Making BBS Anonymous Credentials eIDAS 2.0 Compliant

Nicolas Desmoulins[1], Antoine Dumanois[1], Seyni Kane[1,2], and Jacques Traoré[1]

[1] Orange Innovation, Applied Crypto Group, France.
{nicolas.desmoulins, antoine.dumanois, seyni.kane, jacques.traore}@orange.com
[2] SAMOVAR, Télécom SudParis, Institut Polytechnique de Paris, France.

Abstract. eIDAS 2.0 (electronic IDentification, Authentication and trust Services) is a very ambitious regulation aimed at equipping European citizens with a personal digital identity wallet (EU Digital Identity Wallet) on a mobile phone that not only needs to achieve a high level of security, but also needs to be available as soon as possible for a large number of citizens and respect their privacy (as per GDPR - General Data Protection Regulation).
In this paper, we introduce the foundations of a digital identity wallet solution that could help move closer to this objective by leveraging the proven anonymous credentials system BBS (Eurocrypt 2023), also known as BBS+, but modifying it to avoid the limitations that have hindered its widespread adoption, especially in certified infrastructures requiring trusted hardware implementation.
In particular, the solution we propose, which we call BBS#, does not rely, contrary to BBS/BBS +, on bilinear maps and pairing-friendly curves (which are not supported by existing hardware) and only depends on the hardware implementation of well-known digital signature schemes such as ECDSA (ISO/IEC 14888-3) or ECSDSA (also known as ECSchnorr, ISO/IEC 14888-3) using classical elliptic curves. More precisely, BBS# can be rolled out without requiring any change in existing hardware or the algorithms that hardware supports.
BBS#, which is proven secure in the random oracle model, retains the well-known security property (unforgeability of the credentials under the (gap) q-SDH assumption) and anonymity properties (multi-show full unlinkability and statistical anonymity of presentation proofs) of BBS/BBS+.
By implementing BBS# on several smartphones using different secure execution environments, we show that it is possible to achieve eIDAS 2.0 transactions which are not only efficient (around 70 ms on Android StrongBox), secure and certifiable at the highest level but also provide strong (optimal) privacy protection for all European ID Wallet users.

1 Introduction

The so-called eIDAS 2.0 European regulation adopted May 20th, 2024, is a very ambitious one, aiming at changing the digital (and physical) life of European

citizens and corporations alike by providing them a personal digital identity wallet on a mobile phone to perform transactions on their behalf, not only for eGov services, but also for any daily transaction, including very critical ones, like payments. In order to achieve these ambitious goals, these wallets and the associated architectural framework need to simultaneously ensure security (of the digital credentials issued to users) and privacy (of the usage of these credentials) but also reach (the ability to work for as many users and as many technical environments as possible) and a proper user experience (UX). Given the impacts of the technical choices that will be made on hundreds of millions of individuals, intense discussions continue on the best solutions to deploy to comply with these requirements. Although the security aspect has gathered most of the focus of initial talks by designated experts, privacy and data protection is only recently emerging as a major aspect that still needs to be properly tackled, the goal obviously being to add privacy without relinquishing on any of the other aspects.

To solve the privacy issues raised by the EU Digital Identity Wallet (EUDI Wallet), renowned cryptographers have proposed the use of anonymous credentials[1]. Introduced by David Chaum [14], anonymous credentials systems allow users to obtain a credential from an issuer and then, later, prove possession of this credential, in an unlinkable way, without revealing any additional information. This primitive has attracted a lot of interest as it complies with data minimization principles that consist in preventing the disclosure of irrelevant and unnecessary information. Typically, an anonymous credentials system is expected to enable users to reveal a subset of the attributes associated with their credentials while keeping the remaining ones hidden (*selective disclosure*)[2]. For example, they can prove that they have a driving license, so that they can access sites reserved for adults, without having to reveal their identity or date of birth.

This handset of cryptographers specifically recommended to use the BBS/BBS+ [34] family of anonymous credentials, which are efficient, mathematically proven secure, and are currently the object of a standardization effort [25].

However, the European Commission did not consider this solution[3] mainly because BBS/BBS+ uses bilinear maps and pairing-friendly curves (which are not supported by trusted phone hardware) but also because BBS/BBS+ does not use SOG-IS[4] sanctioned protocols for the implementation of the holder binding feature. This feature states that only the legitimate holder of a credential shall be able to perform transactions with that credential. In practice, this is achieved by binding that credential to a private key stored in a trusted hardware (or Secure Element) of the credential holder's mobile device and making presentation of such a credential impossible without that private key.

[1] https://github.com/eu-digital-identity-wallet/eudi-doc-architecture-and-reference-framework/discussions/211#discussioncomment-9882388

[2] Current solutions such as ISO mDL (ISO/IEC 18013-5) provide selective disclosure but all credentials presentations remain traceable by colluding issuers and verifiers. Therefore, they do not adequately protect users' privacy.

[3] https://github.com/eu-digital-identity-wallet/eudi-doc-architecture-and-reference-framework/discussions/211#discussioncomment-9882388

[4] https://www.sogis.eu/

1.1 Related Work on Pairing-Free Anonymous Credentials

One of the most prevalent pairing-free anonymous credentials systems is Microsoft's U-Prove [29] which is based on a blind signature scheme due to Brands [9]. It is quite efficient, as it works in prime-order groups, and supports the selective disclosure of attributes. Unfortunately, U-Prove does not provide multi-show unlinkability (multiple presentations of the same credential are linkable). Besides, Benhamouda et al. [7] have shown that U-Prove issuance protocol is vulnerable to a parallel attack (ROS attack). Specifically, a user simultaneously running a large number (l) of blind issuance sessions with the issuer would be able, after these l sessions, to forge an additional U-Prove credential for the same set of attributes (and thus fraudulently obtain $l+1$ different U-Prove credentials instead of just l). Very recently, Orrù et al. [28] provided a variant of the U-Prove issuance protocol that is secure in a concurrent setting.

Baldimsti and Lysyanskaya have proposed a slightly less efficient pairing-free anonymous attribute-based credentials system [2]. However, similarly to U-Prove, this system is one-show (i.e. credential presentations are linkable if a credential is used more than once).

IBM's Identity Mixer, commonly known as Idemix [27], is built on the Camenisch-Lysyanskaya (CL) signature scheme [10]. Unlike the above cited credentials systems, Idemix provides multi-show unlinkability but at the cost of a less efficient proof of possession. Indeed, the used CL signatures are based on the Strong RSA assumption [3]. This implies large RSA parameters making Idemix unsuitable for current (constrained) Secure Elements.

Chase et al. [13] have opted for the use of symmetric key primitives, instead of digital signatures, to achieve better performance. More precisely, they used algebraic Message Authentication Codes (MACs), which relies on group operations rather than block ciphers or hash functions, as the main building block of their credentials systems. Their two proposals, denoted $\mathrm{MAC}_{\mathrm{GGM}}$ and $\mathrm{MAC}_{\mathrm{DDH}}$, assume that the credential issuer and the verifier share a secret key. In such a setting, the anonymous credentials system is referred to as Keyed-Verification Anonymous Credentials (KVAC). The main drawback of their pairing-free KVAC systems is that they are tailored to specific settings in which the issuer also acts as verifier, as in the case of e-government or public transportation. They are not suited to the more general setting, envisioned for the EUDI Wallet, in which the issuer and the verifier are two distinct entities that do not necessarily share a secret key.

In [4], Barki et al. proposed a more efficient KVAC based on a different algebraic MAC called $\mathrm{MAC}_{\mathrm{BB}}$. They also shown how to turn their KVAC system into an efficient publicly verifiable anonymous credentials system. Unfortunately, they have to rely on pairings to transform their KVAC into a publicly verifiable anonymous credentials system.

1.2 Motivation

In this paper, our objective is to design an efficient anonymous credential (AC) system that meets the requirements put forth in the eIDAS 2.0 regulation. This

means that our AC system should be pairing-free, and the holder binding feature should be implementable on certified hardware (typically Secure Elements or Hardware Security Modules) using SOG-IS certified digital signature algorithms (ECDSA or ECSDSA/ECSchnorr).

To this end, we first introduce a new algebraic MAC scheme which is the secret key and pairing-free variant of BBS [34] which we naturally call MAC_{BBS}.[5] Next, we use it to construct a practical pairing-free KVAC, which is proven secure under classical assumptions.

Next, we show how to turn it into an efficient pairing-free and publicly verifiable anonymous credentials system. To achieve this, we use a recent technique (Oblivious Issuance of Proofs) developed by Orrù et al. [28].

Finally, we show how to distribute the holder's computations, during the verifiable presentation of their credentials to a verifier, between a mobile application and the Secure Element embedded in their mobile phone. We call the resulting anonymous credential system BBS#.

To demonstrate its efficiency and suitability for the EUDI Wallet, we implemented BBS# on trusted mobile hardware. We found that eIDAS 2.0 transactions can be performed in less than 100 ms when using Android StrongBox SE on the user's smartphone.

1.3 Organization

This paper is structured as follows. Section 2 introduces our main notation and the necessary building blocks. In particular, we introduce our algebraic MAC scheme called MAC_{BBS}. In Section 3, we describe our pairing-free KVAC based on MAC_{BBS} and prove its security in the random oracle model. Next, in Section 4 we explain how our KVAC can be turned into a traditional (pairing-free) public-key anonymous credential system. In Section 5, we explain how to distribute the computations on the user's side between a SE and a wallet application on their mobile phone. Finally, Section 6 presents the efficiency and complexity evaluations, along with the implementation benchmarks of BBS#.[6]

2 Preliminaries

2.1 Notation

We introduce some notation used throughout this document. To state that x is chosen uniformly at random from the set S, we use one of the two following notations $x \xleftarrow{R} S$ or $x \in_R S$. In addition, $\overrightarrow{m}$ will denote the vector or list

[5] Orrù recently proposed independently the same algebraic MAC scheme in eprint 2024/1552.

[6] Obviously, BBS# also works in a pairing-based setting, with pairing-friendly curves. In addition, BBS# is compatible with different data formats, such as ISO mDL (ISO/IEC 18013-5).

$(m_1, m_2, \ldots, m_n)$, also written as $\{m_i\}_{i=1}^n$. λ will denote the security parameter and 1^λ will represent the security parameter in unary form. All algorithms are probabilistic, unless otherwise indicated. By $y \leftarrow \mathcal{A}(x_1, x_2, \ldots, x_n)$, we denote the action of running $\mathcal{A}$ on inputs $(x_1, x_2, \ldots, x_n)$ and assigning the output to y. We write $y \leftarrow \mathcal{A}^{\mathcal{O}}(x_1, x_2, \ldots, x_n)$ to indicate that $\mathcal{A}$ is an algorithm, with oracle access to some algorithm or set of algorithms $\mathcal{O}$, that takes as inputs $(x_1, x_2, \ldots, x_n)$, and assigns the output to y.

We use the term "Experiment" in the context of security definitions and proofs. An experiment will be denoted **Exp**. An experiment **Exp** in which an adversary $\mathcal{A}$ interacts with a challenger $\mathcal{C}$ to break the security property prop of a scheme Schem is denoted by $\mathbf{Exp}_{\mathcal{A}}^{\text{prop}}(1^\lambda)$.

2.2 Zero-Knowledge Proofs (ZKP)

A Zero-Knowledge Proof of Knowledge (ZKPK) [22] is an interactive protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$, where the prover attempts to convince the verifier of the knowledge of some secrets verifying a given statement, without revealing any information about the said secrets. A ZKPK should satisfy three properties, namely (i) *completeness* (*i.e.* a valid prover should be able to convince an honest verifier with overwhelming probability), (ii) *soundness* (*i.e.* a malicious prover should be rejected with overwhelming probability), (ii) *zero-knowledge* (*i.e.* the proof reveals no information about the secret(s)). In our constructions, we use as building blocks non-interactive zero-knowledge proofs of knowledge (or signatures of knowledge, SoK for short), obtained with a heuristic transformation such as Fiat-Shamir [18]. We use the Camenisch-Stadler notation [12], where, for example, $\pi := \text{SoK}\{\alpha, \beta : y = g^\alpha \wedge z = g^\beta\}[m]$ denotes a signature of knowledge of secrets α, β, verifying the statement on the right side of the colon. The signature of knowledge itself is generated on the message m. If the message is empty, we use the following notation to denote this signature of knowledge: $\pi := \text{PoK}\{\alpha, \beta : y = g^\alpha \wedge z = g^\beta\}$.

In the random oracle model (ROM) [6], one can use the forking lemma [30] to extract the secrets from such a signature of knowledge if correct care is taken that the prover can indeed be efficiently rewound. Moreover, in the ROM one can simulate such signatures of knowledge for unknown secrets [30].

2.3 Oblivious Issuance of Proofs (OIP)

For our (pairing-free) anonymous credentials scheme, we will use a specific ZKPK, namely a *proof of equality of discrete logarithms* [15], denoted π_{DLEQ}, that can be requested *anonymously* and issued *obliviously* [28], i.e., in such a way that it cannot be linked back to the interaction that produced it: $\pi_{\text{DLEQ}} := \text{PoK}\{\alpha : B = A^\alpha \wedge h = g^\alpha\}$, where α is the prover's secret, g and h are two public generators of a cyclic group $\mathbb{G}$ and A and B are two generators of $\mathbb{G}$ satisfying $B = A^\alpha$ but which are unknown (blinded) to the prover.
More precisely, the proof will be issued in such a way that the prover will not be able to link $(A, B, \pi_{\text{DLEQ}})$ to its respective issuance (*obliviousness*) and the

verifier will not be able after the issuance of l such proofs, even in a *concurrent manner*, to forge, on its own, a new valid proof (*one-more unforgeability*).

The resulting proof is transferable and can be verified non-interactively by anyone.

2.4 Signature Schemes with Key Blinding

To protect their privacy, users in our anonymous credential scheme will make use of specific signature schemes that support *key blinding* a.k.a., *key randomization* [19]. Signature schemes with this property have the advantage that one can randomize or blind the original key pair (sk, pk) to a new random key-pair (sk', pk') and sign a message m with the seemingly unrelated key (sk'). Of course, in our context, the user will have to prove (in ZK) that pk' is a randomized version of a public key pk that has been certified by a given issuer. The main goal of this randomization is to ensure that a verifier will not be able to trace a user from the signatures the latter issued. In other words, the former should not be able to distinguish between two signatures using two fresh keys obtained from the randomization of the same long-term key sk and two signatures using two fresh keys but obtained from the randomization of two distinct long-term keys sk and sk^*.

Obviously, the signatures generated by a user should be unforgeable and this should even hold when the adversary is allowed to learn message / signature pairs made with respect to randomized public keys that they have chosen (*unforgeability*).

We propose two concrete signature schemes with key blinding (see Section 5): ECSDSA (a.k.a. ECSchnorr) [24] with *additive blinding* and ECDSA [24] with *multiplicative blinding*.

2.5 Computational Hardness Assumptions

The security of our MAC scheme and KVAC system relies on a set of computational hardness assumptions. In what follows, $\mathbb{G}$ denotes a cyclic group of prime order p, where p is a λ-bit prime and λ a security parameter.

Discrete Logarithm (DL) Assumption. The Discrete Logarithm assumption holds in $\mathbb{G}$ if it is computationally hard, given a generator $g \in \mathbb{G}$ and an element $y \in \mathbb{G}$, to compute the integer $x \in \mathbb{Z}_p$ such that $y = g^x$. The advantage w.r.t. an adversary $\mathcal{A}$ in breaking this assumption will be denoted $\mathrm{Adv}^{DL}_{\mathbb{G},\mathcal{A}}(1^\lambda)$.

Decisional Diffie-Hellman (DDH) Assumption. The Decisional Diffie-Hellman assumption holds in $\mathbb{G}$ if it is hard, given a generator $g \in_R \mathbb{G}$, two elements $g^a,\ g^b \in_R \mathbb{G}$ and a candidate $X \in \mathbb{G}$ to decide whether $X = g^{ab}$ or not. This is equivalent to decide, given $g,\ h, g^a,\ h^b$, whether $a = b \mod p$ or not. The advantage w.r.t. an adversary $\mathcal{A}$ in breaking this assumption will be denoted $\mathrm{Adv}^{ddh}_{\mathbb{G},\mathcal{A}}(1^\lambda)$.

gap Discrete Logarithm (gap DL) Assumption. The gap Discrete Logarithm assumption holds in $\mathbb{G}$ if DL is hard even in the presence of a DDH oracle for the DL challenge (*i.e.* $y = g^x$), that is an oracle which given a quadruple $(g,\ h, g^x,\ h^y)$ answers whether $x = y \mod p$ or not. The advantage w.r.t. an adversary $\mathcal{A}$ in breaking this assumption will be denoted $\mathrm{Adv}^{gap\ dl}_{\mathbb{G},\mathcal{A}}(1^\lambda)$.

q-Strong Diffie-Hellman (q-SDH) Assumption. The q-Strong Diffie-Hellman assumption holds in $\mathbb{G}$ if it is hard, given a generator $g \in_R \mathbb{G}$ and $(g^x, g^{x^2}, \ldots, g^{x^q}) \in_R \mathbb{G}^q$ as input, to output a pair $(c,\ g^{\frac{1}{x+c}}) \in \mathbb{Z}_p^* \times \mathbb{G}$. The advantage w.r.t. an adversary $\mathcal{A}$ in breaking this assumption will be denoted $\mathrm{Adv}^{q-sdh}_{\mathbb{G},\mathcal{A}}(1^\lambda)$.

This assumption is believed to be hard even if the adversary is given access to a DDH oracle for the underlying DL challenge (*i.e.* $y = g^x$).

*gap q-Strong Diffie-Hellman (*gap *q-SDH) Assumption.* The gap q-Strong Diffie-Hellman assumption holds in $\mathbb{G}$ if q-SDH is hard even in the presence of a DDH oracle for the DL challenge (the integer $x \in \mathbb{Z}_p$). The advantage w.r.t. an adversary $\mathcal{A}$ in breaking this assumption will be denoted $\mathrm{Adv}^{gap\ q-sdh}_{\mathbb{G},\mathcal{A}}(1^\lambda)$.

q-Discrete Logarithm (q-DL) Assumption. The q-Discrete Logarithm assumption holds in $\mathbb{G}$ if it is hard to recover x uniformly distributed over $\mathbb{Z}_p$, given as input $(g^x, g^{x^2}, \ldots, g^{x^q}) \in_R \mathbb{G}^q$. The advantage w.r.t. an adversary $\mathcal{A}$ in breaking this assumption will be denoted $\mathrm{Adv}^{q-dl}_{\mathbb{G},\mathcal{A}}(1^\lambda)$.

Algebraic Group model (AGM). In the algebraic group model [21], it is assumed that adversaries know the representation of any group element they return. This means that, after having received a list of group elements $X_1, X_2, \ldots, X_n$, whenever the adversary returns a group element X, it must also return a list of coefficients $\alpha_1, \alpha_2, \ldots, \alpha_n$ such that $X = \prod_i X_i^{\alpha_i}$. We call such adversaries *algebraic*.

The q-SDH assumption trivially implies the slightly more standard q-DL assumption. The converse is not known to be true in general, but it is true for algebraic adversaries [5].

2.6 Message Authentication Codes (MACs)

A Message Authentication Code (MAC) for a block of n messages is an authentication tag computed using a secret key that is shared between the issuer and the verifier. More formally, a MAC scheme consists of the following four algorithms:

- Setup$(1^\lambda, n)$: On input a security parameter λ and an integer n, this algorithm creates the public parameters pp of a MAC scheme generating authentication tags on sets of n messages $\{m_i\}_{i=1}^n$.
- KeyGen(pp): On input the public parameters pp, this algorithm generates the secret key sk that is shared between the issuer and the verifier.

- MAC($pp, sk, \{m_i\}_{i=1}^n$): On input the public parameters pp, a secret key sk and n messages $\{m_i\}_{i=1}^n$, this algorithm outputs a MAC, also known as a tag, and denoted by τ, on the set of n messages $\{m_i\}_{i=1}^n$.
- Verify(pp, sk, $\{m_i\}_{i=1}^n, \tau$): On input the public parameters pp, the secret key sk, a set of n messages $\{m_i\}_{i=1}^n$ and a tag τ, this algorithm outputs either 1 (valid) or 0 (invalid).

UF-CMVA Security. Usually, a probabilistic MAC scheme is considered secure if it is *unforgeable under chosen message and verification attack* (UF-CMVA). In other words, the adversary $\mathcal{A}$ can query two oracles: $\mathcal{O}$MAC and $\mathcal{O}$Verify. $\mathcal{O}$MAC provides her with a valid MAC on any set of n messages $\overrightarrow{m} = \{m_i\}_{i=1}^n$ of her choice whereas $\mathcal{O}$Verify enables her to check the validity of any pair $(\overrightarrow{m}, \tau)$. Such an adversary should not be able to compute a pair $(\overrightarrow{m}', \tau')$ where τ' is a valid MAC on the set of n messages $\overrightarrow{m}'$ that has not already been queried to the $\mathcal{O}$MAC oracle.

A yet stronger security notion for probabilistic MACs, denoted sUF-CMVA, exists. In such a variant, the adversary wins even if $\overrightarrow{m}'$ has already been queried to the $\mathcal{O}$MAC oracle, as long as the oracle did not produce the pair $(\overrightarrow{m}', \tau')$. Fig. 1 details the sUF-CMVA experiment $\mathrm{Exp}_{\mathcal{A}}^{\mathrm{sUF-CMVA}}(1^\lambda)$ between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. The adversary's success probability, denoted by $\mathrm{Adv}_{\mathcal{A}}^{\mathrm{sUF-CMVA}}(1^\lambda)$ is defined as $\Pr[\mathrm{Exp}_{\mathcal{A}}^{\mathrm{sUF-CMVA}}(1^\lambda) = 1]$.

$\underline{\mathrm{Exp}_{\mathcal{A}}^{\mathrm{sUF-CMVA}}(1^\lambda)}$

1. $pp \leftarrow \mathbf{Setup}(1^\lambda, n)$;
2. $sk \leftarrow \mathbf{KeyGen}(pp)$;
3. $(\overrightarrow{m}', \tau') \leftarrow \mathcal{A}^{\mathcal{O}}(pp)$, where $\mathcal{O} = (\mathcal{O}\mathrm{MAC}, \mathcal{O}\mathrm{Verify})$;
4. if $(\overrightarrow{m}', \tau')$ was obtained following a call to the $\mathcal{O}$MAC oracle, then return 0.
5. Return Verify$(pp, sk, \overrightarrow{m}', \tau)$

Fig. 1: sUF-CMVA security

2.7 An Algebraic MAC Scheme Based on BBS

Our pairing-free KVAC is based on a variant of Barki et al.'s scheme [4] called $\mathrm{MAC}_{\mathrm{BB}}$. Our variant, which we call $\mathrm{MAC}_{\mathrm{BBS}}$, produces shorter authentication tags than $\mathrm{MAC}_{\mathrm{BB}}$ and can be seen as the secret key (MAC) variant of BBS [34]. $\mathrm{MAC}_{\mathrm{BBS}}$ works as follows:

- Setup($1^\lambda, n$): creates the system public parameters $pp = (\mathbb{G},\ p, \widetilde{g}, g_0,\ g_1, g_2, \ldots, g_n)$ where $\mathbb{G}$ is a cyclic group of prime order p, a λ-bit prime, and $\widetilde{g}, g_0,\ g_1, g_2, \ldots, g_n$ are random generators of $\mathbb{G}$.

- KeyGen(pp): selects a random value $x \in_R \mathbb{Z}_p$ as the issuer's secret key and optionally computes the corresponding public key $PK_I = \tilde{g}^x$.
- MAC($pp, x, \{m_i\}_{i=1}^n$): takes as input a set of n messages $\overrightarrow{m} = \{m_i\}_{i=1}^n$ and computes $A = (g_0 g_1^{m_1} g_2^{m_2} \dots g_n^{m_n})^{\frac{1}{x+e}}$, where $e \in_R \mathbb{Z}_p$. The MAC on $\overrightarrow{m}$ consists of the pair $(A,\ e)$.
- Verify($pp,\ x,\ \{m_i\}_{i=1}^n, A,\ e$): checks the validity of the authentication tag $\tau = (A,\ e)$ with respect to the set of n messages $\{m_i\}_{i=1}^n$. The authentication tag $\tau = \ (A,\ e)$ is valid on $\{m_i\}_{i=1}^n$ only if $(g_0 g_1^{m_1} g_2^{m_2} \dots g_n^{m_n})^{\frac{1}{x+e}} = A$.

Theorem 1. *(Adapted from [34] Theorem 2) In the Algebraic Group Model* $\mathrm{MAC}_{\mathrm{BBS}}$ *is sUF-CMVA secure under the gap DL and gap q-DL assumptions. More precisely, for every algebraic sUF-CMVA adversary* $\mathcal{A}$ *issuing at most* q *requests to* $\mathcal{O}$MAC*, there exist adversaries* $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *such that*

$$\mathrm{Adv}_{\mathcal{A}}^{\mathrm{sUF-CMVA}}(1^\lambda) \leq \mathrm{Adv}_{\mathbb{G},\mathcal{B}_1}^{gap\ q-dl}\left(1^\lambda\right) + \mathrm{Adv}_{\mathbb{G},\mathcal{B}_2}^{gap\ dl}\left(1^\lambda\right) + \frac{1}{p}$$

The adversaries $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *have running times comparable to* $\mathcal{A}$*.*

Remark 1. The proof of this theorem follows along the lines of Theorem 2 of [34]. The major difference is that the cited theorem holds in the AGM with a pairing. The pairing allows to simulate a DDH oracle for the DL challenge x (the issuer's secret key). In our context, we have no pairings, but a DDH oracle provided by our gap-DL and gap-q-DL challengers. We can therefore easily check that Theorem 2 of [34] also holds in our (pairing-free) context.
$\mathrm{MAC}_{\mathrm{BBS}}$ can also be proven secure in the standard model, under the gap q-SDH assumption (along the lines of Theorem 1 of [34]). However, the corresponding proof is not tight, as it incurs a multiplicative loss equal to the number of $\mathcal{O}$MAC queries.

Remark 2. A particular feature of $\mathrm{MAC}_{\mathrm{BBS}}$ is that anyone can verify the validity of a given MAC by himself (i.e. without neither knowing the private key x nor querying the $\mathcal{O}$Verify oracle). In fact, a MAC on $\overrightarrow{m} = \{m_i\}_{i=1}^n$ consists of a pair $(A,\ e)$ such that $A = (g_0 g_1^{m_1} g_2^{m_2} \dots g_n^{m_n})^{\frac{1}{x+e}}$. This implies that $A^{x+e} = g_0 g_1^{m_1} g_2^{m_2} \dots g_n^{m_n}$ and hence $B = g_0 g_1^{m_1} g_2^{m_2} \dots g_n^{m_n} A^{-e} = A^x$. Therefore, if the issuer of the MAC $(A,\ e)$ also provides the following ZKPK
$\pi_{\mathrm{DLEQ}} := \mathrm{PoK}\{\alpha : B = A^\alpha \wedge PK_I = \tilde{g}^\alpha\}$, then anyone will be able to check if the MAC is valid.

Remark 3. Our pairing-free (public-key) anonymous credential scheme (section 4) heavily relies on the fact that the above discrete logarithm equality proof π_{DLEQ} can be requested anonymously and issued obliviously on a randomized version (A^l, B^l) of the pair (A, B) ([28]).

3 A Keyed-Verification Anonymous Credentials System Based on $\mathrm{MAC}_{\mathrm{BBS}}$

Anonymous credential is a broad notion that usually covers any system that allows an Identity Provider (also called an *Issuer*) to issue a credential (we will sometimes call them *Verifiable Credentials* or *VC* for short) on user's attributes such that (1) the users can later prove that their attributes are certified and (2) the elements revealed by the users when they *show* (we will sometimes also say *present* and call this procedure a *Verifiable Presentation* or *VP* for short) their credential to a verifier (or service provider) cannot be linked to a specific issuance (unless the revealed attributes allow to do so). To better protect their privacy, a user should be able to reveal to a verifier (also called a Relying Party, RP for short) only the attributes *strictly necessary* for the requested service. This property is known as *selective disclosure* of attributes.

Traditional anonymous credentials schemes rely on public-key primitives (namely, digital signatures), with the issuer and verifier being two distinct entities. KVACs proposed by Chase et al. [13] are the symmetric counterpart of anonymous credentials schemes, using symmetric key primitives (algebraic MACs) and are tailored to settings where the issuer of credentials is also the verifier or more generally where the issuer and the verifier share the private issuance key.

In this section, we first define Keyed-Verification Anonymous Credentials (KVAC) systems as well as their requirements (but only for the specific use case of selective disclosure which is the main privacy use case envisioned in eIDAS 2.0). Next, we detail our new KVAC system that is built upon $\mathrm{MAC}_{\mathrm{BBS}}$. Our KVAC significantly differs from the one of Barki et al.'s [4] in that all the cryptographic algorithms and cryptographic computations performed on the user's side with our KVAC are supported by current certified secure elements embedded on existing mobile phones.

3.1 Syntax

In this section, we follow the syntax of (multi-show) anonymous credentials systems from [31]. However, we adapt it to our specific (keyed verification) setting.

A KVAC is defined through the following algorithms which involve three entities: a user $\mathcal{U}$ (also called a holder), an issuer I and a verifier $\mathcal{V}$ (who shares a secret key with I).

- Setup$(1^\lambda, n)$: This algorithm takes as input a security parameter λ and a bound n on the number of attributes to certify and outputs the public parameters of the system pp.
- IssKeyGen(pp): This algorithm takes as input the system public parameters pp and outputs an issuer's secret key sk_I and optionally a corresponding public key pk_I.
- UserKeygen(pp): This algorithm takes as input the system public parameters pp and returns a user's key pair (sk, pk).

- (Obtain(sk, pk_I, $\{m_i\}_{i=1}^n$), Issue(pk, sk_I, $\{m_i\}_{i=1}^n$): To obtain an anonymous credential on a set of attributes $\{m_i\}_{i=1}^n$, the user, running Obtain, interacts with the issuer, running Issue. The former algorithm additionally requires the user's secret key sk and optionally the issuer's public key pk_I, whereas the latter requires pk and sk_I. If the protocol does not abort, the user gets a credential σ.
- (Show(pk_I, sk, $\{m_i\}_{i=1}^n, \mathcal{D}, \sigma$), Verify($sk_I$, $\{m_i\}_{i\in\mathcal{D}}$): These algorithms are run by a user and a verifier (which is also the Issuer in our setting), respectively, who interact during execution. Show enables the user to prove that a subset $\{m_i\}_{i\in\mathcal{D}}$ of his attributes, with $\mathcal{D} \subset [1, n]$, has been certified. It takes as input the credential σ, the optional issuer's public key pk_I, the whole set of attributes $\{m_i\}_{i=1}^n$ along with the intended subset $\mathcal{D}$. The Verify algorithm only takes as input sk_I and the subset $\{m_i\}_{i\in\mathcal{D}}$ and returns either 1 (accept) or 0 (reject).

3.2 Security Model

The security model that we consider here is the one from [31] that we adapt to our specific setting where the issuer and the verifier share the issuance private key sk_I.

Besides *correctness*, (keyed verification) anonymous credentials systems must fulfill two additional security requirements: *unforgeability* (it is not possible to successfully *show* a credential that was not previously obtained during an execution of the interactive protocol ($Obtain$, $Issue$)) and *anonymity* (no information about the user is disclosed beyond the attributes that the user agreed to reveal during the execution of $Show$).

We define these two requirements through the experiments described in Fig. 2 which use the following oracles along with two sets: HU, the set containing the identities of *honest* users, and CU, the set containing the identities of *corrupt* users. Following [31], we additionally define the set Att that stores $\{j, \{m_i\}_{i=1}^n\}$ each time a credential is generated for user j on $\{m_i\}_{i=1}^n$ by the oracles $\mathcal{O}$ObtIss and $\mathcal{O}$Issue below. We say that $\{j, \{m_i\}_{i\in\mathcal{D}}\} \subset$ Att if $\exists\{j, \{m'_i\}_{i=1}^n\} \in$ Att with $m'_i = m_i$ for all $i \in \mathcal{D}$.

- $\mathcal{O}$HU(j): on input an identity j, this oracle returns $\perp$ if $j \in$ HU $\cup$ CU. Else it generates a key pair $(sk_j, pk_j) \leftarrow UserKeygen(pp)$ and returns pk_j. The identity j is then added to HU.
- $\mathcal{O}$CU(j, pk_j): on input an identity j and optionally a public key pk_j, this oracle registers a new corrupted user with public key pk_j if $j \notin$ HU and returns sk_j and all associated credentials otherwise. In the latter case, j is removed from HU. In all cases, j is added to CU.
- $\mathcal{O}$ObtIss($j, \{m_i\}_{i=1}^n$): on input an identity $j \in$ HU and a set of attributes $\{m_i\}_{i=1}^n$, this oracle runs ($Obtain(sk_j$, pk_I, $\{m_i\}_{i=1}^n)$, $Issue(pk_j$, sk_I, $\{m_i\}_{i=1}^n$) and stores the resulting output. The elements $\{j, \{m_i\}_{i=1}^n\}$ are then added to Att. If $j \notin$ HU, the oracle returns $\perp$.

- $\mathcal{O}\text{Obtain}(j, \{m_i\}_{i=1}^n)$: on input an identity $j \in \text{HU}$ and a set of attributes $\{m_i\}_{i=1}^n$, this oracle runs $Obtain(sk_j, pk_I, \{m_i\}_{i=1}^n)$ and stores the resulting output. If $j \notin \text{HU}$, the oracle returns $\perp$. This oracle is used by an adversary impersonating the issuer to issue a credential to an honest user.
- $\mathcal{O}\text{Issue}(j, \{m_i\}_{i=1}^n)$: on input an identity $j \in \text{CU}$ and a set of attributes $\{m_i\}_{i=1}^n$, this oracle runs $Issue(pk_j, sk_I, \{m_i\}_{i=1}^n)$ and stores the resulting output. The elements $\{j, \{m_i\}_{i=1}^n\}$ are then added to Att. If $j \notin \text{CU}$, the oracle returns $\perp$. This oracle is used by an adversary playing the role of a malicious user to obtain a credential from an honest issuer.
- $\mathcal{O}Show(k, \mathcal{D})$: Let $\sigma^{(k)}$ be the credential issued on $\left\{m_i^{(k)}\right\}_{i=1}^n$ for a user j_k during the k-th query to $\mathcal{O}\text{Obtain}$ or $\mathcal{O}\text{ObtIss}$. If $j_k \notin \text{HU}$, this oracle returns $\perp$. Else, this oracle runs $Show(pk_I, sk_{j_k}, \left\{m_i^{(k)}\right\}_{i=1}^n, \mathcal{D}, \sigma^{(k)})$, with the adversary playing a malicious verifier.
- $\mathcal{O}Verify(\{m_i\}_{i\in\mathcal{D}}, \mathcal{D})$: on input a set $\mathcal{D} \subset [1, n]$ and a set of attributes $\{m_i\}_{i\in\mathcal{D}}$, this oracle runs $Verify(sk_I, \{m_i\}_{i\in\mathcal{D}})$ with the adversary playing the role of a malicious user.

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

1. $pp \leftarrow \textbf{Setup}(1^\lambda, n)$;
2. $(sk_I, pk_I) \leftarrow IssKeyGen(pp)$;
3. $\{m_i\}_{i\in\mathcal{D}} \leftarrow \mathcal{A}^{\mathcal{O}\text{HU},\mathcal{O}\text{CU},\mathcal{O}\text{ObtIss},\mathcal{O}\text{Issue},\mathcal{O}Show,\mathcal{O}Verify}(pk_I)$;
4. $b \leftarrow (\mathcal{A}(\,), Verify(sk_I, \{m_i\}_{i\in\mathcal{D}})$;
5. If $\{j, \{m_i\}_{i\in\mathcal{D}}\} \subset$ Att with $j \in$ CU or if $b = 0$, return 0;
6. Return 1;

Anonymity
$\underline{\text{Exp}_{\mathcal{A}}^{ano}(1^\lambda, n)}$

1. $pp \leftarrow \textbf{Setup}(1^\lambda, n)$;
2. $(sk_I, pk_I) \leftarrow IssKeyGen(pp)$;
3. $b \xleftarrow{R} \{0,\ 1\}$;
4. $(j_0, j_1, \{m_i\}_{i\in\mathcal{D}}) \leftarrow \mathcal{A}^{\mathcal{O}\text{HU},\mathcal{O}\text{CU},\mathcal{O}\text{Obtain},\mathcal{O}Show}(sk_I)$;
5. If $\{j_{b'}, \{m_i\}_{i\in\mathcal{D}}\} \not\subset$ Att for $b' \in \{0,\ 1\}$, return 0;
6. $(Show(pk_I, sk_{j_b}, \left\{m_i^{(j_b)}\right\}_{i=1}^n, \mathcal{D}, \sigma^{(j_b)}), \mathcal{A}(\,))$;
7. $b^* \leftarrow \mathcal{A}^{\mathcal{O}\text{HU},\ \mathcal{O}\text{CU},\ \mathcal{O}\text{Obtain},\mathcal{O}Show}(sk_I)$;
8. If $\mathcal{O}$CU has been queried on $j_{b'}$ for $b' \in \{0,\ 1\}$, return 0;
9. Return $(b^* = b)$;

Fig. 2: Security requirements for KVAC

Correctness. A showing of a credential σ with respect to a set $\{m_i\}_{i\in\mathcal{D}}$ always verify if σ was honestly issued on $\{m_i\}_{i=1}^{n}$, with $\mathcal{D} \subset [1, n]$.

Unforgeability. A KVAC is unforgeable if the adversary's success probability, denoted by $\mathrm{Adv}^{\mathrm{uf}}_{\mathcal{A}}(1^\lambda,\ n)$ and defined as

$$\left|\Pr[\mathrm{Exp}^{\mathrm{uf}}_{\mathcal{A}}(1^\lambda,\ n)\ =\ 1]\right|$$

is negligible for any polynomial time adversary $\mathcal{A}$.

Anonymity. A KVAC is anonymous if the adversary's success probability, denoted by $\mathrm{Adv}^{\mathrm{ano}}_{\mathcal{A}}(1^\lambda,\ n)$ and defined as

$$\left|\Pr[\mathrm{Exp}^{\mathrm{ano}}_{\mathcal{A}}\left(1^\lambda,\ n\right) - \tfrac{1}{2}]\right|$$

is negligible for any polynomial time adversary $\mathcal{A}$.

Following [31] and [16], our definition assumes that the issuer's key pair (sk_I, pk_I) (where pk_I is optional in a KVAC) is honestly generated and then sent to the adversary in contrast to [20], for example, which lets the adversary generate its own key pair. However, to satisfy the definition in [20], $IssKeyGen$ could output, in addition to the public key pk_I, a non-interactive zero-knowledge proof of knowledge of the secret key corresponding to pk_I.

3.3 Our Construction

Based on the designed $\mathrm{MAC}_{\mathrm{BBS}}$ scheme, we construct a KVAC system involving a user $\mathcal{U}$ (also called a holder), an issuer I and a verifier $\mathcal{V}$ (who holds the issuer's secret key sk_I). We would however like to emphasize that this KVAC system is just a stepping stone to the publicly verifiable anonymous credentials system BBS#.

3.4 Intuition of our Construction

In our approach, the issuer creates a $\mathrm{MAC}_{\mathrm{BBS}}$ authentication tag σ on the user's public pk (of a signature scheme supporting key blinding / randomization) and on their attributes $\{m_i\}_{i=1}^{n}$. The tag σ represents the user's credential and authenticates both the user's attributes and their public key pk. During a Verifiable Presentation of their attributes (or a subset of them) to the verifier, the user will first randomize their public key pk (either additively if ECSDSA is used on the user's secure cryptographic device or multiplicatively in the case of ECDSA) as well as their verifiable credential σ. We denote by pk_{Blind} and σ_{Blind} respectively, these randomized versions. The user will then first generate a SoK $\pi_{HolderBinding}$ (π_{HB} for short) of the private key associated to pk_{Blind} on a *nonce* generated by the verifier (to guarantee the freshness of the VP) and then a ZKP $\pi_{Validity}$ proving knowledge of : (a) two random factors (r, r'), (b) a credential σ and (c) a public pk such that (1) σ_{Blind} is a randomized version of σ under the random factor r, (2) pk_{Blind} is a randomized version of pk under the random factor r',

and (3) σ is a valid MAC_{BBS} authentication tag on the (disclosed) attributes requested by the verifier. The proof $\pi_{HolderBinding}$ is, as its name indicates, a proof that the VP comes from the user who truly holds the credential σ (underlying σ_{Blind}), which certifies the attributes disclosed to the verifier (*holder binding*).

- Setup($1^\lambda, n$): On input a security parameter λ and a bound n on the number of attributes to certify, this algorithm generates the public parameters $pp = (\mathbb{G},\ p, g, \widetilde{g}, g_0,\ g_1, g_2, \ldots, g_n,\ H,\ F)$, where $\mathbb{G}$ is a cyclic group of prime order p, a λ-bit prime, and $g, \widetilde{g}, g_0,\ g_1, g_2, \ldots, g_n,\ H,\ F$ are random generators of $\mathbb{G}$. For $i \in \{1, \ldots, n\}$, g_i is associated with a specific type of attributes (*e.g.* age, gender, etc.). This will help to differentiate attributes and avoid any ambiguity. Note that, from now on, all computations involving exponents are computed modulo p (i.e. mod p).
- IssKeyGen(pp): On input the system public parameters pp, this algorithm selects a random value $sk_I \in_R \mathbb{Z}_p$ and computes the corresponding public key $pk_I = \tilde{g}^{sk_I}$ and optionally a ZKPK $\pi_I := \text{PoK}\{\alpha : pk_I = \tilde{g}^\alpha\}$ proving knowledge of the private key sk_I.
- UserKeygen(pp): To generate a key pair (sk, pk) for a user, this algorithm selects a random value $sk \in_R \mathbb{Z}_p$ and computes the corresponding public key $pk = g^{sk}$. In practice, the key pair (sk, pk) will be managed by the user's cryptographic device (also called WSCD for Wallet Secure Cryptographic Device in the eIDAS 2.0 terminology).
- (Obtain($sk, pk_I, \{m_i\}_{i=1}^n$), Issue($pk, sk_I, \{m_i\}_{i=1}^n$): To obtain an anonymous credential on a set of attributes $\{m_i\}_{i=1}^n$, the user first sends their public key pk along with a signature of knowledge of sk on a challenge ch, chosen by the Issuer, to guarantee the freshness of this SoK: $\pi_U = \text{SoK}\{\alpha : pk = g^\alpha\}[ch]$. If this SoK, which can be generated using for example the Schnorr's protocol [32], is correct, then the issuer randomly picks $e \in_R \mathbb{Z}_p$, computes $C_m = g_0 pk \prod_{i=1}^n g_i^{m_i}$ and $A = (\ C_m)^{\frac{1}{sk_I + e}}$. The issuer may also build a ZKPK $\pi_{\text{DLEQ}} := \text{PoK}\{\alpha : B = A^\alpha \wedge pk_I = \tilde{g}^\alpha\}$ where $B = \ C_m A^{-e} = A^{sk_I}$. Then the issuer returns the pair (A, e) along with the proof π_{DLEQ} to the user. If the proof is valid, the user sets their anonymous credential σ as $\sigma = (A, e)$.
- (Show($pk_I, sk, \{m_i\}_{i=1}^n, \mathcal{D}, \sigma$), $Verify(sk_I, \{m_i\}_{i\in\mathcal{D}})$: To anonymously prove that they hold a credential on $\{m_i\}_{i\in\mathcal{D}}$, the user engages in an interactive protocol with the verifier $\mathcal{V}$.
  - Show($pk_I, sk, \{m_i\}_{i=1}^n, \mathcal{D}, \sigma$). The user will first randomize their public key (so that neither the issuer nor the verifier can trace them from this key). To do this, they will randomly pick an integer r in $\mathbb{Z}_p$ and compute using this value: $pk_{Blind} = g^{sk+r}$ and $\pi_{HB} := \text{SoK}\{\alpha: pk_{Blind} = g^\alpha\}[nonce]$, where $nonce \in_R \{0,1\}^\mu$, is a random value sent by the verifier (to guarantee the freshness of the VP). They will then "randomize" their MAC_{BBS} authentication tag σ, to also prevent the issuer and the verifier from tracing them from this element, and "adapt" it to be on pk_{Blind}

and the $\{m_i\}_{i=1}^n$. To do this, they will choose integers $r_1, r_2 \in_R \mathbb{Z}_p^*$ and compute:

1. $\overline{A} = A^{r_1 \times r_2}$,
2. $D = C_m{}^{r_2}$
3. $\overline{B} = \overline{A}^{-e} D^{r_1} = \overline{A}^{sk_I}$
4. $r_3 = r_2^{-1} \mod p$
5. $\pi_{validity} = \text{SoK}\{\alpha, \beta, \gamma, \delta, \{\theta_i\}_{i \notin \mathcal{D}} : \overline{B} = \overline{A}^{\alpha} D^{\beta} \wedge g_0 pk_{Blind} \prod_{i \in \mathcal{D}} g_i^{m_i} = D^{\gamma} \prod_{i \notin \mathcal{D}} g_i^{\theta_i} g^{\delta}\}[nonce]$

   We have the following two equalities, hence the ZKP $\pi_{validity}$: $\overline{B} = \overline{A}^{-e} D^{r_1}$ and $g_0 pk_{Blind} \prod_{i \in \mathcal{D}} g_i^{m_i} = D^{r_3} \prod_{i \notin \mathcal{D}} g_i^{-m_i} g^r$.
6. The user transmits $VP = (\{m_i\}_{i \in \mathcal{D}}, pk_{Blind}, \pi_{HB}, \overline{A}, \overline{B}, D, \pi_{validity})$ to the verifier.

- Verify$(sk_I, \{m_i\}_{i \in \mathcal{D}})$. Upon receipt of $VP = (\{m_i\}_{i \in \mathcal{D}}, pk_{Blind}, \pi_{HB}, \overline{A}, \overline{B}, D, \pi_{validity})$, $\mathcal{V}$ first checks than π_{HB} is a valid SoK on *nonce*, and then verifies that $\pi_{validity}$ is valid. If so, $\mathcal{V}$ uses sk_I to check whether $\overline{B} = \overline{A}^{sk_I}$. $\mathcal{V}$ is convinced that $\mathcal{U}$ really holds a valid credential on the disclosed attributes $\{m_i\}_{i \in \mathcal{D}}$ if, and only if, all these checks succeed. We prove this fact in Appendix C. In other words, we prove that $VP = (\{m_i\}_{i \in \mathcal{D}}, pk_{Blind}, \pi_{HB}, \overline{A}, \overline{B}, D, \pi_{validity})$ constitutes a proof of knowledge of a blinding factor r, a private key sk and of a valid credential, $\sigma = (A, e)$, on the subset $\{m_i\}_{i \in \mathcal{D}}$ and on a public key $pk = pk_{Blind} g^{-r}$.

3.5 Security Analysis

The unforgeability of our KVAC system directly relies on the one of MAC_{BBS} and on the DL assumption. Anonymity holds statistically/unconditionally, which means that anonymity is preserved even against possible future quantum adversaries. This is formally stated by the following theorem.

Theorem 2. *Our KVAC system is unforgeable in the ROM if* MAC_{BBS} *is sUF-CMVA secure and if the DL assumption holds in* $\mathbb{G}$.
Our KVAC system is anonymous if π_{DLEQ} *is a sound proof system and if* π_{HB} *and* $\pi_{validity}$ *are zero-knowledge proof systems.*

Proof of Unforgeability. Let $\mathcal{A}$ be an adversary against the unforgeability of our KVAC. During the experiment, $\mathcal{A}$ returns a set of attributes $\{m_i\}_{i \in \mathcal{D}}$ and then proves possession of a credential on this set. To be valid, the forged credential should not have been issued to a corrupt user. However, honest users could possess a credential on such attributes, which leads us to consider two different cases in our proof. Let sk be the underlying secret key whose knowledge is proved by the adversary when it shows the credential on $\{m_i\}_{i \in \mathcal{D}}$, we distinguish two types of adversaries:

- Type 1: $\exists j \in$ HU such that $sk = sk_j$
- Type 2: $\forall j \in$ HU, $sk \neq sk_j$.

Lemma 1. *Any type 1 adversary $\mathcal{A}$ succeeding with probability ϵ can be converted into an adversary against the DL assumption in $\mathbb{G}$ succeeding with probability $\frac{\epsilon}{q}$, where q is a bound on the number of users.*

Proof. Let $(g,\ g^s)$ be a DL challenge. Our reduction $\mathcal{R}$ generates the public parameters $pp = (\mathbb{G}, p, g, \widetilde{g}, g_0, g_1, g_2, \ldots, g_n, H, F)$ and the issuer's key pair and returns pk_I to $\mathcal{A}$. Since we consider a type 1 adversary, we know that there is an index j such that $\mathcal{A}$ will try to impersonate the j-th honest user. Our reduction $\mathcal{R}$ then makes a guess on $j \in [1,q]$ and proceeds as follows.

- $\mathcal{O}$HU: Let i be the index query to this oracle. If $i \neq j$, then $\mathcal{R}$ proceeds as usual. Else it returns $pk_j = g^s$.
- $\mathcal{O}$CU: If $\mathcal{R}$ receives a corruption query on an honest user i, it returns sk_i if $i \neq j$ and aborts otherwise.
- $\mathcal{O}$ObtIss : $\mathcal{R}$ knows the issuer's secret key sk_I and so perfectly simulates the issuer's side of this protocol. It can also play the role of any honest user i if $i \neq j$. Else it simulates the proof of knowledge of sk_j.
- $\mathcal{O}$Issue: $\mathcal{R}$ knows the issuer's secret key sk_I and so can perfectly answer to any query.
- $\mathcal{O}Show$: If the queried credential belongs to $i \neq j$, then $\mathcal{R}$ can perform the Show protocol defined above. Else, it chooses a random value r and simulates the proof π_{HB} of knowledge of the private key $sk_j + r$ and runs the other steps of the protocol.
- $\mathcal{O}Verify$: Since $\mathcal{R}$ knows the issuer's secret key sk_I, it can answer to any query of $\mathcal{A}$.

One can note that the experiment is perfectly simulated if the guess on j is correct, which occurs with probability $\frac{1}{q}$. In such a case, a successful adversary $\mathcal{A}$ proves knowledge of $sk_j = s$ when it shows its forged credential. $\mathcal{R}$ can then run the extractors of the proofs of knowledge π_{HB} and $\pi_{validity}$ to recover s, that it returns as a valid solution to the DL problem. The probability of success of $\mathcal{R}$ is then $\frac{\epsilon}{q}$.

Lemma 2. *Any type 2 adversary $\mathcal{A}$ can be converted into an adversary against the sUF-CMVA security of* $\mathrm{MAC}_{\mathrm{BBS}}$ *succeeding with the same probability.*

Proof. Our reduction $\mathcal{R}$ runs the sUF-CMVA security experiment for $\mathrm{MAC}_{\mathrm{BBS}}$ for the parameters (λ, n) with its challenger $\mathcal{C}$ and receives from $\mathcal{C}$ the corresponding public parameters $pp = (\mathbb{G},\ p, \widetilde{g}, g_0,\ g_1, g_2, \ldots, g_n,\ g_{n+1})$. The underlying private signing key will be denoted sk_I in the sequel. $\mathcal{R}$ generates two additional random generators H and F and defines g as g_{n+1}. The reduction $\mathcal{R}$ then returns $pp = (\mathbb{G},\ p, g, \widetilde{g}, g_0,\ g_1, g_2, \ldots, g_n,\ H,\ F)$ to $\mathcal{A}$. $\mathcal{R}$ can then answer oracle queries as follows.

- $\mathcal{O}$HU: $\mathcal{R}$ proceeds as usual and stores the corresponding secret key.
- $\mathcal{O}$CU: Here again $\mathcal{R}$ proceeds as usual.
- $\mathcal{O}$ObtIss : Let $j \in$ HU and $\{m_i\}_{i=1}^n$ be the input of this oracle. The reduction recovers the secret sk_j that it has generated for the user j and then submits $(sk_j, m_1, m_2, \ldots, m_n)$ to its MAC_{BBS} challenger $\mathcal{C}$. It then receives a tag (A, e) on $(sk_j, m_1, m_2, \ldots, m_n)$, where $A = (g_0 g^{sk_j} \prod_{i=1}^n g_i^{m_i})^{\frac{1}{sk_I + e}}$, and stores the resulting credential (A, e).
- $\mathcal{O}$Issue: Let $j \in$ CU and $\{m_i\}_{i=1}^n$ be the input of this oracle. $\mathcal{R}$ extracts sk_j from the proof of knowledge π_U generated by $\mathcal{A}$ and then proceeds as previously to obtain a tag (A, e) on $(sk_j, m_1, m_2, \ldots, m_n)$. $\mathcal{R}$ then simulates the corresponding proof π_{DLEQ} and returns the tag (A, e) to $\mathcal{A}$.
- $\mathcal{O}Show$: Let $\sigma^{(k)}$ be the credential issued on $\left\{m_i^{(k)}\right\}_{i=1}^n$ for an honest user j_k. Let j_k be the input of this oracle. As $\mathcal{R}$ knows the secret key of this honest user, it can perfectly simulate this oracle.
- $\mathcal{O}Verify$: $\mathcal{R}$ first checks the validity of π_{HB} and $\pi_{validity}$ and if they are valid extracts from these proofs, a tag $\sigma = (A', e')$, a private key sk' and a set of messages $\{m'_i\}_{i=1}^n$ (see Appendix C). It then queries the $\mathcal{O}$Verify oracle, provided by its MAC_{BBS} challenger $\mathcal{C}$ to check whether $\sigma = (A', e')$ is a valid MAC_{BBS} on $\{sk', \{m'_i\}_{i=1}^n\}$. $\mathcal{R}$ then returns the result of all these checks to $\mathcal{A}$.

$\mathcal{R}$ can handle any oracle query and never aborts. Therefore, at the end of the game, $\mathcal{A}$ is able, with some probability ϵ, to prove possession of a credential on $\{m_i\}_{i \in \mathcal{D}}$. $\mathcal{R}$ extracts from the SoK π_{HB} and $\pi_{validity}$ contained in the $Show$ protocol the underlying private key sk', a set of messages $\{m'_i\}_{i=1}^n$, and the credential $\sigma = (A', e')$ on $\{sk', \{m'_i\}_{i=1}^n\}$ (see Appendix C).

Since we here consider a type 2 adversary, sk' must be different from sk_j, for any honest user j. Moreover, to be considered as an attack against unforgeability, no credential owned by corrupt users can be valid on this set of messages (see step 5 of $\text{Exp}_{\mathcal{A}}^{uf}(1^\lambda, n)$). This means that, for any credential on $\{sk_j, \{m_i\}_{i=1}^n\}$ with $j \in$ CU, we either have $sk_j \neq sk'$, or $\exists l \in [1, n]$ such that $m_l \neq m'_l$. In both cases, this means that $\sigma = (A', e')$ is a valid forgery on $\{sk', \{m'_i\}_{i=1}^n\}$ against MAC_{BBS}.

Proof of Anonymity. Anonymity holds statistically.

Lemma 3. *Our KVAC system provides statistical anonymity. More precisely, for any adversary $\mathcal{A}$ we have*

$$\text{Adv}_{\mathcal{A}}^{\text{ano}}(1^\lambda,\ n) \leq \frac{1}{2} + \frac{n_{Obtain} q_{\mathcal{H}_1}}{p^2} + \frac{n_{Show} q_{\mathcal{H}_3}}{2^\mu p^2} + \frac{n_{Show} q_{\mathcal{H}_4}}{2^\mu p^5} + \frac{2(q_{\mathcal{H}_2}+1)}{p} + \frac{4}{p}$$

where μ is the bit length of nonce, n_{Obtain} is the total number of $\mathcal{O}$Obtain queries, n_{Show} is the total number of $\mathcal{O}$Show queries, and $q_{\mathcal{H}_i}$, the number of $\mathcal{H}_i$ queries in the ROM.

Proof. Recall that in this experiment the issuer's key pair (sk_I, pk_I), where pk_I is optional in our KVAC, is honestly generated and then sent to the adversary $\mathcal{A}$. We use Shoup's game hopping technique [33], where proofs are organized as sequences of games, for this proof.
Game 0. This is exactly the anonymity game described in Fig. 2.
Game 1. This is the same game as Game 0 except that we remove the computation of π_U inside $\mathcal{O}$Obtain requests and make $\mathcal{R}$ simulates it, using the zero-knowledge of the proof π_U (which is described in Appendix B). More precisely, $\mathcal{R}$ picks c and ρ and then deduce com$'$ and the input to $\mathcal{H}_1$. $\mathcal{R}$ aborts if this query is not fresh. Except with probability $\frac{\mathrm{q}_{\mathcal{H}_1}}{\mathrm{p}^2}$, this query is fresh, and we can program $\mathcal{H}_1$. Hence, we get:

$$\mathrm{Adv}^{\mathrm{ano}}_{\mathcal{A}}\left(1^{\lambda},\ n\right) \leq \mathrm{Adv}^{G_1}_{\mathcal{A}}\left(1^{\lambda},\ n\right) + \frac{n_{\mathrm{Obtain}} q_{\mathcal{H}_1}}{p^2}$$

.

Game 2. This is the same game as Game 1 except that we remove the computation of π_{HB} inside $\mathcal{O}$Show requests and make $\mathcal{R}$ simulates it, using the zero-knowledge of the proof π_{HB} (which is described in Appendix B). More precisely, $\mathcal{R}$ picks c and ρ and then deduce com' and the input to $\mathcal{H}_3$. $\mathcal{R}$ aborts if this query is not fresh. Except with probability $\frac{q_{\mathcal{H}_3}}{2^{\mu}p^2}$, this query is fresh, and we can program $\mathcal{H}_3$. Hence, we get:

$$\mathrm{Adv}^{G_1}_{\mathcal{A}}\left(1^{\lambda},\ n\right) \leq \mathrm{Adv}^{G_2}_{\mathcal{A}}\left(1^{\lambda},\ n\right) + \frac{n_{\mathrm{Show}} q_{\mathcal{H}_3}}{2^{\mu}p^2}$$

.

Game 3. This is the same game as Game 2 except that we remove the computation of $\pi_{Validity}$ inside $\mathcal{O}$Show requests and make $\mathcal{R}$ simulates it, using the zero-knowledge of the proof $\pi_{validity}$ (which is described in Appendix B). More precisely, $\mathcal{R}$ picks c, ρ_1, ρ_2, ρ_3, $resp$ and $\{resp_i\}_{i \notin \mathcal{D}}$ then deduce com'_1, com'_2 and the input to $\mathcal{H}_4$. $\mathcal{R}$ aborts if this query is not fresh. Except with probability $\frac{q_{\mathcal{H}_4}}{2^{\mu}p^4}$, this query is fresh, and we can program $\mathcal{H}_4$. Hence, we get:

$$\mathrm{Adv}^{G_2}_{\mathcal{A}}\left(1^{\lambda},\ n\right) \leq \mathrm{Adv}^{G_3}_{\mathcal{A}}\left(1^{\lambda},\ n\right) + \frac{n_{Show} q_{\mathcal{H}_4}}{2^{\mu}p^5}$$

.

Game 4. This is the same game as Game 3 except that we abort if for one of the credential $(\mathrm{A}_{\mathrm{j_b}}, \mathrm{e}_{\mathrm{j_b}})$, $\mathrm{b} \in \{0,1\}$, obtained during the execution of $\mathcal{O}\mathrm{Obtain}(j_b, \{m_i\}_{i=1}^n)$, we have $A_{j_b}^{sk_I} \neq B_{j_b}$ where $B_{j_b} = C_m A^{-e}$ and $C_m = g_0 pk_{j_b} \prod_{i=1}^n g_i^{m_i}$. Here we use the secret key sk_I which comes from the honest key generation of sk_I for this check. The difference between the two games Game 3 and Game 4 is bounded by the soundness of π_{DLEQ} (described in Appendix B); i.e. the probability that $\mathcal{A}$ can issue a credential (A, e) on a public key pk

and attributes $\{m_i\}_{i=1}^n$ such that the verification of π_{DLEQ} passes but $\mathrm{A}^{\mathrm{sk_I}} \neq B$ where $B = C_m A^{-e}$ and $C_m = g_0 pk \prod_{i=1}^n g_i^{m_i}$. To bound this probability, we look at every query $\mathcal{H}_2$ to the game. Each query defines $(\mathrm{A},\ \mathrm{B},\ \mathrm{com}_1, com_2)$ and the value of $\mathrm{c} = \mathcal{H}_2(\mathrm{pp},\ \mathrm{A},\ \mathrm{B},\ \widetilde{\mathrm{g}},\ \mathrm{pk_I}, com_1, com_2)$ is uniformly distributed. If $A^{sk_I} \neq B$, to pass the verification $\mathcal{A}$ should choose c and ρ randomly and computes $com_2 = \mathrm{A}^{\rho}\mathrm{B}^{-\mathrm{c}}$ and expects that the random oracle will output c on the query $(pp,\ A,\ B,\ \tilde{g},\ pk_I, com_1, com_2)$. The probability to succeed is $\frac{1}{\mathrm{p}}$. If $\mathcal{A}$ did not query the random oracle on $(pp,\ A,\ B,\ \tilde{g},\ pk_I, com_1, com_2)$, the probability that the verification passes for any π_{DLEQ} is also $\frac{1}{p}$. Hence, the soundness advantage is bounded by $\frac{2(q_{\mathcal{H}_2}+1)}{\mathrm{p}}$. We deduce that:

$$\mathrm{Adv}_{\mathcal{A}}^{G_3}\left(1^{\lambda},\ n\right) \leq \mathrm{Adv}_{\mathcal{A}}^{G_4}\left(1^{\lambda},\ n\right) + \frac{2(q_{\mathcal{H}_2}+1)}{p}$$

Game 5. This is the same game as Game 4 except that we choose two random values $(sk^*,\ e^*)$ in $\mathbb{Z}_p$ and using the secret key sk_I (which comes from the honest key generation of sk_I), we compute a valid credential (A^*, e^*) on $\{sk^*,\ \{m_i\}_{i=1}^n\}$, that we will use as the challenge pair of this game. We abort this game if a component of the pair $(sk^*,\ e^*)$ is equal to its corresponding one in the challenge pairs (sk_{j_b}, e_{j_b}), $b \in \{0, 1\}$. Then we execute a Show with $\mathcal{A}$, using this random credential $\sigma^* = (A^*, e^*)$ instead of the one of the user j_b. We show in Appendix D that $\mathcal{R}$ perfectly simulates this game using this random credential (A^*, e^*). It is therefore clear that the output of $(Show(pk_I,\ sk^*,\ \{m_i^*\}_{i=1}^n, \mathcal{D}, \sigma^*),\ \mathcal{A}(\ ))$ reveals no information about b. Therefore, we get $\mathrm{Adv}_{\mathcal{A}}^{G_5}\left(1^{\lambda},\ n\right) = \frac{1}{2}$, and we have:

$$\mathrm{Adv}_{\mathcal{A}}^{G_4}\left(1^{\lambda},\ n\right) \leq \mathrm{Adv}_{\mathcal{A}}^{G_5}\left(1^{\lambda},\ n\right) + \frac{4}{p}$$

4 From KVAC to Pairing-Free Anonymous Credentials

The main drawback of KVAC systems is that they are tailored to specific settings in which the issuer also acts as a verifier, as in the case of e-government or public transportation. They are not suited to the more general setting in which the issuer and the verifier are two distinct entities.

In this section, we explain how to turn our (pairing-free) KVAC system into a (pairing-free) public key anonymous credential system. Thereby, a user will be able to prove possession of a credential to any entity (i.e., without the latter necessarily knowing the issuer's private key).

In BBS/BBS+ based anonymous credentials schemes, pairings are used by the verifier to check whether the following equality $\overline{B} = \overline{A}^{sk_I}$ holds or not, where sk_I is the issuer's private key (see $Verify(sk_I, \{m_i\}_{i\in\mathcal{D}})$, in section 3.3).

We propose below three options to let any verifier perform this check without using pairings.

4.1 Option 1.

The first option is to let the verifier ask the issuer to check whether this equality, $\overline{B} = \overline{A}^{sk_I}$, holds or not. As $\overline{A}$ and $\overline{B}$ have been randomized by the user (they consist of the randomization of his credential values A and B), the Issuer cannot trace back the user from these values. Obviously, the issuer should prove to the verifier whether this equality holds or not. This can be done, for example, by using the classical Chaum-Pedersen ZKP of discrete logarithms equality π_{DLEQ} [15] when the equality holds or by using, for example, the proof of the inequality of discrete logarithms of Camenisch and Shoup otherwise [11].
A similar approach has been adopted in the card payment sector to enable a *point-of-sale terminal* to check the validity of a smart card transaction[7] online with the *issuer* (the cardholder's bank).

4.2 Option 2.

The second option is to let the user *anonymously* request from the issuer, during the *Show* protocol, a blind proof (a.k.a. an Oblivious Proof [28]), π_{DLEQ}, showing that $\overline{\boldsymbol{B}} = \overline{\boldsymbol{A}}^{sk_I}$ that will be sent, along the VP, to the Verifier. By blind, we mean that the issuer, although contributing to the generation of this proof (as only they know sk_I), will be unable, given such a proof, to determine for which user it was intended. This proof can be verified by anyone using solely the issuer's public key.
This approach (option 2) is similar to the one used in the context of centralized / federated identity management systems (IMS). In fact, in a federated IMS when a user wants to authenticate at a RP (or prove that they hold the attributes requested by that RP), the user is redirected to their IDP (issuer) in order to obtain a *token* (signed by the issuer), which the user can present to the RP as a proof that they have authenticated to the issuer (or that they hold the requested attributes). However, we would like to point out that, unlike federated IMS, with Option 2, neither the issuer nor the RP (even if they collude) will be able to track or link the user's activity. Indeed, since the user *anonymously* requests the blind proof π_{DLEQ}, a *time-correlation attack* will not work.

4.3 Option 3.

The user generates several pairs $(A_i = A^{l_i}, B_i = B^{l_i})$ and *anonymously* requests from the issuer, in advance, blind proofs π_{DLEQ} showing that $B_i = A_i^{sk_I}$ and stores these blind proofs for future use (and only uses them in the rare cases where both the user and the verifier are offline).

The blind proof π_{DLEQ} can be obtained using the Chaum-Pedersen seminal blind signature protocol [15], which is standardized in the ISO/IEC standard

[7] which roughly consists of a MAC computed by the card on the payment data elements such as the transaction amount and transaction date

18370-2 ([17], mechanism 4) and which represents the core cryptographic mechanism used in the anonymous credential scheme U-Prove. This blind signature protocol can be seen, in fact, as an Oblivious Issuance of proof (OIP for short), for the proof π_{DLEQ}.

Unfortunately, Benhamouda et al [7] have shown that this OIP is vulnerable to a parallel attack (ROS attack). Specifically, a user simultaneously running a large number (l) of Chaum-Pedersen blind signatures sessions with the issuer would be able, after these l sessions, to forge an additional valid signature/proof (and thus fraudulently obtain $l+1$ DL equality proofs instead of just l).

We therefore consider for Option 2 and Option 3, the Oblivious Issuance Proof proposed by Orrù et al. [28], which is *one more unforgeable* even in the concurrent setting. This OIP which we have adapted to our context and which we denote Blind π_{DLEQ} is described in Appendix B.
Orrù et al. proved that Blind π_{DLEQ} is one-more unforgeable (meaning that an adversary cannot forge blind π_{DLEQ} proofs even in the concurrent setting) under the $q+1$-DL assumption in the AGM and ROM models (Theorem 3 of [28]).
They also show that Blind π_{DLEQ} is *perfectly* oblivious provided that $\overline{B} = \overline{A}^x$ (Theorem 5 of [28]). The case $\bar{B} \neq \bar{A}^x$ could occur if, during issuance, a malicious issuer is successful in forging a false ZKP π_{DLEQ} that $A^x = B$ (whereas the equality does not hold). The success of this attack is bounded by the soundness of π_{DLEQ}. By perfectly oblivious, we mean that an adversary, even with unlimited computational power, will not be able to link back ($\overline{A}, \overline{B}$, π_{DLEQ}) to the corresponding issuing session.

This approach (option 3) is similar to that described in ISO mDL, where a user can obtain several verifiable credentials at once (in batch)[8] to prevent colluding RPs from tracing them. However, our option 3 provides *full unlinkability*, unlike the ISO mDL batch credential issuance approach.

In practice, Option 1 will be the preferred one, whereas Option 3 will be used in the rare cases where both the user and the verifier are offline. Option 2 could be used when the verifier is offline, but not the user (who would be online).

5 Distributed Computations on the User's Side

In practice, Secure Elements (SE) are relatively closed devices. Although most of them support common digital signature algorithms such as ECDSA, developers do not have the ability to implement new cryptographic functionalities for security reasons. Therefore, it is difficult to use these SEs for purposes other than what they were originally designed for (for example, to generate ECDSA signatures). As a result, an SE cannot "randomize" its own public and private keys because it has not been programmed to perform such operations. It cannot carry out these basic operations, even though they may seem straightforward: such as generating a random value r and computing $sk_{Blind} = sk + r \mod p$, its randomized private key. Similarly, an SE cannot generate the SoK π_{HB}, that

[8] A different credential must be used for each new VP.

is, a signature of knowledge of the discrete logarithm of pk_{Blind} in the base g (that is, of the private key $sk + r \mod p$).

In the following, we explain how the Secure Element (SE) of the user's mobile device and the associated mobile wallet application (referred to as M-Wallet) can jointly randomize the public key pk_{Blind} and compute the SoK π_{HB}. It is important to note that in practice only the secure hardware (SE) knows the private key sk corresponding to the user's public key pk.

We propose two variants of BBS#: in the first one, we assume that the digital signature algorithm supported by the SE is ECSchnorr [24], while in the second one, we assume that it is ECDSA [24].

5.1 Joint Computation of PK_{Blind} and π_{HB} with ECSchnorr

We assume that the SE supports the classical ECSchnorr digital signature algorithm [9], also known as ECSDSA in ISO/IEC 14888-3 standard. We will use an *additive blinding* of the SE private key sk.

It should be noted that in this standard, the so-called "weak" version of the Fiat-Shamir heuristic is implemented; however, in certain contexts, this version is vulnerable to an attack introduced by Bernhard et al. [8]. Although this attack does not apply in our context, we will nevertheless indicate how to use the so-called Strong version of the Fiat-Shamir heuristic (Strong FS) with ECSDSA (as specified in ISO/IEC 14888-3 standard). The attack by Bernhard et al. [8] does not apply to non-interactive proofs using the Strong version of the Fiat-Shamir heuristic.

The joint computation of the

$$\text{SoK}\pi_{HB} = \text{SoK}\{\alpha : pk_{Blind} = g^{\alpha}\}[nonce, pk_{Blind}]$$

could be performed in the following way (see Fig. 3):

1. The M-Wallet chooses a random value r and computes $pk_{Blind} = g^r pk = g^{sk+r}$ and transmits it to the SE along with the *nonce* sent by the verifier.
2. The SE will first compute a signature of knowledge (denoted π) of the discrete logarithm of pk in the base g (that is, its private key sk): $\pi = \text{SoK}\{\alpha : pk = g^{\alpha}\}[nonce, pk_{Blind}]$. The algorithm called ECSDSA will be used to compute this signature. This signature of knowledge is computed as follows using the ECSDSA algorithm. The SE generates a random value ω and computes $T = g^{\omega}$, $c = \mathcal{H}(T, nonce, pk_{Blind})$ and $\rho = \omega + csk \mod p$ where $\mathcal{H}$ denotes a cryptographic hash function (e.g., SHA-256). The SoK π consists of the pair (c, ρ): $\pi = (c, \rho)$. It is valid if $c' = \mathcal{H}\left(g^{\rho} \times pk^{-c}, nonce, pk_{Blind}\right) = c$ and invalid otherwise.
3. The SE transmits the SoK to the M-Wallet
4. The M-Wallet computes $\rho_{Blind} = \rho + c \times r = \omega + c \times sk + c \times r = \omega + c \times (sk + r) \mod p$.

[9] We would like to emphasize that the interactive version of ECSchnorr, which would greatly ease the holder binding feature, is currently not deployed on SE's.

The SoK $\pi_{HB} = (c,\ \rho_{Blind})$ is a valid ECSDSA signature on $(nonce, pk_{Blind})$ with respect to the public key pk_{Blind}.

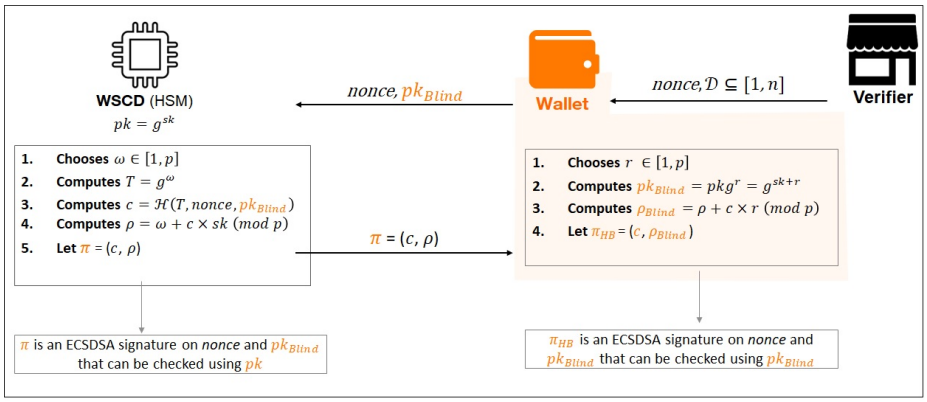


Fig. 3: Joint computation of pk_{Blind} and π_{HB} with ECSchnorr

5.2 Security of ECSchnorr Splitting

The security of the above distributed computation is well studied. In [19], Fleischhacker et al. proved that ECSDSA with additive blinding is unforgeable under re-randomized keys in the ROM if the DL problem in $\mathbb{G}$ is hard ([19], Theorem 1).

5.3 Joint Computation of pk_{Blind} and π_{HB} with ECDSA

This time, we will assume that the SE supports the classic digital signature algorithm ECDSA [FIPS186-4, ISO/IEC 14888 3]. We will use this time a multiplicative blinding of the SE private key sk.

The joint computation of the ECDSA signature (π_{HB}) on the message $(nonce, pk_{Blind})$ using the private key $sk_{Blind} = sk \times r \mod p$, could be performed in the following way (see Fig. 4) :

1. The M-Wallet chooses a random value r and computes $pk_{Blind} = pk^{r} = g^{sk \times r}$ and $M = r^{-1} \times \mathcal{H}(nonce, pk_{Blind}) \mod p$ and transmits M to the SE after authenticating itself with the latter.
2. The SE chooses a random value $k \in \mathbb{Z}_p^*$ and calculates $g^k = (i,\ j)$[10]. Let $x = i \mod p$.

[10] Here, we are abusively using multiplication (instead of addition) to denote the group operation in $\mathbb{G}$. The element g^k is therefore considered (abusively) as a point on the underlying elliptic curve.

3. If $x = 0$ then go back to step 2.
4. The SE calculates $\rho = k^{-1}(M + sk \times x) \mod p$.
5. If $\rho = 0$ go back to step 2. Otherwise, the SE transmits $\sigma_0 = (x,\ \rho)$ to the M-Wallet.
6. The M-Wallet computes
   $\rho_{Blind} = r \times \rho = k^{-1}(\mathcal{H}(nonce, pk_{Blind}) + sk_{Blind} \times x) \mod p$

The signature $\pi_{HB} = (x, \rho_{Blind})$ is a valid ECDSA signature on $(nonce, pk_{Blind})$ with respect to the public key pk_{Blind}.

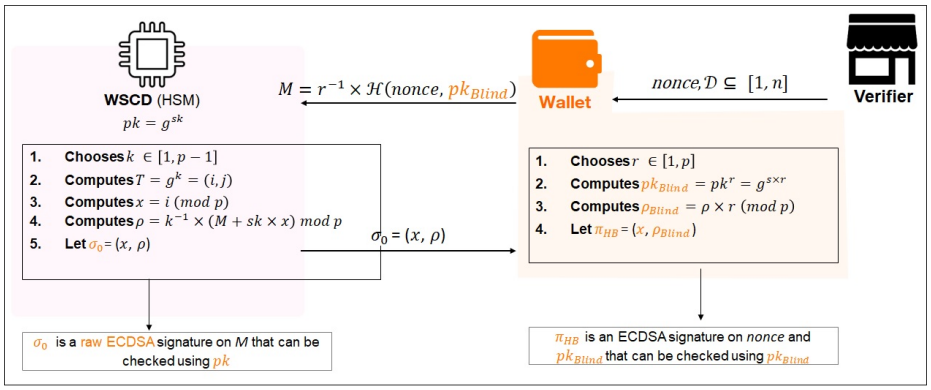


Fig. 4: Joint computation of pk_{Blind} and π_{HB} with ECDSA

As with ECSchnorr, the VP consists of:

$$VP = (\{m_i\}_{i \in \mathcal{D}},\ pk_{Blind},\ \pi_{HB},\ \overline{A}, \overline{B}, D,\ \pi_{validity}).$$

Only the SoK $\pi_{validity}$ differs when ECDSA is used on the user's side. In the ECDSA case, the proof $\pi_{validity}$ would be the following :

$$\pi_{validity} = \text{PoK}\{\alpha, \beta, \gamma, \delta, \{\tau_i\}_{i \notin \mathcal{D}} :\ \overline{B} = \overline{A}^{-\alpha} D^{\beta} \wedge F = PK_{Blind}{}^{\delta} D^{\gamma} \prod_{i \notin \mathcal{D}} g_i^{-\tau_i} \mod p\}$$

where, $F = g_0 \prod_{i \in \mathcal{D}} g_i^{m_i}$. We have the following two equalities, hence the validity proof $\pi_{validity}$: $\overline{B} = \overline{A}^{-e} D^{r_1}$ and $F = PK_{Blind}{}^{\tilde{r}} D^{r_3} \prod_{i \notin \mathcal{D}} g_i^{-m_i}$ where, $\tilde{r} = -r^{-1} \mod p$ et $r_3 = r_2^{-1} \mod p$.

5.4 Security of ECDSA Splitting

The particularity of our splitting technique described in Fig. 4 is that it makes use of *raw ECDSA* on the WSCD / SE side (*i.e.*, the signing queries are on $\mathcal{H}(m)$ instead on plain m) instead of the classical ECDSA (where the signing queries are on plain m).

Raw ECDSA is, however, supported by a majority of WSCDs: the iOS/Secure Enclave, Android-/HBK+Strongbox, TPMs, PKCS11 based HSMs for example, all support this functionality.

Another specificity of our splitting technique is that we include pk_{Blind} in the message to be signed by the WSCD (unlike SECDSA [35]). Without pk_{Blind}, our splitting technique would be vulnerable to a simple related-key attack (see for example [26] section 4.2)[11].

The security of our splitting mechanism, which is proven in the following, relies on the security of raw ECDSA[12] (*one-more unforgeability*).

By one-more unforgeability, we mean that an attacker after having requested l raw ECDSA signatures on l chosen hash values $H_1, H_2, \cdots, H_l$, will not be able to generate an additional signature on a hash value H that was not requested.

Proof (sketch). Suppose that an attacker $\mathcal{A}$ can break *the security of our splitting technique for ECDSA*. The attacker makes l raw ECDSA signing queries to the WSCD (with public key pk) on values $M_1, M_2, \cdots, M_l$, and generates $l+1$ *valid forgeries*, *i.e.*, $l+1$ tuples $(m_i, pk_i,\ r_i, \sigma_i)$ where σ_i is a valid ECDSA signature on (m_i, pk_i) with respect to the public key pk_i.

By valid forgeries, we mean a forge that can be used later by $\mathcal{A}$ to produce valid VP's that will be accepted by any verifier. As a valid VP is a proof of knowledge of a valid credential (see Appendix C)[13], this means that $\mathcal{A}$ knows a public key pk, that has been certified by an issuer, and for which pk_i is a randomized version: $pk_i = pk^{r_i}$.

We distinguish two types of adversary:

1. Type 1: $\exists\ i,\ j$ in [1, l+1] such that $r_i^{-1}\mathcal{H}(m_i,\ pk^{r_i}) = r_j^{-1}\mathcal{H}\left(m_j,\ pk^{r_j}\right)$.
2. Type 2: $r_i^{-1}\mathcal{H}(m_i,\ pk^{r_i}) \neq r_j^{-1}\mathcal{H}(m_j,\ pk^{r_j}), \forall\ i,\ j$ in [1, l+1].

If $\mathcal{H}$ is modeled as a random oracle, the advantage of any type 1 adversary $\mathcal{A}$ of finding such 'collisions' is less than $\frac{lq_{\mathcal{H}}}{p}$, where $q_{\mathcal{H}}$ is a bound on the number of queries to $\mathcal{H}$ made by $\mathcal{A}$.

Any type 2 adversary $\mathcal{A}$ can be converted into an adversary against the one-more unforgeability of raw ECDSA. Indeed, if $\sigma_i = (x_i,\ \rho^i_{Blind})$ is a valid signature on (m_i, pk_i) with private key $r_i sk$, then it follows that $\sigma_0^i = (x_i,\ r_i^{-1}\rho^i_{Blind})$ is a raw ECDSA signature on $M_i = r_i^{-1}\mathcal{H}(m_i,\ pk^{r_i})$ with private key sk.

[11] Which unfortunately applies to SECDSA [35].

[12] Groth and Shoup have shown that raw ECDSA signatures are one-more unforgeable in the *elliptic curve generic group model* [23].

[13] Although the proof in Appendix C applies when ECSchnorr is used on the WSCD side, it also applies when ECDSA is used on the WSCD side under the reasonable assumption that an ECDSA signature is a proof of knowledge of the underlying private signing key. This assumption is tacitly assumed in practice in the context of Self-Sovereign Identity (SSI).

As we have supposed that all the $M_i's$ are distinct for type 2 adversaries, then $\mathcal{A}$ would have break the one-more unforgeability of raw ECDSA.

Our splitting technique is therefore secure under the assumption that raw ECDSA signatures are unforgeable.

6 Performance

In this section, we present the overall performance of BBS#. We provide a detailed benchmark of our scheme across various devices, demonstrating its efficiency and suitability for the EUDI Wallet. Additionally, we conduct a comparative analysis in Appendix A in which we compare the efficiency of BBS# with $\mathrm{ISO-mDL/SD-JWT}$ (using the ECDSA signature scheme) and $\mathrm{PQ-ABC}$ [1], a recent post-quantum anonymous credential scheme that appeared at ACM CCS 2024. Our comparison focuses on key size and computation time for credential issuance and presentation. Furthermore, we provide a comparative analysis of the security and privacy offered by each scheme, including their resistance to quantum attacks (see Appendix A).

6.1 BBS# Benchmarks

In this section, we present the timings obtained from our C++ implementation of the BBS# protocol, as detailed in Table 1. These measures were taken on three different smartphones: an iPhone and two Android phones:

- iPhone X with a 2.39GHz Hexa-core processor, running iOS 16. The phone provides a "Secure Enclave" for cryptographic operations.
- Samsung A52S (SM-A528B) with a 2.4GHz Octa-core processor, running Android 14. The Keystore of this Android smartphone is TEE based.
- Samsung S24 Ultra (SM-S928B) with a 3.4GHz Octa-core processor, running Android 15. The Keystore of this Android smartphone is based on a "StrongBox", which offers hardware security comparable to a smartcard.

The NONEwithECDSA (a.k.a. raw ECDSA) signature generation is done by the mobile secure execution environment (SE) of the smartphone, namely the Keystore on Android and Secure Enclave on iOS. This means that the private key is managed and protected by that SE and may require biometric authentication for each use.

After randomization of the signature, the resulting SHA256withECDSA signature was verified using the standard cryptographic API provided by Android and iOS environments.

The operations specific to BBS# are done by our own C++ library, which uses MCL (https://github.com/herumi/mcl) for arithmetic operations over the elliptic curve. While MCL is "a portable and fast pairing-based cryptography library", it also supports the NIST curves. In our case, it was configured to use the NIST secp256r1 curve. As we have tested the whole BBS# protocol with

the use of Keystore/Secure Enclave on smartphones, we can confirm that it is indeed compatible.

For the measurements detailed in Table 1, we assume that the issuer will certify 10 attributes of the user and that the user will choose to reveal only one of them to a verifier. This attribute may be a tag "18" set to 1 (indicating that the user is over "18"), which, for example, the user may reveal to provide a proof of majority.

Each measure over the cryptographic library is the average over 1000 executions, except for the NONEwithECDSA signature generation (by the Keystore/Secure Enclave), which is the average over 100 executions. We did not use any parallelization of the operations.

For these tests, we ran the complete protocol on the smartphone. In a real wallet application, only the operations marked as (wallet) are supposed to run on it.

We have identified different steps of the validity proof generation protocol between the issuer and user/wallet with numbers (step1, step2, step3, and step4).[14] During this protocol, the wallet must generate some random values that will also be used to randomize the credentials. This corresponds to the line "randomize credentials step1 (wallet)" in Table 1.

These benchmarks confirm that BBS# is very fast on these smartphones.

In conclusion, the most costly operation is the NONEwithECDSA signature done by the SE of the smartphone, particularly when done by a StrongBox, but it remains fast enough not to be noticeable by a user.

7 Conclusion

The EU Digital Identity Wallet (EUDI Wallet), introduced by the eIDAS 2.0 regulation, is a digital identity solution for securely presenting personal identification data (PID) and verifiable credentials (a.k.a., Qualified Electronic Attestations of Attributes, QEAAs for short, in the eIDAS 2.0 terminology). eIDAS 2.0 mandates selective disclosure and unlinkability for privacy protection.

Anonymous credentials, which allow holders to prove statements about their identity in a privacy preserving way, is likely to become the key technology to meet the stringent requirements (no pairing-based cryptography and hardware-based holder binding) put forth in the eIDAS 2.0 regulation. Unfortunately, current efficient anonymous credentials protocols, such as BBS/BBS +, do not meet these requirements: they either make use of bilinear maps and pairing-friendly curves (which are not supported by current certified secure elements) or would require, to be rolled out, changes in these certified hardware or the algorithms they support, to implement the holder binding feature.

In this paper, we introduced BBS#, a pairing-free variant of the anonymous credentials scheme BBS, which can be used with 'classic' (non-pairing

[14] For our demonstrator, we have used Chaum-Pedersen's seminal blind signature protocol [15], rather than the OIP described in Appendix E, for the blind proof π_{DLEQ}. However, the performance of these two protocols is almost identical.

Table 1: Timings in milliseconds (ms) of the whole BBS# protocol obtained on different smartphones

Operations	**Devices**		
PKBlind generation	A52S	S24 Ultra	iPhone X
ECDSA public key randomization (wallet)	0.33	0.18	0.24
ECDSA prepare data to sign (wallet)	0.035	0.02	0.12
NONEwithECDSA signature generation (SE)	11.3	69	11.4
ECDSA signature randomization (wallet)	0.014	0.016	0.25
Issuance	A52S	S24 Ultra	iPhone X
BBS+ signature over attributes (issuer)	1.3	1	1.75
Verification of issuer signature (wallet)	1.42	1.1	2.7
Blind validity proof generation (user $\leftrightarrow$ issuer)	A52S	S24 Ultra	iPhone X
step1 (issuer)	0.39	0.30	0.54
step2 (wallet)	0.96	0.73	1.33
step3 (issuer)	$<$0.01	$<$0.01	$<$0.01
step4 (wallet)	$<$0.01	$<$0.01	$<$0.01
VP generation	A52S	S24 Ultra	iPhone X
Randomize credentials step1 (wallet)	0.59	0.46	0.77
Randomize credentials step2 (wallet)	1.17	0.85	1.6
Total generation time	1.76	1.31	2.37
VP Verification	A52S	S24 Ultra	iPhone X
ECDSA signature	0.26	0.11	0.18
Blind validity proof	0.51	0.36	0.6
Blind credentials signature	1.02	0.73	1.4
Total verification time	1.8	1.2	2.2

friendly) elliptic curves, and more importantly with current 'Wallet Secure Cryptographic Devices' (WSCD): iOS/Secure Enclave, Android-/HBK+Strongbox, TPMs, PKCS11 based HSMs.

BBS# is provably secure; it inherits the security of BBS, of Oblivious Issuance of Proofs and of the security of ECDSA with multiplicative key randomization.

Finally, our implementation results confirm the efficiency and suitability of BBS# for the forthcoming EUDI Wallet.

References

1. Argo, S., Güneysu, T., Jeudy, C., Land, G., Roux-Langlois, A., Sanders, O.: Practical post-quantum signatures for privacy. Cryptology ePrint Archive (2024)
2. Baldimtsi, F., Lysyanskaya, A.: Anonymous credentials light. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. pp. 1087–1098 (2013)
3. Barić, N., Pfitzmann, B.: Collision-free accumulators and fail-stop signature schemes without trees. In: International conference on the theory and applications of cryptographic techniques. pp. 480–494. Springer (1997)
4. Barki, A., Brunet, S., Desmoulins, N., Traoré, J.: Improved algebraic macs and practical keyed-verification anonymous credentials. In: Selected Areas in Cryptography–SAC 2016: 23rd International Conference, St. John's, NL, Canada, August 10-12, 2016, Revised Selected Papers 23. pp. 360–380. Springer (2017)
5. Bauer, B., Fuchsbauer, G., Loss, J.: A classification of computational assumptions in the algebraic group model. In: Annual International Cryptology Conference. pp. 121–151. Springer (2020)
6. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Proceedings of the 1st ACM Conference on Computer and Communications Security. pp. 62–73 (1993)
7. Benhamouda, F., Lepoint, T., Loss, J., Orrù, M., Raykova, M.: On the (in) security of ros. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 33–53. Springer (2021)
8. Bernhard, D., Pereira, O., Warinschi, B.: How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios. In: Advances in Cryptology–ASIACRYPT 2012: 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings 18. pp. 626–643. Springer (2012)
9. Brands, S.A.: An efficient off-line electronic cash system based on the representation problem (1993)
10. Camenisch, J., Lysyanskaya, A.: A signature scheme with efficient protocols. In: Security in Communication Networks: Third International Conference, SCN 2002 Amalfi, Italy, September 11–13, 2002 Revised Papers 3. pp. 268–289. Springer (2003)
11. Camenisch, J., Shoup, V.: Practical verifiable encryption and decryption of discrete logarithms. In: Annual International Cryptology Conference. pp. 126–144. Springer (2003)
12. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups. In: Annual international cryptology conference. pp. 410–424. Springer (1997)

13. Chase, M., Meiklejohn, S., Zaverucha, G.: Algebraic macs and keyed-verification anonymous credentials. In: Proceedings of the 2014 acm sigsac conference on computer and communications security. pp. 1205–1216 (2014)
14. Chaum, D.: Security without identification: Transaction systems to make big brother obsolete. Communications of the ACM **28**(10), 1030–1044 (1985)
15. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Annual international cryptology conference. pp. 89–105. Springer (1992)
16. Durak, F.B., Marco, L., Talayhan, A., Vaudenay, S.: Non-transferable anonymous tokens by secret binding. Cryptology ePrint Archive (2024)
17. Fal', O.: Standardization in information technology security. Cybernetics and Systems Analysis **53**, 78–82 (2017)
18. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Conference on the theory and application of cryptographic techniques. pp. 186–194. Springer (1986)
19. Fleischhacker, N., Krupp, J., Malavolta, G., Schneider, J., Schröder, D., Simkin, M.: Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. Cryptology ePrint Archive, Paper 2015/395 (2015), https://eprint.iacr.org/2015/395
20. Fuchsbauer, G., Hanser, C., Slamanig, D.: Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. Journal of Cryptology **32**, 498–546 (2019)
21. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part II 38. pp. 33–62. Springer (2018)
22. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems. In: Providing sound foundations for cryptography: On the work of shafi goldwasser and silvio micali, pp. 203–225 (2019)
23. Groth, J., Shoup, V.: On the security of ECDSA with additive key derivation and presignatures. Cryptology ePrint Archive, Paper 2021/1330 (2021), https://eprint.iacr.org/2021/1330
24. ISO: Iso/iec 14888-3: 2018 it security techniques-digital signatures with appendix-part 3: discrete logarithm based mechanisms (2018)
25. Looker, T., Kalos, V., Whitehead, A., Lodder, M.: The BBS Signature Scheme. Internet-Draft draft-irtf-cfrg-bbs-signatures-05, Internet Engineering Task Force (Dec 2023), https://datatracker.ietf.org/doc/draft-irtf-cfrg-bbs-signatures/05/, work in Progress
26. Morita, H., Schuldt, J.C., Matsuda, T., Hanaoka, G., Iwata, T.: On the security of the schnorr signature scheme and dsa against related-key attacks. In: Information Security and Cryptology-ICISC 2015: 18th International Conference, Seoul, South Korea, November 25-27, 2015, Revised Selected Papers 18. pp. 20–35. Springer (2016)
27. Neven, G.: Ibm identity mixer (idemix). In: NIST Meeting on Privacy Enhancing Technology, Zurich, Switzerland. pp. 8–9 (2011)
28. Orrù, M., Tessaro, S., Zaverucha, G., Zhu, C.: Oblivious issuance of proofs. In: Annual International Cryptology Conference. pp. 254–287. Springer (2024)
29. Paquin, C., Zaverucha, G.: U-prove cryptographic specification v1. 1. Technical Report, Microsoft Corporation (2011)
30. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. Journal of cryptology **13**, 361–396 (2000)

31. Sanders, O.: Efficient redactable signature and application to anonymous credentials. In: IACR International Conference on Public-Key Cryptography. pp. 628–656. Springer (2020)
32. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Advances in Cryptology—CRYPTO'89 Proceedings 9. pp. 239–252. Springer (1990)
33. Shoup, V.: Sequences of games: a tool for taming complexity in security proofs. cryptology eprint archive (2004)
34. Tessaro, S., Zhu, C.: Revisiting bbs signatures. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 691–721. Springer (2023)
35. Verheul, E.: SECDSA: Mobile signing and authentication under classical "sole control". Cryptology ePrint Archive, Paper 2021/910 (2021), https://eprint.iacr.org/2021/910

A Comparative Analysis

A.1 Efficiency

We provide a comparison of *space efficiency* in bytes for BBS#, $\mathrm{ISO-mDL/SD-JWT}$ (using the ECDSA signature scheme on both the Holder's and Issuer's sides), and $\mathrm{PQ-ABC}$, as shown in Table 2. We also present a comparison of *time efficiency*, detailed in Table 3. Our estimates are based on counting the number of multi-exponentiations required to perform an operation for BBS# and $\mathrm{ISO-mDL/SD-JWT}$. We use the notation $\mathbb{E}_{\mathbb{G}}$ to denote the cost of an exponentiation/scalar multiplication in $\mathbb{G}$, which is evaluated at $63\mu s$ on an Intel(R) Core(TM) i7-8565U CPU running at 1.80GHz, $0.2ms$ on a Samsung S10e over the secp256r1 curve, and $50ms$ on a Javacard 2.2.2 SIM card, Global Platform 2.2 compliant, over the secp256r1 curve.

$\mathrm{PQ-ABC}$ is benchmarked on an Intel Core i7 12800H CPU running at 4.6 GHz. N.A. in Table 3 indicates that current WSCDs do not support the computations involved in $\mathrm{PQ-ABC}$ [1].

In Table 2 and Table 3, the parameter N denotes the number of signed attributes, and in Table 2, the parameter U denotes the number of undisclosed attributes.

We do not consider operations in $\mathbb{Z}_p$ since their cost is negligible compared to the other operations.

Table 2: Space Efficiency (bytes)

Schemes	Private Key (Holder, Issuer)	Public Key (Holder, Issuer)	Credential Size	Presentation Proof
BBS#	(32, 32)	(32, 32)	128	$416+U$ x 32
$\mathrm{ISO-mDL}$	(32, 32)	(32, 32)	64	$64+N$ x 32
$\mathrm{PQ-ABC}$	(0.25 KB, 10 KB)	(2.38 KB, 47.53 KB)	6.81 KB	79.58 KB

Table 3: Time Efficiency. $N = 10$

Schemes	Credential Issuance	Present WSCD	Present Wallet	Verify Presentation
BBS#	$N\mathbb{G}_1$ ($630\mu s$)	$1\mathbb{G}_1$ ($50ms$)	$(N+9)\mathbb{G}_1$ ($3.8ms$)	$(N+12)\mathbb{G}_1$ ($1.4ms$)
$ISO-mDL$	$1\mathbb{G}_1$ ($63\mu s$)	$1\mathbb{G}_1$ ($50ms$)	-	$2\mathbb{G}_1$ ($126\mu s$)
$PQ-ABC$	$400ms$	N.A.	$355ms$	$147ms$

A.2 Functionality and Properties Comparison

In Table 4, we compare the three aforementioned protocols (BBS#, $ISO-mDL$ with ECDSA, and $PQ-ABC$) in terms of the level of security and privacy they provide.

The *NPQ Assumption* refers to classical assumptions that are not quantum resistant, such as the q-SDH assumption in the case of BBS#. *NPQ security* indicates that the scheme does not offer a quantum-resistant level of security. *Unconditional or Everlasting Privacy* means that the scheme provides anonymity even against an adversary with unbounded computational power. Additionally, the $ISO-mDL$ is implemented with ECDSA on both the Holder's and Issuer's sides

Table 4: Security and Privacy Comparison

Shemes	Credential Unforgeability	VP Unlinkability Colluding RPs	VP Unlinkability Colluding RP-Issuer	VP Unforgeability
BBS#	NPQ Assumption	Unconditional Privacy	Unconditional Privacy	NPQ Assumption
SD-JWT and mDL	Unknown Assumption (NPQ security)	No	No	Unknown Assumption (NPQ security)
$PQ-ABC$	PQ Assumption	PQ Assumption	PQ Assumption	PQ Assumption

B A Zero Knowledge Proofs of Knowledge

In this Appendix, we describe the main ZKPK used in our construction, namely π_U (Fig. 5), π_{DLEQ} (Fig. 6), π_{HB} (Fig. 7) and $\pi_{validity}$ (Fig. 8). The $\mathcal{H}_i$'s will denote suitable hash functions.

Prover(pp, sk, pk)

1. $\alpha \in_R \mathbb{Z}_p$;
2. $com = g^{\alpha}$;
3. $c = \mathcal{H}_1(pp,\ pk,\ com)$;
4. $\rho = \alpha + csk \mod p$;
5. Return (c, ρ);

Verifier(pp, pk)

1. $com' = g^{\rho} pk^{-c}$;
2. $c' = \mathcal{H}_1(pp,\ pk,\ com')$;
3. Return $(c' == c)$;

Fig. 5: π_U

Prover$(pp, A, B, \tilde{g}, sk_I, pk_I)$

1. $\alpha \in_R \mathbb{Z}_p$;
2. $com_1 = \tilde{g}^{\alpha}$;
3. $com_2 = A^{\alpha}$;
4. $c = \mathcal{H}_2(pp,\ A,\ B,\ \tilde{g}, pk_I, com_1, com_2)$;
5. $\rho = \alpha + csk_I \mod p$;
6. Return (c, ρ);

Verifier(pp, A, B, pk_I)

1. $com'_1 = \tilde{g}^{\rho} {pk_I}^{-c}$;
2. $com'_2 = A^{\rho} B^{-c}$;
3. $c' = \mathcal{H}_2(pp,\ A,\ B,\ \tilde{g}, pk_I, com'_1, com'_2)$;
4. Return $(c' == c)$;

Fig. 6: π_{DLEQ}

Prover(pp, r, sk, pk_{Blind})

1. $\alpha \in_R \mathbb{Z}_p$;
2. $com = g^{\alpha}$;
3. $c = \mathcal{H}_3(pp,\ nonce,\ pk_{Blind},\ com)$;
4. $\rho = \alpha + c(sk + r) \mod p$;
5. Return (c, ρ);

Verifier(pp, A, B, pk_I)

1. $com' = g^{\rho} pk_{Blind}^{-c}$;
2. $c' = \mathcal{H}_3(pp,\ nonce,\ pk_{Blind},\ com')$;
3. Return $(c' == c)$;

Fig. 7: π_{HB}

Prover$(pp, \overline{A}, \overline{B}, D, \{m_i\}_{i=1}^n, pk_{Blind}, e, r_1, r_2, r_3, r, \mathcal{D})$

1. $\alpha, \beta, \gamma, \delta, \{\tau_i\}_{i \notin \mathcal{D}} \in_R \mathbb{Z}_p$;
2. $com_1 = \overline{A}^{\alpha} D^{\beta}$;
3. $com_2 = D^{\gamma} \prod_{i \notin \mathcal{D}} g_i^{\tau_i} g^{\delta}$;
4. $c = \mathcal{H}_4(pp,\ nonce,\ \overline{A},\ \overline{B},\ D,\ \{m_i\}_{i=1}^n, pk_{Blind}, com_1, com_2)$;
5. $\rho_1 = \alpha - ce \mod p$;
6. $\rho_2 = \beta + cr_1 \mod p$;
7. $\rho_3 = \gamma + cr_3 \mod p$;
8. For $i \notin \mathcal{D}$;
9. $resp_i = \tau_i - cm_i \mod p$;
10. $resp = \delta + cr \mod p$;
11. Return $(c, \rho_1, \rho_2, \rho_3, \{resp_i\}_{i \notin \mathcal{D}},\ resp)$;

Verifier$(pp, \overline{A}, \overline{B}, D, \{m_i\}_{i=1}^n, pk_{Blind}, \mathcal{D}, \{m_i\}_{i \in \mathcal{D}})$

1. $com_1' = \overline{A}^{\rho_1} D^{\rho_2} \overline{B}^{-c}$;
2. $com_2' = D^{\rho_3} \prod_{i \notin \mathcal{D}} g_i^{resp_i} g^{resp} P^{-c}$ where $P = g_0 pk_{Blind} \prod_{i \in \mathcal{D}} g_i^{m_i}$;
3. $c' = \mathcal{H}_4(pp,\ nonce,\ \overline{A},\ \overline{B},\ D,\ \{m_i\}_{i=1}^n, pk_{Blind}, com_1', com_2')$;
4. Return $(c' == c)$;

Fig. 8: $\pi_{validity}$

C Proof of Knowledge of a Valid Credential

We will show that if a VP is accepted by a verifier, then this implies that the user knows a valid VC on the attributes $\{m_i\}_{i\in\mathcal{D}}$, as well as the private key associated with that VC.

Proof: Let $VP = (\{m_i\}_{i\in\mathcal{D}}, pk_{Blind}, \pi_{HB}, \overline{A}, \overline{B}, D, \pi_{validity})$ be a given VP. We recall that $\pi_{HB} = \text{SoK}\{\alpha : \ pk_{Blind} = g^{\alpha}\}[nonce]$. Therefore, π_{HB} is a proof of knowledge of the discrete logarithm of PK_{Blind} in the base g. By using the *extractor* for this proof of knowledge, we can extract sk_{Blind} such that:

$$pk_{Blind} = g^{sk_{Blind}} \tag{1}$$

By using the extractor for the $\pi_{validity}$ proof, we can extract values $e, r, \ r_1, r_3$, $\{m_i\}_{i=1}^{n}$ such that:

$$\overline{B} = \overline{A}^{-e} D^{r_1} \tag{2}$$

$$g_0 pk_{Blind} \prod_{i\in\mathcal{D}} g_i^{m_i} = D^{r_3} \prod_{i\notin\mathcal{D}} g_i^{-m_i} g^{r} \tag{3}$$

We also know that if the VP is accepted by the verifier, then:

$$\overline{B} = \ \overline{A}^{sk_I} \tag{4}$$

(2) and (4) therefore implies that:

$$\overline{A}^{sk_I+e} = D^{r_1} \tag{5}$$

If $r_1 = 0 \mod p$, (5) implies that $sk_I = -e \mod p$ and therefore our reduction has found the private key of the issuer. This is impossible under the discrete logarithm assumption. Therefore, we will assume that $r_1 \neq 0 \mod p$.
From (3) and (1), we can deduce that:

$$D^{r_3} = g_0 g^{sk_{Blind}-r} \prod_{i=1}^{n} g_i^{m_i} \tag{6}$$

1. If $r_3 = 0 \mod p$ then $g_0 = g^{r-sk_{Blind}} \prod_{i=1}^{n} g_i^{-m_i}$. We have therefore found a representation of g_0 in the base $(g, \{g_i\}_{i=1}^{n})$. This is also impossible under the discrete logarithm assumption. Therefore, we will assume that $r_3 \neq 0 \mod p$.
2. If $r_3 \neq 0 \mod p$, from the equalities (5) and (6), we obtain:

$$\left(\overline{A}^{r_3 r_1^{-1}}\right)^{sk_I+e} = g_0 g^{sk_{Blind}-r} \prod_{i=1}^{n} g_i^{m_i} \tag{7}$$

Therefore, we have extracted a valid MAC_{BBS} authentication tag $(\overline{A}^{r_3 r_1^{-1}}, e)$ on $(sk_{Blind} - r, \{m_i\}_{i=1}^{n})$.

Under the gap DL and gap q-DL assumptions, the user therefore holds a valid MAC_{BBS} authentication tag (*i.e.*, a VC) on the $\{m_i\}_{i\in D}$. The $\{m_i\}_{i\in D}$ disclosed to the verifier have thus been certified by the issuer and presented by the user to whom the corresponding VC was issued (as the user proved that he knows the underlying key $sk = sk_{Blind} - r$).

D Proof of Game 5

We show that $\mathcal{R}$, in *Game 5* of the anonymity proof, perfectly simulates this game using the random credential (A^*, e^*). In other words, this means that the output of $(Show(pk_I,\ sk^*,\ \{m_i^*\}_{i=1}^n, \mathcal{D}, \sigma^*),\ \mathcal{A}(\ ))$ reveals no information about b.

Let $VP = (\{m_i^*\}_{i\in\mathcal{D}}, pk_{Blind}, \pi_{HB}, \overline{A}, \overline{B}, D, \pi_{validity})$ be the VP generated using the credential $\sigma^* = (A^*, e^*)$ instead of the one of the user j_b.

Let $VC' = (A,\ e)$ be the VC obtained by the honest user j_b on the secret key sk and the attributes $(m_1,\ m_2,\ m_3, \ldots,\ m_n)$. We suppose that for all $i \in \mathcal{D}$, we have: $m_i = m_i^*$.

As D is an element of a cyclic group of order p and $D \neq 1$ (the neutral element of this group, considered here as multiplicative), this implies that there exists an integer $r_2' \in \mathbb{Z}_p^*$ such that $D = C_m^{r_2'}$ where $C_m = g_0 pk \prod_{i=1}^n g_i^{m_i}$.

For the same reasons, there exists an integer $r_1' \in \mathbb{Z}_p^*$ such that $\overline{A} = A^{r_1' r_2'}$.

Let us show that $\overline{B} = \overline{A}^{-e} D^{r_1'} = \overline{A}^{sk_I}$.

By definition: $A^{sk_I + e} = C_m$ and thus $A^{sk_I} = C_m A^{-e}$. This therefore implies that:

$$\overline{A}^{sk_I} = A^{sk_I r_1' r_2'} = C_m^{r_1' r_2'} A^{-e r_1' r_2'} = D^{r_1'} \overline{A}^{-e} = \overline{B}$$

Furthermore, there exists $r \in \mathbb{Z}_p^*$ such that $pk_{Blind} = g^{sk^* + r^*} = g^{sk+r}$. π_{HB} is therefore a valid SoK produced with the private key $sk^* + r^* = sk + r \mod p$.

Let us also show that: $g_0 pk_{Blind} \prod_{i\in\mathcal{D}} g_i^{m_i^*} = D^{r_3'} \prod_{i\notin\mathcal{D}} g_i^{-m_i} g^r$ where $r_3' = r'^{-1}_2 \mod p$.

By definition:

$$D^{r_3'} = C_m = g_0 pk \prod_{i=1}^n g_i^{m_i} = g_0 pk_{Blind} \prod_{i=1}^n g_i^{m_i} g^{-r}$$

Therefore, we have $g_0 pk_{Blind} \prod_{i\in\mathcal{D}} g_i^{m_i^*} = D^{r_3'} \prod_{i\notin\mathcal{D}} g_i^{-m_i} g^r$since $m_i = m_i^*$ for all $i \in \mathcal{D}$.

Given that the proof $\pi_{validity}$ is *witness-indistinguishable*, it reveals no information (even to an attacker with unbounded computational power) about the elements $(\alpha, \beta, \gamma, \delta, \{\tau_i\}_{i\notin\mathcal{D}})$ used to produce the proof $\pi_{validity}$.

Therefore, an attacker, even with unbounded computational power, has no way to determine which credential was used to generate the $VP = (\{m_i^*\}_{i\in\mathcal{D}},\ pk_{Blind}, \pi_{HB}, \overline{A}, \overline{B}, D,\ \pi_{validity})$. This concludes the proof that that the output of $(Show(pk_I,\ sk^*,\ \{m_i^*\}_{i=1}^n, \mathcal{D}, \sigma^*),\ \mathcal{A}(\ ))$ reveals no information about b.

E Oblivious Issuance Proof of Discrete Logarithm Equality

In this Appendix section, we describe how to issue securely and obviously a π_{DLEQ} proof, even in a concurrent setting. We use the Oblivious Issuance Proof

proposed by Orrù et al. ([28], Fig. 9), with slight adaptations to our pairing-free anonymous credentials system. In particular, the pair $(\overline{A}, \overline{B})$ used on the user's side should be computed as in a Show protocol. More precisely if the pair (A, e) represents the user's credential on their public key pk and their attributes $\{m_i\}_{i=1}^n$, they should choose two integers $r_1, r_2 \in \mathbb{Z}_p^*$ and compute:

- $\overline{A} = A^{r_1 \times r_2}$,
- $D = C_m{}^{r_2}$, where $C_m = g_0 pk \prod_{i=1}^n g_i^{m_i}$.
- $\overline{B} = \overline{A}^{-e} D^{r_1} = \overline{A}^{sk_I}$, where sk_I is the issuer's private key.

In the sequel, G and H will denote two public random generators (in pp) and $\mathcal{H}$ a suitable hash function.

E.1 Verification of the Proof $\pi_{\mathbf{DLEQ}}$.

The verification algorithm of π_{DLEQ} is given in Fig. 10

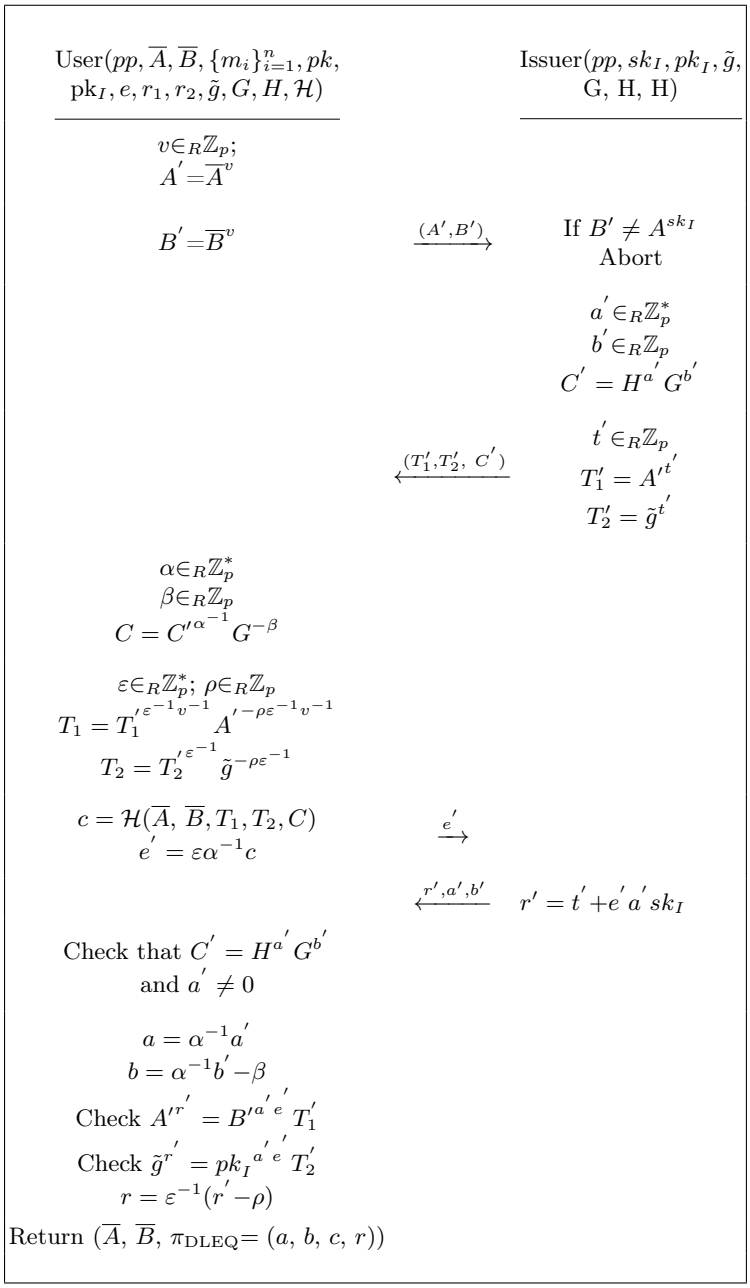


Fig. 9: Oblivious Issuance of the Proof π_{DLEQ}

Verifier$(pp, \overline{A}, \overline{B}, \pi_{\text{DLEQ}} = (a, b, c, r), pk_I, \tilde{g}, G, H, \mathcal{H})$

1. $C^* = H^a G^b$;
2. $T_1^* = \overline{A}^r \overline{B}^{-ca}$;
3. $T_2^* = \tilde{g}^r {pk_I}^{-ca}$;
4. $c^* = \mathcal{H}(\overline{A}, \overline{B}, T_1^*, T_2^*, C)$;
5. Return $(c^* == c)$

Fig. 10: Verification of the proof π_{DLEQ}