the privacy requirement with respect to the anonymity and unlinkability properties.

Second, we consider the case of malicious users trying to override their rights. That is, malicious users may attempt to deviate from the protocol or to provide invalid inputs. As such, we consider the malicious user security model mainly against the unforgeability requirement, as presented in Sect. 4.2.1.

4.2.1 Unforgeability

The unforgeability property means that unless the private key of the issuer (resp. the user) is known, it is not possible to forge a valid credential – in case of Issue (resp. the presentation token of the user – in case of Show). This property also covers non frameability and ensures that even if users collude, they cannot frame a user who did not generate a valid presentation token. We thus define unforgeability based on three security games between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$, that simulates the system procedures to interact with the adversary.

Definition 4. ***Unforgeability*** – *We say that $\mathcal{HABS}$ satisfies the unforgeability property, if for every PPT adversary $\mathcal{A}$, there exists a negligible function ϵ such that:*

$$Pr[\boldsymbol{Exp}_{\mathcal{A}}^{unforg}(1^{\xi}) = 1] \leq \epsilon(\xi)$$

where $\boldsymbol{Exp}_{\mathcal{A}}^{unforg}$ is the security experiment against the unforgeability property, with respect to MC-Game, MU-Game and Col-Game introduced hereafter.

On the one hand, *MC-Game*, formally defined hereafter, enables to capture the behaviour of malicious users trying to forge a valid credential. That is, during the first phase, ***Phase I***, the challenger $\mathcal{C}$ runs the $\mathcal{HABS}$.Setup algorithm, gives the public parameters *params* to the adversary $\mathcal{A}$ and proceeds as follows:

- *Keygen*: the challenger $\mathcal{C}$ runs the $\mathcal{HABS}$.KeyGen algorithm, in order to get the key pairs of the issuer, the inspector, and a user (u). The key pair of the user (pk_u, sk_u) is sent to the adversary.
- *Issue-Query*: the adversary $\mathcal{A}$ can request $\mathcal{C}$, as many times as he wants, for getting the credential result C_i (for session i) obtained from the $\mathcal{HABS}$.Issue algorithm applied over the public key pk_u, and a set of attributes $\mathcal{S}_i$.

Then, in ***Phase II***, $\mathcal{C}$ requests the adversary to provide a valid credential over a set of attributes $\mathcal{S}$ (such that $\mathcal{S}$ has not been output during the previous *Issue-Query* phase). Thus, $\mathcal{A}$ runs *ForgeCred* and tries to compute a valid

credential C^*. The adversary $\mathcal{A}$ wins the game if he provides a valid credential. That is, the $\mathcal{HABS}$.Obtain(C^*, sk_u, pk_{is}) algorithm returns an *accept*.

On the other hand, *MU-Game* and *Col-Game* security games enable to capture the behaviour of a malicious user, trying a forgery of the presentation token, either on his own (i.e.; *MU-Game*) or by colluding with other legitimate users (i.e.; *Col-Game*).

First, the *MU-Game* is formally defined as follows: during ***Phase I***, the challenger $\mathcal{C}$ runs the $\mathcal{HABS}$.Setup algorithm, gives the public parameters *params* to the adversary $\mathcal{A}$ and proceeds as follows:

- *Keygen*: the challenger $\mathcal{C}$ runs the $\mathcal{HABS}$.KeyGen algorithm, in order to get the key pairs of the issuer, the inspector, and a user (u). The key pair of the user (pk_u, sk_u) is sent to the adversary.
- *Issue*: the challenger $\mathcal{C}$ runs the $\mathcal{HABS}$.Issue algorithm over the public key pk_u, and a set of attributes $\mathcal{S}$. He sends to the adversary $\mathcal{A}$ the set of attributes $\mathcal{S}$, the credential C and a predicate Υ such that $\Upsilon(\mathcal{S}) = 1$.
- *Show-Query*: the adversary $\mathcal{A}$ can request as many times as he wants the $\mathcal{HABS}$.Show over the predicate Υ, the private key of the user sk_u, a randomly generated message m_i (for session i), and a sub-set of his attributes $\mathcal{S}'$ where $\Upsilon(\mathcal{S}') = 1$. Each request i results in a signature Σ_i.

Then, in ***Phase II***, the challenger $\mathcal{C}$ requests the adversary to provide a valid signature over a randomized message m (such that m has not been output during the previous *Show-Query* phase). Thus, the adversary $\mathcal{A}$ executes *ForgeSig* and tries to compute a valid signature Σ^*.

Second, the *Col-Game*, considered as a sub-case of the *MU-Game*, is formally defined as follows: the challenger $\mathcal{C}$ first runs the $\mathcal{HABS}$.Setup algorithm, gives the public parameters *params* to the adversary $\mathcal{A}$ and proceeds such as:

- *Keygen*: the challenger $\mathcal{C}$ runs the $\mathcal{HABS}$.KeyGen algorithm, in order to get the key pairs of the issuer, the inspector, and two users u_1 and u_2. Both key pairs obtained (pk_{u_1}, sk_{u_1}) and (pk_{u_2}, sk_{u_2}) are sent to the adversary $\mathcal{A}$.
- *Issue*: $\mathcal{C}$ runs the $\mathcal{HABS}$.Issue algorithm over the public key pk_{u_k} $(k \in \{1, 2\})$, and a set of attributes $\mathcal{S}_k$ where $\mathcal{S}_1$ and $\mathcal{S}_2$ are disjoint and non empty. He sends to $\mathcal{A}$ the set of attributes $\mathcal{S}_k$, the obtained credential C_k, a random m and a predicate Υ for which $\Upsilon(\mathcal{S}_k) \neq 1$, but $\Upsilon(\mathcal{S}_1 \cup \mathcal{S}_2) = 1$.
- *Show-Query*: $\mathcal{A}$ can request as many times as he wants the $\mathcal{HABS}$.Show algorithm over the private key sk_{u_k}, the message m, a sub-set of his attributes $\mathcal{S}'_k$ and a predicate Υ_i where $\Upsilon_i(\mathcal{S}'_k) = 1$ to get back a signature Σ_{ik}.

During the second phase, $\mathcal{C}$ requests the adversary to provide a signature over message m and predicate Υ. As such, $\mathcal{A}$ tries to compute a valid signature σ^*.

We say that the AC scheme is unforgeable if the probability that the $\mathcal{HABS}$.Verify procedure in the *MU-Game* and *Col-Game* returns *accept* is negligible.

4.2.2 Privacy

The privacy property covers the anonymity, the issue-show and multi-show requirements, as defined in Sect. 2. In this section, we define three realistic privacy games – *PP-Game*, *MS-Game* and *IS-Game* – based on an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ where $\mathcal{A}$ has only access to public data, except in one game where he has access to credentials. Thus, $\mathcal{A}$ cannot run on his own the $\mathcal{HABS}$.Obtain $\leftrightarrow$ Issue, or $\mathcal{HABS}$.Show $\leftrightarrow$ Verify algorithms, but has to request the results of these algorithms to the challenger $\mathcal{C}$ which is responsible for simulating the system procedures.

Definition 5. ***Privacy*** – *We say that $\mathcal{HABS}$ satisfies the privacy property, if for every PPT adversary $\mathcal{A}$, there exists a negligible function ϵ such that:*

$$Pr[\boldsymbol{Exp}_{\mathcal{A}}^{priv}(1^{\xi}) = 1] = \frac{1}{2} \pm \epsilon(\xi)$$

where $\boldsymbol{Exp}_{\mathcal{A}}^{priv}$ is the security experiment against the privacy property, with respect to PP-Game, MS-Game and IS-Game introduced hereafter.

We formally define our three games as follows: during the first phase, ***Phase I***, $\mathcal{C}$ runs the $\mathcal{HABS}$.Setup algorithm, gives the global public parameters *params* to $\mathcal{A}$ and proceeds as follows:

- *Keygen*: the challenger $\mathcal{C}$ runs the $\mathcal{HABS}$.KeyGen algorithm to get the pair of keys (pk_{is}, sk_{is}) and (pk_{u_j}, sk_{u_j}) (j is for user u_j, $j \in \{1, 2\}$). $\mathcal{C}$ sends the public key of the issuer pk_{is} to the adversary $\mathcal{A}$.
- *Issue*: the challenger $\mathcal{C}$ runs the $\mathcal{HABS}$.Issue algorithm over the public key pk_{u_j} ($j \in \{1, 2\}$), and a set of attributes $\mathcal{S}$ ($\mathcal{S}$=$\mathcal{S}_1$=$\mathcal{S}_2$). $\mathcal{C}$ gets the credential $\mathcal{C}_j$, and only sends the set of attributes $\mathcal{S}_j$ to $\mathcal{A}$.
- *Show-Query*: $\mathcal{A}$ can request $\mathcal{C}$ as many times as he wants, for getting the result of $\mathcal{HABS}$.Show algorithm applied on user u_j (only index j is given to $\mathcal{C}$), with respect to some message m_{jk}, predicate Υ_{jk} and set of attributes $\mathcal{S}'_{jk}$ selected by $\mathcal{A}$ (where $\mathcal{S}'_{jk} \subset \mathcal{S}_j$). $\mathcal{A}$ gets back the presentation token Σ_{jk}.

Afterwards, during ***Phase II***, $\mathcal{A}$ can select one of the following games:

- *PP-Game* – for proving the anonymity property. $\mathcal{A}$ selects $j \in \{1, 2\}$, and generates a message m, a predicate Υ and a subset of attributes $\mathcal{S}_{jk}$ ($k \in \{1, 2\}$) such that $\mathcal{S}_{jk} \subset \mathcal{S}_j$, $\mathcal{S}_{j1} \neq \mathcal{S}_{j2}$, $\Upsilon(\mathcal{S}_{jk}) = 1$, and the triplet $(m, \Upsilon, \mathcal{S}_{jk})$ has never been output during the *Show-Query* phase. $\mathcal{A}$ then sends m, Υ and $\mathcal{S}_{jk}$ to $\mathcal{C}$ which chooses a random bit $b \in \{1, 2\}$, runs $\mathcal{HABS}$.Show over m, Υ, attributes $\mathcal{S}_{jb}$, and sk_{u_j}. $\mathcal{C}$ sends back to $\mathcal{A}$ the obtained presentation token Σ_{jb}. The adversary $\mathcal{A}$ wins the game if he is able to guess the value of b, i.e. the set of attributes $\mathcal{S}_{jb}$ used to derive the presentation token.
- *MS-Game* – for proving the multi-show property. The adversary $\mathcal{A}$ selects $j \in \{1, 2\}$ and generates a message m, a predicate Υ and a subset of attributes $\mathcal{S}'$, such that $\mathcal{S}' \subset \mathcal{S}$, $\Upsilon(\mathcal{S}') = 1$. Note that the triplet $(m, \Upsilon, \mathcal{S}')$ has never been output during the *Show-Query* phase. $\mathcal{A}$ then sends m, Υ, and $\mathcal{S}'$ to

the challenger $\mathcal{C}$ which chooses a random bit $b \in \{1, 2\}$, runs $\mathcal{HABS}$.Show over (m, Υ), the attributes' set $\mathcal{S}'$ and private key sk_{u_b}. $\mathcal{C}$ sends back the presentation token Σ_b to $\mathcal{A}$. The adversary $\mathcal{A}$ wins the game if he is able to guess the value of b, i.e. the user u_b having generated the presentation token.
- *IS-Game* – for proving the issue-show property. The adversary $\mathcal{A}$ generates a message m, a predicate Υ such that $\Upsilon(\mathcal{S}) = 1$, such as the triplet $(m, \Upsilon, \mathcal{S})$ has never been output during the *Show-Query* phase. $\mathcal{A}$ then sends m, Υ and $\mathcal{S}$ to the challenger $\mathcal{C}$ which chooses a random bit $b \in \{1, 2\}$, runs $\mathcal{HABS}$.Show for user u_b over m, Υ, $\mathcal{S}$, and sk_{u_b}. $\mathcal{C}$ sends back to $\mathcal{A}$ Σ_b and credentials C_1 and C_2. The adversary $\mathcal{A}$ wins the game if he is able to guess the value of b, i.e. to which credential C_b the presentation token refers to.

4.2.3 Anonymity Removal

Our $\mathcal{HABS}$ system should fulfill the inspection property meaning that the `trace` algorithm is able to return the right identity of the actual user, for each verified tuple $(m, \Upsilon, \Sigma, pk_{is})$. As the unforgeability Subsect. 4.2.1 already takes care of subcases of anonymity removal, this section focuses only on the *IA-Game* leading an adversary $\mathcal{A}$ to successfully pass the $\mathcal{HABS}$.Show $\leftrightarrow$ Verify procedure, while the inspector is unable to trace the identity of the signature originator.

The *IA-Game* is formally defined as follows: during the first phase, ***Phase I***, the challenger $\mathcal{C}$ runs the $\mathcal{HABS}$.Setup and $\mathcal{HABS}$.KeyGen algorithms to get the key pairs of the issuer, the inspector and a user u_1 indexed as 1. It gives the public parameters *params* and the key pair (pk_{u_1}, sk_{u_1}) to the adversary $\mathcal{A}$ with a predicate Υ, and a random message m.

- *Keygen*: the adversary $\mathcal{A}$ runs the $\mathcal{HABS}$.KeyGen algorithm, in order to get the key pair (pk_{u_1}, sk_{u_1}).
- *Issue*: the adversary $\mathcal{A}$ requests $\mathcal{C}$ for getting the result of $\mathcal{HABS}$.Issue algorithm over the public key pk_{u_1} and a set of attributes $\mathcal{S}$ such as $\Upsilon(\mathcal{S}) = 1$. He gets back the credential C.
- *Show-Query*: $\mathcal{A}$ can request as many times as he wants the $\mathcal{HABS}$.Show over the predicate Υ, the private key sk_{u_1}, the message m, and a sub-set of his attributes $\mathcal{S}_i$ where $\Upsilon(\mathcal{S}_i) = 1$. Each request i results in a signature Σ_i.

Then, during ***Phase II***, $\mathcal{C}$ requests the adversary to provide a valid but untraceable signature over message m and predicate Υ. As such, $\mathcal{A}$ runs *ForgeProof* and the adversary $\mathcal{A}$ tries to compute a signature Σ^*, such as $\mathcal{HABS}$.Verify $(m, \Upsilon, \Sigma, pk_{is}) = 1$ and $\mathcal{HABS}$.`trace`$(\Sigma, sk_{ins}) = \perp$ or k $(k \neq 1)$.

We say that the AC scheme is resistant to inspection abuse attack if the probability that the $\mathcal{HABS}$.Inspec procedure in the *IA-Game* returns *accept* is negligible.

5 Concrete Construction

In this section, we give a concrete attribute based signature scheme that fulfills the features introduced in Sect. 3 and that can be used to design a secure anonymous credential system.

5.1 Mathematical Background

We first introduce the access structure in Sect. 5.1.1. Then, in Sect. 5.1.2, we present the bilinear maps. Finally, we introduce security assumptions.

5.1.1 Access Structures

Definition 6 *(**Access Structure** [1]). Let $\mathcal{P} = \{P_1, P_2, \cdots, P_n\}$ be a set of parties, and a collection $\mathbb{A} \subseteq 2^{\{P_1,P_2,\cdots,P_n\}}$ is called monotone if $\forall B, C \subseteq 2^{\{P_1,P_2,\cdots,P_n\}}$: if $B \in \mathbb{A}$ and $B \subseteq C$ then $C \in \mathbb{A}$. An access structure is a collection $\mathbb{A}$ of non-empty subsets of $\{P_1, P_2, \cdots, P_n\}$; i.e. $\mathbb{A} \subseteq 2^{\{P_1,P_2,\cdots,P_n\}} \setminus \{\emptyset\}$. The sets in $\mathbb{A}$ are called authorized sets, and the sets not in $\mathbb{A}$ are called unauthorized sets.*

We note that in recent ABS schemes, the parties are considered as the attributes.

Definition 7 *(**Linear Secret Sharing Schemes (LSSS)** [1]). A secret sharing scheme Π over a set $\mathcal{P} = \{P_1, P_2, \cdots, P_n\}$ is called linear (over $\mathbb{Z}_p$) if:*

1. *the share for each party forms a vector over $\mathbb{Z}_p$;*
2. *there exists a matrix M with l rows called the sharing generating matrix for Π. For each $i \in [1, l]$, we let the function ρ define the party labeling the row i of the matrix M as $\rho(i)$. When we consider the column vector $\boldsymbol{v} = (v_1, \cdots, v_k)^T$, where $v_1 = s \in \mathbb{Z}_p$ is the secret to be shared, and $v_t \in \mathbb{Z}_p$, where $t \in [2, k]$ are chosen randomly, then $M \cdot \boldsymbol{v}$ is the vector of l shares of s according to Π. The share $\lambda_i = (M \cdot v)_i$ belongs to the party $\rho(i)$.*

Suppose that Π is an LSSS for the access structure $\mathbb{A}$. Let S be an authorized set, such as $S \in \mathbb{A}$, and $I \subseteq \{1, 2, \cdot, l\}$ is defined as $I = \{i : \rho(i) \in S\}$. If $\{\lambda_i\}_{i \in I}$ are valid shares of a secret s according to Π, there exist constants $\{w_i \in \mathbb{Z}_p\}_{i \in I}$, that can be computed in a polynomial time, such as $\sum_{i \in I} \lambda_i w_i = s$ [1].

We note that any monotonic boolean formula can be converted into LSSS representation. Generally, boolean formulas are used to describe the access policy, and equivalent LSSS matrix is used to sign and verify the signature. We must note that the labeled matrix in Definition 7 is also called monotone span program [16].

Definition 8 *(**Monotone Span Programs (MSP)** [16,19]). A Monotone Span Program (MSP) is the tuple $(\mathbb{K}, M, \rho, \boldsymbol{t})$, where $\mathbb{K}$ is a field, M is a $l \times c$ matrix (l is the number of rows and c is the numbers of columns), $\rho : [l] \to [n]$ is the labeling function and $\boldsymbol{t}$ is the target vector. The size of the MSP is the number l of rows.*

As ρ is the function labeling each row i of M to a party $P_{\rho(i)}$, each party can be considered as associated *to one or more rows. For any set of parties $S \subseteq \mathcal{P}$, the sub-matrix consisting of rows associated to parties in S is denoted M_S.*

The span of a matrix M, denoted $span(M)$ is the subspace generated by the rows of M, i.e.; all vectors of the form $\boldsymbol{v} \cdot M$. An MSP is said to compute an access structure $\mathcal{A}$ if:

$$S \in \mathcal{A} \quad iff \quad t \in span(M_S)$$

In other words:

$$\mathcal{A}(S) = 1 \Longleftrightarrow \exists \boldsymbol{v} \in \mathbb{K}^{1\times l} : \boldsymbol{v}M = \boldsymbol{t}$$

5.1.2 Bilinear Maps

Let $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ be three cyclic groups of prime order p. Let g_1, g_2 be generators of respectively $\mathbb{G}_1$ and $\mathbb{G}_2$. A bilinear map $\hat{e}$ is a map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ satisfying the following properties: (i) bilinearity: for all $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$, (ii) non-degeneracy: $\hat{e}(g_1, g_2) \neq 1$ and (iii) there is an efficient algorithm to compute $\hat{e}(g_1, g_2)$ for any $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_1$.

5.1.3 Complexity Assumptions

For our construction, we consider the following complexity assumptions:

- ***q-Diffie Hellman Exponent Problem (q-DHE)*** – Let $\mathbb{G}$ be a group of a prime order p, and g is a generator of $\mathbb{G}$. The q-DHE problem is, given a tuple of elements $(g, g_1, \cdots, g_q, g_{q+2}, \cdots, g_{2q})$, such that $g_i = g^{\alpha^i}$, where $i \in \{1, \cdots, q, q+2, \cdots, 2q\}$ and $\alpha \xleftarrow{R} \mathbb{Z}_p$, there is no efficient probabilistic algorithm $\mathcal{A}_{qDHE}$ that can compute the missing group element $g_{q+1} = g^{\alpha^{q+1}}$.
- ***Computational Diffie Hellman Assumption (CDH)*** – Let $\mathbb{G}$ be a group of a prime order p, and g is a generator of $\mathbb{G}$. The CDH problem is, given the tuple of elements (g, g^a, g^b), where $\{a, b\} \xleftarrow{R} \mathbb{Z}_p$, there is no efficient probabilistic algorithm $\mathcal{A}_{CDH}$ that computes g^{ab}.

5.2 Overview

In this section, we review the procedures and algorithms of $\mathcal{HABS}$. Our proposal is composed of seven algorithms defined as follows:

- SETUP: this algorithm takes as input the security parameter ξ and outputs the public parameters *params*. As presented in Sect. 4.1, we suppose that the public parameters includes the public key of the inspector and are considered as an auxiliary input to all $\mathcal{HABS}$ algorithms
  Global Public Parameters params – the SETUP algorithm first generates an asymmetric bilinear group environment such as $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e})$ where $\hat{e}$ is an asymmetric pairing function such as $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. Random generators $g_1, \{u_i\}_{i\in[1,U]} \in \mathbb{G}_1$ (i.e.; U is the maximum number of attributes supported by the span program) and $g_2 \in \mathbb{G}_2$ are also generated, together with $\alpha \in \mathbb{Z}_p$. Let $h_1 := {g_1}^{\alpha} \in \mathbb{G}_1$ and $h_2 := {g_2}^{-\alpha} \in \mathbb{G}_2$. Let $\mathcal{H}$ be a cryptographic hash function. The global parameters of our system are as follows:

  $$params = \{\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, p, g_1, \{u_i\}_{i\in[1,U]}, g_2, h_1, h_2, \mathcal{H}\}$$

  We note that the secret key of the inspector is $sk_{ins} = \alpha$.

- KeyGen – this algorithm outputs a pair of private and public keys for each participating entity. In our proposal, each entity (i.e.; issuer and user) has a pair of private and public keys. That is, the user has a pair of keys (sk_u, pk_u) where sk_u is randomly chosen in $\mathbb{Z}_p$ and pk_u is the couple $(X_u, Y_u) = (g_1{}^{sk_u}, \hat{e}(g_1, g_2)^{sk_u})$. The issuer has a pair of secret and public keys (sk_{is}, pk_{is}). The issuer secret key sk_{is} is the couple defined as $sk_{is} = (s_{is}, x_{is})$ where s_{is} is randomly chosen in $\mathbb{Z}_p$ and $x_{is} = g_1{}^{s_{is}}$. The issuer public key pk_{is} is the couple $(X_{is}, Y_{is}) = (\hat{e}(g_1, g_2)^{s_{is}}, h_2{}^{s_{is}})$.
- Issue: this algorithm is performed by the issuer in order to issue the credential to the user with respect to a pre-shared set of attributes $\mathcal{S} \subset \mathbb{S}$ ($\mathbb{S}$ is referred to as the attribute universe). The set of attributes $\mathcal{S}$ is defined as follows: $\mathcal{S} = \{a_1, a_2, \cdots, a_N\}$, where N is the number of attributes.
  The Issue algorithm takes as input the public key of the user pk_u, a set of attributes $\mathcal{S}$ and the private key of the issuer sk_{is}. It outputs the credential C defined as $C = (C_1, C_2, C_3, \{C_{4,i}\}_{i\in[1,N]}) = (x_{is} \cdot [X_u{}^{\mathcal{H}(\mathcal{S})^{-1}}] \cdot h_1{}^r, g_1{}^{-r}, g_2{}^r, \{u_i{}^r\}_{i\in[1,N]})$, where $\mathcal{H}(\mathcal{S}) = \mathcal{H}(a_1)\mathcal{H}(a_2)\cdots\mathcal{H}(a_N)$, r is an integer randomly selected by the issuer and $u_i{}^r$ presents the secret key associated to the attribute a_i, where $i \in [1, N]$.
- Obtain: this algorithm is executed by the user. It takes as input the credential C, the public key of the user pk_u, the public key of the issuer pk_{is} and the set of attributes $\mathcal{S}$. The correctness of the obtained credential is given by Eq. 1, as follows:

$$\hat{e}(C_1, g_2) \stackrel{?}{=} X_{is} \cdot \hat{e}(X_u^{\mathcal{H}(\mathcal{S})^{-1}}, g_2) \cdot \hat{e}(h_1, C_3) \tag{1}$$

- Show: this algorithm is performed by the user, in order to authenticate with the verifier. That is, when the user wants to access a service, he sends a request to the verifier. As such, the verifier sends his presentation policy. The presentation policy is given by a randomized message m, a predicate Υ and the set of attributes that have to be revealed. The user has to sign the message m with respect to the predicate Υ satisfying a sub-set of his attributes $\mathcal{S}$. As presented in Sect. 3, the message m should be different for each authentication session.

  In the following, we denote by $\mathcal{S}_R$, the set of attributes revealed to the verifier, and $\mathcal{S}_H$ the set of non-revealed attributes, such as $\mathcal{S} = \mathcal{S}_R \cup \mathcal{S}_H$.

  Let the signing predicate Υ can be represented by an LSSS access structure (M, ρ), i,e; M is an $l \times k$ matrix, and ρ is an injective function that maps each row of the matrix M to an attribute. The Show algorithm takes in input the user secret key sk_u, the credential C, the attribute set $\mathcal{S}$, the message m and the predicate Υ such that $\Upsilon(\mathcal{S}) = 1$. The showing process is as follows:
  1. The user should first blind his credential C in the following way: the user first selects at random an integer $r' \in \mathbb{Z}_p$ and sets $C'_1 = C_1 \cdot h_1{}^{r'} = x_{is} \cdot X_u{}^{\mathcal{H}(\mathcal{S})^{-1}} \cdot h_1{}^r \cdot h_1{}^{r'} = x_{is} \cdot X_u{}^{\mathcal{H}(\mathcal{S})^{-1}} \cdot h_1{}^{r+r'}$, $C'_2 = C_2 \cdot g_1{}^{-r'} = g_1{}^{-(r+r')}$ and $C'_3 = C_3 \cdot g_2{}^{r'} = g_2{}^{r+r'}$.
     Then, the user blinds the secret value associated to each attribute required in the access policy such that: $\forall a_i \in \mathcal{S}, u'_i = u_i{}^r \cdot u_i{}^{r'} = u_i{}^{r+r'}$. Thus,

the new blinded credential C' presents the tuple $(C'_1, C'_2, C'_3, C'_{4,i}) = (x_{is} \cdot {X_u}^{\mathcal{H}(\mathcal{S})^{-1}} \cdot {h_1}^{r+r'}, {g_1}^{-(r+r')}, {g_2}^{r+r'}, {u_i}^{r+r'})$.

2. As the user's attributes $\mathcal{S}$ satisfies Υ, the user can find a vector $\boldsymbol{v} = (v_1, \cdots, v_l)$ that satisfies $\boldsymbol{v}M = (1, 0, \cdots, 0)$ according to Definition 8.
3. For each attribute a_i, where $i \in [1, l]$, the user first computes $\omega_i = {C'_3}^{v_i}$. Then, he calculates $B = \prod_{i=1}^{l} (u'_{\rho(i)})^{v_i}$.
4. Afterwards, the user selects a random r_m and computes the couple $(\sigma_1, \sigma_2) = (C'_1 \cdot B \cdot {g_1}^{r_m m}, {g_1}^{r_m})$.
   We note that the user may not have the secret value of each attribute mentioned in Υ. But, in this case, $v_i = 0$ and thus the value is not needed.
5. Finally, the user computes an accumulator on non-revealed attributes, using his secret key such as $A = {g_2}^{\frac{sk_u \mathcal{H}(\mathcal{S}_H)^{-1}}{r_m}}$. Then, he outputs a presentation token Σ, which mainly includes the signature of the message m with respect to the predicate Υ such that $\Sigma = (\Omega, \sigma_1, \sigma_2, C'_1, C'_2, A, \mathcal{S}_R)$.
   We note that $\Omega = \{\omega_1, \cdots, \omega_l\}$ is the set of committed elements' values of the vector $\boldsymbol{v}$, based on the credential's item C'_3.

– Verify: this algorithm is performed by the verifier. It takes as input the public key of the issuer pk_{is}, the presentation token Σ, the set of revealed attributes $\mathcal{S}_R$, the message m and the signing predicate Υ corresponding to $(M_{l \times k}, \rho)$. It outputs a bit $b \in \{0, 1\}$. The verifier proceeds as follows:
  First, the verifier checks the received set of revealed attributes $\mathcal{S}_R$, and computes an accumulator A_R such as $A_R = {\sigma_2}^{\mathcal{H}(\mathcal{S}_R)^{-1}}$.
  Then, the verifier chooses at random $k-1$ values from $\mathbb{Z}_p$, denoted by $\mu_2, \cdots, \mu_k$ respectively and sets the vector $\boldsymbol{\mu} = (1, \mu_2, \cdots, \mu_k)$.
  Consequently, the verifier calculates $\tau_i = \sum_{j=1}^{k} \mu_j M_{i,j}$ where $M_{i,j}$ is an element of the matrix M. Finally, the verifier checks the correctness of the received presentation token (Eq. 2):

$$\hat{e}(\sigma_1, g_2) \stackrel{?}{=} X_{is} \hat{e}(A_R, A) \hat{e}(C'_2, h_2) \prod_{i=1}^{l} \hat{e}(u_{\rho(i)} {h_1}^{\tau_i}, \omega_i) \hat{e}(\sigma_2, {g_2}^{m}) \tag{2}$$

– Inspec: this algorithm is performed by the inspector, the authority in possession of the secret sk_{ins}. The inspector can decrypt Elgamal ciphertext (C'_1, C'_2) to retrieve $\varpi^* = C'_1 \cdot {C'_2}^{\alpha}$. Then, the inspector uses the issuer table in order to retrieve an entry $({u_j}^*, pk_j, {Y_{u_j}}^{\mathcal{H}(\mathcal{S})^{-1}})$, such that $\hat{e}(\varpi^*, g_2) \cdot [X_{is}]^{-1} = \hat{e}({X_u}^{\mathcal{H}(\mathcal{S})^{-1}}, g_2)$. The proof of validity of such an inspection procedure is done by proving that the decryption is correctly done, using the knowledge of sk_{ins} (Eq. 3).

$$\hat{e}(\varpi^*, g_2) \cdot {X_{is}}^{-1} \stackrel{?}{=} \hat{e}({X_u}^{-\mathcal{H}(\mathcal{S})}, g_2) \tag{3}$$

6 Security Analysis

In this section, we first prove that $\mathcal{HABS}$ provides the security requirements defined in Sect. 4.2. Then, we discuss an extension to support multiple issuers.

6.1 Security of the Main Scheme

The security of our main scheme $\mathcal{HABS}$ relies on the following Theorems:

Theorem 1. ***Correctness*** *– $\mathcal{HABS}$ is correct if for all* $(params) \leftarrow$ SETUP(ξ)*, all pair of public and private keys* $\{(pk_{is}, sk_{is}), (pk_u, sk_u)\} \leftarrow$ KEYGEN$(params)$*, all attribute sets* $\mathcal{S}$*, all credentials* $C \leftarrow$ ISSUE $(\mathcal{S}, sk_{is}, pk_u)$*, all claiming predicates* Υ *such as* $\Upsilon(\mathcal{S}) = 1$*, all presentation tokens* $\Sigma \leftarrow$ SHOW (C, sk_u, m, Υ) *and all proofs* $\varpi \leftarrow \texttt{trace}(sk_{ins}, \sigma, pk_{is})$*, we have* OBTAIN $(C, sk_u, pk_{is}, \mathcal{S}) = 1$*,* VERIF $(\Sigma, m, \Upsilon, pk_{is}) = 1$ *and* $\texttt{judge}(\varpi) = 1$*.*

Theorem 2. ***Unforgeability*** *– $\mathcal{HABS}$ satisfies the unforgeability requirement, under the CDH, q-DHE and DLP assumptions.*

Theorem 3. ***Privacy*** *– $\mathcal{HABS}$ achieves the privacy requirement, with respect to the anonymity and unlinkability properties.*

Theorem 4. ***Anonymity Removal*** *– Our attribute based credential system $\mathcal{HABS}$ achieves the inspection feature, with respect to IA-Game.*

For detailed security proofs, please refer to http://www-public.tem-tsp.eu/~laurenm/ABS-AC/securityanalysis.pdf

6.2 Homomorphism to Support Multiple Issuers

As presented in Sect. 3.3, when a user requests multiple authorities to issue credentials with respect to his attributes, the different sessions are linked through the user's public key. To satisfy the unlinkability property of AC schemes between several issuance sessions, a novel ABS issuance procedure has to be designed, leading us to extend our proposal to support pseudonym systems and public key masking during the issuance procedure, presented hereafter. Also our construction is demonstrated to support an homomorphism property helpful for defining a new $\mathcal{HABS}$.agg algorithm, and a modified $\mathcal{HABS}$.VERIFY algorithm.

ASSUMPTIONS – Extra assumptions are requested for the support of multiple issuers: (i) all the issuing authorities AA_j share the same public parameters $params$, but have distinct key pairs (sk_{is_j}, pk_{is_j}), (ii) the public parameters $params$ include the secrets u_i relative to the attributes that might be certified by diverse issuers, (iii) the user is provided with one pseudonym nym_j per authority, and enables the user to authenticate to the issuers with different identities. For consistency among obtained credentials (i.e. C_1), each pseudonym nym_j should rely on the private key of the user and the related issuing authority AA_j and the $\mathcal{HABS}$.ISSUE should be extended with works of Chase and Chow [7] and Chase *et al.* [8] for masking the public key of the user.

HOMOMORPHISM CONSTRUCTION – For simplicity reasons, the reasoning next is limited to two issuers IS_i and IS_j, but it can be easily extended to n (different) issuer(s), where $n \geq 2$. Let us then assume that a user receives two signed sets of attributes from two different attribute authorities IS_i and IS_j.

The user receives $C_i = \mathcal{HABS}.\text{Issue}\ ({sk_{is}}^{(i)}, \mathcal{S}_i)$ and $C_j = \mathcal{HABS}.\text{Issue}\ ({sk_{is}}^{(j)}, \mathcal{S}_j)$, from IS_i and IS_j, respectively. The sets of attributes are represented by $\mathcal{S}_i = \{a_{i,1}, \cdots, a_{i,n_i}\}$ and $\mathcal{S}_j = \{a_{j,1}, \cdots, a_{j,n_j}\}$, where n_k is the number of attributes in the set $\mathcal{S}_k$ and $k \in \{i, j\}$.

The idea is to aggregate credentials C_i and C_j to form a new C_R covering the attributes $\mathcal{S} = \mathcal{S}_i \cup \mathcal{S}_j$. We define the agg algorithm as follows:

agg – this algorithm takes as input two credentials C_i and C_j corresponding to the sets of attributes $\mathcal{S}_i$ and $\mathcal{S}_j$ respectively, and the public keys of issuers ${pk_{is}}^i$ and ${pk_{is}}^j$. It outputs a resulting signed commitment C_R, where C_R is a signature over the union of the two sets of attributes $\mathcal{S}_i$ and $\mathcal{S}_j$. We note that the agg algorithm has to fulfill the *correctness* and *homomorphism* properties.

Recall that the credential C_k, obtained from the issuer IS_k, is denoted by $C_k = (C_1, C_2, C_3, \{C_{l,4}\}_{l\in[1,n_l]})^{(k)} = (x_{is_k}[{X_u}^{\mathcal{H}(\mathcal{S}_k)^{-1}}]{h_1}^{r_k}, {g_1}^{-r_k}, {g_2}^{r_k}, \{{u_l}^{r_k}\}_{l\in[1,N]})$, where $k \in \{i, j\}$ and n_l is the number of certified attributes by the issuer IS_k.

Let us define the following theorem defining the aggregation algorithm:

Theorem 5. *Let us consider the algorithms* $\mathcal{HABS}.\text{Issue}$, $\mathcal{HABS}.\text{Obtain}$, $\mathcal{HABS}.\text{Show}$ *and* $\mathcal{HABS}.\text{Verify}$ *defined in Sect. 5.2. Let* $\mathcal{HABS}.\texttt{agg}$ *be the aggregation algorithm such as:*

$$\texttt{agg}(C^{(i)}, C^{(j)}, \mathcal{S}_i, \mathcal{S}_j, {pk_{is}}^i, {pk_{is}}^j) = \mathcal{HABS}.\text{Issue}(pk_u, \mathcal{S}_i \cup \mathcal{S}_j), a.sk_{is_i} + b.sk_{is_j}) \tag{4}$$

where a and b are two integers that might be computed by the user based on the union set $\mathcal{S}_i \cup \mathcal{S}_j$.

That theorem and homomorphism property come directly from the following Lemma 1 which expresses $\mathcal{H}(S_i \cup S_j)$ based on $\mathcal{H}(S_i)$ and $\mathcal{H}(S_j)$ in order to write $C_{\{1,S_i\cup S_j\}}$ with respect to ${C_1}^{(i)}$ and ${C_1}^{(j)}$.

Lemma 1. *Given the hash function $\mathcal{H}$ and for every sets of attributes S_i and S_j, there exist two integers a and b, such that $\mathcal{H}(S_i \cup S_j)^{-1} = a\mathcal{H}(S_i)^{-1} + b\mathcal{H}(S_j)^{-1}$.*

Proof. Referring to the Bezout's lemma, the gcd satisfies the following property:

$$gcd(\mathcal{H}(S_i), \mathcal{H}(S_j)) = b\mathcal{H}(S_i) + a\mathcal{H}(S_j) \tag{5}$$

where a and b are two non zero integers (a and b are called Bezout coefficients). In addition, the gcd and lcm satisfy Eq. 6 such that

$$gcd(\mathcal{H}(S_i), \mathcal{H}(S_j)) * lcm(\mathcal{H}(S_i), \mathcal{H}(S_j)) = \mathcal{H}(S_i)\mathcal{H}(S_j) \tag{6}$$

As such, using Eq. 6, we have:

$$lcm(\mathcal{H}(S_i), \mathcal{H}(S_j))^{-1} = \frac{gcd(\mathcal{H}(S_i), \mathcal{H}(S_j))}{\mathcal{H}(S_i)\mathcal{H}(S_j)} = \frac{b\mathcal{H}(S_i) + a\mathcal{H}(S_j)}{\mathcal{H}(S_i)\mathcal{H}(S_j)} = b\mathcal{H}(S_j)^{-1} + a\mathcal{H}(S_i)^{-1} \tag{7}$$

On the other side, we write $\mathcal{H}(S_i \cup S_j)$ as follows:

$$\mathcal{H}(S_i \cup S_j) = \prod_{a_k \in S_i \cup S_j} \mathcal{H}(a_k) = lcm(\prod_{a_k \in S_i} \mathcal{H}(a_k), \prod_{a_k \in S_j} \mathcal{H}(a_k)) = lcm(\mathcal{H}(S_i), \mathcal{H}(S_j)) \tag{8}$$

6.3 Proof of Homomorphism

In order to prove the homomorphism property with respect to the union operator, we first express $[{C_1}^{(i)}]^a \cdot [{C_1}^{(j)}]^b$, denoted by RS, as a function of $\mathcal{S}_i \cup \mathcal{S}_j$, sk_{is_i} and sk_{is_j}, as follows:

$$\begin{aligned} RS &= [x_{is_i} \cdot [{X_u}^{\mathcal{H}(\mathcal{S}_i)^{-1}}] \cdot {h_1}^{r_i}]^a \cdot [x_{is_j} \cdot [{X_u}^{\mathcal{H}(\mathcal{S}_j)^{-1}}] \cdot {h_1}^{r_j}]^b \\ &= {g_1}^{a.s_{is_i}+b.s_{is_j}} \cdot [{X_u}^{a\mathcal{H}(\mathcal{S}_i)^{-1}+b\mathcal{H}(\mathcal{S}_j)^{-1}}] \cdot {h_1}^{a.r_i+b.r_j} \\ &= {g_1}^{a.s_{is_i}+b.s_{is_j}} \cdot [{X_u}^{\mathcal{H}(\mathcal{S}_i \cup \mathcal{S}_j)^{-1}}] \cdot {h_1}^{a.r_i+b.r_j} \end{aligned}$$

Similarly, we can write the elements of the resulting credential C_R, such that $C_R = (C_{1,\mathcal{S}_i\cup\mathcal{S}_j}, C_{2,\mathcal{S}_i\cup\mathcal{S}_j}, C_{3,\mathcal{S}_i\cup\mathcal{S}_j}, \{C_{l,4,\mathcal{S}_i\cup\mathcal{S}_j}\}_{l\in[1,N]})$, where $C_{1,\mathcal{S}_i\cup\mathcal{S}_j} = [{C_1}^{(i)}]^a \cdot [{C_1}^{(j)}]^b = {x_{is_i}}^a \cdot {x_{is_j}}^b \cdot [{X_u}^{\mathcal{H}(\mathcal{S}_i\cup\mathcal{S}_j)}] \cdot {h_1}^{a.r_i+b.r_j}$, $C_{2,\mathcal{S}_i\cup\mathcal{S}_j} = [{C_2}^{(i)}]^a \cdot [{C_2}^{(j)}]^b = {g_1}^{-(a.r_i+b.r_j)}$ $C_{3,\mathcal{S}_i\cup\mathcal{S}_j} = [{C_3}^{(i)}]^a \cdot [{C_3}^{(j)}]^b = {g_2}^{a.r_i+b.r_j}$ and $\{C_{l,4,\mathcal{S}_i\cup\mathcal{S}_j}\}_{l\in[1,N]} = \{{u_l}^{a.r_i+b.r_j}\}_{l\in[1,N]}$, (i.e.; N is the maximum number of attributes).

The form of the aggregated credential $C_{1,\mathcal{S}_i\cup\mathcal{S}_j}, C_{2,\mathcal{S}_i\cup\mathcal{S}_j}, C_{3,\mathcal{S}_i\cup\mathcal{S}_j}$, $\{C_{l,4,\mathcal{S}_i\cup\mathcal{S}_j}\}_{l\in[1,N]}$ is similar to the individual credentials like C_i, thus leading to the aggregated presentation token Σ_R by applying exactly the same $\mathcal{HABS}$Show algorithm. The obtained Σ_R is as follows: $\Sigma_R = (\Omega_R, \sigma_{1,R}, \sigma_{2,R}, C'_{1,R}, C'_{2,R}, A, \mathcal{S}_R)$.

6.4 Proof of Correctness

We show how the verifier can rely on the aggregated presentation token Σ_R, to authenticate the user (u), with respect to his access policy Υ, such as $\Upsilon(\mathcal{S}_i \cup \mathcal{S}_j) = 1$, where $\mathcal{S}_k$ presents the set of attributes certified by the issuer IS_k, $k \in \{i, j\}$. Using the properties of the pairing function $\hat{e}$, we can easily prove the correctness of Eq. 9:

$$\hat{e}(\sigma_{1,R}, g_2) \stackrel{?}{=} {X_{is_i}}^a {X_{is_j}}^b \hat{e}(A_R, A)\hat{e}(C'_{2,R}, h_2) \prod_{i=1}^{l} \hat{e}(u_{\rho(i)} {h_1}^{\tau_i}, \omega_i)\hat{e}(\sigma_{2,R}, {g_2}^m) \qquad (9)$$

where a and b are two integers as defined in Lemma 1.

By equivalence to Eq. 2, we can consider that $D = a.r_i + b.r_j + r'$ presents the quantity $R = r + r'$. Thus, for proving the correctness of Eq. 9, let us denote by Ⓢ the quantity $\hat{e}(\sigma_{1,R}, g_2)$:

$$\begin{aligned} Ⓢ &= \hat{e}({x_{is_i}}^a {x_{is_j}}^b \cdot {X_u}^{\mathcal{H}(\mathcal{S}_i\cup\mathcal{S}_j)} \cdot {h_1}^D \cdot \prod_{i=1}^{l} (u_{\rho(i)})^{Dv_i} \cdot {g_1}^{r_m m}, g_2) \\ &= \hat{e}(x_{is_i}, g_2)^a \cdot \hat{e}(x_{is_j}, g_2)^b \cdot \hat{e}({X_u}^{\mathcal{H}(\mathcal{S}_i\cup\mathcal{S}_j)}, g_2) \cdot \hat{e}({h_1}^D, g_2) \cdot \hat{e}({g_1}^{r_m m}, g_2) \cdot \hat{e}(\prod_{i=1}^{l} {u_{\rho(i)}}^{Dv_i}, g_2) \\ &= {X_{is_i}}^a \cdot {X_{is_j}}^b \cdot \hat{e}({g_1}^{\mathcal{H}(\mathcal{S}_R)^{-1}}, [{g_2}^{sk_u}]^{\mathcal{H}(\mathcal{S}_H)^{-1}}) \cdot \hat{e}(C'_{2,R}, h_2) \cdot \hat{e}(\sigma_2, {g_2}^m) \cdot \prod_{i=1}^{l} \hat{e}(u_{\rho(i)}, \omega_i) \\ &= {X_{is_i}}^a {X_{is_j}}^b \hat{e}(A_R, A)\hat{e}(C'_{2,R}, h_2) \prod_{i=1}^{l} \hat{e}(u_{\rho(i)} {h_1}^{\tau_i}, \omega_i)\hat{e}(\sigma_{2,R}, {g_2}^m) \end{aligned}$$

Table 1. Comparisons between $\mathcal{HABS}$ and the related works

Scheme	Keys (N attributes per credential)						Issuance procedure		
	Groups		*params*	Credential size			User	Issuer	Bw
[15]	$\mathbb{Z}_n$: RSA: 1024		$\mathcal{O}(N)$	$\mathcal{O}(1)$	$\simeq 2600 + 1024$		$\mathcal{O}(N)$: $\mathbb{Z}_n$		
[20]	$p = 1024 : q = 128$		$\mathcal{O}(N)$	$\mathcal{O}(1)$	$3 \cdot \lvert\mathbb{G}_q\rvert + 128$		$\mathcal{O}(N)$: $\mathbb{G}_q$		$\mathcal{O}(1)$
[6]	$\mathbb{F}_p : \lvert\mathbb{G}_1\rvert \simeq 170$		$\mathcal{O}(1)$	$\mathcal{O}(N)$	$(2N + 2) \cdot (\lvert\mathbb{G}_1\rvert + \lvert\mathbb{Z}_p\rvert)$		$\mathcal{O}(1)$	$\mathcal{O}(N) : \mathbb{G}_1$	
$\mathcal{HABS}$	$\mathbb{F}_p : \lvert\mathbb{G}_1\rvert \simeq 170$		$\mathcal{O}(N)$	$\mathcal{O}(N)$	$(N + 3) \cdot \lvert\mathbb{G}_1\rvert$		$\mathcal{O}(1)$	$\mathcal{O}(N) : \mathbb{G}_1$	
	Presentation Procedure								
	User	Verifier	Bw.	User	Verifier	Bw.	User	Verifier	Bw.
	single-use			*single-use*		*l-out-of-N attributes*	*K-use*	*N attributes*	
[15]	$\mathcal{O}(N)$: $\mathbb{Z}_n$		$\mathcal{O}(1)$	$\mathcal{O}(N)$		$\mathcal{O}(N - l)$	$\mathcal{O}(N)$: $\mathbb{Z}_n$		$\mathcal{O}(1)$
[20]	$\mathcal{O}(N)$		$\mathcal{O}(N)$	$\mathcal{O}(l)$		$\mathcal{O}(N)$	$\mathcal{O}(KN)$: $\mathbb{Z}_q$		$\mathcal{O}(KN)$
[6]	$\mathcal{O}(1)$	$2 \cdot \mathcal{O}(N)$	$\mathcal{O}(N)$	$\mathcal{O}(1)$	$2 \cdot \mathcal{O}(N)$	$\mathcal{O}(l)$	$\mathcal{O}(N)$	$2 \cdot \mathcal{O}(N)$	$\mathcal{O}(N)$
$\mathcal{HABS}$	$\mathcal{O}(k)^*$			$\mathcal{O}(k)^*$			$\mathcal{O}(k)^*$		

This proves the correctness of our $\mathcal{HABS}$.Verify, while considering a multi-issuers setting according to the `agg` algorithm.

7 Comparison

In this section, we give a quantitative comparison between related works and our anonymous credential system based on attribute based signatures $\mathcal{HABS}$. That is, we give in Table 1 several elements of comparison between our construction and most closely related anonymous credential systems, with respect to processing and communication overhead.

The first column underlines the algebraic structure for each AC system. It may be an RSA environment [15], $\mathbb{Z}_n$ with a subgroup of order q [20], or bilinear groups $\hat{e}(\mathbb{G}_1, \mathbb{G}_2)$ over a base field $\mathbb{F}_p$ [6].

We denote by N the maximum number of attributes issued by an authority into a single credential. The bandwidth, for issuing and showing protocols, presents the exchanged quantity of data during protocols' running.

The credential size presents the size of public keys or a certificate. The memory consumption for credentials is given with asymptotic complexity and some concrete size in bits. Table 1 also details the processing complexity at the issuer, user and verifier sides, while considering the number of operations in the underlying algebraic structures. As presented before, Table 1 shows that our $\mathcal{HABS}$ is a direct signature, and thus the issuance procedure is rather interesting, compared to IBM Identity Mixer [15] and U-Prove [20] solutions. The [6] construction presents also a direct sanitizable signature applications for anonymous credential systems. However, $\mathcal{HABS}$ presents an interesting overhead, for the showing protocol, compared to existing solutions. That is, the computation and communication overhead depends only on attributes required for satisfying the access policy of the verifier, referred to as k in Table 1, whereas we denote by K the set

of attributes that have to be disclosed with respect to presentation policy of the verifier. In addition, our attribute-based construction $\mathcal{HABS}$ bring multiple-use credentials, likely as [6,15] with an interesting processing overhead, compared to the UProve's technology which is a single-use credentials' solution.

8 Conclusion

In this paper, we proposed a new way to design anonymous credential systems, based on the use of attribute based signatures. Our anonymous certification system $\mathcal{HABS}$ enables a user to anonymously authenticate with a verifier, while providing only required information for the service provider, with respect to its presentation policy. Indeed, $\mathcal{HABS}$ supports a flexible selective disclosure mechanism with no-extra processing cost, which is directly inherited from the expressiveness of attribute based signatures for defining access policies.

Additionally, our proposal is deliberately designed to ensure unlinkability between the different sessions while preserving the anonymity of the user. An extension of $\mathcal{HABS}$ is also detailed to preserve users' privacy with ensuring the unlinkability between multiple issuers. Finally, a quantitative comparison of $\mathcal{HABS}$ with most closely-related technologies shows the interesting processing and communication cost of our construction, especially due to the application of direct attribute based signatures for the issuing protocol.

References

1. Beimel, A.: Secret sharing and key distribution. In: Research Thesis (1996)
2. Brands, S.A.: Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy. MIT Press, Cambridge (2000)
3. Camenisch, J., Krenn, S., Lehmann, A., Mikkelsen, G.L., Neven, G., Pederson, M.O.: Scientific comparison of abc protocols: Part i - formal treatment of privacy-enhancing credential systems (2014)
4. Camenisch, J.L., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 93–118. Springer, Heidelberg (2001)
5. Camenisch, J.L., Lysyanskaya, A.: A signature scheme with efficient protocols. In: Cimato, S., Galdi, C., Persiano, G. (eds.) SCN 2002. LNCS, vol. 2576, pp. 268–289. Springer, Heidelberg (2003)
6. Canard, S., Lescuyer, R.: Protecting privacy by sanitizing personal data: a new approach to anonymous credentials. In: ASIA CCS 2013 (2013)
7. Chase, M., Chow, S.S.M.: Improving privacy and security in multi-authority attribute-based encryption. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, pp. 121–130 (2009)
8. Chase, M., Kohlweiss, M., Meiklejohn, S., Lysyanskaya, A.: Malleable signatures: complex unary transformations and delegatable anonymous credentials (2013)
9. Chaum, D.: Blind signatures for untraceable payment. In: Advances in Cryptology: Proceedings of Crypto 1982 (1982)

10. El Kaafarani, A., Ghadafi, E., Khader, D.: Decentralized traceable attribute-based signatures. In: Benaloh, J. (ed.) CT-RSA 2014. LNCS, vol. 8366, pp. 327–348. Springer, Heidelberg (2014)
11. Europe, C.: Proposal for a regulation of the european parliament and of the council on the protection of individuals with regard to the processing of personal data and on the free movement of such data. In: General Data Protection Regulation, January 2016
12. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987)
13. Herranz, J., Laguillaumie, F., Libert, B., Ràfols, C.: Short attribute-based signatures for threshold predicates. In: Dunkelman, O. (ed.) CT-RSA 2012. LNCS, vol. 7178, pp. 51–67. Springer, Heidelberg (2012)
14. House, W.: Enhancing online choice, efficiency, security, and privacy. In: National Strategy for Trusted Identities in Cyberspace, April 2011
15. IBM: Ibm identity mixer, idemix (2012)
16. Karchmer, M., Wigderson, A.: On span programs. In: Proceedings of the 8th IEEE Structure in Complexity Theory (1993)
17. Langheinrich, M.: Privacy by design - principles of privacy-aware ubiquitous systems. In: Abowd, G.D., et al. (eds.) UbiComp 2001. LNCS, vol. 2201, pp. 273–291. Springer, Heidelberg (2001)
18. Li, J., Au, M.H., Susilo, W., Xie, D., Ren, K.: Attribute-based signature and its applications. In: ASIACCS 2010 (2010)
19. Maji, H.K., Prabhakaran, M., Rosulek, M.: Attribute-based signatures. Cryptology ePrint Archive, Report 2010/595 (2010)
20. Microsoft.: U-prove community technology (2013)
21. Okamoto, T., Takashima, K.: Efficient attribute-based signatures for non-monotone predicates in the standard model. In: PKC 2011 (2011)
22. Shahandashti, S.F., Safavi-Naini, R.: Threshold attribute-based signatures and their application to anonymous credential systems. In: Preneel, B. (ed.) AFRICACRYPT 2009. LNCS, vol. 5580, pp. 198–216. Springer, Heidelberg (2009)
23. Zhang, Y., Feng, D.: Efficient attribute proofs in anonymous credential using attribute-based cryptography. In: Chim, T.W., Yuen, T.H. (eds.) ICICS 2012. LNCS, vol. 7618, pp. 408–415. Springer, Heidelberg (2012)