Server-Aided Anonymous Credentials

Rutchathon Chairattana-Apirom[1(✉)], Franklin Harding[2], Anna Lysyanskaya[2], and Stefano Tessaro[1]

[1] Paul G. Allen School of Computer Science & Engineering, University of Washington, Seattle, USA
{rchairat,tessaro}@cs.washington.edu
[2] Brown University, Providence, RI, USA
{franklin_harding,anna_lysyanskaya}@brown.edu

Abstract. This paper formalizes the notion of server-aided anonymous credentials (SAACs), a new model for anonymous credentials (ACs) where, in the process of showing a credential, the holder is helped by additional auxiliary information generated in an earlier (anonymous) interaction with the issuer. This model enables lightweight instantiations of *publicly verifiable* and *multi-use* ACs from pairing-free elliptic curves, which is important for compliance with existing national standards. A recent candidate for the EU Digital Identity Wallet, BBS#, roughly adheres to the SAAC model we have developed; however, it lacks formal security definitions and proofs.

In this paper, we provide rigorous definitions of security for SAACs, and show how to realize SAACs from the weaker notion of keyed-verification ACs (KVACs) and special types of oblivious issuance protocols for zero-knowledge proofs. We instantiate this paradigm to obtain two constructions: one achieves statistical anonymity with unforgeability under the Gap q-SDH assumption, and the other achieves computational anonymity and unforgeability under the DDH assumption.

1 Introduction

Anonymous credentials (ACs), introduced by Chaum [25], allow a user (or *holder*) to obtain a credential from an *issuer*. Typically, a credential is associated with a number of attributes, such as the credential's expiration date, or the credential holder's date of birth. This credential can be *shown* to a verifier unlinkably, i.e. such that it cannot be linked to the transaction in which it was issued, and different showings of the same credential cannot be linked to each other. Further, a showing only reveals the minimum necessary amount of information about the attributes—typically, that these attributes satisfy a certain relevant predicate (e.g., that the holder is not a minor, that they have a valid driver's license, etc.).

ACs were first practically realized by Camenisch and Lysyanskaya [17–19]. In the standard approach to designing ACs [32,33], a credential is a *signature* on

The full version of this work can be found at [21].

Y. Tauman Kalai and S. F. Kamara (Eds.): CRYPTO 2025, LNCS 16005, pp. 291–324, 2025.
https://doi.org/10.1007/978-3-032-01887-8_10

the user's attributes, generated by the issuer via a secure protocol that protects the privacy of the user's attributes. Credentials are shown via a zero-knowledge proof of knowledge of a credential whose attributes satisfy the relevant predicate. In principle, one can build ACs from any signature scheme by using generic zero-knowledge proof systems, but in a practical instantiation, a digital signature scheme which enables efficient realizations of such proofs is a better approach. Examples include RSA- and pairing-based CL signatures [18,19], as well as pairing-based BBS signatures [3,12,19,41].

Systems using ACs have been proposed over the years, such as Microsoft's U-Prove [13,39] and IBM's IDEMIX [20]. Recently, credentials have regained popularity as components of decentralized/self-sovereign identity services like Hyperledger Indy, Veramo and Okapi. These come with ongoing companion standardization efforts by the IETF [31] and the World Wide Web Consortium (W3C). Technology policy, especially that of the EU and its member states, has mandated privacy-preserving authentication [1,2] for which anonymous credentials appear to be the right solution [7].

Credentials based on pairing-free elliptic curves. Elliptic-curve-based cryptography has outperformed and outpaced cryptographic constructions based on RSA. Especially desirable from the practical point of view – both for efficiency reasons and because of standardized curves – is elliptic-curve-based cryptography that does not require pairing-friendly curves [5,10]. The lack of suitable standards[1], in particular, often prevents the use of pairing-based solutions in the public sector, where ACs find a natural use case. Other natural application scenarios are web applications and anonymous browsing, and pairings are often not supported by browser libraries such as NSS and BoringSSL. Unfortunately, however, the only approach to (multi-show) ACs based on pairing-free curves relies on generic zero-knowledge proofs, and is mostly very costly, and this is due to the fact that pairing-free signature schemes are inherently non-algebraic (as proved e.g. in [26]). To overcome this inherent barrier, prior works have considered different settings where pairing-free ACs are possible:

- *Blind signatures with attributes.* Baldimtsi and Lysyanskaya [4] presented an approach extending the notion of blind signatures to include attributes, formalizing ideas implicit in U-Prove [39]. The resulting construction gives a use-once AC, referred to as "AC light" (ACL), i.e., one needs to interact with the issuer to obtain as many copies of the credential as the number of intended showings. This also introduces a tradeoff between privacy and efficiency: either each user needs to get as many copies of the ACL credential as a reasonable upper bound on the lifetime use of the credential, or it needs to get credentials reissued upon running out of them, revealing the rate of credential use.
- *Keyed-Verification Anonymous Credentials (KVAC).* The single-use aspect of ACL can be a feature, but is mostly a bottleneck. Chase, Meiklejohn and Zaverucha [23] considered multi-use credentials in an alternative setting where

[1] For example, the IETF draft for pairing-friendly curves expired in 2023 [40].

the issuer and the verifier are the same entity, and provided pairing-free solutions that rely on the lack of public verifiability when showing credentials. The resulting schemes are very practical, and are widely adopted in the Signal messaging system [24].

This paper: Server-aided anonymous credentials. This paper formalizes an alternative model for multi-use credentials in which efficient pairing-free credentials are possible, and which we refer to as *Server-Aided Anonymous Credentials* (or SAAC, for short). In contrast to KVAC, SAAC enable publicly verifiable showing of credentials, and this is achieved by allowing the holder to interact with the issuer's helper server to generate additional helper proofs. To preserve anonymity, this interaction with the helper is entirely oblivious (in a way related, but not formally equivalent, to the work of Orrú et al. [37]): the helper server does not need to verify anything about the user it is interacting with, and can neither link the interaction to any other by the same user, nor learn anything about the user's credential attributes. The extra cost of this interaction with the helper is limited, in particular as the generation of these proofs can be performed offline, and not at the time of showing the credential.

The helper flow is somewhat natural in the context of credentials. In OAuth 2.0 [28], the industry-standard authorization protocol for the web, users obtain a *refresh token* and must query that refresh token to an issuer to obtain *access tokens* which they can later spend. However, in the setting of anonymous credentials, the use of a helper server was, to the best of our knowledge, only recently brought up in the BBS# white paper [36,42]. BBS# is an industry white paper that explores several ideas for the development of a European Digital Identity Wallet.[2] However, it does not contain a formal security model or analysis. As a result, we are the first to provide the foundations behind such an approach, as well as provably secure solutions.

This work develops a formal treatment of SAAC, for which we give security definitions. We also develop generic constructions that lift KVACs, which are not meant to be publicly verifiable, to SAAC with the help of specific protocols for oblivious issuance of zero-knowledge proofs. Interestingly, our security needs for the latter are weaker than those considered by the recent work of Orrù et al. [37], as our helper protocol is not required to resist strong attacks such as ROS [9], and thus we can prove security based on a standard cryptographic assumption without relying on the algebraic group model (AGM) [29].

We instantiate our framework with two concrete constructions: A first solution based on BBS (without pairings), which we prove unforgeable, in the random-oracle (RO) model, under the Gap q-SDH assumption, and statistically anonymous. We also present a second instantiation for which both unforgeability and anonymity hold under the DDH assumption in the RO model. Our security analysis is in the random oracle model [8], but does not make any use of the AGM or any other ideal group model.

[2] BBS# includes other ideas besides including a helper server; and in particular integration with an HSM, which are outside the scope of this paper.

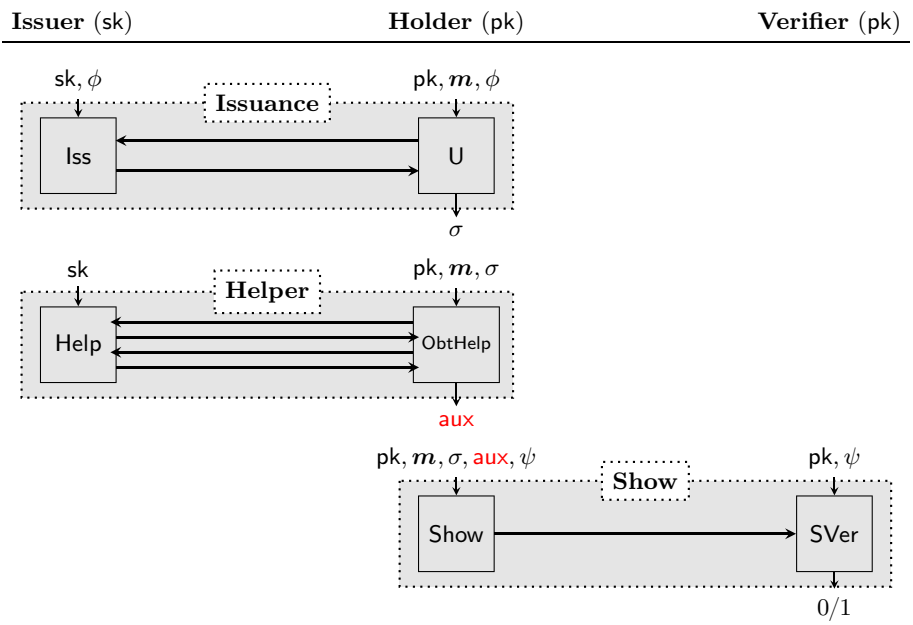


Fig. 1. Server-Aided Anonymous Credentials. Illustration of the SAAC setting. Note that the secret and public keys (sk, pk) are generated by the KeyGen algorithm, which is not described here. Also, we allow each showing to be linked to some additional value nonce, which is a joint input of Show and SVer, and this is not illustrated here.

The next section provides a detailed overview of our contributions.

1.1 Overview of This Paper

We now give a detailed overview of our results and contributions. This section also serves as a roadmap for the paper.

Syntax for SAAC. We provide a definition of *Server-Aided Anonymous Credentials* (SAAC). A SAAC scheme is parameterized by a set of predicates Φ, and consists of a number of protocols, involving the *issuer*, the *credential holders*, and the *verifier*. The setting is also defined in Fig. 1. For simplicity, both issuance and showing predicates come from the same space Φ using our syntax.

- **Key generation.** The issuer generates a secret-key/public-key pair (sk, pk) by running the key generation algorithm.
- **Issuance.** A credential σ is issued to the holder as the output of an interaction with the issuer—in the same way as with a classical credential system. The issuer's input is sk, whereas the holder's inputs are pk and a vector of attributes $\boldsymbol{m}$. Further, their shared input is an issuance predicate $\phi \in \Phi$. The intuition (which will be a consequence of our security notions we introduce

below) is that the credential is only issued if indeed $\phi(\boldsymbol{m}) = 1$, and that the issuer only learns ϕ where $\phi(\boldsymbol{m}) = 1$. The holder's output is a credential σ.

- **Helper protocol.** The main new component is a *helper protocol* between a holder and the issuer. The issuer's input is sk, whereas the holder's inputs are pk, a vector of attributes $\boldsymbol{m}$, along with a credential σ for it. The protocol outputs a string aux, which we refer to as the *helper information* to the holder, and produces no output for the issuer.
- **Credential showing and verification.** Showing and verification are similar to those in any (publicly verifiable) credential system, in that the user can select a showing predicate $\psi \in \Phi$, an attribute vector $\boldsymbol{m}$, and a corresponding credential σ, and produce some showing message τ which can be verified (under the public key pk and given ψ) to assess that indeed $\psi(\boldsymbol{m}) = 1$. But in addition to this, we allow the process of creating τ to also depend on helper information aux output by the helper protocol. Looking ahead once again to our definitions, unlinkability is meant to hold as long as each showing uses a freshly generated aux. *But crucially*, we note that aux does not depend on ψ, and thus can be precomputed by running the helper at any prior time after receiving the credential σ *and* it is obtained via a privacy-preserving protocol that will ensure that an execution of the protocol generating aux cannot be linked to the credential showing using this aux.

Here, predicates model information about the attributes which is revealed either at issuance or at showing—in both cases, it is only revealed that $\phi(\boldsymbol{m}) = 1$. The most relevant class of predicates describes *selective disclosure*. As part of the showing protocol, the user sends a list of indices $\boldsymbol{I} = (i_1, \ldots, i_k)$ and a list of disclosed attributes $\boldsymbol{a} \in \mathcal{M}^\ell$ which determines the predicate $\phi_{\boldsymbol{I},\boldsymbol{a}}$ given by $\phi_{\boldsymbol{I},\boldsymbol{a}}(m_1, \ldots, m_\ell) = 1$ if $a_{i_j} = m_{i_j}$ for all $j \in [k]$, and otherwise 0.

Unforgeability of SAAC. We formalize a strong notion of *unforgeability* for a SAAC scheme which postulates that a malicious holder can only convince the verifier to accept a showing for a predicate ϕ such that the holder has previously obtained a credential for some attribute vector $\boldsymbol{m}$ such that $\phi(\boldsymbol{m}) = 1$.

A definitional challenge is that a malicious holder may arbitrarily deviate from the protocol when interacting with the issuer, and therefore, care must be taken to ensure that the set of attribute vectors for which a credential was issued is well-defined. To this end, our definition relies on an *extractor* which, whenever a malicious message μ from the holder is successfully answered by the issuer (run on input ϕ), extracts attribute vector $\boldsymbol{m}$ from μ such that $\phi(\boldsymbol{m}) = 1$. The holder wins if a verifier is convinced by a showing for a predicate ϕ^* not satisfied by any of the extracted attribute vectors.

Furthermore, we allow the malicious holder to leverage additional types of interactions:

- **Helper interaction.** The malicious holder can interact as they please, in a fully concurrent and arbitrarily interleaved way, with the helper protocol.
- **Honest showings.** The malicious holder can obtain honest showings of credentials; the winning condition disallows a win for the adversary by simply replaying a showing of an honest user's credential.

Our unforgeability notion, however, does not require that the helper protocol is run for a successful showing. One could envision that the helper protocol serves some rate-limiting purpose, but effectively our formalism and our instantiations allow re-use of the helper string aux (at the cost of losing anonymity), and thus the rate-limiting effect is inconsequential. As a result of not making such a (in our view, unnecessary) restriction in the definition, we get the benefit that existing (multi-show, helper-free) anonymous credential systems immediately satisfy our definition.

Anonymity of SAAC. Our anonymity notion is meant to protect the credential holder from an adversary that controls the issuer (and thus both the issuance and the helper processes), and that is also shown credentials. The only information that is leaked *at issuance* is that the predicate ϕ holds for the attribute vector $\boldsymbol{m}$, and the only information leaked at showing is that the holder has a credential for some vector $\boldsymbol{m}$ satisfying the predicate ϕ. Crucially, we need to ensure that the helper protocol interaction is unlinkable to a particular showing of a credential, a fact which is also guaranteed by the security definition.

A generic construction. Our main contribution is a generic construction that lifts a KVAC scheme to a SAAC scheme. Informally, KVAC differ from a regular credential system in that the credential is meant to be verified by the same party that issued it; i.e. verification of the showing of a credential requires the secret key. Unlike in SAAC, no helper is involved. Despite not requiring the issuer's public key for verification, the public key of KVAC allows the issuer to prove to their holders that the credential was issued correctly. Several constructions of KVAC have been given in the literature [6,14,23].

Our generic construction replaces the keyed verification of a KVAC scheme with a non-interactive proof that the showing message satisfies the verification algorithm. The helper protocol will be an oblivious issuance of proof (oNIP) [37] protocol, which allows the holder to obtain the proof without leaking its showing message. Implementing this construction requires a KVAC scheme with a specific structure where showing and verification are done in two steps:

- **Key-dependent verification.** The holder first uses its attributes $\boldsymbol{m}$ and credential σ to compute a key-dependent showing message τ_{key} and a state st which are *independent of the predicate* ϕ. The verifier can then verify τ_{key} *using its secret key* sk.
- **Public verification.** The holder then continues showing using its state st to compute public showing message τ_{pub}, which is *dependent on the predicate* ϕ and can be bound to some additional value nonce. Then, $(\tau_{\mathsf{key}}, \tau_{\mathsf{pub}}, \phi, \mathsf{nonce})$ can be publicly verified using pk. (Note that both key-dependent and public verification needs to return 1.)

The key-dependent verification defines a relation $\mathsf{R_V}$ with statement $(\mathsf{pk}, \tau_{\mathsf{key}})$ and witness sk such that (1) the key sk corresponds to pk based on the key generation, and (2) τ_{key} is a valid key-dependent showing message when verified by sk. Then, using an oNIP protocol for the relation $\mathsf{R_V}$ (refer to Sect. 4.1 for the

deviation from the prior oNIP formalization in [37]), we arrive at the following SAAC construction:

- **Key generation and issuance** are exactly those of the KVAC scheme.
- **Helper protocol.** First, the holder computing the key-dependent showing message τ_{key} and a state st. Then, the issuer and the holder runs the oNIP protocol with the holder obtaining a proof π_{V} attesting that τ_{key} is valid with respect to sk. The helper information aux contains $(\tau_{\mathsf{key}}, \pi_{\mathsf{V}}, \mathsf{st})$.
- **Showing.** To show that the holder's credential satisfies a predicate ϕ, the holder computes the public showing message τ_{pub} for ϕ with *the additional value* nonce *set as* π_{V}. The final showing message contains $(\tau_{\mathsf{key}}, \tau_{\mathsf{pub}}, \pi_{\mathsf{V}})$.
- **Verification.** The verifier checks the validity of the proof π_{V} with respect to τ_{key} and the KVAC showing message $(\tau_{\mathsf{key}}, \tau_{\mathsf{pub}})$ with respect to ϕ and π_{V}.

It is important that τ_{pub} is dependent on π_{V}. Otherwise, the showing message is malleable. In particular, a malicious holder can forge by obtaining an honest user's showing message and requesting a new π_{V} through the helper. With that said, the security of our generic SAAC construction still requires other properties.

Achieving unforgeability. At a high level, unforgeability of the generic SAAC construction requires the following properties:

- *The proof π_{V} is sound.* This ensures that a valid forgery $(\tau_{\mathsf{key}}, \tau_{\mathsf{pub}}, \pi_{\mathsf{V}})$ contains τ_{key} that is valid with respect to the issuer's secret key sk. However, soundness by itself only guarantees that *there exists* a secret key sk' (not necessarily sk) that verifies τ_{key}. Hence, we require an additional property for KVAC, *denoted validity of key generation*, which is implied if each public key corresponds to a unique secret key. This ensures that τ_{key} is valid with respect to the issuer's secret key sk.
- *Helper protocol does not leak* sk. A malicious holder should not be able to distinguish between interactions with an honest helper or interactions with a simulator. Looking ahead, the simulator may require some sk-dependent computation, e.g., checking whether sk verifies a rerandomized statement. Hence, we formalize instead the $\mathcal{O}$*-zero-knowledge* property, where the simulator is assisted by an oracle $\mathcal{O}$ embedded with sk.
- *Unforgeability of KVAC.* We require a stronger than standard unforgeability for KVAC with the following main changes:
  1. Instead of a verification oracle, the adversary has access to *the same oracle $\mathcal{O}$ from $\mathcal{O}$-zero-knowledge of oNIP.* This is for our reduction to successfully run the simulator discussed above. For our instantiations, the oracle $\mathcal{O}$ can be used to simulate the verification oracle as well.
  2. Similarly to SAAC unforgeability, the adversary can query honest users' showing messages. Each query access, however, is split into two steps: first the adversary obtains an honest τ_{key}, then it adaptively chooses both the predicate ϕ it wants the honest user to show *and* the nonce it wants to be tied to the message, and gets τ_{pub} in response.

One challenge to securely instantiate our generic construction is to balance the strength of $\mathcal{O}$. Notably, if $\mathcal{O}$ reveals too much information about sk, the KVAC would be insecure; in contrast, if it reveals too little, the oNIP would be insecure.

Achieving anonymity. Anonymity of our SAAC construction follows from anonymity of KVAC and obliviousness of oNIP, with the following modifications made to the definitions.

- *Obliviousness of oNIP.* To satisfy our simulation-based definition of SAAC anonymity, we require a simulation-based obliviousness definition. However, in our instantiations, we are able to show obliviousness only when honest users request proofs for valid statements; specifically, $(\mathsf{pk}, \tau_{\mathsf{key}})$ must be in the language induced by the relation $\mathsf{R_V}$. Hence, we *require an extra property of KVAC* which ensures that even under a malicious issuer, if the user obtains a credential and does not abort, it should be able to produce a valid τ_{key} (in the sense that $(\mathsf{pk}, \tau_{\mathsf{key}})$ is in the induced language).
- *Anonymity of KVAC.* Similar to anonymity of SAAC , we require that both during issuance and during showing, the only information leaked to the adversary is that the relevant predicate ϕ is satisfied by the attributes $\boldsymbol{m}$. For showing, the adversary chooses the predicate ϕ and the value nonce adaptively, after obtaining the key-dependent value τ_{key}.

We refer the readers to Sect. 4 for the formalization of KVAC and oNIP required and our generic construction.

Instantiation from BBS. Our first SAAC instantiation is inspired by the KVAC by Barki et al. [6], which builds upon an algebraic message authentication code (MAC) based on BBS/BBS+ signatures [3,12,41]. The scheme is based on a pairing-free group $\mathbb{G}$ of prime order p and generator G. The secret and public keys are $x \in \mathbb{Z}_p$ and $X = xG$, respectively. A credential for attributes $\boldsymbol{m} \in \mathbb{Z}_p^\ell$ is of the form $(A \in \mathbb{G}, e \in \mathbb{Z}_p, s \in \mathbb{Z}_p)$ such that $A = (x+e)^{-1}C$, where $C = G + \sum_{i=1}^{\ell} m_i H_i + sH_{\ell+1}$ and $H_1, \ldots, H_{\ell+1}$ are public parameters. To show, the holder rerandomizes $A, B = C - eA$, and C into $\tilde{A}, \tilde{B}, \tilde{C}$ and proves knowledge of the underlying attributes with a valid credential via CDL proofs [15]. To verify the showing message, one uses the secret key x to check that $(G, X, \tilde{A}, \tilde{B})$ form a valid Diffie-Hellman tuple. By giving an oNIP for this relation (adapting Orrù et al. [37]), we turn this KVAC into SAAC. Note that our oNIP is zero-knowledge with respect to the restricted DDH oracle $\mathrm{rDDH}(x, \cdot)$ which checks that its input (A, B) satisfies $xA = B$.[3]

In order to use Barki et al.'s KVAC, however, we need to show that it satisfies our required (stronger) security notions. Specifically, recall that our unforgeability notions allows the adversary to (1) query the restricted DDH oracle embedded with the secret key and (2) view showing messages of honest users (in the manner described above). We show that this stronger version of unforgeability holds

[3] This oracle is exactly the key-dependent verification.

in the ROM under the Gap-q-SDH assumption. This "gap" assumption is necessary for simulating the restricted DDH oracle. Note that Barki et al. already require Gap-q-SDH to simulate the verification oracle.

The efficiency of the resulting SAAC is comparable to that of Barki et al.'s KVAC (see Table 1). For more details on this instantiation, we refer the readers to Sect. 5.1.

Instantiation from DDH. Sacrificing some efficiency (see Table 1), our second SAAC instantiation *completely removes* the dependency on *a gap q-type assumption* and only relies on the much more standard DDH assumption. Our starting point is the KVAC scheme introduced by Chase, Meiklejohn, and Zaverucha [23], building upon an algebraic MAC. We then give a corresponding oNIP protocol for the algebraic relation induced by the key-dependent verification. Similar to the BBS-based instantiation, the zero-knowledge of this oNIP is proved with respect to a simulator with access to an oracle, which we denote $\mathcal{O}_{\mathsf{SVerDDH}}$ (and will define later on in Sect. 5.2), that essentially runs the key-dependent verification of this KVAC with the embedded secret key.

This KVAC was already known to be provably secure but under a weaker definition not suitable for our generic construction. To address this gap, we made the following contributions (and refer the readers to Sect. 5.2 for more details):

1. We revisited the unforgeability of the underlying MAC and gave a new proof (albeit using similar techniques) for the security against adversaries who have access to the oracle $\mathcal{O}_{\mathsf{SVerDDH}}$ instead of the verification oracle. Additionally, this new security still implies the standard UFCMVA security of MACs.
2. Building on the unforgeability of the MAC, we showed unforgeability of the resulting KVAC scheme in the ROM. As we require unforgeability against adversaries who can see honest users' showings, there were several technical difficulties to overcome. Mainly, the reduction (to unforgeability of the algebraic MAC) needs to be constructed so that it can simulate the honest users' showings correctly, but still extract a valid MAC forgery from the adversary.
3. We gave a more efficient blind issuance protocol. In particular, our issuer's communication is independent of the number of attributes compared to the one sketched in [23] which contains a linear number of group elements.

2 Preliminaries

Notations. We use λ as the security parameter. We denote $[n..m] = \{n, n+1, \ldots, m\}$ for any $n \leqslant m \in \mathbb{Z}$ and $[n] = [1..n]$ for any $n \in \mathbb{N}$. We denote vectors using bold-sized letters (e.g., $\boldsymbol{v}, \boldsymbol{H}$). If $\boldsymbol{u} = (u_1, \ldots, u_n)$ and $\boldsymbol{v} = (v_1, \ldots, v_m)$, then $u \| v := (u_1, \ldots, u_n, v_1, \ldots, v_m)$. Denote $x \leftarrow a$ as assigning value a to a variable x. Denote $a \leftarrow\!\!\$\ S$ as uniformly sampling a from a finite set S. We denote $y \leftarrow\!\!\$\ \mathsf{A}(x)$ as running a (probabilistic) algorithm A on input x with fresh randomness and $[\mathsf{A}(x)]$ as the set of possible outputs of A; $(y_1, y_2) \leftarrow\!\!\$\ \langle \mathsf{A}(x_1) \rightleftharpoons \mathsf{B}(x_2) \rangle$ denotes a pair of interactive algorithms A, B with inputs x_1, x_2 and outputs y_1, y_2 respectively. We often use the words *messages* and *attributes* interchangably.

Table 1. Comparison of group-based KVAC, AC, and BSA schemes and our highlighted SAAC instantiations. The number of attributes is ℓ. Showing size depends on the number of disclosed attributes and is given as a close-to-tight upper-bound. Denote $\mathbb{G}$ and $\mathbb{Z}_p$ as the sizes of group elements and scalars, respectively. All security analyses assume the ROM. *: Showing requires two rounds of communication with the helper server (helper interactions can be batched). This is "multi-show" in the sense that the user does not have to re-prove that their attributes satisfy an issuance predicate, which may be expensive, to compute a showing (in contrast to, e.g., ACL). †: Only BBS is pairing-based and $\mathbb{G}_1$ denotes the size of a source group element. ‡: The DDH-based version is less efficient.

Scheme	Publicly Verifiable	Multi-Show	Credential Size	Showing Size	Security: Unforgeability	Security: Anonymity
CMZ14 [23]	No	Yes	$2\mathbb{G}$	$(\ell+2)\mathbb{G}$ $+(2\ell+2)\mathbb{Z}_p$	GGM / DDH‡	DDH
BBDT16 [6]	No	Yes	$2\mathbb{G}+2\mathbb{Z}_p$	$3\mathbb{G}$ $+(\ell+7)\mathbb{Z}_p$	Gap-q-SDH	Statistical
$\mathsf{KVAC}_{\mathsf{wBB}}$ [14]	No	Yes	$(\ell+1)\mathbb{G}$	$2\mathbb{G}$ $+(\ell+1)\mathbb{Z}_p$	ℓ-SCDHI	Statistical
μCMZ [38]	No	Yes	$2\mathbb{G}$	$(\ell+2)\mathbb{G}$ $+(2\ell+2)\mathbb{Z}_p$	AGM + 3-DL	Statistical
μBBS [38]	No	Yes	$1\mathbb{G}+1\mathbb{Z}_p$	$2\mathbb{G}$ $+(\ell+4)\mathbb{Z}_p$	AGM + q-DL	Statistical
MBS+25 [35]	No	Yes	$(\ell+2)\mathbb{G}$	$2\mathbb{G}$	GGM	Statistical
ACL [4]	Yes	No	$2\mathbb{G}+6\mathbb{Z}_p$	$2\mathbb{G}$ $+(\ell+8)\mathbb{Z}_p$	DL+AGM	DDH
$\mathsf{SAAC}_{\mathsf{BBS}}$ (Sec. 5.1)	Yes	Yes*	$1\mathbb{G}+2\mathbb{Z}_p$	$3\mathbb{G}$ $+(\ell+8)\mathbb{Z}_p$	Gap-q-SDH	Statistical
$\mathsf{SAAC}_{\mathsf{DDH}}$ (Sec. 5.2)	Yes	Yes*	$4\mathbb{G}$	$(\ell+6)\mathbb{G}+$ $(4\ell+11)\mathbb{Z}_p$	DDH	DDH
BBS [41]†	Yes	Yes	$1\mathbb{G}_1+1\mathbb{Z}_p$	$2\mathbb{G}_1$ $+(\ell+3)\mathbb{Z}_p$	q-SDH	Statistical

Group parameter generator. A group parameter generator is a probabilistic polynomial time algorithm GGen taking as input 1^λ and outputting a cyclic group $\mathbb{G}$ of $\Theta(\lambda)$-bit prime order p with a generator G. We assume that standard group operations in $\mathbb{G}$ can be performed in polynomial time in λ and adopt *additive notation* (i.e., $A+B$ for applying group operation on $A, B \in \mathbb{G}$).

Cryptographic assumptions. In Fig. 2, we define games for Decisional Diffie-Hellman (DDH), Discrete Logarithm (DL), and a pairing-free analog of the q-*Strong Diffie-Hellman* assumption [11] augmented with a *restricted* DDH oracle. Denote the advantage of an adversary $\mathcal{A}$ against these assumptions as

$$\mathsf{Adv}^{(\mathrm{DL},(q,\mathrm{rDDH})\text{-SDH})}_{\mathsf{GGen}}(\mathcal{A},\lambda) := \mathsf{Pr}[(\mathrm{DL}/(q,\mathrm{rDDH})\text{-SDH})^{\mathcal{A}}_{\mathsf{GGen}}(\lambda)=1]\,,$$
$$\mathsf{Adv}^{\mathrm{ddh}}_{\mathsf{GGen}}(\mathcal{A},\lambda) := \left|\mathsf{Pr}[\mathrm{DDH}^{\mathcal{A}}_{\mathsf{GGen},0}(\lambda)=1]-\mathsf{Pr}[\mathrm{DDH}^{\mathcal{A}}_{\mathsf{GGen},1}(\lambda)=1]\right|.$$

Relations and non-interactive proofs. Let $\mathsf{R} \subseteq \mathcal{X} \times \mathcal{W}$ be a relation and $\mathcal{L}_{\mathsf{R}} := \{x \in \mathcal{X} | \exists w \in \mathcal{W} : (x,w) \in \mathsf{R}\}$ denotes its induced language. A non-

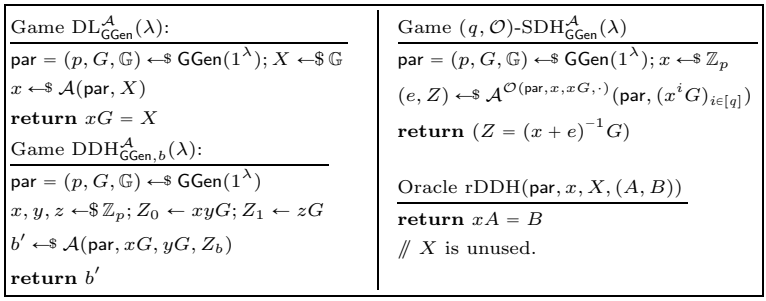


Fig. 2. Games DDH, DL, and $(q, \mathcal{O})$-SDH, and a definition of the oracle rDDH.

interactive zero-knowledge (NIZK) proof system for a relation R is a tuple of algorithms $(\mathsf{NIZK.Prove}^{\mathsf{H}}, \mathsf{NIZK.Ver}^{\mathsf{H}})$ with access to a random oracle $\mathsf{H} : \{0,1\}^* \to \mathcal{R}$ with the following syntax:

- $\pi \leftarrow\!\$\ \mathsf{NIZK.Prove}^{\mathsf{H}}(x, w)$: outputs a proof π on input $(x, w) \in \mathsf{R}$.
- $0/1 \leftarrow \mathsf{NIZK.Ver}^{\mathsf{H}}(x, \pi)$: verifies a proof π for statement x.

We require a NIZK to be correct, sound, zero-knowledge, and optionally straight-line extractable knowledge-sound for a relaxed relation $\widetilde{\mathsf{R}} \supseteq \mathsf{R}$. We refer to the full version for formal security definitions of NIZKs.

3 Server-Aided Anonymous Credentials

In this section, we introduce Server-Aided Anonymous Credentials (SAAC), with the syntax and security definitions given in Sects. 3.1 and 3.2, respectively. SAAC allow a user to obtain a credential for its attributes through a (blind) issuance protocol and to anonymously show that it owns a credential for attributes which satisfies some specified predicate. However, in contrast to anonymous credentials (AC), the user may request the issuer to help produce helper information which to be used to produce a publicly-verifiable showing message. This is modeled as an unlinkable helper protocol *independent* of the predicate specified during showing. Users may then ask for several pieces of helper information ahead of time and spend them later during showing.

3.1 Syntax

A server-aided anonymous credential scheme $\mathsf{SAAC} = \mathsf{SAAC}[\Phi, \mathcal{M}]$ defined with respect to a predicate class family $\Phi = \{\Phi_{\mathsf{par}}\}_{\mathsf{par}}$[4] and an attribute space $\mathcal{M} = \{\mathcal{M}_{\mathsf{par}}\}_{\mathsf{par}}$ consists of the following algorithms.

[4] Alternatively, one can define the scheme with respect to two classes of predicates Φ_{Iss} and Φ_{Show} which model predicates accepted during issuance and showing. Here, we define our SAAC syntax with respect to a single class of predicates $\Phi = \Phi_{\mathsf{Iss}} \cup \Phi_{\mathsf{Show}}$ covering both issuance and showing predicate classes. For our constructions, we consider the class of selective disclosure predicates for both issuance and showing.

- $\mathsf{par} \leftarrow\!\!\$\ \mathsf{SAAC.Setup}(1^\lambda, 1^\ell)$ outputs public parameters par which defines the attribute space $\mathcal{M} = \mathcal{M}_{\mathsf{par}}$ and a corresponding class of predicates $\Phi = \Phi_{\mathsf{par}}$. For succinctness, we will abuse the notation and omit the subscript par.
- $(\mathsf{sk}, \mathsf{pk}) \leftarrow\!\!\$\ \mathsf{SAAC.KeyGen}(\mathsf{par})$ outputs the secret and public key pair.
- $(\perp, \sigma) \leftarrow\!\!\$\ \langle \mathsf{SAAC.Iss}(\mathsf{par}, \mathsf{sk}, \phi) \rightleftharpoons \mathsf{SAAC.U}(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}, \phi) \rangle$ is an interactive protocol between the issuer and the user where at the end, the user obtains a credential σ for its vector of attributes $\boldsymbol{m} \in \mathcal{M}^\ell$, which satisfies a predicate $\phi \in \Phi$ (i.e., $\phi(\boldsymbol{m}) = 1$). We consider a round-optimal issuance protocol consisting of the following algorithms:
  - $(\mu, \mathsf{st}^u) \leftarrow\!\!\$\ \mathsf{SAAC.U}_1(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}, \phi)$ outputs a protocol message and a state.
  - $\mathsf{imsg} \leftarrow\!\!\$\ \mathsf{SAAC.Iss}(\mathsf{par}, \mathsf{sk}, \mu, \phi)$ outputs issuer's message imsg, and if the issuer aborts, we say that $\mathsf{imsg} = \perp$.
  - $\sigma \leftarrow\!\!\$\ \mathsf{SAAC.U}_2(\mathsf{st}^u, \mathsf{imsg})$ outputs a credential σ for the attributes $\boldsymbol{m}$.
- $(\perp, \mathsf{aux}) \leftarrow\!\!\$\ \langle \mathsf{SAAC.Helper}(\mathsf{par}, \mathsf{sk}) \rightleftharpoons \mathsf{SAAC.ObtHelp}(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}, \sigma) \rangle$ is a r-round protocol where the user interacts with the issuer to obtain a helper information aux. Formally, the protocol execution is of the following format:

$$
\begin{aligned}
&(\mathsf{umsg}_1, \mathsf{st}^u) \leftarrow\!\!\$\ \mathsf{SAAC.ObtHelp}_1(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}, \sigma)\ , \\
&(\mathsf{hmsg}_1, \mathsf{st}^h) \leftarrow\!\!\$\ \mathsf{SAAC.Helper}_1(\mathsf{par}, \mathsf{sk}, \mathsf{umsg}_1)\ , \\
&\left.\begin{aligned}(\mathsf{umsg}_i, \mathsf{st}^u) &\leftarrow\!\!\$\ \mathsf{SAAC.ObtHelp}_i(\mathsf{st}^u, \mathsf{hmsg}_{i-1})\ , \\ (\mathsf{hmsg}_i, \mathsf{st}^h) &\leftarrow\!\!\$\ \mathsf{SAAC.Helper}_i(\mathsf{st}^h, \mathsf{umsg}_i)\ ,\end{aligned}\right\} \quad \textbf{for } i = 2, \ldots, r \\
&\mathsf{aux} \leftarrow\!\!\$\ \mathsf{SAAC.ObtHelp}_{r+1}(\mathsf{st}^u, \mathsf{hmsg}_r)\ .
\end{aligned}
$$

- $\tau \leftarrow\!\!\$\ \mathsf{SAAC.Show}(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}, \sigma, \mathsf{aux}, \phi, \mathsf{nonce})$ outputs a showing τ of the credential σ issued for attributes $\boldsymbol{m}$ such that $\phi(\boldsymbol{m}) = 1$.
- $0/1 \leftarrow \mathsf{SAAC.SVer}(\mathsf{par}, \mathsf{pk}, \tau, \phi, \mathsf{nonce})$ outputs a bit.

In the showing and verification algorithms, we allow the showing message τ to be bound to some additional value nonce (which in some cases is the token identifier or a nonce chosen by the verifier). We do not require a credential verification algorithm, since the credential itself might not be publicly verifiable, and a secret key credential verification is not required for our security properties.

Correctness. A SAAC scheme is η-correct if for any $\lambda, \ell = \ell(\lambda) \in \mathbb{N}$, any $\mathsf{par} \in [\mathsf{SAAC.Setup}(1^\lambda, 1^\ell)]$, any $(\mathsf{sk}, \mathsf{pk}) \in [\mathsf{SAAC.KeyGen}(\mathsf{par})]$, any attributes $\boldsymbol{m} \in \mathcal{M}_{\mathsf{par}}^\ell$, any $\mathsf{nonce} \in \{0,1\}^*$, and any predicates $\phi, \phi' \in \Phi_{\mathsf{par}}$ such that $\phi(\boldsymbol{m}) = \phi'(\boldsymbol{m}) = 1$, the following experiment returns 1 with probability at least $1 - \eta(\lambda)$.

$$
\begin{aligned}
&(\perp, \sigma) \leftarrow\!\!\$\ \langle \mathsf{SAAC.Iss}(\mathsf{par}, \mathsf{sk}, \phi) \rightleftharpoons \mathsf{SAAC.U}(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}, \phi) \rangle \\
&(\perp, \mathsf{aux}) \leftarrow\!\!\$\ \langle \mathsf{SAAC.Helper}(\mathsf{par}, \mathsf{sk}) \rightleftharpoons \mathsf{SAAC.ObtHelp}(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}, \sigma) \rangle \\
&\tau \leftarrow\!\!\$\ \mathsf{SAAC.Show}(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}, \sigma, \mathsf{aux}, \phi', \mathsf{nonce}) \\
&\textbf{return } \mathsf{SAAC.SVer}(\mathsf{par}, \mathsf{pk}, \tau, \phi', \mathsf{nonce})\ .
\end{aligned}
$$

3.2 Security Definitions

We consider two main security notions for anonymous credentials: unforgeability and anonymity. At the end of the section, we define an additional security notion, denoted integrity, and discuss its importance.

UNFORGEABILITY. A SAAC scheme is unforgeable if there exists an extractor $\mathsf{Ext} = (\mathsf{Ext}_{\mathsf{Setup}}, \mathsf{Ext}_{\mathsf{Iss}})$ such that

1. The distribution of par from the setup algorithm and $\mathsf{Ext}_{\mathsf{Setup}}$ are indistinguishable, i.e., for any adversary $\mathcal{A}$, the following advantage is bounded

$$\mathsf{Adv}^{\mathsf{par\text{-}indist}}_{\mathsf{SAAC},\mathsf{Ext}}(\mathcal{A},\lambda) := |\mathsf{Pr}[\mathcal{A}(\mathsf{par}) = 1 | \mathsf{par} \leftarrow\!\!\$\ \mathsf{SAAC.Setup}(1^\lambda, 1^\ell)] - \mathsf{Pr}[\mathcal{A}(\mathsf{par}) = 1 | (\mathsf{par}, \mathsf{td}) \leftarrow\!\!\$\ \mathsf{Ext}_{\mathsf{Setup}}(1^\lambda, 1^\ell)]| \ .$$

2. Denote the advantage of any adversary $\mathcal{A}$ in the unforgeability game, defined in Fig. 3 with respect to Ext (more discussion on the game below), as

$$\mathsf{Adv}^{\mathsf{unf}}_{\mathsf{SAAC},\mathsf{Ext}}(\mathcal{A},\lambda) := \mathsf{Pr}[\mathrm{UNF}^{\mathcal{A}}_{\mathsf{SAAC},\mathsf{Ext}}(\lambda) = 1] \ .$$

We now discuss in more detail our unforgeability game. First, the game generates public parameters par and a trapdoor td using the extractor along with the secret and public keys $(\mathsf{sk}, \mathsf{pk})$. Then, it runs the adversary $\mathcal{A}$ (acting as a malicious user) which can arbitrarily interleave the execution of the following oracles.

Issuance oracle Iss**.** The adversary $\mathcal{A}$ can request a credential to be issued via the blind issuance protocol modeled with Iss. In this oracle, the game extracts the underlying attributes $\boldsymbol{m}$ using $\mathsf{Ext}_{\mathsf{Iss}}$. The game keeps track of the attributes of which a credential has been issued so far.

Helper oracles $\mathrm{Help}_1, \ldots, \mathrm{Help}_r$**.** The adversary can run multiple helper protocol sessions with the issuer, with each identified with the session ID sid.

New user oracle NewUsr**.** The adversary can request generation of a credential for attributes $\boldsymbol{m}$ satisfying the predicate ϕ for *honest users*. The adversary do not see the credential σ_{cid} generated from this oracle, but can identify them in SH with a credential ID cid.

Showing oracle SH**.** The adversary specifies the credential ID cid (which links to $\boldsymbol{m}_{\mathsf{cid}}$ and σ_{cid}) along with the predicate ϕ and a value nonce. Then, the game will compute τ by running (1) the helper protocol with the honest user (using $\boldsymbol{m}_{\mathsf{cid}}$ and σ_{cid}) and (2) the showing algorithm Show using the helper information aux obtained from the protocol, the predicate ϕ, and the given value nonce. The tuple $(\phi, \mathsf{nonce}, \tau)$ is recorded by the game.

Finally, $\mathcal{A}$ **wins** the game if one of the following occurs:

- During issuance, the issuer does not abort **and** the extractor extracts attributes $\boldsymbol{m}$ that do not satisfy the predicate ϕ specified at issuance. This prevents adversaries who try to request credentials for unauthorized attributes.

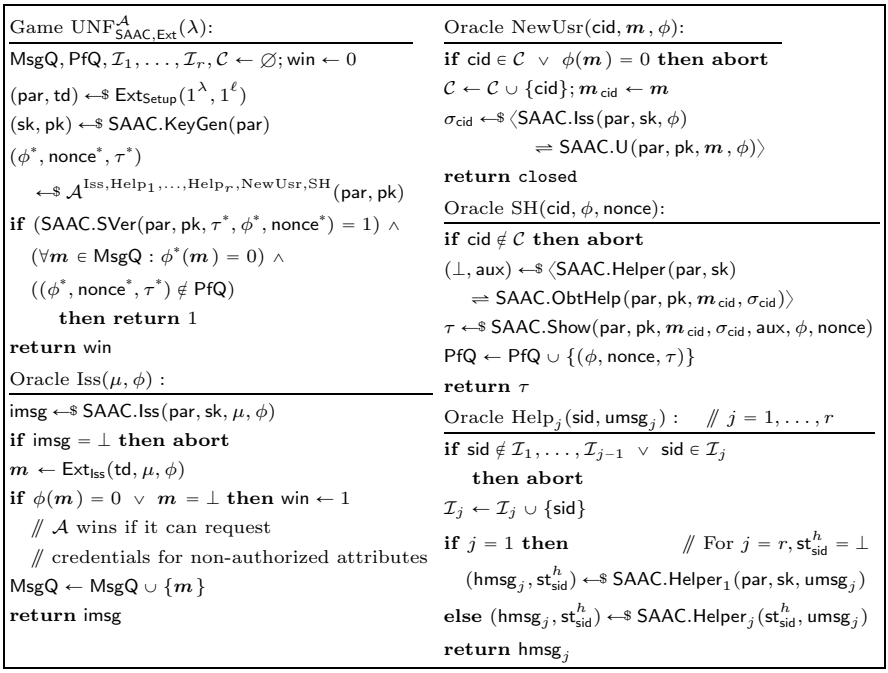


Fig. 3. Unforgeability game for $\mathsf{SAAC} = \mathsf{SAAC}[\Phi, \mathcal{M}]$. We assume that all the predicates output by $\mathcal{A}$ are in Φ.

- They output a tuple $(\phi^*, \mathsf{nonce}^*, \tau^*)$ of which the game considers a forgery if (1) τ^* is valid with respect to the predicate ϕ^* and the value nonce^*, (2) ϕ^* is not satisfied by any of the extracted attributes, and (3) they do not replay honest users' showing messages.

Below, we discuss the design choices for our unforgeability definition.

On the adversary winning if the extractor fails. We require this winning condition for two *important* reasons:

The extracted attributes should satisfy the predicate. Consider a similar game *where the issuance oracle aborts* if the extracted attributes do not satisfy the predicate. It is possible that a SAAC is secure with respect to an extractor that always aborts. In particular, the adversary will not get any credential, so the security only prevents key-only attacks. Hence, we cannot simply allow the game nor the issuer oracle to abort when the extraction fails.

Credentials should only be granted for authorized attributes. Consider the game that only extracts and record the attributes into MsgQ *without aborting*. One could construct a SAAC scheme where the issuer algorithm ignores the predicate and always computes imsg. An adversary can then request credentials for unauthorized attributes, a scenario which should not be allowed.

On the (non-)requirement of the helper interaction. Our unforgeability notion only aims to prevent malicious holders from showing credentials that do not correspond to their attributes, and does not prevent a situation where a user is able to show a credential without helper interaction. In a way, we view SAAC as a relaxed notion of multi-show AC where the helper protocol helps us achieve public verification, as a consequence standard AC satisfies SAAC notion. We note that our instantiations require at least one helper interaction to output a publicly verifiable showing message.

The NewUsr **and** SH **oracles** model adversaries who can obtain showing messages of honest users. This is to provide a non-malleability guarantee where the adversary cannot forge by modifying previous showing messages of honest users. This scenario is also considered by the unforgeability of Privacy-Enhancing Attribute-Based Signatures (PABS) from [16] and the extractability security of KVAC given in [38], but not in the original KVAC unforgeability definition [23].

Honest users reusing aux**.** As mentioned in the overview, it is possible that the helper information aux is reused at the cost of anonymity. However, we assume that honest users do not reuse the helper information and do not consider an adversary who forges a showing *by forcing honest users to reuse a helper information* aux. One may argue that (a) such situation can occur given a bug in the system or (b) honest users might not care about their anonymity. However, we see (a) as an implementation problem. For (b), such users could instead use the more convenient (and efficient) non-anonymous credentials systems.

Adversary's power over the honest users. We consider adversaries who can see only the final showing message τ of honest users. We leave the consideration of a stronger model of adversaries (e.g., one that can view the transcript between the user and the helper or intercept user's messages) for future work.

ANONYMITY. No adversary can distinguish between interactions with *an honest user* and interactions with a simulator Sim. In particular, a SAAC is anonymous if there exists a simulator $\mathsf{Sim} = (\mathsf{Sim}_{\mathsf{Setup}}, \mathsf{Sim}_{\mathsf{U}}, \mathsf{Sim}_{\mathsf{ObtH}}, \mathsf{Sim}_{\mathsf{Show}})$ such that

1. The distribution of par from the setup algorithm and $\mathsf{Sim}_{\mathsf{Setup}}$ are indistinguishable, i.e., for any adversary $\mathcal{A}$, the following advantage is bounded

$$\mathsf{Adv}^{\mathsf{par\text{-}indist}}_{\mathsf{SAAC},\mathsf{Sim}}(\mathcal{A},\lambda) := |\mathsf{Pr}[\mathcal{A}(\mathsf{par}) = 1 | \mathsf{par} \leftarrow\!\!\$\ \mathsf{SAAC.Setup}(1^\lambda, 1^\ell)] - \mathsf{Pr}[\mathcal{A}(\mathsf{par}) = 1 | (\mathsf{par}, \mathsf{td}) \leftarrow\!\!\$\ \mathsf{Sim}_{\mathsf{Setup}}(1^\lambda, 1^\ell)]| \ .$$

2. The advantage of $\mathcal{A}$, denoted $\mathsf{Adv}^{\mathsf{anon}}_{\mathsf{SAAC},\mathsf{Sim}}(\mathcal{A},\lambda)$ and defined as follows, in the anonymity game described in Fig. 4 with respect to Sim, is bounded

$$|\mathsf{Pr}[\mathrm{Anon}^{\mathcal{A}}_{\mathsf{SAAC},\mathsf{Sim},0}(\lambda) = 1] - \mathsf{Pr}[\mathrm{Anon}^{\mathcal{A}}_{\mathsf{SAAC},\mathsf{Sim},1}(\lambda) = 1]| \ .$$

For readability, we give more detail on our anonymity game below. The adversary (acting as a malicious issuer) will first receive *both the public parameters* par *and the trapdoor* td generated by the simulator and will do the following:

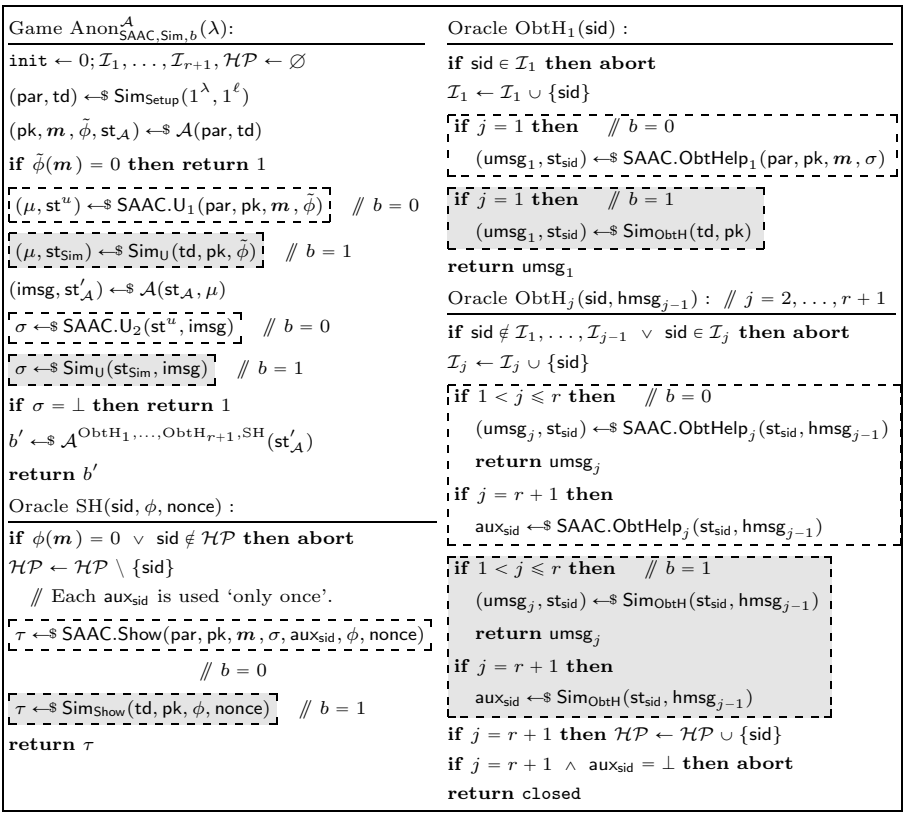


Fig. 4. Anonymity game for $\mathsf{SAAC} = \mathsf{SAAC}[\Phi, \mathcal{M}]$, parameterized with a simulator Sim and a bit b. We denote case $b = 0$ in the dashed boxes and case $b = 1$, denoted in the dashed and highlighted boxes. When querying the oracle SH, the adversary specifies a helper information $\mathsf{aux}_{\mathsf{sid}}$ via input sid. We assume all predicates output by $\mathcal{A}$ are in Φ.

Determine $\mathsf{pk}, \boldsymbol{m}, \tilde{\phi}$: The adversary determines its (possibly malicious) public key pk, the attributes $\boldsymbol{m}$, and the issuance predicate $\tilde{\phi}$ for which the honest user will use to request a credential. The user (or the simulator) then computes a protocol message μ and sends them to the adversary.

Finish credential issuance: The adversary sends imsg which lets the honest user derive a credential σ or abort. The simulator needs to correctly simulate the abort as well.

The adversary then outputs a guess b' after interacting with the following oracles.

Obtain-help oracles $\mathrm{ObtH}_1, \ldots \mathrm{ObtH}_{r+1}$: The adversary forces the user holding σ to request a helper information. In these oracles, the adversary would interact with either (a) the honest user, who knows the attributes $\boldsymbol{m}$ and the credential σ, or (b) the simulator, who knows neither the attributes nor the credential. At the end, the honest user will either abort or receive a helper

information $\mathsf{aux}_{\mathsf{sid}}$ tied to the session ID sid. On the other hand, the simulator would only need to simulate the abort correctly.

Showing oracle SH**:** The adversary specifies a helper information (via sid), a predicate ϕ, and a value nonce, such that the honest user computes τ via $\mathsf{SAAC.Show}$ using the helper information $\mathsf{aux}_{\mathsf{sid}}$, the attributes $\boldsymbol{m}$ satisfying ϕ and the credential σ. *Each helper information is restricted to be used only once.* In contrast, the simulator only takes as input the trapdoor td, the public key pk, and the predicate ϕ.

We stress that, in oracle SH, the simulator *does not* depend on the helper information $\mathsf{aux}_{\mathsf{sid}}$ nor the attributes and credential of the honest user. This captures the fact that the helper protocol sessions and the final showing messages are unlinkable, as the simulator is independent of the session ID sid.

Moreover, although we stated the anonymity game with respect to *a single honest user*, the multi-user/session security, where the adversary interacts with multiple credential holders, is also defined and proved in the full version.

Integrity. In the full version, we consider an additional security property, denoted *integrity*, which ensures that a malicious issuer cannot convince a user that they have been issued a valid credential and helper information, when in fact, these cannot be used to create a valid showing for some adversarially-chosen (valid) predicate. This protects against a scenario where a user does not immediately compute a showing and check that it is valid, perhaps because they do not yet know the predicate that they want to show the credential for. We show that a weak notion of integrity follows from correctness and anonymity.

Remark 3.1 (Revocation). We do not consider revocation of credentials in this work and see it as an interesting open problem. A possible (and not-so-efficient) approach is to have the issuer to maintain a public list of allowed user identities (which will be one of the users' attributes), and at showing time, the user additionally shows with a predicate saying their attributes contains an identity on this public list.

4 Generic Construction from Keyed-Verification Anonymous Credentials

In this section, we introduce our building blocks, keyed-verification anonymous credentials (KVAC) and oblivious proof issuance protocol (oNIP), in Sect. 4.1, and give a generic construction of SAAC in Sect. 4.2.

4.1 Building Blocks

In this subsection, we give the syntax and definitions related to our building blocks and point out several distinctions from prior works. These include (1) global parameters generator, (2) syntax for relations and languages for oNIP, (3) KVAC syntax and definitions, and (4) oNIP syntax and definitions.

Global parameters generator. Inspired by the formalization in [16], we define global parameters generator $\mathsf{Gen}(1^\lambda)$, a probabilistic algorithm which generates public parameters par_g. Note that par_g are shared by both of our building blocks KVAC and oNIP. In practice, an example for Gen is a group parameters generator GGen which outputs a group description $(p, G, \mathbb{G})$. In our instantiations, the underlying building blocks KVAC and oNIP may require the global parameters to be generated with some trapdoor td_g, used to simulate components of *both building blocks* in the security proofs. In that case, we need a simulator $\mathsf{Sim}_{\mathsf{Gen}}$ which returns $(\mathsf{par}_g, \mathsf{td}_g)$ such that par_g is indistinguishable from Gen.

Syntax on relations for oblivious proof issuance. Particularly for this section, we use a similar syntax for relations and languages from [37]. In [37], a relation R contains tuples of the form $((X, Y, Z), x)$, denoting X the statement, x the witness, Y an *argument* and Z an *augmented statement*. In our case, a relation contains tuples $((X, Y), x)$ and we instead call Y an *augmented statement*, containing both (Y, Z) in their syntax. We denote the relation $\mathsf{Core}(\mathsf{R}) := \{(X, x) | \exists Y : ((X, Y), x) \in \mathsf{R}\}$ and the induced language $\mathcal{L}_{\mathsf{R}} := \{(X, Y) | \exists x : ((X, Y), x) \in \mathsf{R}\}$. The membership $(X, x) \in \mathsf{Core}(\mathsf{R})$ can be efficiently checked.

Keyed-verification anonymous credentials. A keyed-verification anonymous credential (KVAC) scheme $\mathsf{KVAC} = \mathsf{KVAC}[\mathsf{Gen}, \Phi, \mathcal{M}]$, defined with respect to Gen, a predicate family Φ and an attribute space $\mathcal{M}$, consists of the following algorithms.

- $\mathsf{par}_{\mathsf{KVAC}} \leftarrow\!\!\$\ \mathsf{KVAC.Setup}(1^\ell, \mathsf{par}_g)$ takes as input par_g and outputs public parameters $\mathsf{par}_{\mathsf{KVAC}}$ defining the an attribute space $\mathcal{M} = \mathcal{M}_{\mathsf{par}_{\mathsf{KVAC}}}$ and a predicate class $\Phi = \Phi_{\mathsf{par}_{\mathsf{KVAC}}}$. *We assume that* $\mathsf{par}_{\mathsf{KVAC}}$ *contains* par_g.
- $(\mathsf{sk}, \mathsf{pk}) \leftarrow\!\!\$\ \mathsf{KVAC.KeyGen}(\mathsf{par}_{\mathsf{KVAC}})$ outputs the secret/public key pair.
- $(\bot, \sigma) \leftarrow\!\!\$\ \langle \mathsf{KVAC.Iss}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{sk}, \phi) \rightleftharpoons \mathsf{KVAC.U}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}, \boldsymbol{m}, \phi) \rangle$ is a round-optimal protocol with similar syntax to SAAC's issuance (see Sect. 3.1).
- $\tau = (\tau_{\mathsf{key}}, \tau_{\mathsf{pub}}) \leftarrow\!\!\$\ \mathsf{KVAC.Show}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}, \boldsymbol{m}, \sigma, \phi, \mathsf{nonce})$ outputs a showing message τ. The showing algorithm is split into the two algorithms.
  - $(\tau_{\mathsf{key}}, \mathsf{st}) \leftarrow\!\!\$\ \mathsf{KVAC.Show}_{\mathsf{key}}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}, \boldsymbol{m}, \sigma)$ outputs a state st and a key-dependent showing message τ_{key}.
  - $\tau_{\mathsf{pub}} \leftarrow\!\!\$\ \mathsf{KVAC.Show}_{\mathsf{pub}}(\mathsf{st}, \phi, \mathsf{nonce})$ outputs a message τ_{pub} showing the credential σ issued for attributes $\boldsymbol{m}$ such that $\phi(\boldsymbol{m}) = 1$.
- $0/1 \leftarrow \mathsf{KVAC.SVer}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{sk}, \mathsf{pk}, (\tau_{\mathsf{key}}, \tau_{\mathsf{pub}}), \phi, \mathsf{nonce})$ outputs a bit. Similar to showing, verification also splits into key-dependent and public verification as follows. The output bit is determined by $b_0 \wedge b_1$.
  - $b_0 \leftarrow \mathsf{KVAC.SVer}_{\mathsf{key}}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{sk}, \tau_{\mathsf{key}})$ verifies τ_{key} using sk.
  - $b_1 \leftarrow \mathsf{KVAC.SVer}_{\mathsf{pub}}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}, \tau_{\mathsf{key}}, \tau_{\mathsf{pub}}, \phi, \mathsf{nonce})$ verifies τ_{key} and τ_{pub}.

One distinction from prior works' syntax is the split in showing and verification algorithms into key-dependent and public parts. In the showing algorithm, the showing message τ_{pub} is bound to an additional value nonce (which in some cases can be a token identifier or a nonce chosen by the verifier). For our generic SAAC construction, we require that τ_{key} is independent of the predicate ϕ and nonce.

This syntax is applicable to some existing KVAC schemes (e.g., [6,23]), but not for some others [35] where the predicate-dependent parts of the showing message require the secret key to verify. The key-dependent verification algorithm $\mathsf{KVAC.SVer}_{\mathsf{key}}$ induces a relation

$$\mathsf{R}_{\mathsf{V},\mathsf{par}_g} := \left\{ (((\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}), \tau_{\mathsf{key}}), \mathsf{sk}) : \begin{array}{l} \mathsf{par}_{\mathsf{KVAC}} = (\mathsf{par}_g, \cdot) \wedge \\ (\mathsf{sk}, \mathsf{pk}) \in [\mathsf{KVAC.KeyGen}(\mathsf{par}_{\mathsf{KVAC}})] \wedge \\ \mathsf{KVAC.SVer}_{\mathsf{key}}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{sk}, \tau_{\mathsf{key}}) = 1 \end{array} \right\}.$$

The relation contains a statement $((\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}), \tau_{\mathsf{key}})$ and a witness sk such that $\mathsf{par}_{\mathsf{KVAC}}$ contains par_g, $(\mathsf{sk}, \mathsf{pk})$ can be generated from $\mathsf{KVAC.KeyGen}(\mathsf{par}_{\mathsf{KVAC}})$, and τ_{key} is valid with respect to sk. Checking $(\mathsf{sk}, \mathsf{pk}) \in [\mathsf{KVAC.KeyGen}(\mathsf{par}_{\mathsf{KVAC}})]$ can be done efficiently, e.g., interpreting sk as random coins used to generate pk. We denote $\mathcal{L}_{\mathsf{V},\mathsf{par}_g}$ as the induced language of $\mathsf{R}_{\mathsf{V},\mathsf{par}_g}$.

Then, we require a KVAC scheme to satisfy the following properties. We refer the readers to the full version for the standard definitions of parameter indistinguishability for various algorithms and the η-correctness property which is defined similarly to that of SAAC's (without the helper). Later on in Sect. 5, we modify some existing KVAC schemes to fit to our definitions.

Unforgeability. Let $\mathcal{O}(\mathsf{par}_g, \mathsf{sk}, (\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}), \cdot)$ be an oracle embedded with $\mathsf{par}_g, \mathsf{par}_{\mathsf{KVAC}}, \mathsf{sk}, \mathsf{pk}$, and taking a to-be-determined input. A KVAC scheme is $\mathcal{O}$-unforgeable if there exists an extractor $\mathsf{Ext} = (\mathsf{Ext}_{\mathsf{Setup}}, \mathsf{Ext}_{\mathsf{Iss}})$ such that

1. The distribution of $\mathsf{par}_{\mathsf{KVAC}}$ from $\mathsf{KVAC.Setup}(\mathsf{par}_g)$ and $\mathsf{Ext}_{\mathsf{Setup}}(\mathsf{par}_g)$ for $\mathsf{par}_g \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)$ are indistinguishable.
2. The following advantage of $\mathcal{A}$ in the unforgeability game, defined in Fig. 5 with respect to the oracle $\mathcal{O}$ and the extractor Ext, is bounded. $\mathsf{Adv}^{\mathsf{unf}}_{\mathsf{KVAC},\mathsf{Ext},\mathcal{O}}(\mathcal{A}, \lambda) := \Pr[\mathrm{UNF}^{\mathcal{A}}_{\mathsf{KVAC},\mathsf{Ext},\mathcal{O}}(\mathcal{A}, \lambda) = 1]$.

The KVAC unforgeability game is defined similarly to SAAC unforgeability with the following exceptions: no helper oracle is involved, the adversary can query the oracle $\mathcal{O}$ which parameterized the game, and the adversary can request honest users' showing messages adaptively by first querying $\mathrm{SH}_{\mathsf{key}}$ and then $\mathrm{SH}_{\mathsf{pub}}$ with a predicate ϕ and a value nonce. The adversary's goal is still to forge a valid $(\phi^*, \mathsf{nonce}^*, \tau^*)$ for a predicate ϕ^* not satisfied by any extracted attributes and without replaying honest users' showings.

Compared to the original KVAC unforgeability in [23], we rely on an extractor instead of having the adversary reveals the attributes, but we do not give the adversary access to a verification oracle. Compared to the extractability definition of KVAC in [38], we do not require an extractor for the final forgery. In their game, the issuer oracle also extracts the underlying attributes; however, the game aborts if they do not satisfy the predicate, instead of allowing the adversary to win (as in our case).

Anonymity. A KVAC scheme is anonymous if there exists a simulator $\mathsf{Sim}_{\mathsf{Gen}}$ which generates par_g indistinguishable from Gen and a simulator $\mathsf{Sim} = (\mathsf{Sim}_{\mathsf{Setup}}, \mathsf{Sim}_{\mathsf{U}}, \mathsf{Sim}_{\mathsf{Show}})$ such that

1. The distribution of $\mathsf{par}_{\mathsf{KVAC}}$ from $\mathsf{KVAC.Setup}(\mathsf{par}_g)$ and $\mathsf{Sim}_{\mathsf{Setup}}(\mathsf{par}_g)$ for $\mathsf{par}_g \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)$ are indistinguishable.

Game $\mathrm{UNF}^{\mathcal{A}}_{\mathsf{KVAC},\mathsf{Ext},\mathcal{O}}(\lambda)$:

$\mathsf{MsgQ}, \mathsf{PfQ}, \mathcal{C}, \mathcal{S} \leftarrow \varnothing$; $\mathsf{sctr}, \mathsf{win} \leftarrow 0$

$\mathsf{par}_g \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)$; $(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{td}) \leftarrow\!\!\$\ \mathsf{Ext}_{\mathsf{Setup}}(1^\ell, \mathsf{par}_g)$

$(\mathsf{sk}, \mathsf{pk}) \leftarrow\!\!\$\ \mathsf{KVAC.KeyGen}(\mathsf{par}_{\mathsf{KVAC}})$

$(\tau^*, \phi^*, \mathsf{nonce}^*) \leftarrow\!\!\$$
$\mathcal{A}^{\mathrm{Iss},\mathrm{NewUsr},\mathrm{SH}_{\mathsf{key}},\mathrm{SH}_{\mathsf{pub}},\mathcal{O}(\mathsf{par}_g,\mathsf{sk},(\mathsf{par}_{\mathsf{KVAC}},\mathsf{pk}),\cdot)}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk})$

if $(\mathsf{KVAC.SVer}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{sk}, \mathsf{pk}, \tau^*, \phi^*, \mathsf{nonce}^*) = 1) \wedge$
$(\forall \boldsymbol{m} \in \mathsf{MsgQ} : \phi^*(\boldsymbol{m}) = 0) \wedge$
$((\phi^*, \mathsf{nonce}^*, \tau^*) \notin \mathsf{PfQ})$ **then**
return 1

return win

Oracle $\mathrm{Iss}(\mu, \phi)$:

$\mathsf{imsg} \leftarrow\!\!\$\ \mathsf{KVAC.Iss}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{sk}, \mu, \phi)$

if $\mathsf{imsg} = \bot$ **then abort**

$\boldsymbol{m} \leftarrow \mathsf{Ext}_{\mathsf{Iss}}(\mathsf{td}, \mu, \phi)$

if $\boldsymbol{m} = \bot \ \vee\ \phi(\boldsymbol{m}) = 0$ **then**
$\mathsf{win} \leftarrow 1$ // $\mathcal{A}$ wins if it can request
// credentials for non-authorized attributes

$\mathsf{MsgQ} \leftarrow \mathsf{MsgQ} \cup \{\boldsymbol{m}\}$

return imsg

Oracle $\mathrm{NewUsr}(\mathsf{cid}, \boldsymbol{m}, \phi)$:

if $\mathsf{cid} \in \mathcal{C} \ \vee\ \phi(\boldsymbol{m}) = 0$ **then**
return $\bot$

$\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathsf{cid}\}$; $\boldsymbol{m}_{\mathsf{cid}} \leftarrow \boldsymbol{m}$

$\sigma_{\mathsf{cid}} \leftarrow\!\!\$\ \langle \mathsf{KVAC.Iss}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{sk}, \phi)$
$\rightleftharpoons \mathsf{KVAC.U}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}, \boldsymbol{m}, \phi)\rangle$

return `closed`

Oracle $\mathrm{SH}_{\mathsf{key}}(\mathsf{cid})$:

if $\mathsf{cid} \notin \mathcal{C}$ **then abort**

$\mathsf{sctr} \leftarrow \mathsf{sctr} + 1$

$(\tau_{\mathsf{key},\mathsf{sctr}}, \mathsf{st}_{\mathsf{sctr}}) \leftarrow\!\!\$$
$\mathsf{KVAC.Show}_{\mathsf{key}}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}, \boldsymbol{m}_{\mathsf{cid}}, \sigma_{\mathsf{cid}})$

return $(\mathsf{sctr}, \tau_{\mathsf{key},\mathsf{sctr}})$

Oracle $\mathrm{SH}_{\mathsf{pub}}(\mathsf{sid}, \phi, \mathsf{nonce})$:

if $\mathsf{sid} \in \mathcal{S} \ \vee\ \mathsf{sid} > \mathsf{sctr}$ **then abort**

$\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathsf{sid}\}$

$\tau_{\mathsf{pub}} \leftarrow\!\!\$\ \mathsf{KVAC.Show}_{\mathsf{pub}}(\mathsf{st}_{\mathsf{sid}}, \phi, \mathsf{nonce})$

$\tau \leftarrow (\tau_{\mathsf{key},\mathsf{sid}}, \tau_{\mathsf{pub}})$

$\mathsf{PfQ} \leftarrow \mathsf{PfQ} \cup \{(\phi, \mathsf{nonce}, \tau)\}$

return τ_{pub}

Game $\mathrm{Anon}^{\mathcal{A}}_{\mathsf{KVAC},\mathsf{Sim}_{\mathsf{Gen}},\mathsf{Sim},b}(\lambda)$:

$\mathsf{sctr} \leftarrow 0; \mathcal{S} \leftarrow \varnothing$

$(\mathsf{par}_g, \mathsf{td}_g) \leftarrow\!\!\$\ \mathsf{Sim}_{\mathsf{Gen}}(1^\lambda)$

$(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{td}_{\mathsf{KVAC}}) \leftarrow\!\!\$\ \mathsf{Sim}_{\mathsf{Setup}}(1^\ell, \mathsf{par}_g)$

$\mathsf{td} \leftarrow (\mathsf{td}_g, \mathsf{td}_{\mathsf{KVAC}})$

$(\mathsf{pk}, \boldsymbol{m}, \tilde{\phi}, \mathsf{st}_{\mathcal{A}}) \leftarrow\!\!\$\ \mathcal{A}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{td})$

if $\tilde{\phi}(\boldsymbol{m}) = 0$ **then return** 1

$(\mu, \mathsf{st}^u) \leftarrow\!\!\$\ \mathsf{KVAC.U}_1(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}, \boldsymbol{m}, \tilde{\phi})$ // $b = 0$

$(\mu, \mathsf{st}_{\mathsf{Sim}}) \leftarrow\!\!\$\ \mathsf{Sim}_{\mathsf{U}}(\mathsf{td}, \mathsf{pk}, \tilde{\phi})$ // $b = 1$

$(\mathsf{imsg}, \mathsf{st}'_{\mathcal{A}}) \leftarrow\!\!\$\ \mathcal{A}(\mathsf{st}_{\mathcal{A}}, \mu)$

$\sigma \leftarrow\!\!\$\ \mathsf{KVAC.U}_2(\mathsf{st}^u, \mathsf{imsg})$ // $b = 0$

$\sigma \leftarrow\!\!\$\ \mathsf{Sim}_{\mathsf{U}}(\mathsf{st}_{\mathsf{Sim}}, \mathsf{imsg})$ // $b = 1$

if $\sigma = \bot$ **then return** 1

$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{SH}_{\mathsf{key}},\mathrm{SH}_{\mathsf{pub}}}(\mathsf{st}'_{\mathcal{A}})$

return b'

Oracle $\mathrm{SH}_{\mathsf{key}}()$:

$\mathsf{sctr} \leftarrow \mathsf{sctr} + 1$

$(\tau_{\mathsf{key},\mathsf{sctr}}, \mathsf{st}_{\mathsf{sctr}})$ // $b = 0$
$\leftarrow\!\!\$\ \mathsf{KVAC.Show}_{\mathsf{key}}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}, \boldsymbol{m}, \sigma)$

$(\tau_{\mathsf{key},\mathsf{sctr}}, \mathsf{st}_{\mathsf{sctr}}) \leftarrow\!\!\$\ \mathsf{Sim}_{\mathsf{Show}}(\text{“key”}, \mathsf{td}, \mathsf{pk})$
// $b = 1$

return $(\mathsf{sctr}, \tau_{\mathsf{key},\mathsf{sctr}})$

Oracle $\mathrm{SH}_{\mathsf{pub}}(\mathsf{sid}, \phi, \mathsf{nonce})$:

if $\phi(\boldsymbol{m}) = 0 \ \vee\ \mathsf{sid} \in \mathcal{S} \ \vee\ \mathsf{sid} > \mathsf{sctr}$
then abort

$\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathsf{sid}\}$

$\tau_{\mathsf{pub}} \leftarrow\!\!\$\ \mathsf{KVAC.Show}_{\mathsf{pub}}(\mathsf{st}_{\mathsf{sid}}, \phi, \mathsf{nonce})$ // $b = 0$

$\tau_{\mathsf{pub}} \leftarrow\!\!\$\ \mathsf{Sim}_{\mathsf{Show}}(\text{“pub”}, \mathsf{st}_{\mathsf{sid}}, \phi, \mathsf{nonce})$ // $b = 1$

return τ_{pub}

Fig. 5. Unforgeability and anonymity game for $\mathsf{KVAC} = \mathsf{KVAC}[\mathsf{Gen}, \Phi, \mathcal{M}]$ on the top and bottom, respectively. We note that both the adversary and the simulator are given access to the global trapdoor td_g and KVAC trapdoor $\mathsf{td}_{\mathsf{KVAC}}$. We assume that all the predicates output by $\mathcal{A}$ are in Φ.

2. No adversary can distinguish between interactions with an honest user and interactions with the simulator Sim. The advantage of $\mathcal{A}$'s in the anonymity game in Fig. 5 is $\mathsf{Adv}^{\mathsf{anon}}_{\mathsf{KVAC},\mathsf{Sim}_{\mathsf{Gen}},\mathsf{Sim}}(\mathcal{A},\lambda) := |\mathsf{Pr}[\mathrm{Anon}^{\mathcal{A}}_{\mathsf{KVAC},\mathsf{Sim}_{\mathsf{Gen}},\mathsf{Sim},0}(\lambda)=1] - \mathsf{Pr}[\mathrm{Anon}^{\mathcal{A}}_{\mathsf{KVAC},\mathsf{Sim}_{\mathsf{Gen}},\mathsf{Sim},1}(\lambda)=1]|$.

The anonymity game of KVAC's is similar to that of SAAC's without the helper, except that we split the showing oracle into $\mathrm{SH}_{\mathsf{key}}$ and $\mathrm{SH}_{\mathsf{pub}}$. This allows the adversary to adaptively choose the predicate ϕ and value nonce depending on τ_{key}. Compared to the anonymity definition in [23], our definition incorporates blind issuance and considers maliciously generated key.

Integrity of issued credentials. No adversary can force the honest user to output an invalid showing message even when the public key pk is adversarially chosen and the public parameters $\mathsf{par}_{\mathsf{KVAC}}$ are sampled with a trapdoor using the simulator $\mathsf{Sim}_{\mathsf{Gen}}$ and Sim (defined in the anonymity definition). Denote the integrity advantage of $\mathcal{A}$ as $\mathsf{Adv}^{\mathsf{integ}}_{\mathsf{KVAC},\mathsf{Sim}_{\mathsf{Gen}},\mathsf{Sim}}(\mathcal{A},\lambda) :=$

$$\mathsf{Pr}\left[\begin{array}{l|l} \sigma \neq \bot \wedge & (\mathsf{par}_g, \mathsf{td}_g) \leftarrow\!\!\$\ \mathsf{Sim}_{\mathsf{Gen}}(1^\lambda) \\ (\mathsf{pk}, \tau_{\mathsf{key}}) \notin \mathcal{L}_{\mathsf{V},\mathsf{par}_g} & (\mathsf{par}_{\mathsf{KVAC}}, \mathsf{td}_{\mathsf{KVAC}}) \leftarrow\!\!\$\ \mathsf{Sim}_{\mathsf{Setup}}(1^\ell, \mathsf{par}_g) \\ & (\mathsf{pk}, \boldsymbol{m}, \phi, \mathsf{st}) \leftarrow\!\!\$\ \mathcal{A}(\mathsf{par}_{\mathsf{KVAC}}, (\mathsf{td}_g, \mathsf{td}_{\mathsf{KVAC}})) \\ & \textbf{if } \phi(\boldsymbol{m}) = 0 \textbf{ then abort} \\ & (\bot, \sigma) \leftarrow\!\!\$\ \langle \mathcal{A}(\mathsf{st}) \rightleftharpoons \mathsf{KVAC.U}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}, \boldsymbol{m}, \phi)\rangle \\ & (\tau_{\mathsf{key}}, \mathsf{st}) \leftarrow\!\!\$\ \mathsf{KVAC.Show}_{\mathsf{key}}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}, \boldsymbol{m}, \sigma) \end{array}\right] .$$

Validity of key generation with respect to Ext: For any $\lambda, \ell = \ell(\lambda) \in \mathbb{N}$, $\mathsf{par}_g \in [\mathsf{Gen}(1^\lambda)]$, $(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{td}) \in [\mathsf{Ext}_{\mathsf{Setup}}(1^\ell, \mathsf{par}_g)]$ and $((\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}), \tau_{\mathsf{key}}) \in \mathcal{L}_{\mathsf{V},\mathsf{par}_g}$, for any sk that corresponds to pk (i.e., $(\mathsf{sk}, \mathsf{pk}) \in [\mathsf{KVAC.KeyGen}(\mathsf{par}_{\mathsf{KVAC}})]$), we have $((\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}), \tau_{\mathsf{key}}), \mathsf{sk}) \in \mathsf{R}_{\mathsf{V},\mathsf{par}_g}$. This property ensures that for any τ_{key} that is valid for some secret key sk which corresponds to the public key pk, it should also be valid for any other secret key sk' corresponding to pk. This property is satisfied if *the secret key is unique for each public key.*

Oblivious issuance of non-interactive proofs. An oblivious issuance of non-interactive proofs $\mathsf{oNIP} = \mathsf{oNIP}[\mathsf{Gen}, \mathsf{R}]$ defined with respect to Gen and a family of relations $\mathsf{R} = \{\mathsf{R}_{\mathsf{par}_g}\}_{\mathsf{par}_g}$ consists of the following algorithms.

- $\mathsf{par}_{\mathsf{oNIP}} \leftarrow\!\!\$\ \mathsf{oNIP.Setup}(\mathsf{par}_g)$ outputs public parameters $\mathsf{par}_{\mathsf{oNIP}}$. The input par_g defines the relation $\mathsf{R} = \mathsf{R}_{\mathsf{par}_g}$, omitting subscript par_g when clear from the context. *We also assume that* $\mathsf{par}_{\mathsf{oNIP}}$ *contains* par_g.
- $(\bot, \pi) \leftarrow\!\!\$\ \langle \mathsf{oNIP.Iss}(\mathsf{par}_{\mathsf{oNIP}}, x, X) \rightleftharpoons \mathsf{oNIP.U}(\mathsf{par}_{\mathsf{oNIP}}, X, Y)\rangle$ is a r-round interactive protocol starting with the user algorithm $\mathsf{oNIP.U}_1$ and concluding with $\mathsf{oNIP.U}_{r+1}$ outputting the proof π.
- $0/1 \leftarrow \mathsf{oNIP.Ver}(\mathsf{par}_{\mathsf{oNIP}}, (X, Y), \pi)$ outputs a bit.

Our syntax deviates from [37] in that the user algorithm does not output an augmented statement Z, but the user takes as input the augmented statement Y (which we think of as (Y, Z) in their work). We require oNIP to satisfy the following properties, but unlike [37], unforgeability is not required for our generic

construction. We refer the readers to the full version for the standard definitions of η-correctness, parameter-indistinguishability, and soundness.

Zero-knowledge. Let $\mathcal{O}(\mathsf{par}_g, x, X, \cdot)$ be a deterministic oracle embedded with par_g (which defines $\mathsf{R}_{\mathsf{par}_g}$), and statement and witness X, x and taking in a to-be-determined input. An oNIP is $\mathcal{O}$-Zero-knowledge if there exists a simulator $\mathsf{Sim} = (\mathsf{Sim}_{\mathsf{Setup}}, \mathsf{Sim}_{\mathsf{Iss}})$, such that no adversary can distinguish between an honest issuer using the witness x from a simulator who does not know the witness. Unconventionally, our simulator Sim is assisted by the oracle $\mathcal{O}$ embedded with x, modeling witness-dependent computation that is not efficiently simulatable (e.g., checking if a rerandomized statement is in the language). The advantage of $\mathcal{A}$ in the ZK game in Fig. 6 is $\mathsf{Adv}^{\mathsf{zk}}_{\mathsf{oNIP},\mathsf{Sim},\mathcal{O}}(\mathcal{A},\lambda) := |\mathsf{Pr}[\mathrm{ZK}^{\mathcal{A}}_{\mathsf{oNIP},\mathsf{Sim},\mathcal{O},0}(\lambda)=1] - \mathsf{Pr}[\mathrm{ZK}^{\mathcal{A}}_{\mathsf{oNIP},\mathsf{Sim},\mathcal{O},1}(\lambda)=1]|$.

Obliviousness for valid statements. An oNIP is oblivious for valid statements if there exists a simulator $\mathsf{Sim}_{\mathsf{Gen}}$ generating par_g indistinguishable from Gen and a simulator $\mathsf{Sim} = (\mathsf{Sim}_{\mathsf{Setup}}, \mathsf{Sim}_{\mathsf{U}}, \mathsf{Sim}_{\mathrm{Pf}})$ such that

1. The distribution of $\mathsf{par}_{\mathsf{oNIP}}$ from $\mathsf{oNIP.Setup}(\mathsf{par}_g)$ and $\mathsf{Sim}_{\mathsf{Setup}}(\mathsf{par}_g)$ for $\mathsf{par}_g \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)$ are indistinguishable.
2. The adversary $\mathcal{A}$, given the simulation trapdoor, cannot distinguish between an honest user who obtains the proof from the issuance protocol and a simulator who simulates the proof independent of the protocol. Importantly, the simulator only gets the 'core' statement X but not the 'augmented' statement Y_{sid} during the protocol. The advantage of $\mathcal{A}$ in the obliviousness game in Fig. 6 is defined as $\mathsf{Adv}^{\mathsf{oblv}}_{\mathsf{oNIP},\mathsf{Sim}_{\mathsf{Gen}},\mathsf{Sim}}(\mathcal{A},\lambda) := |\mathsf{Pr}[\mathrm{OBLV}^{\mathcal{A}}_{\mathsf{oNIP},\mathsf{Sim}_{\mathsf{Gen}},\mathsf{Sim},0}(\lambda)=1] - \mathsf{Pr}[\mathrm{OBLV}^{\mathcal{A}}_{\mathsf{oNIP},\mathsf{Sim}_{\mathsf{Gen}},\mathsf{Sim},1}(\lambda)=1]|$.

Our obliviousness definition is simulation-based instead of the definition in [37]. Further, it only applies for statements in the language and not any statements.

4.2 Construction

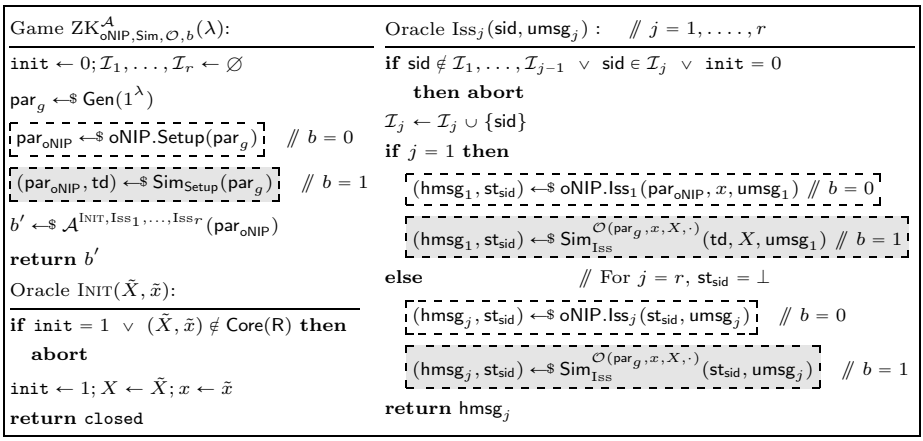
We construct below a SAAC scheme $\mathsf{SAAC} = \mathsf{SAAC}[\mathsf{Gen}, \mathsf{KVAC}, \mathsf{oNIP}]$ for predicate family Φ and attribute space $\mathcal{M}$, using $\mathsf{KVAC} = \mathsf{KVAC}[\mathsf{Gen}, \Phi, \mathcal{M}]$ and $\mathsf{oNIP} = \mathsf{oNIP}[\mathsf{Gen}, \mathsf{R}_{\mathsf{V}}]$ for the relation family R_{V} defined by the $\mathsf{KVAC.SVer}_{\mathsf{key}}$ algorithm. The main idea is to replace the key-dependent verification $\mathsf{KVAC.SVer}_{\mathsf{key}}$ with a proof generated from oNIP.

Setup: $\mathsf{SAAC.Setup}(1^\lambda)$: $\mathsf{par}_g \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)$, $\mathsf{par}_{\mathsf{KVAC}} \leftarrow\!\!\$\ \mathsf{KVAC.Setup}(1^\ell, \mathsf{par}_g)$, and $\mathsf{par}_{\mathsf{oNIP}} \leftarrow\!\!\$\ \mathsf{oNIP.Setup}(\mathsf{par}_g)$. Return $\mathsf{par} = (\mathsf{par}_{\mathsf{KVAC}}, \mathsf{par}_{\mathsf{oNIP}})$

Key generation and Issuance: These are defined exactly as those of KVAC.

Helper protocol: $\langle \mathsf{SAAC.Helper}(\mathsf{par}, \mathsf{sk}) \rightleftharpoons \mathsf{SAAC.ObtHelp}(\mathsf{par}, \mathsf{pk}, \sigma)\rangle$ is defined as follows:

- First, $\mathsf{SAAC.ObtHelp}$ runs $(\tau_{\mathsf{key}}, \mathsf{st}) \leftarrow\!\!\$\ \mathsf{KVAC.Show}_{\mathsf{key}}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}, \boldsymbol{m}, \sigma)$.
- Then, SAAC.Helper and $\mathsf{SAAC.ObtHelp}$ run the protocol $(\bot, \pi_{\mathsf{V}}) \leftarrow\!\!\$\ \langle \mathsf{oNIP.Iss}(\mathsf{par}_{\mathsf{oNIP}}, \mathsf{sk}, (\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk})) \rightleftharpoons \mathsf{oNIP.U}(\mathsf{par}_{\mathsf{oNIP}}, (\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}), \tau_{\mathsf{key}})\rangle$.



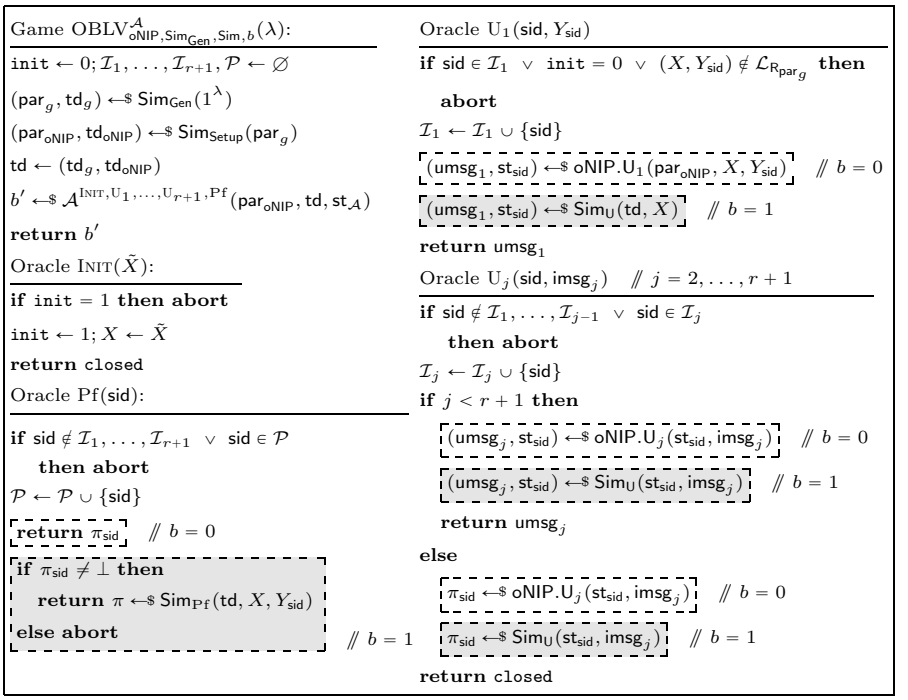


Fig. 6. Zero-knowledge and obliviousness games of $\mathsf{oNIP} = \mathsf{oNIP}[\mathsf{Gen}, \mathsf{R}]$ on the top and bottom, respectively. The ZK game is parameterized by the simulator Sim with access to the oracle $\mathcal{O}$. As with the KVAC's anonymity definition, both the adversary and the simulator in OBLV game are given access to the global trapdoor td_g and oNIP trapdoor $\mathsf{td}_{\mathsf{oNIP}}$. Crucially, the OBLV simulator gets the 'core' statement X but not the 'augmented' statement Y during the protocol.

- Finally, $\mathsf{SAAC.ObtHelp}$ returns $\mathsf{aux} = (\tau_{\mathsf{key}}, \pi_{\mathsf{V}}, \mathsf{st})$.

Show: $\mathsf{SAAC.Show}(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}, \sigma, \mathsf{aux} = (\tau_{\mathsf{key}}, \pi_{\mathsf{V}}, \mathsf{st}), \phi, \mathsf{nonce})$: computes $\tau_{\mathsf{pub}} \leftarrow\!\!\$$ $\mathsf{KVAC.Show}_{\mathsf{pub}}(\mathsf{st}, \phi, (\pi_{\mathsf{V}}, \mathsf{nonce}))$ and returns $\pi = (\tau_{\mathsf{key}}, \tau_{\mathsf{pub}}, \pi_{\mathsf{V}})$.

Verify: $\mathsf{SAAC.SVer}(\mathsf{par}, \mathsf{pk}, \pi = (\tau_{\mathsf{key}}, \tau_{\mathsf{pub}}, \pi_{\mathsf{V}}), \phi, \mathsf{nonce})$: returns $b_0 \wedge b_1$ where

- $b_0 \leftarrow \mathsf{oNIP.Ver}(\mathsf{par}, (\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}), \tau_{\mathsf{key}}, \pi_{\mathsf{V}})$
- $b_1 \leftarrow \mathsf{KVAC.SVer}_{\mathsf{pub}}(\mathsf{par}, \mathsf{pk}, (\tau_{\mathsf{key}}, \tau_{\mathsf{pub}}), \phi, (\pi_{\mathsf{V}}, \mathsf{nonce}))$

The following theorem then establishes the properties of our generic SAAC construction. Correctness follows from the correctness of both building blocks. We refer to the overview Sect. 1.1 for a proof sketch and to the full version for the formal proofs and concrete security bounds.

Theorem 4.1. *Let* $\ell = \ell(\lambda)$ *and* Gen *be a global parameters generator,* KVAC *be a keyed-verification anonymous credential, and* oNIP *be an oblivious proof issuance protocol for the relation family* R_{V} *induced by* $\mathsf{KVAC.SVer}_{\mathsf{key}}$*. Then, the server-aided anonymous credential scheme* $\mathsf{SAAC} = \mathsf{SAAC}[\mathsf{Gen}, \mathsf{KVAC}, \mathsf{oNIP}]$ *is*

- $(\eta_{\mathsf{KVAC}} + \eta_{\mathsf{oNIP}})$*-correct if* KVAC *is* η_{KVAC}*-correct and* oNIP *is* η_{oNIP}*-correct.*
- *Unforgeable if there exists an oracle* $\mathcal{O}$ *such that* oNIP *is* $\mathcal{O}$*-zero-knowledge and sound and* KVAC *satisfies* $\mathcal{O}$*-unforgeability and validity of key generation with respect to the same extractor* Ext*.*
- *Anonymous if there exist simulators* $\mathsf{Sim}_{\mathsf{Gen}}, \mathsf{Sim}_{\mathsf{oNIP}}, \mathsf{Sim}_{\mathsf{KVAC}}$ *such that* oNIP *is oblivious with respect to* $\mathsf{Sim}_{\mathsf{Gen}}$ *and* $\mathsf{Sim}_{\mathsf{oNIP}}$*, and* KVAC *satisfies anonymity and integrity with respect to* $\mathsf{Sim}_{\mathsf{Gen}}$ *and* $\mathsf{Sim}_{\mathsf{KVAC}}$*.*

5 Instantiations

In this section, we give two SAAC instantiations in pairing-free groups from two KVAC schemes based on algebraic MACs. In Sect. 5.1, we give a scheme adapting Barki et al.'s KVAC [6] based on BBS-MAC. In Sect. 5.2, we give less efficient scheme adapting ideas from Chase et al.'s DDH-based KVAC [23]. For both instantiations, we construct a suitable oNIP protocol.

Both KVACs use three proof systems: Π_{com} proving that committed attributes satisfy an issuance predicate, Π_{σ} proving correct issuance of credentials, and Π_{pub} used for showing a credential.[5] Except for the Π_{com} which is instantiated from the Fischlin transform [27,30], Π_{σ} and Π_{pub} are obtained by applying the Fiat-Shamir transform to Σ-protocols for linear relations [34] (see the proof system Lin in Fig. 7). Crucially, the prover of Π_{pub} takes as input a string nonce which is hashed by H. This is *necessary to achieve our stronger KVAC security and ensure non-malleability of honest users' showings* in our SAAC instantiations. We refer the full version for more details on these NIZKs.

[5] We will refer to them as $\Pi_{\mathsf{com}}, \Pi_{\sigma}, \Pi_{\mathsf{pub}}$ for both instantiations, but note that they are different proof systems.

$\mathsf{Lin.Prove}^{\mathsf{H}}((M \in \mathbb{G}^{n\times m}, \boldsymbol{Y} \in \mathbb{G}^n), \boldsymbol{x} \in \mathbb{Z}_p^m, \mathsf{nonce})$	$\mathsf{Lin.Ver}^{\mathsf{H}}((M \in \mathbb{G}^{n\times m}, \boldsymbol{Y} \in \mathbb{G}^n), \pi, \mathsf{nonce})$
$\boldsymbol{r} \leftarrow\$ \mathbb{Z}_p^m; \boldsymbol{R} \leftarrow M\boldsymbol{r}; c \leftarrow \mathsf{H}(M, \boldsymbol{Y}, \boldsymbol{R}, \mathsf{nonce})$	$(c, \boldsymbol{s}) \leftarrow \pi$
$\boldsymbol{s} \leftarrow \boldsymbol{r} + c \cdot \boldsymbol{x}$	$\boldsymbol{R} \leftarrow M\boldsymbol{s} - c \cdot \boldsymbol{Y}$
return $\pi := (c, \boldsymbol{s})$	**return** $\mathsf{H}(M, \boldsymbol{Y}, \boldsymbol{R}, \mathsf{nonce}) = c$

Fig. 7. NIZK proof system $\mathsf{Lin} = \mathsf{Lin}[\mathsf{H}, \mathbb{G}]$ for $\mathsf{R}_{\mathbb{G}} := \{((M, \boldsymbol{Y}), \boldsymbol{x}) : \boldsymbol{Y} = M\boldsymbol{x}\}$. The prover optionally takes an input nonce which will also be hashed by H.

5.1 Instantiation from BBS

In this section, we instantiate our SAAC construction with a KVAC based on the BBS MAC, which can be seen as a variant of Barki et al.'s KVAC [6], and a corresponding oNIP scheme. Following the syntax in Sect. 4.1, we note that our *global parameters generator* is exactly the group generator GGen and the simulator $\mathsf{Sim}_{\mathsf{Gen}}$ simply runs GGen and does not output any trapdoor.

BBS-BASED KVAC. We first describe the $\mathsf{KVAC}_{\mathsf{BBS}}$ scheme in Fig. 8, which can be seen as a variant of the KVAC from [6]. The credential for the attributes $\boldsymbol{m} = (m_i)_{i=1}^{\ell}$ is computed as $(A := (x+e)^{-1}C, e \leftarrow\$ \mathbb{Z}_p, s \leftarrow\$ \mathbb{Z}_p)$ where $x \in \mathbb{Z}_p$ is the secret key, $C = G + \sum_{i=1}^{\ell} m_i H_i + sH_{\ell+1}$, and $H_1 \ldots, H_{\ell+1} \in \mathbb{G}$ are parts of the public parameters. To show a credential, a holder can sample $r, r' \leftarrow\$ \mathbb{Z}_p$ and compute $\tilde{C} \leftarrow rC$, $\tilde{A} \leftarrow r'rA$, and $\tilde{B} \leftarrow r'\tilde{C} - e\tilde{A}$. The holder sends to the issuer $(\tilde{A}, \tilde{B}, \tilde{C})$, along with a proof of knowledge of $e, r, r', \boldsymbol{m}$ (using CDL proofs [15]), and the issuer can check that $x\tilde{A} = \tilde{B}$. To bind a value nonce to the showing message, the hash computation in Π_{pub} also takes nonce as an input. We emphasize that this is crucial for the security of our final SAAC construction. Our KVAC makes use of proof systems $\Pi_{\mathsf{com}}, \Pi_{\sigma}$, and Π_{pub} for the following relations (implicitly parameterized by the group description), respectively:

$$\mathsf{R}_{\mathsf{com}} := \{((\boldsymbol{H}, C, \psi), (s, \boldsymbol{m})) : C = sH_{\ell+1} + \textstyle\sum_{i=1}^{\ell} m_i H_i \wedge \psi(\boldsymbol{m}) = 1\}$$

$$\mathsf{R}_{\sigma} := \{((X, A, B), x) : xG = X \wedge xA = B\}$$

$$\mathsf{R}_{\mathsf{pub}} := \left\{ ((\tilde{A}, \tilde{B}, \tilde{C}, \boldsymbol{H}_{\mathrm{priv}}, Y), (e, r', r'', \hat{\boldsymbol{m}}, s) : \begin{matrix} r''\tilde{C} + \langle \boldsymbol{H}_{\mathrm{priv}}, (\hat{\boldsymbol{m}} \| s) \rangle = Y \wedge \\ \tilde{B} = r'\tilde{C} - e\tilde{A} \end{matrix} \right\} .$$

We require Π_{com} to be straightline-extractable for the relaxed relation $\widetilde{\mathsf{R}}_{\mathsf{com}}$ which in addition to statement and witness $((\boldsymbol{H}, C, \psi), (s, \boldsymbol{m})) \in \mathsf{R}_{\mathsf{com}}$ also accepts witnesses $(s \| \boldsymbol{m}) \neq \boldsymbol{0}$ such that $0_{\mathbb{G}} = \sum_{i=1}^{\ell} m_i H_i + sH_{\ell+1}$. For the relations R_{σ} and $\mathsf{R}_{\mathsf{pub}}$, we explicitly define the corresponding linear maps as follows: $M^{\sigma}_{G,A} := (G, A)^T$ and $M^{\mathsf{pub}}_{\tilde{C}, H_{\mathrm{priv},1}, \ldots, H_{\mathrm{priv},k}, \tilde{A}} := \begin{pmatrix} \tilde{C} & H_{\mathrm{priv},1} & \cdots & H_{\mathrm{priv},k} & 0 & 0 \\ 0 & 0 & \cdots & 0 & \tilde{C} & -\tilde{A} \end{pmatrix}$. We point out that $\mathsf{SVer}_{\mathsf{key}}$ induces the following DLEQ relation (parameterized by $\mathsf{par}_g = (p, G, \mathbb{G})$ which we will omit) for which we give a corresponding oNIP protocol.

$$\mathsf{R}_{\mathsf{dleq}} := \{((X, (\tilde{A}, \tilde{B})), x) : X = xG \ \wedge \ \tilde{B} = x\tilde{A}\} , \tag{1}$$

The augmented statement is $(\tilde{A}, \tilde{B})$, and the core relation $\mathsf{Core}(\mathsf{R}_{\mathsf{dleq}})$ contains public-secret key pairs $(X = xG, x)$ defined by the key generation of $\mathsf{KVAC}_{\mathsf{BBS}}$.

$\underline{\mathsf{KVAC}_{\mathsf{BBS}}.\mathsf{Setup}(1^\ell, \mathsf{par}_g = (p, G, \mathbb{G}))}$
Select $\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2 : \{0,1\}^* \to \mathbb{Z}_p$
$\boldsymbol{H} = (H_i)_{i=1}^{\ell+1} \leftarrow\$\ \mathbb{G}^{\ell+1}$
$\Pi_\sigma \leftarrow \mathsf{Lin}[\mathsf{H}_1, \mathbb{G}]$; $\Pi_{\mathsf{pub}} \leftarrow \mathsf{Lin}[\mathsf{H}_2, \mathbb{G}]$
return $\mathsf{par} = (p, G, \mathbb{G}, \boldsymbol{H}, \mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2)$

$\underline{\mathsf{KVAC}_{\mathsf{BBS}}.\mathsf{KeyGen}(\mathsf{par})}$
$x \leftarrow\$\ \mathbb{Z}_p$; $X \leftarrow xG$
return $(\mathsf{sk} \leftarrow x, \mathsf{pk} \leftarrow X)$

$\underline{\mathsf{KVAC}_{\mathsf{BBS}}.\mathsf{Iss}(\mathsf{par}, x, \psi, \mu = (C, \pi_{\mathsf{com}}))}$
if $C + G = 0_{\mathbb{G}} \vee \Pi_{\mathsf{com}}.\mathsf{Ver}^{\mathsf{H}_0}((\boldsymbol{H}, C, \psi), \pi_{\mathsf{com}}) = 0$ **then abort**
$e \leftarrow\$\ \mathbb{Z}_p$; $A \leftarrow (x+e)^{-1}(G + C)$; $B \leftarrow C - eA$
$\pi_\sigma \leftarrow \Pi_\sigma.\mathsf{Prove}^{\mathsf{H}_1}((M^\sigma_{G,A}, (X, B)), x)$
return $\mathsf{imsg} \leftarrow (A, e, \pi_\sigma)$

$\underline{\mathsf{KVAC}_{\mathsf{BBS}}.\mathsf{SVer}_{\mathsf{key}}(\mathsf{par}, x, \tau_{\mathsf{key}} = (\tilde{A}, \tilde{B}))}$
return $x\tilde{A} = \tilde{B}$

$\underline{\mathsf{KVAC}_{\mathsf{BBS}}.\mathsf{SVer}_{\mathsf{pub}}(\mathsf{par}, X, \tau_{\mathsf{key}}, \tau_{\mathsf{pub}}, \phi_{\boldsymbol{I},\boldsymbol{a}}, \mathsf{nonce})}$
parse $(\tilde{A}, \tilde{B}) \leftarrow \tau_{\mathsf{key}}$; $(\tilde{C}, \pi_{\mathsf{pub}}) \leftarrow \tau_{\mathsf{pub}}$
$\boldsymbol{H}_{\mathrm{priv}} \leftarrow (H_i)_{i \in [\ell+1] \setminus \boldsymbol{I}}$
$Y \leftarrow G + \langle \boldsymbol{a}, (H_i)_{i \in \boldsymbol{I}} \rangle$
return $\Pi_{\mathsf{pub}}.\mathsf{Ver}^{\mathsf{H}_2}((M^{\mathsf{pub}}_{\tilde{C}, \boldsymbol{H}_{\mathrm{priv}}, \tilde{A}}, (Y, \tilde{B})), \pi_{\mathsf{pub}}, (\phi_{\boldsymbol{I},\boldsymbol{a}}, \mathsf{nonce}))$

$\underline{\mathsf{KVAC}_{\mathsf{BBS}}.\mathsf{U}_1(\mathsf{par}, X, \boldsymbol{m} \in \mathbb{Z}_p^\ell, \psi)}$
$s \leftarrow\$\ \mathbb{Z}_p$; $C \leftarrow sH_{\ell+1} + \sum_{i=1}^{\ell} m_i H_i$
if $C + G = 0_{\mathbb{G}}$ **then abort**
$\pi_{\mathsf{com}} \leftarrow \Pi_{\mathsf{com}}.\mathsf{Prove}^{\mathsf{H}_0}((\boldsymbol{H}, C, \psi), (s, \boldsymbol{m}))$
return $\mu := (C, \pi_{\mathsf{com}})$

$\underline{\mathsf{KVAC}_{\mathsf{BBS}}.\mathsf{U}_2(\mathsf{imsg} = (A, e, \pi_\sigma))}$
$B \leftarrow G + C - eA$
if $\Pi_\sigma.\mathsf{Ver}^{\mathsf{H}_1}((M^\sigma_{G,A}, (X, B)), \pi_\sigma) = 0$ **then abort**
return $\sigma \leftarrow (A, e, s)$

$\underline{\mathsf{KVAC}_{\mathsf{BBS}}.\mathsf{Show}_{\mathsf{key}}(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}, \sigma = (A, e, s))}$
$r, r' \leftarrow\$\ \mathbb{Z}_p^*$
$\tilde{C} \leftarrow r(G + sH_{\ell+1} + \sum_{i=1}^{\ell} m_i H_i)$
$\tilde{A} \leftarrow r'rA$; $\tilde{B} \leftarrow r'\tilde{C} - e\tilde{A}$
return $\tau_{\mathsf{key}} := (\tilde{A}, \tilde{B})$

$\underline{\mathsf{KVAC}_{\mathsf{BBS}}.\mathsf{Show}_{\mathsf{pub}}(\phi_{\boldsymbol{I},\boldsymbol{a}}, \mathsf{nonce})}$
if $\phi_{\boldsymbol{I},\boldsymbol{a}}(\boldsymbol{m}) = 0$ **then abort**
$\boldsymbol{H}_{\mathrm{priv}} \leftarrow (H_i)_{i \in [\ell] \setminus \boldsymbol{I}}$
$Y \leftarrow G + \langle (m_i)_{i \in \boldsymbol{I}}, (H_i)_{i \in \boldsymbol{I}} \rangle$
$\pi_{\mathsf{pub}} \leftarrow \Pi_{\mathsf{pub}}.\mathsf{Prove}^{\mathsf{H}_2}((M^{\mathsf{pub}}_{\tilde{C}, \boldsymbol{H}_{\mathrm{priv}}, \tilde{A}}, (Y, \tilde{B})), (r^{-1}, (m_i)_{i \in [\ell] \setminus I}, r', s, e), (\phi_{\boldsymbol{I},\boldsymbol{a}}, \mathsf{nonce}))$
return $\tau_{\mathsf{pub}} := (\tilde{C}, \pi_{\mathsf{pub}})$

Fig. 8. Scheme $\mathsf{KVAC}_{\mathsf{BBS}} = \mathsf{KVAC}_{\mathsf{BBS}}[\mathsf{GGen}]$. The proof systems $\Pi_{\mathsf{com}}, \Pi_\sigma, \Pi_{\mathsf{pub}}$ are NIZKs for $\mathsf{R}_{\mathsf{com}}, \mathsf{R}_\sigma, \mathsf{R}_{\mathsf{pub}}$ defined in Section 5.1, respectively. States are omitted for readability – subsequent algorithms can use values defined before (e.g. $\mathsf{KVAC}_{\mathsf{BBS}}.\mathsf{U}_2$ can use variables from $\mathsf{KVAC}_{\mathsf{BBS}}.\mathsf{U}_1$). In $\mathsf{Show}_{\mathsf{pub}}$, the value nonce is bound to π_{pub}.

The following theorem, proved in the full version, establishes the security properties of $\mathsf{KVAC}_{\mathsf{BBS}}$. Note that the KVAC unforgeability adversary has access to a restricted DDH oracle rDDH defined in Fig. 2.

Theorem 5.1. *Let* GGen *be a group generator outputting groups of prime order* $p = p(\lambda)$*. Then,* $\mathsf{KVAC}_{\mathsf{BBS}} = \mathsf{KVAC}_{\mathsf{BBS}}[\mathsf{GGen}]$ *satisfies correctness, anonymity and integrity of issued credentials in the ROM with respect to the same simulator* Sim*, and* rDDH*-unforgeability in the ROM under* (q, rDDH)*-SDH assumption and validity of key generation with respect to the same extractor* Ext*.*

oNIP for BBS-based instantiation. Here, we sketch the protocol $\mathsf{oNIP}_{\mathsf{BBS}} = \mathsf{oNIP}[\mathsf{GGen}, \mathsf{R}_{\mathsf{dleq}}]$ for the family of relations $\mathsf{R}_{\mathsf{dleq}}$, defined in Eq. (1), and refer to the full version for the full description. The protocol starts by the user sending a rerandomized statement $(A = \tilde{A} + \beta G, B = \tilde{B} + \beta X)$ to the issuer. The issuer first checks that $(X, (A, B))$ is actually in the language $\mathcal{L}_{\mathsf{R}_{\mathsf{dleq}}}$. Then, the two parties interact in a blinded Σ-protocol to compute an OR-proof that (1) $(X, (A, B)) \in \mathcal{L}_{\mathsf{R}_{\mathsf{dleq}}}$ or (2) the issuer knows the discrete logarithm of

public parameters $W \in \mathbb{G}$. At the end of the protocol, the user obtains a proof π for its statement of choice $(\tilde{A}, \tilde{B})$. This protocol is similar to a recent blind signature scheme [22] and the oNIP for $\mathsf{R}_{\mathsf{dleq}}$ in [37], except that in their cases the issuer computes $B = xA$ for the user who sends A. The following theorem establishes the security properties of $\mathsf{oNIP}_{\mathsf{BBS}}$ with the proof given in the full version.

Theorem 5.2. *Let* GGen *be a group generator outputting groups of prime order* $p = p(\lambda)$*,* rDDH *be a restricted DDH oracle, and* $\mathsf{Sim}_{\mathsf{Gen}}$ *be the simulator for the global parameters generator. Then,* $\mathsf{oNIP}_{\mathsf{BBS}} = \mathsf{oNIP}_{\mathsf{BBS}}[\mathsf{GGen}, \mathsf{R}_{\mathsf{dleq}}]$ *satisfies perfect correctness, soundness in the ROM assuming* DL*, perfect* rDDH*-zero-knowledge, and perfect obliviousness for valid statements with respect to* $\mathsf{Sim}_{\mathsf{Gen}}$*.*

The following corollary follows from Theorems 4.1, 5.1 and 5.2. Although we do not formally show this, strong integrity of $\mathsf{SAAC}_{\mathsf{BBS}}$ follows from the public key of $\mathsf{KVAC}_{\mathsf{BBS}}$ fixing an underlying secret key and soundness of Π_σ ensuring that the issued credential is valid.

Corollary 5.3. *Let* $\mathsf{SAAC}_{\mathsf{BBS}} = \mathsf{SAAC}[\mathsf{GGen}, \mathsf{KVAC}_{\mathsf{BBS}}, \mathsf{oNIP}_{\mathsf{BBS}}]$ *be a SAAC scheme from* $\mathsf{KVAC}_{\mathsf{BBS}}$ *and* $\mathsf{oNIP}_{\mathsf{BBS}}$ *according to Theorem 4.1. Then,* $\mathsf{SAAC}_{\mathsf{BBS}}$ *satisfies correctness, unforgeability in the ROM assuming* $(q,$rDDH$)$-SDH*, and anonymity in the ROM.*

Efficiency. In addition to the concrete sizes in Table 1, we also consider the computational costs of showing (without the helper) and verification of $\mathsf{SAAC}_{\mathsf{BBS}}$, which are $\ell + 4$ (helper protocol includes 19 and 5 exponentiations for the user and issuer, resp.) and $\ell + 12$ exponentiations, resp. This is comparable to those of pairing-based BBS which requires $\ell + 7$ exponentiations for showing and $\ell + 5$ exponentiations + 2 pairing evaluations for verification.

5.2 Instantiation from DDH

In this section, we instantiate our generic construction with a DDH-based KVAC by Chase, Meiklejohn, and Zaverucha's [23] and a corresponding oNIP scheme. Following the syntax in Sect. 4.1, our global parameters generator, denoted $\mathsf{Gen}_{\mathsf{DDH}}(1^\lambda)$, runs $(p, G, \mathbb{G}) \leftarrow\!\!\$\ \mathsf{GGen}(1^\lambda)$, samples $H \leftarrow\!\!\$\ \mathbb{G}^*$, and sets $\mathsf{par}_g = (p, G, \mathbb{G}, H)$. For security of both KVAC and oNIP, we fix the simulator $\mathsf{Sim}_{\mathsf{Gen}}$ which samples $H = vG$ with a trapdoor $v \leftarrow\!\!\$\ \mathbb{Z}_p^*$.

DDH-based KVAC. We first introduce the DDH-based KVAC in [23], building on top of an algebraic MAC where a tag for a vector of attributes $(m_i)_{i=1}^\ell$ is $(S_w, S_x, S_y, S_z) := (U \leftarrow\!\!\$\ \mathbb{G}, (x_0 + \sum_{i=1}^\ell x_i m_i)U, (y_0 + \sum_{i=1}^\ell y_i m_i)U, zU)$ with the secret key containing scalars $(x_i)_{i=0}^\ell$, $(y_i)_{i=0}^\ell$, and z. The issuer's public key includes $(X_i = x_i H, Y_i = y_i H)_{i=1}^\ell$ with H being the public parameters. For blind issuance, a user ElGamal encrypts each of their attributes, and the issuer homomorphically creates a tag for the user to decrypt.

To show a credential: the user randomizes the tag as $(S'_w = rS_w, C_x = rS_x + r_xH, C_y = rS_y + r_yH, S'_z = rS_z)$ for $r \leftarrow\$ \mathbb{Z}_p^*, r_x, r_y \leftarrow\$ \mathbb{Z}_p$. Then, the user computes commitments $C_i = m_iU' + r_iG$ to their attributes. With U' and $(C_i)_{i=1}^{\ell}$, the issuer can use their secret key to compute (for example) $\tilde{V}_x = x_0U' + \sum_{i=1}^{\ell} x_iC_i = (x_0 + \sum_{i=1}^{\ell} x_im_i)U' + \sum_{i=1}^{\ell} r_iX_i$ which is close to C_x, but with added randomness from the blinding. Hence, the user also sends $\Gamma_x := \sum_{i=1}^{\ell} r_iX_i - r_xH$ (and similarly Γ_y). The issuer checks that $C_x + \Gamma_x = \tilde{V}_x$ (respectively for y_i and $C_y, \Gamma_y, \tilde{V}_y$). This is the key-dependent part of the verification. The user also includes a publicly verifiable proof of knowledge of representations of $(C_i)_{i=1}^{\ell}, \Gamma_x, \Gamma_y$.

Our $\mathsf{KVAC}_{\mathsf{DDH}}$, described in Fig. 9, made these changes to their scheme:

1. **Public key:** In [23], Pedersen commitments of x_0, y_0, z are included in the public key, allowing the issuer to prove correct credential issuance. In this case, the underlying secret key is not uniquely determined (binding is computational), which is insufficient for our SAAC compiler. We instead include ElGamal ciphertexts of x_0, y_0 (security is not affected) and publish $Z = zH$ in the clear. For the latter, we noticed that revealing Z does not affect the underlying MAC's security, saving us one group element.[6]
2. **Blind Issuance:** In [23], users individually encrypt each m_i, and let the issuer computes and sends ciphertexts of S_x, S_y. We observe that pk contains $X_i = x_iH$, $Y_i = y_iH$ for $i \in [\ell]$, so the user can compute ciphertexts of $\sum_{i=1}^{\ell} m_iX_i$ and $\sum_{i=1}^{\ell} m_iY_i$, while the issuer can still compute ciphertexts of S_x, S_y. Now, the issuer's communication is independent of ℓ as it only has to compute a proof with respect to a smaller witness.

Our KVAC makes use of proof systems $\Pi_{\mathsf{com}}, \Pi_{\sigma}$, and Π_{pub} for the relations $\mathsf{R}_{\mathsf{com}}, \mathsf{R}_{\sigma}, \mathsf{R}_{\mathsf{pub}}$, respectively defined below, respectively.

$$\mathsf{R}_{\mathsf{com}} := \left\{ \begin{array}{c} ((\widetilde{E}_x, \widetilde{E}_y, D, (X_i)_{i=1}^{\ell}, (Y_i)_{i=1}^{\ell}, \psi), \\ (u_x, u_y, \boldsymbol{m} = (m_i)_{i=1}^{\ell})) \end{array} : \begin{array}{c} \widetilde{E}_x = (u_xG, u_xD + \sum_{i=1}^{\ell} m_iX_i) \\ \widetilde{E}_y = (u_yG, u_yD + \sum_{i=1}^{\ell} m_iY_i) \\ \psi(\boldsymbol{m}) = 1 \end{array} \right\}$$

$$\mathsf{R}_{\sigma} := \left\{ \begin{array}{c} ((E_x, E_y, D, S_w, S_z, \\ \widetilde{E}_x, \widetilde{E}_y, Z, \mathsf{ct}_x, \mathsf{ct}_y), \\ (z, x_0, y_0, r', t_x, t_y, \gamma_x, \gamma_y)) \end{array} : \begin{array}{c} Z = zH, r'S_w = G, S_z = zS_w \\ \widetilde{E}_x = r'E_x - (\gamma_0 G, \gamma_0 D + x_0 H) \\ \widetilde{E}_y = r'E_y - (\gamma_0 G, \gamma_0 D + y_0 H) \\ \mathsf{ct}_x = (t_xG, t_xH + x_0G) \\ \mathsf{ct}_y = (t_yG, t_yH + y_0G) \end{array} \right\}$$

$$\mathsf{R}_{\mathsf{pub}} := \left\{ \begin{array}{c} (((m_i)_{i\in I}, (X_i)_{i=1}^{\ell}, (Y_i)_{i=1}^{\ell}, \\ S_w, (C_i)_{i=1}^{\ell}, \Gamma_x, \Gamma_y), \\ ((m_i)_{i\in[\ell]\setminus I}, (r_i)_{i=1}^{\ell}, r_x, r_y)) \end{array} : \begin{array}{c} \forall i \in [\ell] : C_i = m_iS_w + r_iH \\ \Gamma_x = (\sum_{i=1}^{\ell} r_iX_i) - r_xH \\ \Gamma_y = (\sum_{i=1}^{\ell} r_iY_i) - r_yH \end{array} \right\}.$$

We note that Π_{com} is straightline-extractable for a relaxed relation $\widetilde{\mathsf{R}}_{\mathsf{com}} \supseteq \mathsf{R}_{\mathsf{com}}$ which also accepts witness $(u_x = 0, u_y = 0, \boldsymbol{m} \neq \boldsymbol{0})$ where $0_{\mathbb{G}} = \sum_{i=1}^{\ell} m_iX_i =$

[6] Intuitively, this is because (U, zU) is included in every tag anyways.

$\mathsf{KVAC}_{\mathsf{DDH}}.\mathsf{Setup}(1^\ell, \mathsf{par}_g = (p, G, \mathbb{G}, H))$

Select $\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2 : \{0,1\}^* \to \mathbb{Z}_p$

$\Pi_\sigma \leftarrow \mathsf{Lin}[\mathsf{H}_1, \mathbb{G}]$; $\Pi_{\mathsf{pub}} \leftarrow \mathsf{Lin}[\mathsf{H}_2, \mathbb{G}]$

return $\mathsf{par} = (p, G, \mathbb{G}, H, \mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2)$

$\mathsf{KVAC}_{\mathsf{DDH}}.\mathsf{KeyGen}(\mathsf{par})$

$\boldsymbol{x}, \boldsymbol{y} \leftarrow\$\ \mathbb{Z}_p^{\ell+1}; z, t_x, t_y \leftarrow\$\ \mathbb{Z}_p; \mathsf{sk} \leftarrow (\boldsymbol{x}, \boldsymbol{y}, z, t_x, t_y)$

$\mathsf{ct}_x \leftarrow (t_x G, t_x H + x_0 G); \mathsf{ct}_y \leftarrow (t_y G, t_y H + y_0 G)$

$\mathsf{pk} \leftarrow (\boldsymbol{X} := (X_i)_{i=1}^\ell, \boldsymbol{Y} := (Y_i)_{i=1}^\ell, Z, \mathsf{ct}_x, \mathsf{ct}_y)$

return $(\mathsf{sk}, \mathsf{pk})$

$\mathsf{KVAC}_{\mathsf{DDH}}.\mathsf{Iss}(\mathsf{par}, x, \psi, \mu = (\widetilde{E}_x, \widetilde{E}_y, D, \pi_{\mathsf{com}}))$

if $\Pi_{\mathsf{com}}.\mathsf{Ver}^{\mathsf{H}_0}((\widetilde{E}_x, \widetilde{E}_y, D, \boldsymbol{X}, \boldsymbol{Y}, \psi), \pi_{\mathsf{com}}) = 0$
  then abort

$r \leftarrow\$\ \mathbb{Z}_p^*; \gamma_x, \gamma_y \leftarrow\$\ \mathbb{Z}_p; S_w \leftarrow rH, S_z \leftarrow rZ$

$E_x \leftarrow r((\gamma_x G, \gamma_x D + x_0 H) + \widetilde{E}_x)$

$E_y \leftarrow r((\gamma_y G, \gamma_y D + y_0 H) + \widetilde{E}_y)$

$\pi_\sigma \leftarrow \Pi_\sigma.\mathsf{Prove}^{\mathsf{H}_1}((M^\sigma_{G,H,S_w,D,E_x,E_y},$
  $(\widetilde{E}_x, \widetilde{E}_y, Z, \mathsf{ct}_x, \mathsf{ct}_y)), (z, x_0, y_0, r^{-1}, t_x, t_y, \gamma_x, \gamma_y))$

return $(S_w, E_x, E_y, S_z, \pi_\sigma)$

$\mathsf{KVAC}_{\mathsf{DDH}}.\mathsf{SVer}_{\mathsf{key}}(\mathsf{par}, \mathsf{sk}, \tau_{\mathsf{key}})$

$(S'_w, S'_z, (C_i)_{i=1}^\ell, C_x, C_y, \Gamma_x, \Gamma_y) \leftarrow \tau_{\mathsf{key}}$

return $S'_w \neq 0_{\mathbb{G}} \wedge S'_z = zS'_w$
  $\wedge\ \Gamma_x + C_x = (x_0 S'_w + \sum_{i=1}^\ell x_i C_i)$
  $\wedge\ \Gamma_y + C_y = (y_0 S'_w + \sum_{i=1}^\ell y_i C_i)$

$\mathsf{KVAC}_{\mathsf{DDH}}.\mathsf{SVer}_{\mathsf{pub}}(\mathsf{par}, \mathsf{pk}, \tau_{\mathsf{key}}, \pi_{\mathsf{pub}}, \phi_{\boldsymbol{I},\boldsymbol{a}}, \mathsf{nonce})$

return $\Pi_{\mathsf{pub}}.\mathsf{Ver}^{\mathsf{H}_2}((M^{\mathsf{pub}}_{G,H,S'_w,\boldsymbol{X},\boldsymbol{Y}}, ((C_i)_{i\in[\ell]\setminus\boldsymbol{I}},$
  $(C_i - a_i S'_w)_{i\in\boldsymbol{I}}, \Gamma_x, \Gamma_y)), \pi_{\mathsf{pub}}, (\phi_{\boldsymbol{I},\boldsymbol{a}}, \mathsf{nonce}))$

Oracle $\mathcal{O}_{\mathsf{SVerDDH}}(\mathsf{par}, \mathsf{sk}, S_w, S_z, (C_i)_{i=1}^\ell, \zeta_x, \zeta_y)$

return $S_z = zS_w \wedge \zeta_x = x_0 S_w + \sum_{i=1}^\ell x_i C_i \wedge$
  $\zeta_y = y_0 S_w + \sum_{i=1}^\ell y_i C_i \wedge S_w \neq 0_{\mathbb{G}}$

$\mathsf{KVAC}_{\mathsf{DDH}}.\mathsf{U}_1(\mathsf{par}, \mathsf{pk}, \boldsymbol{m} \in \mathbb{Z}_p^\ell, \psi)$

$d, u_x, u_y \leftarrow\$\ \mathbb{Z}_p; D \leftarrow dG$

$\widetilde{E}_x \leftarrow (u_x G, u_x D + \sum_{i=1}^\ell m_i X_i)$

$\widetilde{E}_y \leftarrow (u_y G, u_y D + \sum_{i=1}^\ell m_i Y_i)$

$\pi_{\mathsf{com}} \leftarrow \Pi_{\mathsf{com}}.\mathsf{Prove}^{\mathsf{H}_0}((\widetilde{E}_x, \widetilde{E}_y, D, \boldsymbol{X}, \boldsymbol{Y}, \psi),$
  $(u_x, u_y, \boldsymbol{m}))$

return $\mu := (\widetilde{E}_x, \widetilde{E}_y, D, \pi_{\mathsf{com}})$

$\mathsf{KVAC}_{\mathsf{DDH}}.\mathsf{U}_2(\mathsf{imsg} = (S_w, E_x, E_y, S_z, \pi_\sigma))$

if $\Pi_\sigma.\mathsf{Ver}^{\mathsf{H}_1}((M^\sigma_{G,H,S_w,D,E_x,E_y},$
  $(\widetilde{E}_x, \widetilde{E}_y, Z, \mathsf{ct}_x, \mathsf{ct}_y), \pi_\sigma) = 0$
  then abort

$(E_{x,0}, E_{x,1}) \leftarrow E_x; (E_{y,0}, E_{y,1}) \leftarrow E_y$

$S_x \leftarrow E_{x,1} - dE_{x,0}; S_y \leftarrow E_{y,1} - dE_{y,0}$

return $\sigma \leftarrow (S_w, S_x, S_y, S_z)$

$\mathsf{KVAC}_{\mathsf{DDH}}.\mathsf{Show}_{\mathsf{key}}(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}, \sigma)$

$r', r_x, r_y \leftarrow\$\ \mathbb{Z}_p; \boldsymbol{r} := (r_i)_{i=1}^\ell \leftarrow\$\ \mathbb{Z}_p^\ell$

$(S'_w, S'_x, S'_y, S'_z) \leftarrow r'\sigma$

for $i \in [\ell] : C_i \leftarrow m_i S'_w + r_i H$

$C_x \leftarrow S'_x + r_x H; C_y \leftarrow S'_y + r_y H$

$\Gamma_x \leftarrow \sum_{i=1}^\ell r_i X_i - r_x H$

$\Gamma_y \leftarrow \sum_{i=1}^\ell r_i Y_i - r_y H$

return $(S'_w, S'_z, (C_i)_{i=1}^\ell, C_x, C_y, \Gamma_x, \Gamma_y)$

$\mathsf{KVAC}_{\mathsf{DDH}}.\mathsf{Show}_{\mathsf{pub}}(\phi_{\boldsymbol{I},\boldsymbol{a}}, \mathsf{nonce})$

for $i \in \boldsymbol{I} : C'_i \leftarrow C_i - a_i S'_w$

$\pi_{\mathsf{pub}} \leftarrow \Pi_{\mathsf{pub}}.\mathsf{Prove}^{\mathsf{H}_2}((M^{\mathsf{pub}}_{G,H,S'_w,\boldsymbol{X},\boldsymbol{Y}},$
  $((C_i)_{i\in[\ell]\setminus\boldsymbol{I}}, (C'_i)_{i\in\boldsymbol{I}}, \Gamma_x, \Gamma_y)),$
  $((m_i)_{i\in[\ell]\setminus\boldsymbol{I}}, \boldsymbol{r}, r_x, r_y), (\phi_{\boldsymbol{I},\boldsymbol{a}}, \mathsf{nonce}))$

return π_{pub}

Fig. 9. Scheme $\mathsf{KVAC}_{\mathsf{DDH}} = \mathsf{KVAC}_{\mathsf{DDH}}[\mathsf{Gen}_{\mathsf{DDH}}]$ and oracle $\mathcal{O}_{\mathsf{SVerDDH}}$. $\Pi_{\mathsf{com}}, \Pi_\sigma, \Pi_{\mathsf{pub}}$ are NIZKs for $\mathsf{R}_{\mathsf{com}}, \mathsf{R}_\sigma, \mathsf{R}_{\mathsf{pub}}$ defined in Section 5.2, respectively. States are omitted for readability – subsequent algorithms can use values defined before (e.g. $\mathsf{KVAC}_{\mathsf{BBS}}.\mathsf{U}_2$ can use variables from $\mathsf{KVAC}_{\mathsf{BBS}}.\mathsf{U}_1$). In $\mathsf{Show}_{\mathsf{pub}}$, the value nonce is bound to π_{pub}.

$\sum_{i=1}^\ell m_i Y_i$. For R_σ and $\mathsf{R}_{\mathsf{pub}}$, let $M^\sigma_{G,H,S_w,D,E_x,E_y}$ and $M^{\mathsf{pub}}_{G,H,S_w,\boldsymbol{X},\boldsymbol{Y}}$ be matrices defined by the respective relations described above (omitting the explicit representation for brevity), analogously to what was done in Sect. 5.1.

The algorithm $\mathsf{SVer}_{\mathsf{key}}$ induces the relation family $\mathsf{R}_{\mathsf{DDH}}$, parameterized by $\mathsf{par}_g = (p, G, \mathbb{G}, H)$ (which we omit in the subscript), for which we give a corresponding oNIP protocol. $\mathsf{R}_{\mathsf{DDH}}$ contains statements $(\mathsf{pk} = (\boldsymbol{X}, \boldsymbol{Y}, Z, \mathsf{ct}_x, \mathsf{ct}_y),$

$\tau_{\mathsf{key}} = (S_w, (C_i)_{i\in[\ell]}, \zeta_x, \zeta_y, S_z))$[7] and witnesses $\mathsf{sk} = (\boldsymbol{x}, \boldsymbol{y}, z, t_x, t_y)$, such that

$$\begin{array}{c} Z = zH, S_w \neq 0_{\mathbb{G}}, S_z = zS_w, \forall i \in [\ell] : X_i = x_i H, Y_i = y_i H \\ \zeta_x = x_0 S_w + \sum_{i=1}^{\ell} x_i C_i, \zeta_y = y_0 S_w + \sum_{i=1}^{\ell} y_i C_i \\ \mathsf{ct}_x = (t_x G, t_x H + x_0 G), \mathsf{ct}_y = (t_y G, t_y H + y_0 G) \end{array} \tag{2}$$

The following theorem, proved in the full version, establishes the security of $\mathsf{KVAC}_{\mathsf{DDH}}$. In the proof, we first show unforgeability of the underlying MAC against adversaries with access to $\mathcal{O}_{\mathsf{SVerDDH}}$ (defined in Fig. 9), using techniques similar to [23]. Then, we give a reduction from unforgeability of $\mathsf{KVAC}_{\mathsf{DDH}}$ to that of the MAC. Our main contribution is twofold: (1) A careful rewinding argument to extract a MAC forgery from the KVAC forgery; and (2) We show how to simulate showings for an honest user by querying for a tag on a random (and hidden) set of attributes, and that we still reliably extract a fresh forgery.

Theorem 5.4. *Let* $\mathsf{Gen}_{\mathsf{DDH}}$ *be a global parameters generator defined in Sect. 5.2. Then,* $\mathsf{KVAC}_{\mathsf{DDH}} = \mathsf{KVAC}_{\mathsf{DDH}}[\mathsf{Gen}_{\mathsf{DDH}}]$ *satisfies correctness, anonymity assuming DDH and integrity of issued credentials both in the ROM and with respect to the same simulators* $\mathsf{Sim}_{\mathsf{Gen}}$ *and* $\mathsf{Sim}_{\mathsf{KVAC}}$*, and* $\mathcal{O}_{\mathsf{SVerDDH}}$*-unforgeability in the ROM assuming DDH and validity of key generation with respect to the same extractor* Ext*.*

oNIP for DDH-based instantiation. We sketch the protocol $\mathsf{oNIP}_{\mathsf{DDH}} = \mathsf{oNIP}[\mathsf{Gen}_{\mathsf{DDH}}, \mathsf{R}_{\mathsf{DDH}}]$ for the family of relations $\mathsf{R}_{\mathsf{DDH}}$ described in Eq. (2), containing statement pk, an augmented statement τ_{key} and witness sk.

Our $\mathsf{oNIP}_{\mathsf{DDH}}$ construction follows a similar structure to $\mathsf{oNIP}_{\mathsf{BBS}}$ relying on a blinded OR-proof of either (1) membership of the induced language $\mathcal{L}_{\mathsf{R}_{\mathsf{DDH}}}$ or (2) knowledge of discrete logarithm of public parameters W. The key difference lies in the first move, where the user rerandomizes the augmented statement $(S'_w, (C'_i)_{i=1}^{\ell}, \zeta'_x, \zeta'_y, S'_z)$ by computing $S_w = \alpha S'_w, C_i = \alpha C'_i + \beta_i H$ with random scalars $\alpha, \beta_1, \ldots, \beta_\ell$ and uses $\boldsymbol{X}, \boldsymbol{Y}$ in the public key to compute $\zeta_x = \alpha\zeta'_x + \sum_{i=1}^{\ell} \beta_i X_i, \zeta_y = \alpha\zeta'_y + \sum_{i=1}^{\ell} \beta_i Y_i, S_z = \alpha S'_z$, which still preserves the membership of the language. The issuer then checks whether the rerandomized statement is in the language. We refer to the full version for the full protocol description and the proof of the following theorem, establishing the security properties of $\mathsf{oNIP}_{\mathsf{DDH}}$. The proof follows from standard techniques as with $\mathsf{oNIP}_{\mathsf{BBS}}$, except that for obliviousness, we *inherently requires* the global trapdoor v to efficiently simulate honest users without knowing the augmented statement τ_{key}.

Theorem 5.5. *Let* $\mathsf{Gen}_{\mathsf{DDH}}$ *be a global parameters generator defined in Sect. 5.2 and* $\mathcal{O}_{\mathsf{SVerDDH}}$ *be the oracle in Fig. 9. Then,* $\mathsf{oNIP}_{\mathsf{DDH}} = \mathsf{oNIP}[\mathsf{Gen}_{\mathsf{DDH}}, \mathsf{R}_{\mathsf{DDH}}]$ *satisfies perfect correctness, soundness in the ROM assuming* DL*, perfect* $\mathcal{O}_{\mathsf{SVerDDH}}$*-zero-knowledge, and perfect obliviousness for valid statements with respect to the simulator* $\mathsf{Sim}_{\mathsf{Gen}}$*.*

[7] Note that ζ_x and ζ_y represent $C_x + \Gamma_x$ and $C_y + \Gamma_y$ and can be computed from the output τ_{key} of $\mathsf{Show}_{\mathsf{key}}$.

Finally, the following corollary follows from Theorems 4.1, 5.4 and 5.5. Similar to $\mathsf{SAAC}_{\mathsf{BBS}}$, strong integrity of $\mathsf{SAAC}_{\mathsf{DDH}}$ follows from the structure of $\mathsf{KVAC}_{\mathsf{DDH}}$'s public key and soundness of Π_σ.

Corollary 5.6. *Let* $\mathsf{SAAC}_{\mathsf{DDH}} = \mathsf{SAAC}[\mathsf{Gen}_{\mathsf{DDH}}, \mathsf{KVAC}_{\mathsf{DDH}}, \mathsf{oNIP}_{\mathsf{DDH}}]$ *be a SAAC scheme from* $\mathsf{KVAC}_{\mathsf{DDH}}$ *and* $\mathsf{oNIP}_{\mathsf{DDH}}$ *according to Theorem 4.1. Then,* $\mathsf{SAAC}_{\mathsf{DDH}}$ *satisfies correctness, unforgeability, and anonymity (both in the ROM and assuming* DDH*).*

Efficiency. The computational costs of showing (without the helper) and verification of $\mathsf{SAAC}_{\mathrm{DDH}}$ are $4\ell + 2$ (helper protocol includes $18\ell + 47$ and $6\ell + 15$ exponentiations for the user and issuer, resp.) and $11\ell + 22$ exponentiations, resp.

6 Conclusion

This paper introduced the SAAC model and gave two efficient instantiations. We emphasize that despite the requirement of the helper interaction, SAAC is not as restrictive as it may seem to be. This is because (1) the helper information can be requested ahead of time and can be spent later *without* any additional online interaction, and (2) the helper protocol is *independent* of the showing predicate.

We envision that each user would obtain an upper bound B pieces of helper information at regular time increments (e.g., the number of times one uses a digital ID per week, which need not be large). Since the showing predicate and disclosed attributes can be decided later on, and the helper information is very small in size, the space requirements for this are not significant.

In a real-world setting, timing or counting attacks may compromise anonymity if our system is used carelessly. For example, if users always request helper information immediately before showing a credential, then linking helper interactions to showings becomes possible. Or, in a setting where the helper server can identify users, if User-A interacts with the helper 99 times, and User-B interacts only once, then a verifier who sees 2 different showings can be sure that they interacted with user A in one of the interactions. Implementing the system to hide usage patterns (e.g., as discussed earlier) should prevent these attacks.

Our BBS-based instantiation improves considerably upon the state of the art for pairing-free ACs: it is multi-show, the helper interaction is lightweight, and it is provably secure in the ROM. This is in contrast to, e.g., ACL [4], which requires re-proving a (potentially expensive) issuance predicate for each showing, and is only proved secure in the AGM via an involved security proof [29].

Acknowledgements. We thank CRYPTO2025 anonymous reviewers for their feedback. Anna Lysyanskaya was supported by NSF Grants 2312241, 2154170, and 2247305 as well as the Ethereum Foundation. Chairattana-Apirom and Tessaro's research was partially supported by NSF grants CNS-2026774, CNS-2154174, CNS-2426905, a gift from Microsoft, and a Stellar Development Foundation Academic Research Award.

References

1. Architecture proposal for the german eIDAS implementation (2024). https://gitlab.opencode.de/bmi/eudi-wallet/eidas-2.0-architekturkonzept/-/blob/main/architecture-proposal/01-architecture-proposal.md. Accessed 13 Feb 2025
2. The European digital identity wallet architecture and reference framework (2024). https://eu-digital-identity-wallet.github.io/eudi-doc-architecture-and-reference-framework/1.4.0/arf/. Accessed 13 Feb 2025
3. Au, M.H., Susilo, W., Mu, Y.: Constant-size dynamic k-TAA. In: De Prisco, R., Yung, M. (eds.) SCN 2006. LNCS, vol. 4116, pp. 111–125. Springer, Heidelberg (2006). https://doi.org/10.1007/11832072_8
4. Baldimtsi, F., Lysyanskaya, A.: Anonymous credentials light. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013, pp. 1087–1098. ACM Press (2013). https://doi.org/10.1145/2508859.2516687
5. Barker, E., Chen, L., Roginsky, A., Vassilev, A., Davis, R.: Recommendation for pair-wise key-establishment schemes using discrete logarithm cryptography. Technical report NIST Special Publication 800-56A, National Institute of Standards (NIST). https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.pdf. Accessed 13 Feb 2025
6. Barki, A., Brunet, S., Desmoulins, N., Traoré, J.: Improved algebraic MACs and practical keyed-verification anonymous credentials. In: Avanzi, R., Heys, H. (eds.) SAC 2016. LNCS, vol. 10532, pp. 360–380. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69453-5_20
7. Baum, C., et al.: Cryptographers' feedback on the EU digital identity's ARF (2024). https://github.com/user-attachments/files/15904122/cryptographers-feedback.pdf
8. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) ACM CCS 1993, pp. 62–73. ACM Press (1993). https://doi.org/10.1145/168588.168596
9. Benhamouda, F., Lepoint, T., Loss, J., Orrù, M., Raykova, M.: On the (in)security of ROS. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021, Part I. LNCS, vol. 12696, pp. 33–53. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77870-5_2
10. Bernstein, D.J., Lange, T.: Safecurves: choosing safe curves for elliptic-curve cryptography. https://safecurves.cr.yp.to/. Accessed 13 Feb 2025
11. Boneh, D., Boyen, X.: Short signatures without random oracles and the SDH assumption in bilinear groups. J. Cryptol. **21**(2), 149–177 (2007). https://doi.org/10.1007/s00145-007-9005-7
12. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 41–55. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28628-8_3
13. Brands, S.: Rethinking public key infrastructure and digital certificates— building in privacy. Ph.D. thesis, Eindhoven Inst. of Tech. The Netherlands (1999)
14. Camenisch, J., Drijvers, M., Dzurenda, P., Hajny, J.: Fast keyed-verification anonymous credentials on standard smart cards. In: Dhillon, G., Karlsson, F., Hedström, K., Zúquete, A. (eds.) SEC 2019. IAICT, vol. 562, pp. 286–298. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-22312-0_20
15. Camenisch, J., Drijvers, M., Lehmann, A.: Anonymous attestation using the strong diffie hellman assumption revisited. In: Franz, M., Papadimitratos, P. (eds.) Trust

2016. LNCS, vol. 9824, pp. 1–20. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45572-3_1
16. Camenisch, J., Krenn, S., Lehmann, A., Mikkelsen, G.L., Neven, G., Pedersen, M.Ø.: Formal treatment of privacy-enhancing credential systems. In: Dunkelman, O., Keliher, L. (eds.) SAC 2015. LNCS, vol. 9566, pp. 3–24. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31301-6_1
17. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 93–118. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44987-6_7
18. Camenisch, J., Lysyanskaya, A.: A signature scheme with efficient protocols. In: Cimato, S., Persiano, G., Galdi, C. (eds.) SCN 2002. LNCS, vol. 2576, pp. 268–289. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36413-7_20
19. Camenisch, J., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 56–72. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28628-8_4
20. Camenisch, J., Van Herreweghen, E.: Design and implementation of the idemix anonymous credential system. In: Atluri, V. (ed.) ACM CCS 2002, pp. 21–30. ACM Press (2002). https://doi.org/10.1145/586110.586114
21. Chairattana-Apirom, R., Harding, F., Lysyanskaya, A., Tessaro, S.: Server-aided anonymous credentials. Cryptology ePrint Archive, Paper 2025/513 (2025). https://eprint.iacr.org/2025/513
22. Chairattana-Apirom, R., Tessaro, S., Zhu, C.: Pairing-free blind signatures from CDH assumptions. In: Reyzin, L., Stebila, D. (eds.) CRYPTO 2024, Part I. LNCS, vol. 14920, pp. 174–209. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-68376-3_6
23. Chase, M., Meiklejohn, S., Zaverucha, G.: Algebraic MACs and keyed-verification anonymous credentials. In: Ahn, G.J., Yung, M., Li, N. (eds.) ACM CCS 2014, pp. 1205–1216. ACM Press (2014). https://doi.org/10.1145/2660267.2660328
24. Chase, M., Perrin, T., Zaverucha, G.: The Signal private group system and anonymous credentials supporting efficient verifiable encryption. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020, pp. 1445–1459. ACM Press (2020). https://doi.org/10.1145/3372297.3417887
25. Chaum, D.: Blind signatures for untraceable payments. In: Chaum, D., Rivest, R.L., Sherman, A.T. (eds.) Advances in Cryptology, pp. 199–203. Springer, Boston (1983). https://doi.org/10.1007/978-1-4757-0602-4_18
26. Döttling, N., Hartmann, D., Hofheinz, D., Kiltz, E., Schäge, S., Ursu, B.: On the impossibility of purely algebraic signatures. In: Nissim, K., Waters, B. (eds.) TCC 2021. LNCS, vol. 13044, pp. 317–349. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90456-2_11
27. Fischlin, M.: Communication-efficient non-interactive proofs of knowledge with online extractors. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 152–168. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_10
28. Hardt, D.: The OAuth 2.0 Authorization Framework. RFC 6749 (2012). https://doi.org/10.17487/RFC6749. https://www.rfc-editor.org/info/rfc6749
29. Kastner, J., Loss, J., Renawi, O.: Concurrent security of anonymous credentials light, revisited. In: Meng, W., Jensen, C.D., Cremers, C., Kirda, E. (eds.) ACM CCS 2023, pp. 45–59. ACM Press (2023). https://doi.org/10.1145/3576915.3623184

30. Kondi, Y., Shelat, A.: Improved straight-line extraction in the random oracle model with applications to signature aggregation. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022, Part II. LNCS, vol. 13792, pp. 279–309. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-22966-4_10
31. Looker, T., Kalos, V., Whitehead, A., Lodder, M.: The BBS Signature Scheme. Internet-Draft draft-IRTF-CFRG-BBS-signatures-07, Internet Engineering Task Force (2024). https://datatracker.ietf.org/doc/draft-irtf-cfrg-bbs-signatures/07/. Work in Progress
32. Lysyanskaya, A.: Signature schemes and applications to cryptographic protocol design. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts (2002)
33. Lysyanskaya, A., Rivest, R., Sahai, A., Wolf, S.: Pseudonym systems. In: Heys, H., Adams, C. (eds.) Selected Areas in Cryptography. LNCS, vol. 1758 (1999)
34. Maurer, U.: Zero-knowledge proofs of knowledge for group homomorphisms. DCC **77**(2-3), 663–676 (2015). https://doi.org/10.1007/s10623-015-0103-5
35. Mirzamohammadi, O., et al.: Keyed-verification anonymous credentials with highly efficient partial disclosure. Cryptology ePrint Archive, Paper 2025/041 (2025). https://eprint.iacr.org/2025/041
36. Orange Innovation: The BBS# protocol: technical details. https://github.com/eu-digital-identity-wallet/eudi-doc-architecture-and-reference-framework/issues/193#issuecomment-2179355934. Accessed 13 Feb 2025
37. Orrù, M., Tessaro, S., Zaverucha, G., Zhu, C.: Oblivious issuance of proofs. In: Reyzin, L., Stebila, D. (eds.) CRYPTO 2024, Part IX. LNCS, vol. 14928, pp. 254–287. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-68400-5_8
38. Orrù, M.: Revisiting keyed-verification anonymous credentials. Cryptology ePrint Archive, Paper 2024/1552 (2024). https://eprint.iacr.org/2024/1552
39. Paquin, C., Zaverucha, G.: U-prove cryptographic specification v1.1 (revision 3) (2013). https://www.microsoft.com/en-us/research/publication/u-prove-cryptographic-specification-v1-1-revision-3/. Released under the Open Specification Promise. http://www.microsoft.com/openspecifications/en/us/programs/osp/default.aspx
40. Sakemi, Y., Kobayashi, T., Saito, T., Wahby, R.S.: Pairing-Friendly Curves. Internet-Draft draft-IRTF-CFRG-pairing-friendly-curves-11, Internet Engineering Task Force (2022). https://datatracker.ietf.org/doc/draft-irtf-cfrg-pairing-friendly-curves/11/. Work in Progress
41. Tessaro, S., Zhu, C.: Revisiting BBS signatures. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part V. LNCS, vol. 14008, pp. 691–721. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-30589-4_24
42. Traoré, J., Dumanois, A.: BBS# and eIDAS 2.0.: making BBS anonymous credentials eIDAS 2.0 compliant. https://csrc.nist.gov/csrc/media/presentations/2024/wpec2024-3b3/images-media/wpec2024-3b3-slides-antoine-jacques--BBS-sharp-eIDAS2.pdf. Accessed 13 Feb 2025