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, US
{rchairat,tessaro}@cs.washington.edu
[2] Brown University
Providence, RI, US
{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 [Cha82], 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 [CL01, CL03, CL04]. In the standard approach to designing ACs [LRSW99, Lys02], a credential is a *signature* on 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-knolwedge 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 [CL03, CL04], as well as pairing-based BBS signatures [CL04, BBS04, ASM06, TZ23b].

Systems using ACs have been proposed over the years, such as Microsoft's U-Prove [Bra99, PZ13] and IBM's IDEMIX [CV02]. 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 [LKWL24] and the World Wide Web Consortium (W3C). Technology policy, especially that of the EU and its member states, has mandated privacy-preserving authentication [ARF24, Ger24] for which anonymous credentials appear to be the right solution [BBC+24].

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 [BL, BCR$^+$]. The lack of suitable standards, 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 [DHH$^+$21]).

To overcome this inherent barrier, prior works have considered different settings where pairing-free ACs are possible:

- *Blind signatures with attributes.* Baldimtsi and Lysyanskaya [BL13] presented an approach extending the notion of blind signatures to include attributes, formalizing ideas implicit in U-Prove [PZ13]. 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 [CMZ14] considered multi-use credentials in an alternative setting where 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 [CPZ20].

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. [OTZZ24]): 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 [Har12], 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 [TD, Ora]. BBS# is an industry white paper that explores several ideas for the development of a European Digital Identity Wallet.[3] 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. [OTZZ24], as our helper protocol is not required to resist strong attacks such as ROS [BLL$^+$21], and thus we can prove security based on a standard cryptographic assumption without relying on the algebraic group model (AGM) [KLR23].

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

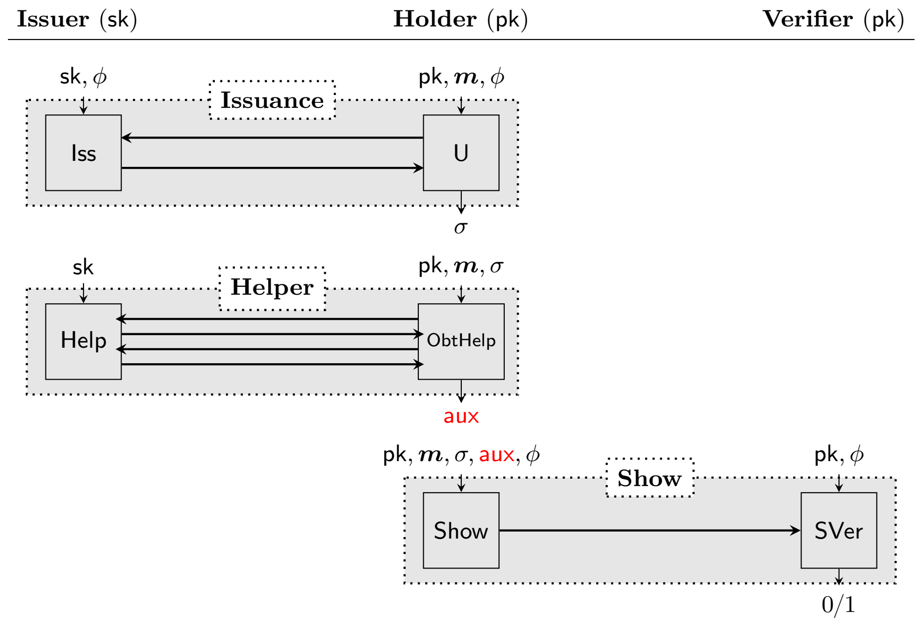

Fig. 1. Server-Aided Anonymous Credentials. Illustration of the SAAC setting. Note that the secret and public keys $(\mathsf{sk}, \mathsf{pk})$ are generated by the $\mathsf{SAAC.KeyGen}$ algorithm, which is not described here. Also, we allow each showing to be linked to some additional value nonce,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

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 [BR93], but does not make any use of the AGM or any other ideal group model.

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 Figure 1.

- **Key generation.** The issuer generates a secret-key/public-key pair $(\mathsf{sk}, \mathsf{pk})$ 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 a 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 and that $\phi(\boldsymbol{m}) = 1$. 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 predicate $\phi \in \Phi$, 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 ϕ) to assess that indeed $\phi(\boldsymbol{m}) = 1$. 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 ϕ, 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 [CMZ14, BBDT16, CDDH19].

Our generic construction replaces the keyed verification of a KVAC scheme with a non-interactive proof that the showing message satisfies the keyed-verification algorithm. The helper protocol will be an oblivious issuance of proof (oNIP) [OTZZ24] 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 $\mathsf{R_V}$ containing statement $(\mathsf{pk}, \tau_{\mathsf{key}})$ and witness sk such that (1) the secret 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 Section 4.1 for the deviation from the prior oNIP formalization in [OTZZ24]), we arrive at the following SAAC construction:

- **Key generation and issuance** are exactly those of the KVAC scheme.
- **Helper protocol.** The helper protocol begins with 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, there are still other requirements for the security of our generic SAAC construction.

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 challenging point in giving a secure instantiation from 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; on the other hand, 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. Here are some 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 additionally *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 (without the helper protocol), 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 Section 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. [BBDT16], which builds upon an algebraic message authentication code (MAC) based on BBS/BBS+ signatures [BBS04, ASM06, TZ23a]. 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 [CDL16]. 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. [OTZZ24]), 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 $xA = B$.[4]

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 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 Section 5.

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 [CMZ14], 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 Section 6), that essentially runs the key-dependent verification of this KVAC with the embedded secret key.

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

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 or no longer allowed by the issuer, 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.

Scheme	Publicly Verifiable	Multi-Show	Credential Size	Helper			Showing Size	Security	
				Usr. Comm	Iss Comm	Rnds		Unforgeability	Anonymity
CMZ14 [CMZ14]	No	Yes	$2\mathbb{G}$	-	-	-	$(\ell+2)\mathbb{G}$ $+(2\ell+2)\mathbb{Z}_p$	GGM	DDH
BBDT16 [BBDT16]	No	Yes	$2\mathbb{G}+2\mathbb{Z}_p$	-	-	-	$3\mathbb{G}$ $+(\ell+7)\mathbb{Z}_p$	Gap-q-SDH	Statistical
$\mathsf{KVAC}_{\mathsf{wBB}}$ [CDDH19]	No	Yes	$(\ell+1)\mathbb{G}$	-	-	-	$2\mathbb{G}$ $+(\ell+1)\mathbb{Z}_p$	ℓ-SCDHI	Statistical
μCMZ [Orr24]	No	Yes	$2\mathbb{G}$	-	-	-	$(\ell+2)\mathbb{G}$ $+(2\ell+2)\mathbb{Z}_p$	AGM + 3-DL	Statistical
μBBS [Orr24]	No	Yes	$1\mathbb{G}+1\mathbb{Z}_p$	-	-	-	$2\mathbb{G}$ $+(\ell+4)\mathbb{Z}_p$	AGM + q-DL	Statistical
MBS+25 [MBS+25]	No	Yes	$(\ell+2)\mathbb{G}$	-	-	-	$2\mathbb{G}$	GGM	Statistical
ACL [BL13]	Yes	No	$2\mathbb{G}+6\mathbb{Z}_p$	-	-	-	$2\mathbb{G}$ $+(\ell+8)\mathbb{Z}_p$	DL+AGM	DDH
$\mathsf{SAAC}_{\mathsf{BBS}}$	Yes	Yes*	$1\mathbb{G}+2\mathbb{Z}_p$	$2\mathbb{G}+1\mathbb{Z}_p$	$3\mathbb{G}+3\mathbb{Z}_p$	2	$3\mathbb{G}$ $+(\ell+8)\mathbb{Z}_p$	Gap-q-SDH	Statistical
$\mathsf{SAAC}_{\mathsf{DDH}}$	Yes	Yes*	$4\mathbb{G}$	$(\ell+4)\mathbb{G}$ $+1\mathbb{Z}_p$	$(2\ell+9)\mathbb{G}$ $+(2\ell+7)\mathbb{Z}_p$	2	$(\ell+6)\mathbb{G}+$ $(4\ell+11)\mathbb{Z}_p$	DDH	DDH
BBS [TZ23a]†	Yes	Yes	$1\mathbb{G}_1+1\mathbb{Z}_p$	-	-	-	$2\mathbb{G}_1$ $+(\ell+3)\mathbb{Z}_p$	q-SDH	Statistical

This KVAC was already known to be provably secure but under a definition that is weaker than what we need to instantiate our generic construction. To address this gap, we made the following contributions:

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 [CMZ14] which contains a linear number of group elements.

For more details on this instantiation, we refer the readers to Section 6.

2 Preliminaries

<u>Notations.</u> 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 often 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.

Game $\mathrm{DL}^{\mathcal{A}}_{\mathsf{GGen}}(\lambda)$:	Game $(q,\mathcal{O})$-$\mathrm{SDH}^{\mathcal{A}}_{\mathsf{GGen}}(\lambda)$
$\mathsf{par} = (p, G, \mathbb{G}) \leftarrow\$ \mathsf{GGen}(1^\lambda)$ $X \leftarrow\$ \mathbb{G}$ $x \leftarrow\$ \mathcal{A}(\mathsf{par}, X)$ **return** $xG = X$ Game $\mathrm{DDH}^{\mathcal{A}}_{\mathsf{GGen},b}(\lambda)$: $\mathsf{par} = (p, G, \mathbb{G}) \leftarrow\$ \mathsf{GGen}(1^\lambda)$ $x, y, z \leftarrow\$ \mathbb{Z}_p$ $Z_0 \leftarrow xyG; Z_1 \leftarrow zG$ $b' \leftarrow\$ \mathcal{A}(\mathsf{par}, xG, yG, Z_b)$ **return** b'	$\mathsf{par} = (p, G, \mathbb{G}) \leftarrow\$ \mathsf{GGen}(1^\lambda)$ $x \leftarrow\$ \mathbb{Z}_p$ $(e, Z) \leftarrow\$ \mathcal{A}^{\mathcal{O}(\mathsf{par}, x, xG, \cdot)}(\mathsf{par}, (x^i G)_{i \in [q]})$ **return** $(Z = (x+e)^{-1} G)$ Oracle $\mathrm{rDDH}(\mathsf{par}, x, X, (A, B))$ **return** $xA = B$ // X is unused.

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

<u>Group parameter generator.</u> 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 $A, B \in \mathbb{G}$).

<u>Cryptographic assumptions.</u> In Figure 2, we define games for Decisional Diffie-Hellman (DDH), Discrete Logarithm (DL), and a pairing-free analog of the *q-Strong Diffie-Hellman* assumption [BB08] augmented with a *restricted* DDH oracle. Denote the advantage of an adversary $\mathcal{A}$ against these assumptions as

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

For modularity of our security proofs, we will rely on the rel-DL and n-DDH (a multi-instance version of DDH) assumptions with the games described in Figure 3. With the corresponding advantage defined as

$$\begin{aligned}\mathsf{Adv}^{\mathrm{rel\text{-}dl}}_{\mathsf{GGen}}(\mathcal{A},\lambda) &:= \Pr[\mathrm{rel\text{-}DL}^{\mathcal{A}}_{\mathsf{GGen}}(\lambda) = 1]\ ,\\ \mathsf{Adv}^{\mathrm{ddh}}_{\mathsf{GGen},n}(\mathcal{A},\lambda) &:= \left|\Pr[\mathrm{n\text{-}DDH}^{\mathcal{A}}_{\mathsf{GGen},0}(\lambda) = 1] - \Pr[\mathrm{DDH}^{\mathcal{A}}_{\mathsf{GGen},1}(\lambda) = 1]\right| .\end{aligned}$$

The following lemmas establish tight reduction between rel-DL and DL and n-DDH and DDH. Lemma 2.2 follows from the random self-reducibility of DDH (see e.g., [EHK+13]).

Lemma 2.1 ([JT20]). *Let $n = n(\lambda)$ and GGen be a group generation algorithm outputs groups of prime order $p = p(\lambda)$. For any $\mathcal{A}$ running in time $t_{\mathcal{A}} = t_{\mathcal{A}}(\lambda)$, there exists $\mathcal{B}$ running in time $t_{\mathcal{A}} + O(n)$ such that*

$$\mathsf{Adv}^{\mathrm{rel\text{-}DL}}_{\mathsf{GGen},n}(\mathcal{A},\lambda) \leqslant \mathsf{Adv}^{\mathrm{dlog}}_{\mathsf{GGen}}(\mathcal{B},\lambda) + \frac{1}{p}$$

Lemma 2.2. *Let $n = n(\lambda)$ and GGen be a group generation algorithm outputs groups of prime order $p = p(\lambda)$. For any $\mathcal{A}$ running in time $t_{\mathcal{A}} = t_{\mathcal{A}}(\lambda)$, there exists $\mathcal{B}$ running in time $t_{\mathcal{A}} + O(n)$ such that*

$$\mathsf{Adv}^{\mathrm{ddh}}_{\mathsf{GGen},n}(\mathcal{A},\lambda) \leqslant \mathsf{Adv}^{\mathrm{ddh}}_{\mathsf{GGen}}(\mathcal{B},\lambda) + \frac{1}{p-1}\ .$$

<u>Random oracles.</u> Most of our analyses assume one or more random oracles, and we will clearly indicate so in the theorem statements. The random oracles are modeled as additional oracles to which the adversary $\mathcal{A}$ is given access.

Game n-DDH$_b(\lambda)$

$\mathsf{par} = (G, p, \mathbb{G}) \leftarrow \mathsf{GGen}(1^\lambda)$:
$x \leftarrow\!\!\$\ \mathbb{Z}_p$
$(y_i)_{i=1}^n, (z_i)_{i=1}^n \leftarrow\!\!\$\ \mathbb{Z}_p^n$
$Z_{i,0} \leftarrow xy_iG; Z_{i,1} \leftarrow z_iG$ for all $i \in [n]$
$b' \leftarrow\!\!\$\ \mathcal{A}(\mathsf{par}, xG, (b_iG)_{i=1}^n, (Z_{i,b})_{i=1}^n)$

Game rel-DL(λ) :

$\mathsf{par} = (G, p, \mathbb{G}) \leftarrow \mathsf{GGen}(1^\lambda)$:
$(X_i)_{i=1}^n \leftarrow\!\!\$\ \mathbb{G}^n$
$(y_i)_{i=0}^n \leftarrow \mathcal{A}(\mathsf{par}, (X_i)_{i=1}^n)$
if $(y_i)_{i=1}^n = 0^n$ **then return** 0
return $\sum_{i=1}^n y_iX_i = y_0G$

Fig. 3. Games n-DDH and rel-DL

Games UFCMA$^{\mathcal{A}}_{\mathsf{MAC},\mathcal{O}}(\lambda)$, $\boxed{\text{UFCMVA}^{\mathcal{A}}_{\mathsf{MAC},\mathcal{O}}(\lambda)}$

$\mathsf{MsgQ} \leftarrow \varnothing$; $\mathsf{par} \leftarrow\!\!\$\ \mathsf{MAC.Setup}(1^\lambda)$
$(\mathsf{sk}, \mathsf{ipk}) \leftarrow\!\!\$\ \mathsf{MAC.KG}(\mathsf{par})$
$(m^*, \sigma^*) \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{MAC}, \mathcal{O}, \boxed{\mathrm{V}}}(\mathsf{par}, \mathsf{ipk})$
if $m^* \notin \mathsf{MsgQ}\ \wedge\ \mathsf{MAC.Ver}(\mathsf{par}, \mathsf{sk}, m^*, \sigma^*) = 1$
  then return 1
return 0

Oracle MAC(m)

$\sigma \leftarrow\!\!\$\ \mathsf{MAC.M}(\mathsf{par}, \mathsf{sk}, m)$
if $\sigma \neq \perp$ **then**
  $\mathsf{MsgQ} \leftarrow \mathsf{MsgQ} \cup \{m\}$
return σ

Oracle V(m, σ)

return $\mathsf{MAC.Ver}(\mathsf{par}, \mathsf{sk}, m, \sigma)$

Fig. 4. Unforgeability under chosen message attack (UFCMA) and unforgeability under chosen message and verification queries (UFCMVA) games

---

<u>Message authentication codes.</u> A message authentication code MAC is a tuple of algorithms ($\mathsf{MAC.Setup}$, $\mathsf{MAC.KG}$, $\mathsf{MAC.M}$, $\mathsf{MAC.Ver}$) with the following syntax:

- The setup algorithm $\mathsf{MAC.Setup}(1^\lambda)$ generates public parameters par. We let the public parameters par define the message space $\mathsf{MAC.M} = \mathsf{MAC.M}(\mathsf{par})$.
- The key generation algorithm $\mathsf{MAC.KG}(\mathsf{par})$ outputs the secret key sk and the issuer's public parameters ipk.
- The randomized MAC algorithm $\mathsf{MAC.M}(\mathsf{par}, \mathsf{sk}, m)$ takes as inputs, the secret key sk and a message $m \in \mathsf{MAC.M}$, and outputs a message authentication code σ.
- The deterministic verification algorithm outputs a bit $\mathsf{MAC.Ver}(\mathsf{par}, \mathsf{sk}, m, \sigma)$.

Note that the issuer's public key ipk is not used in the MAC and verification algorithms, but will be relevant in the keyed-verification anonymous credentials (KVAC) building on algebraic MACs, which we define later on.

Correctness is defined as usual in that for any public parameters par and key sk generated from the setup and key generation algorithms and any message $m \in \mathsf{M}$, the message authentication code $\sigma \leftarrow \mathsf{MAC.M}(\mathsf{par}, \mathsf{sk}, m)$ always satisfies $\mathsf{MAC.Ver}(\mathsf{par}, \mathsf{sk}, m, \sigma) = 1$. We consider two security definitions: *unforgeability under chosen message attack* (UFCMA) and *unforgeability under chosen message and verification queries attack*, which are respectively defined by the games UFCMA$^{\mathcal{A}}_{\mathsf{MAC}}(\lambda)$ and UFCMVA$^{\mathcal{A}}_{\mathsf{MAC}}(\lambda)$ (both given in Figure 4). Additionally, we define UFCMA/UFCMVA in the presence of an arbitrary oracle, denoted $\mathcal{O}$-UFCMA. (Note that for some schemes and oracles that we consider in this paper, $\mathcal{O}$-UFCMA implies UFCMVA.) The corresponding advantage of any adversary $\mathcal{A}$ playing the game ($\mathcal{O}$ is optional) is:

$$\mathsf{Adv}^{\mathsf{ufcma/ufcmva}}_{\mathsf{MAC},\mathcal{O}}(\mathcal{A}, \lambda) := \Pr[(\mathcal{O}\text{-UFCMA}/\mathcal{O}\text{-UFCMVA})^{\mathcal{A}}_{\mathsf{MAC}}(\lambda) = 1]\,.$$

<u>Relations and Σ-protocol.</u> 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 Σ-protocol for a relation R is a tuple of algorithms:

- $\mathsf{Init}(x, w)$: given a statement and witness $(x, w) \in \mathsf{R}$, output a commitment R and a state st.
- $\mathsf{Resp}(\mathsf{st}, c)$: given a challenge $c \in \mathcal{CH}$, output a response z.

- $\mathsf{Verify}(x, R, c, z)$: output a bit $b \in \{0, 1\}$.

The *transcript* (R, c, z) is *valid* for a statement x if $\mathsf{Verify}(x, R, c, z) = 1$. Σ-protocols satisfy correctness, honest-verifier zero-knowledge, special soundness, and high min-entropy.

- **Correctness.** For any $(x, w) \in \mathsf{R}$, $(R, \mathsf{st}) \in [\mathsf{Init}(x, w)], c \in \mathcal{CH}, z \leftarrow \mathsf{Resp}(\mathsf{st}, c)$, $\mathsf{Verify}(x, R, c, z) = 1$.
- **Honest-verifier zero-knowledge (HVZK).** There exists an efficient simulator Sim such that for any $(x, w) \in \mathsf{R}$, $c \in \mathcal{CH}$ the following distributions are identical: $\{(R, c, z) : (R, \mathsf{st}) \leftarrow\$ \mathsf{Init}(x, w), z \leftarrow \mathsf{Resp}(\mathsf{st}, c)\} \equiv \{(R, c, z) : (R, z) \leftarrow\$ \mathsf{Sim}(x, c)\}$
- **Special soundness.** There exists an efficient deterministic extractor Ext such that for any x and two transcripts $(R, c, z), (R, c', z')$ where $c \neq c'$, the output $w \leftarrow \mathsf{Ext}(x, (R, c, z), (R, c', z'))$ is such that $(x, w) \in \mathsf{R}$.
- **High Min-Entropy.** For any $(x, w) \in \mathsf{R}$, $(R, \mathsf{st}) \leftarrow\$ \mathsf{Init}(x, w)$ is such that $2^{-\mathsf{H}_{\mathsf{min}}(R)}$ is negligible, where $\mathsf{H}_{\mathsf{min}}(X) := -\log \max_{x \in \mathcal{X}} \Pr[X = x]$ denotes the min entropy of a random variable X with values drawn from a finite domain $\mathcal{X}$. Moreover, we denote $\mathsf{H}_{\mathsf{min}}(\Sigma) := \min_{x \in \mathcal{L}_{\mathsf{R}}} \mathsf{H}_{\mathsf{min}}(R)$.

<u>Non-interactive zero-knowledge proofs.</u> A non-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.

The proof systems used in this work only rely on the random oracle. We require that NIZK satisfies the following properties:

- **Correctness.** For any $(x, w) \in \mathsf{R}$,
$$\Pr[1 = \mathsf{NIZK.Ver}^{\mathsf{H}}(x, \pi) | \pi \leftarrow\$ \mathsf{NIZK.Prove}^{\mathsf{H}}(x, w)] \geqslant 1 - \eta(\lambda)$$
where the probability is over the random choice of H and the random coins of $\mathsf{NIZK.Prove}$. We denote η as the correctness error.
- **Soundness.** For any adversary $\mathcal{A}$ with bounded access to H, the following advantage is bounded
$$\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{NIZK}}(\mathcal{A}, \lambda) := \Pr\left[x \notin \mathcal{L}_{\mathsf{R}} \ \wedge \ \mathsf{NIZK.Ver}^{\mathsf{H}}(x, \pi) = 1 \,\middle|\, (x, \pi) \leftarrow\$ \mathcal{A}^{\mathsf{H}}(1^\lambda)\right] .$$
- **Zero-knowledge.** There exists a simulator Sim which is allowed to reprogram H such that for any adversary $\mathcal{A}$ with bounded access to H, the following advantage is bounded:
$$\mathsf{Adv}^{\mathsf{zk}}_{\mathsf{NIZK},\mathsf{Sim}}(\mathcal{A}, \lambda) := \left|\Pr[\mathcal{A}^{\mathsf{H}, \mathrm{P}_0}(1^\lambda) = 1] - \Pr[\mathcal{A}^{\mathsf{H}, \mathrm{P}_1}(1^\lambda) = 1]\right| .$$
The oracles $\mathrm{P}_b(x, w)$ does the following: If $(x, w) \notin \mathsf{R}$ then return $\perp$. If $b = 0$, then return $\pi \leftarrow\$ \mathsf{NIZK.Prove}^{\mathsf{H}}(x, w)$. Otherwise, return $\pi \leftarrow\$ \mathsf{Sim}^{\mathsf{H}}(x)$.
- **Relaxed knowledge-soundness.** A NIZK is straight-line extractable knowledge-sound for a *relaxed relation* $\widetilde{\mathsf{R}} \supseteq \mathsf{R}$ if there exists an extractor Ext who has access to the adversary's random oracle queries such that for any adversary $\mathcal{A}$ playing the game KSND (defined in Figure 5), the following advantage is bounded
$$\mathsf{Adv}^{\mathrm{ksnd}}_{\mathsf{NIZK},\mathsf{Ext},\widetilde{\mathsf{R}}}(\mathcal{A}, \lambda) := \Pr[\mathrm{KSND}^{\mathcal{A}}_{\mathsf{NIZK},\mathsf{Ext},\widetilde{\mathsf{R}}}(\lambda) = 1] .$$

<u>Proofs for linear relations.</u> Throughout the paper, we will use Σ-protocol for proving preimage of linear maps over a prime-order group $\mathbb{G}$ [Mau15]. The relation $\mathsf{R}_{\mathbb{G}}$ contains statements of the form $(M \in \mathbb{G}^{n \times m}, \boldsymbol{Y} \in \mathbb{G}^n)$ and the witnesses are $\boldsymbol{x} \in \mathbb{Z}_p^m$ such that $\boldsymbol{Y} = M\boldsymbol{x}$. In particular, we consider the following Σ-protocol $\Sigma_{\mathsf{Lin}} = (\mathsf{Init}, \mathsf{Resp}, \mathsf{Verify})$ described as

- $(\boldsymbol{R}, \mathsf{st}) \leftarrow\$ \mathsf{Init}((M \in \mathbb{G}^{n \times m}, \boldsymbol{Y} \in \mathbb{G}^n), \boldsymbol{x} \in \mathbb{Z}_p^m)$: sample $\boldsymbol{r} \leftarrow\$ \mathbb{Z}_p^m$ and output $(\boldsymbol{R} \leftarrow M\boldsymbol{r}, \mathsf{st} \leftarrow (\boldsymbol{x}, \boldsymbol{r}))$
- $\boldsymbol{z} \leftarrow \mathsf{Resp}(\mathsf{st}, c \in \mathbb{Z}_p)$: output $\boldsymbol{z} \leftarrow \boldsymbol{r} + c\boldsymbol{x}$.

Game $\mathrm{KSND}^{\mathcal{A}}_{\mathsf{NIZK},\mathsf{Ext},\widetilde{\mathsf{R}}}(\lambda)$:	Oracle $\mathsf{H}(\mathrm{str})$:	Oracle $\mathcal{O}_{\mathsf{Ext}}(x, \pi)$:
$\mathsf{win} \leftarrow 0$; $\mathcal{Q} \leftarrow \varnothing$	**if** $\mathsf{T}[\mathrm{str}] \neq \perp$ **then**	**if** $\mathsf{NIZK.Ver}^{\mathsf{H}}(x, \pi) \neq 1$ **then**
Map: $\mathsf{T} \leftarrow [\cdot]$	**return** $\mathsf{T}[\mathrm{str}]$	**return** 0
$\mathcal{A}^{\mathsf{H},\mathcal{O}_{\mathsf{Ext}}}(1^\lambda)$	$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\mathrm{str}\}$	$w \leftarrow \mathsf{Ext}^{\mathsf{H}}(\mathcal{Q}, x, \pi)$
return win	$\mathsf{T}[\mathrm{str}] \leftarrow\$ \mathcal{R}$	**if** $(x, w) \notin \widetilde{\mathsf{R}}$ **then** $\mathsf{win} \leftarrow 1$
	return $\mathsf{T}[\mathrm{str}]$	**return** 1

Fig. 5. Straightline extractable knowledge soundness game for NIZK.

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

Fig. 6. 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.

- $b \leftarrow\$ \mathsf{Verify}((M, \boldsymbol{Y}), \boldsymbol{R}, c, \boldsymbol{z})$: output 1 if and only if $\boldsymbol{R} + c\boldsymbol{Y} = M\boldsymbol{z}$.

Additionally, we will repeatedly use a non-interactive proof system Lin for $\mathsf{R}_{\mathbb{G}}$ which is obtained by applying the Fiat-Shamir transform to Σ_{Lin} (see the description of proof system Lin in Figure 6). Note in particular that the prover and verifier take an additional (and optional) input string nonce which will be an additional input to H.

The following theorem then establishes the security of the proof system Lin in Figure 6. This follows from Fiat-Shamir transform applying to Σ_{Lin} (see e.g., Boneh-Shoup [BS20, Chapter 19-20]).

Theorem 2.3. Lin *satisfies perfect correctness, zero-knowledge, and soundness in the ROM.*

3 Server-Aided Anonymous Credentials

In this section, we introduce Server-Aided Anonymous Credentials (SAAC), with the syntax and security definitions given in Sections 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 the user can use to output a publicly-verifiable showing message. This is modeled as an unlinkable helper protocol, which is 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 $\Phi = \{\Phi_{\mathsf{par}}\}_{\mathsf{par}}$[5] and an attribute space $\mathcal{M} = \{\mathcal{M}_{\mathsf{par}}\}_{\mathsf{par}}$ consists of the following algorithms.

- $\mathsf{par} \leftarrow\$ \mathsf{SAAC.Setup}(1^\lambda, 1^\ell)$ 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.

[5] 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. However, we define our SAAC syntax with respect to a single class of predicates $\Phi = \Phi_{\mathsf{Iss}} \cup \Phi_{\mathsf{Show}}$ covering both predicate classes for issuance and showing. This will be the case for our constructions which consider the class of selective disclosure predicates for both issuance and showing.

- $(\mathsf{sk}, \mathsf{pk}) \leftarrow\$ \mathsf{SAAC.KeyGen}(\mathsf{par})$ 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 the first 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:

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

- $\tau \leftarrow\$ \mathsf{SAAC.Show}(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}, \sigma, \mathsf{aux}, \phi, \mathsf{nonce})$ 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 $\boldsymbol{m} \in \mathcal{M}^\ell_{\mathsf{par}}$, 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

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

2. Denote the advantage of any adversary $\mathcal{A}$ in the unforgeability game, defined in Figure 7 with respect to Ext (more discussion on the game below), as

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

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

$\mathsf{MsgQ}, \mathsf{PfQ}, \mathcal{I}_1, \ldots, \mathcal{I}_r, \mathcal{C} \leftarrow \varnothing; \mathsf{win} \leftarrow 0$
$(\mathsf{par}, \mathsf{td}) \leftarrow\$ \mathsf{Ext}_{\mathsf{Setup}}(1^\lambda, 1^\ell)$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow\$ \mathsf{SAAC.KeyGen}(\mathsf{par})$
$(\phi^*, \mathsf{nonce}^*, \tau^*)$
$\quad \leftarrow\$ \mathcal{A}^{\mathrm{Iss}, \mathrm{Help}_1, \ldots, \mathrm{Help}_r, \mathrm{NewUsr}, \mathrm{SH}}(\mathsf{par}, \mathsf{pk})$
if $(\mathsf{SAAC.SVer}(\mathsf{par}, \mathsf{pk}, \tau^*, \phi^*, \mathsf{nonce}^*) = 1) \wedge$
$\quad (\forall \boldsymbol{m} \in \mathsf{MsgQ} : \phi^*(\boldsymbol{m}) = 0) \wedge$
$\quad ((\phi^*, \mathsf{nonce}^*, \tau^*) \notin \mathsf{PfQ})$
$\quad\quad$ **then return** 1
return win

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

$\mathsf{imsg} \leftarrow\$ \mathsf{SAAC.Iss}(\mathsf{par}, \mathsf{sk}, \mu, \phi)$
if $\mathsf{imsg} = \perp$ **then abort**
$\boldsymbol{m} \leftarrow \mathsf{Ext}_{\mathsf{Iss}}(\mathsf{td}, \mu, \phi)$
if $\phi(\boldsymbol{m}) = 0 \vee \boldsymbol{m} = \perp$ **then** $\mathsf{win} \leftarrow 1$
$\quad$ // $\mathcal{A}$ wins if it can request
$\quad$ // 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 abort**
$\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathsf{cid}\}; \boldsymbol{m}_{\mathsf{cid}} \leftarrow \boldsymbol{m}$
$\sigma_{\mathsf{cid}} \leftarrow\$ \langle \mathsf{SAAC.Iss}(\mathsf{par}, \mathsf{sk}, \phi)$
$\quad\quad \rightleftharpoons \mathsf{SAAC.U}(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}, \phi) \rangle$
return closed

Oracle $\mathrm{SH}(\mathsf{cid}, \phi, \mathsf{nonce})$:

if $\mathsf{cid} \notin \mathcal{C}$ **then abort**
$(\perp, \mathsf{aux}) \leftarrow\$ \langle \mathsf{SAAC.Helper}(\mathsf{par}, \mathsf{sk})$
$\quad \rightleftharpoons \mathsf{SAAC.ObtHelp}(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}_{\mathsf{cid}}, \sigma_{\mathsf{cid}}) \rangle$
$\tau \leftarrow\$ \mathsf{SAAC.Show}(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}_{\mathsf{cid}}, \sigma_{\mathsf{cid}}, \mathsf{aux}, \phi, \mathsf{nonce})$
$\mathsf{PfQ} \leftarrow \mathsf{PfQ} \cup \{(\phi, \mathsf{nonce}, \tau)\}$
return τ

Oracle $\mathrm{Help}_j(\mathsf{sid}, \mathsf{umsg}_j)$: $\quad$ // $j = 1, \ldots, r$

if $\mathsf{sid} \notin \mathcal{I}_1, \ldots, \mathcal{I}_{j-1} \vee \mathsf{sid} \in \mathcal{I}_j$
$\quad$ **then abort**
$\mathcal{I}_j \leftarrow \mathcal{I}_j \cup \{\mathsf{sid}\}$
if $j = 1$ **then** $\quad$ // For $j = r, \mathsf{st}^h_{\mathsf{sid}} = \perp$
$\quad (\mathsf{hmsg}_j, \mathsf{st}^h_{\mathsf{sid}}) \leftarrow\$ \mathsf{SAAC.Helper}_1(\mathsf{par}, \mathsf{sk}, \mathsf{umsg}_j)$
else $(\mathsf{hmsg}_j, \mathsf{st}^h_{\mathsf{sid}}) \leftarrow\$ \mathsf{SAAC.Helper}_j(\mathsf{st}^h_{\mathsf{sid}}, \mathsf{umsg}_j)$
return hmsg_j

Fig. 7. Unforgeability game for $\mathsf{SAAC} = \mathsf{SAAC}[\Phi, \mathcal{M}]$. We assume that all the predicates output by $\mathcal{A}$ are in Φ.

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.
- 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 and other scenarios which we do not consider as an attack on SAAC.

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

The extractor should output attributes satisfying the predicate. Consider a similar game *where the issuance oracle aborts* if the extracted attributes does 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 in this game, 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, and this means that standard AC should satisfy SAAC notion. We note however that, for our instantiations, at least one helper interaction is required to output a 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 [CKL+16] and the extractability security of KVAC given in [Orr24], but not in the original KVAC unforgeability definition [CMZ14].

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 do not consider an adversary who forges a showing *by forcing honest users to reuse a helper information* aux. In our view, honest users should not compromise their anonymity by reusing the helper information aux. One could argue that (a) this can occur given a bug in the system or (b) honest users might not care about their anonymity. However, we see (a) as a problem in the system implementation. For (b), it would be more convenient (and efficient) for such users to instead use 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. Our definition does not cover an adversary that can see the transcript between the user and the helper or intercept user's messages during the helper protocol. We leave the consideration of a stronger (and more complicated) model of adversaries for future work.

ANONYMITY. For 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

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

2. The advantage of $\mathcal{A}$ in the anonymity game, defined in Figure 8 with respect to Sim, is bounded

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

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:

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.

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

$\mathtt{init} \leftarrow 0; \mathcal{I}_1, \ldots, \mathcal{I}_{r+1}, \mathcal{HP} \leftarrow \varnothing$

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

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

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

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

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

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

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

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

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

$b' \leftarrow\!\$\ \mathcal{A}^{\mathrm{ObtH}_1, \ldots, \mathrm{ObtH}_{r+1}, \mathrm{SH}}(\mathsf{st}'_{\mathcal{A}})$

return b'

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

if $\phi(\boldsymbol{m}) = 0 \ \vee\ \mathsf{sid} \notin \mathcal{HP}$ **then abort**

$\mathcal{HP} \leftarrow \mathcal{HP} \setminus \{\mathsf{sid}\}$

// Each $\mathsf{aux}_{\mathsf{sid}}$ is used 'only once'.

$\tau \leftarrow\!\$\ \mathsf{SAAC.Show}(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}, \sigma, \mathsf{aux}_{\mathsf{sid}}, \phi, \mathsf{nonce})$ // $b = 0$

$\tau \leftarrow\!\$\ \mathsf{Sim}_{\mathsf{Show}}(\mathsf{td}, \mathsf{pk}, \phi, \mathsf{nonce})$ // $b = 1$

return τ

Oracle $\mathrm{ObtH}_1(\mathsf{sid})$:

if $\mathsf{sid} \in \mathcal{I}_1$ **then abort**

$\mathcal{I}_1 \leftarrow \mathcal{I}_1 \cup \{\mathsf{sid}\}$

if $j = 1$ **then** // $b = 0$

$(\mathsf{umsg}_1, \mathsf{st}_{\mathsf{sid}}) \leftarrow\!\$\ \mathsf{SAAC.ObtHelp}_1(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}, \sigma)$

if $j = 1$ **then** // $b = 1$

$(\mathsf{umsg}_1, \mathsf{st}_{\mathsf{sid}}) \leftarrow\!\$\ \mathsf{Sim}_{\mathsf{ObtH}}(\mathsf{td}, \mathsf{pk})$

return umsg_1

Oracle $\mathrm{ObtH}_j(\mathsf{sid}, \mathsf{hmsg}_{j-1})$: // $j = 2, \ldots, r+1$

if $\mathsf{sid} \notin \mathcal{I}_1, \ldots, \mathcal{I}_{j-1} \ \vee\ \mathsf{sid} \in \mathcal{I}_j$ **then abort**

$\mathcal{I}_j \leftarrow \mathcal{I}_j \cup \{\mathsf{sid}\}$

if $1 < j \leqslant r$ **then** // $b = 0$

$(\mathsf{umsg}_j, \mathsf{st}_{\mathsf{sid}}) \leftarrow\!\$\ \mathsf{SAAC.ObtHelp}_j(\mathsf{st}_{\mathsf{sid}}, \mathsf{hmsg}_{j-1})$

return umsg_j

if $j = r + 1$ **then**

$\mathsf{aux}_{\mathsf{sid}} \leftarrow\!\$\ \mathsf{SAAC.ObtHelp}_j(\mathsf{st}_{\mathsf{sid}}, \mathsf{hmsg}_{j-1})$

if $\mathsf{aux}_{\mathsf{sid}} = \perp$ **then abort**

if $1 < j \leqslant r$ **then** // $b = 1$

$(\mathsf{umsg}_j, \mathsf{st}_{\mathsf{sid}}) \leftarrow\!\$\ \mathsf{Sim}_{\mathsf{ObtH}}(\mathsf{st}_{\mathsf{sid}}, \mathsf{hmsg}_{j-1})$

return umsg_j

if $j = r + 1$ **then**

$\mathsf{aux}_{\mathsf{sid}} \leftarrow\!\$\ \mathsf{Sim}_{\mathsf{ObtH}}(\mathsf{st}_{\mathsf{sid}}, \mathsf{hmsg}_{j-1})$

if $\mathsf{aux}_{\mathsf{sid}} = \perp$ **then abort**

$\mathcal{HP} \leftarrow \mathcal{HP} \cup \{\mathsf{sid}\}$ // Only occurs for $j = r + 1$

return $\mathtt{closed}$

Fig. 8. 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.

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 is allowed to specify a helper information (via sid) owned by an honest user, 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 $\boldsymbol{m}_{\mathsf{cid}}$ satisfying ϕ and the credential σ_{cid}. *Each helper information is restricted to be used only once.* On the other hand, the simulator only requires the trapdoor td, the public key pk, and the specified predicate ϕ to simulate.

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 satisfied via a hybrid argument. We include the security definition and the proof in Appendix A.

Integrity. The integrity property, formalized in Figure 9, 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

Game $\boxed{\mathrm{Integ}^{\mathcal{A}}_{\mathsf{SAAC,strong}}(1^\lambda)}$ $\boxed{\mathrm{Integ}^{\mathcal{A}}_{\mathsf{SAAC,weak}}(1^\lambda)}$:
$\mathsf{par} \leftarrow \mathsf{SAAC.Setup}(1^\lambda, 1^\ell)$
$\boxed{(\mathsf{pk}, \boldsymbol{m}, \tilde{\phi}, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(\mathsf{par})}$ $\boxed{(\rho, \boldsymbol{m}, \tilde{\phi}, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(\mathsf{par}); (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{SAAC.KeyGen}(\mathsf{par}; \rho)}$
$(\mathsf{st}'_{\mathcal{A}}, \sigma) \leftarrow\$ \langle \mathcal{A}(\mathsf{st}_{\mathcal{A}}) \rightleftharpoons \mathsf{SAAC.U}(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}, \tilde{\phi}) \rangle$
$(\mathsf{st}''_{\mathcal{A}}, \mathsf{aux}) \leftarrow\$ \langle \mathcal{A}(\mathsf{st}'_{\mathcal{A}}) \rightleftharpoons \mathsf{SAAC.ObtHelp}(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}, \sigma) \rangle$
$(\phi, \mathsf{nonce}) \leftarrow \mathcal{A}(\mathsf{st}''_{\mathcal{A}})$
$\tau \leftarrow\$ \mathsf{SAAC.Show}(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}, \sigma, \mathsf{aux}, \phi, \mathsf{nonce})$
return $\tilde{\phi}(\boldsymbol{m}) = \phi(\boldsymbol{m}) = 1 \wedge \sigma \neq \perp \wedge \mathsf{aux} \neq \perp \wedge \mathsf{SAAC.SVer}(\mathsf{par}, \mathsf{pk}, \tau, \phi, \mathsf{nonce}) = 0$

Fig. 9. Strong and weak integrity games of $\mathsf{SAAC} = \mathsf{SAAC}[\Phi, \mathcal{M}]$. The strong version uses the unboxed and dashed code. The weak version uses the unboxed and highlighted code. We assume that $\mathcal{A}$ outputs predicates in Φ.

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 define two variants: *strong integrity*, where the public key can be chosen maliciously; and *weak integrity*, where the adversary reveals its random coins ρ used to generate the key. Denote the integrity advantage of $\mathcal{A}$ as

$$\mathsf{Adv}^{\mathsf{integ}}_{\mathsf{SAAC,(strong/weak)}}(\mathcal{A}, \lambda) := \Pr[\mathrm{Integ}^{\mathcal{A}}_{\mathsf{SAAC,(strong/weak)}}(\lambda) = 1] \;,$$

and in Appendix B, we prove the following theorem.

Theorem 3.1. *If* SAAC *satisfies correctness and anonymity, then* SAAC *satisfies weak integrity.*

Remark 3.2. If generic NIZK proof systems exist, any SAAC satisfying weak integrity can be transformed into a SAAC' satisfying strong integrity. This is because the issuer can publish a proof of knowledge of ρ such that for $(\mathsf{sk}', \mathsf{pk}') \leftarrow \mathsf{SAAC.KeyGen}(\mathsf{par}; \rho)$ the string pk' equals their public key.

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 Section 4.1, and give a generic construction of SAAC in Section 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.

<u>Global parameters generator.</u> Inspired by the formalization in [CKL+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. Denote the distinguishing advantage of $\mathcal{A}$ as

$$\mathsf{Adv}^{\mathsf{par\text{-}indist}}_{\mathsf{Gen},\mathsf{Sim}_{\mathsf{Gen}}}(\mathcal{A}, \lambda) := \left|\Pr[\mathcal{A}(\mathsf{par}_g) = 1 | \mathsf{par}_g \leftarrow\$ \mathsf{Gen}(1^\lambda)] - \Pr[\mathcal{A}(\mathsf{par}_g) = 1 | (\mathsf{par}_g, \mathsf{td}_g) \leftarrow\$ \mathsf{Sim}_{\mathsf{Gen}}(1^\lambda)]\right| \;.$$

<u>SYNTAX ON RELATIONS FOR OBLIVIOUS PROOF ISSUANCE.</u> Particularly for this section, we use a similar syntax for relations and languages from [OTZZ24]. In [OTZZ24], 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. Further, we denote the relation $\mathsf{Core}(\mathsf{R})$ and the induced language $\mathcal{L}_{\mathsf{R}}$ as

$$\begin{aligned}\mathsf{Core}(\mathsf{R}) &:= \{(X, x) : \exists Y \text{ such that } ((X, Y), x) \in \mathsf{R}\}\ ,\\ \mathcal{L}_{\mathsf{R}} &:= \{(X, Y) : \exists x \text{ such that } ((X, Y), x) \in \mathsf{R}\}\ .\end{aligned}$$

The membership $(X, x) \in \mathsf{Core}(\mathsf{R})$ can be efficiently checked.

<u>KEYED-VERIFICATION ANONYMOUS CREDENTIALS.</u> A keyed-verification anonymous credential (KVAC) scheme $\mathsf{KVAC} = \mathsf{KVAC}[\mathsf{Gen}, \Phi, \mathcal{M}]$, defined with respect to the global parameters generator 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.
- $(\perp, \sigma) \leftarrow\$\ \langle \mathsf{KVAC.Iss}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{sk}, \phi) \rightleftharpoons \mathsf{KVAC.U}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}, \boldsymbol{m}, \phi)\rangle$ is a round-optimal protocol with similar syntax to SAAC's issuance (see Section 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 that the showing and verification algorithms are split into two parts: the key-dependent and public verification. 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., [BBDT16, CMZ14]), but not for some others [MBS+25] 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. The membership $(\mathsf{sk}, \mathsf{pk}) \in [\mathsf{KVAC.KeyGen}(\mathsf{par}_{\mathsf{KVAC}})]$ can be efficiently checked (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.

η-Correctness. For any $\lambda, \ell = \ell(\lambda) \in \mathbb{N}$, any global parameters $\mathsf{par}_g \in [\mathsf{Gen}(1^\lambda)]$, any KVAC public paramters $\mathsf{par}_{\mathsf{KVAC}} \in [\mathsf{KVAC.Setup}(1^\ell, \mathsf{par}_g)]$, any keys $(\mathsf{sk}, \mathsf{pk}) \in [\mathsf{KVAC.KeyGen}(\mathsf{par}_{\mathsf{KVAC}})]$, any $\boldsymbol{m} \in \mathcal{M}^\ell$, any $\phi, \phi' \in \Phi$ where $\phi(\boldsymbol{m}) = \phi'(\boldsymbol{m}) = 1$, and any $\mathsf{nonce} \in \{0,1\}^*$, the following experiment returns 1 with probability $1 - \eta(\lambda)$.

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

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

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

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

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

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

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

return win

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

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

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

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

if $\boldsymbol{m} = \perp \vee \phi(\boldsymbol{m}) = 0$ **then**
$\quad \mathsf{win} \leftarrow 1$ // $\mathcal{A}$ wins if it can request
$\quad$ // 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**
$\quad$ **return** $\perp$

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

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

return `closed`

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

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

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

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

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 $\sigma = \perp$ **then return** 1

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

return b'

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

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

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

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

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

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

if $\phi(\boldsymbol{m}) = 0 \vee \mathsf{sid} \in \mathcal{S} \vee \mathsf{sid} > \mathsf{sctr}$
$\quad$ **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. 10. 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.

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. Denote the distinguishing advantage of $\mathcal{A}$ as

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

$$\Pr[\mathcal{A}(\mathsf{par}_{\mathsf{KVAC}}) = 1 \,|\, \mathsf{par}_g \leftarrow\$ \mathsf{Gen}(1^\lambda); (\mathsf{par}_{\mathsf{KVAC}}, \mathsf{td}) \leftarrow\$ \mathsf{Ext}_{\mathsf{Setup}}(1^\ell, \mathsf{par}_g)\,]\,| \ .$$

2. The following advantage of $\mathcal{A}$ in the unforgeability game, defined in Figure 10 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 [CMZ14], 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 [Orr24], 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. , i.e., an adversary $\mathcal{A}$'s advantage is

$$\begin{aligned}\mathsf{Adv}^{\mathsf{par\text{-}indist}}_{\mathsf{KVAC},\mathsf{Sim}}(\mathcal{A}, \lambda) := \big|\Pr\big[\mathcal{A}(\mathsf{par}_{\mathsf{KVAC}}) = 1 \,\big|\, \mathsf{par}_g \leftarrow\$ \mathsf{Gen}(1^\lambda); \mathsf{par}_{\mathsf{KVAC}} \leftarrow\$ \mathsf{KVAC.Setup}(1^\ell, \mathsf{par}_g)\,\big] -\\ \Pr\big[\mathcal{A}(\mathsf{par}_{\mathsf{KVAC}}) = 1 \,\big|\, \mathsf{par}_g \leftarrow\$ \mathsf{Gen}(1^\lambda); (\mathsf{par}_{\mathsf{KVAC}}, \mathsf{td}) \leftarrow\$ \mathsf{Sim}_{\mathsf{Setup}}(1^\ell, \mathsf{par}_g)\,\big]\big| \ .\end{aligned}$$

2. No adversary can distinguish between interactions with an honest user and interactions with the simulator Sim. This property is defined via the anonymity game in Figure 10 with $\mathcal{A}$'s advantage defined as

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

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 [CMZ14], 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

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

Validity of key generation with respect to extractor 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.*

Remark 4.1. At a glance, integrity and validity of key generation, defined with respect to a simulator and an extractor, might seem strong. However, we view them as extensions of anonymity and unforgeability which allows composition with oNIP. Moreover, they are satisfied in our KVAC instantiations. This is because (1) our simulator and extractor generates public parameters that are identically distributed to honestly generated ones, (2) for integrity, the issuer needs to prove that it issued the credential correctly, so an honest user is then likely to get a valid credential allowing them to produce valid τ_{key}, and (3) the public key of these schemes fixes an underlying secret key, which immediately implies validity of key generation.

<u>Oblivious issuance of non-interactive proofs.</u> An oblivious issuance of non-interactive proofs $\mathsf{oNIP} = \mathsf{oNIP}[\mathsf{Gen}, \mathsf{R}]$ defined with respect to a global parameters generator 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*.*
- $(\perp, \pi) \leftarrow\!\!\$\ \langle \mathsf{oNIP.Iss}(\mathsf{par}_{\mathsf{oNIP}}, x, X) \rightleftharpoons \mathsf{oNIP.U}(\mathsf{par}_{\mathsf{oNIP}}, X, Y)\rangle$ 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 [OTZZ24] 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 an oNIP scheme to satisfy the following properties, but unlike [OTZZ24], unforgeability is not required for our generic construction.

Correctness. An oNIP scheme is η_{oNIP}-correct if for any $\lambda \in \mathbb{N}$ and parameters $\mathsf{par}_g \in [\mathsf{Gen}(1^\lambda)], \mathsf{par}_{\mathsf{oNIP}} \in [\mathsf{oNIP.Setup}(\mathsf{par}_g)]$, any $((X, Y), x) \in \mathsf{R}_{\mathsf{par}_g}$, the following experiment returns 1 with probability $1 - \eta_{\mathsf{oNIP}}(\lambda)$.

$$\begin{aligned}&(\perp, \pi) \leftarrow\!\!\$\ \langle \mathsf{oNIP.Iss}(\mathsf{par}_{\mathsf{oNIP}}, x, X) \rightleftharpoons \mathsf{oNIP.U}(\mathsf{par}_{\mathsf{oNIP}}, (X, Y))\rangle\\ &\textbf{return } \mathsf{oNIP.Ver}(\mathsf{par}_{\mathsf{oNIP}}, (X, Y), \pi)\end{aligned}$$

Soundness. Soundness is defined similarly to an NIZK where no adversary can output a statement (X, Y) and a proof π such that π verifies and $(X, Y) \notin \mathcal{L}_{\mathsf{R}}$. Denote the soundness advantage for $\mathcal{A}$ as

$$\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{oNIP}}(\mathcal{A}, \lambda) := \Pr\left[\begin{array}{l}(X, Y) \notin \mathcal{L}_{\mathsf{R}_{\mathsf{par}_g}} \wedge \\ \mathsf{oNIP.Ver}(\mathsf{par}_{\mathsf{oNIP}}, (X, Y), \pi) = 1\end{array}\middle|\begin{array}{l}\mathsf{par}_g \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)\\ \mathsf{par}_{\mathsf{oNIP}} \leftarrow\!\!\$\ \mathsf{oNIP.Setup}(\mathsf{par}_g)\\ (X, Y, \pi) \leftarrow\!\!\$\ \mathcal{A}(\mathsf{par}_{\mathsf{oNIP}})\end{array}\right].$$

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 Figure 11 is

$$\mathsf{Adv}^{\mathsf{zk}}_{\mathsf{oNIP},\mathsf{Sim},\mathcal{O}}(\mathcal{A}, \lambda) := |\Pr[\mathrm{ZK}^{\mathcal{A}}_{\mathsf{oNIP},\mathsf{Sim},\mathcal{O},0}(\lambda) = 1] - \Pr[\mathrm{ZK}^{\mathcal{A}}_{\mathsf{oNIP},\mathsf{Sim},\mathcal{O},1}(\lambda) = 1]|$$

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. Denote the advantage of $\mathcal{A}$ as

$$\begin{aligned}\mathsf{Adv}^{\mathsf{par\text{-}indist}}_{\mathsf{oNIP},\mathsf{Sim}}(\mathcal{A}, \lambda) :=& |\Pr[\mathcal{A}(\mathsf{par}_{\mathsf{oNIP}}) = 1 \,|\, \mathsf{par}_g \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda); \mathsf{par}_{\mathsf{oNIP}} \leftarrow\!\!\$\ \mathsf{oNIP.Setup}(\mathsf{par}_g)\,] -\\ &\Pr[\mathcal{A}(\mathsf{par}_{\mathsf{oNIP}}) = 1 \,|\, \mathsf{par}_g \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda); (\mathsf{par}_{\mathsf{oNIP}}, \mathsf{td}_{\mathsf{oNIP}}) \leftarrow\!\!\$\ \mathsf{Sim}_{\mathsf{Setup}}(\mathsf{par}_g)\,]|\,.\end{aligned}$$

Game $\mathrm{ZK}^{\mathcal{A}}_{\mathsf{oNIP},\mathsf{Sim},\mathcal{O},b}(\lambda)$:

$\mathtt{init} \leftarrow 0; \mathcal{I}_1, \ldots, \mathcal{I}_r \leftarrow \varnothing$

$\mathsf{par}_g \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)$

$\boxed{\mathsf{par}_{\mathsf{oNIP}} \leftarrow\!\!\$\ \mathsf{oNIP.Setup}(\mathsf{par}_g)}$ // $b = 0$

$\boxed{(\mathsf{par}_{\mathsf{oNIP}}, \mathsf{td}) \leftarrow\!\!\$\ \mathsf{Sim}_{\mathsf{Setup}}(\mathsf{par}_g)}$ // $b = 1$

$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{INIT},\mathrm{ISS}_1,\ldots,\mathrm{ISS}_r}(\mathsf{par}_{\mathsf{oNIP}})$

return b'

Oracle $\mathrm{INIT}(\tilde{X}, \tilde{x})$:

if $\mathtt{init} = 1 \ \lor\ (\tilde{X}, \tilde{x}) \notin \mathsf{Core}(\mathsf{R})$ **then**

  abort

$\mathtt{init} \leftarrow 1; X \leftarrow \tilde{X}; x \leftarrow \tilde{x}$

return $\mathtt{closed}$

Oracle $\mathrm{ISS}_j(\mathsf{sid}, \mathsf{umsg}_j)$: // $j = 1, \ldots, r$

if $\mathsf{sid} \notin \mathcal{I}_1, \ldots, \mathcal{I}_{j-1} \ \lor\ \mathsf{sid} \in \mathcal{I}_j \ \lor\ \mathtt{init} = 0$

  then abort

$\mathcal{I}_j \leftarrow \mathcal{I}_j \cup \{\mathsf{sid}\}$

if $j = 1$ **then**

  $\boxed{(\mathsf{hmsg}_1, \mathsf{st}_{\mathsf{sid}}) \leftarrow\!\!\$\ \mathsf{oNIP.Iss}_1(\mathsf{par}_{\mathsf{oNIP}}, \mathsf{sk}, \mathsf{umsg}_1)}$ // $b = 0$

  $\boxed{(\mathsf{hmsg}_1, \mathsf{st}_{\mathsf{sid}}) \leftarrow\!\!\$\ \mathsf{Sim}_{\mathrm{ISS}}^{\mathcal{O}(\mathsf{par}_g, x, X, \cdot)}(\mathsf{td}, X, \mathsf{umsg}_1)}$ // $b = 1$

else // For $j = r$, $\mathsf{st}_{\mathsf{sid}} = \bot$

  $\boxed{(\mathsf{hmsg}_j, \mathsf{st}_{\mathsf{sid}}) \leftarrow\!\!\$\ \mathsf{oNIP.Iss}_j(\mathsf{st}_{\mathsf{sid}}, \mathsf{umsg}_j)}$ // $b = 0$

  $\boxed{(\mathsf{hmsg}_j, \mathsf{st}_{\mathsf{sid}}) \leftarrow\!\!\$\ \mathsf{Sim}_{\mathrm{ISS}}^{\mathcal{O}(\mathsf{par}_g, x, X, \cdot)}(\mathsf{st}_{\mathsf{sid}}, \mathsf{umsg}_j)}$ // $b = 1$

return hmsg_j

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

$\mathtt{init} \leftarrow 0; \mathcal{I}_1, \ldots, \mathcal{I}_{r+1}, \mathcal{P} \leftarrow \varnothing$

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

$(\mathsf{par}_{\mathsf{oNIP}}, \mathsf{td}_{\mathsf{oNIP}}) \leftarrow\!\!\$\ \mathsf{Sim}_{\mathsf{Setup}}(\mathsf{par}_g)$

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

$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{INIT},\mathrm{U}_1,\ldots,\mathrm{U}_{r+1},\mathrm{Pf}}(\mathsf{par}_{\mathsf{oNIP}}, \mathsf{td}, \mathsf{st}_{\mathcal{A}})$

return b'

Oracle $\mathrm{INIT}(\tilde{X})$:

if $\mathtt{init} = 1$ **then abort**

$\mathtt{init} \leftarrow 1; X \leftarrow \tilde{X}$

return $\mathtt{closed}$

Oracle $\mathrm{Pf}(\mathsf{sid})$:

if $\mathsf{sid} \notin \mathcal{I}_1, \ldots, \mathcal{I}_{r+1} \ \lor\ \mathsf{sid} \in \mathcal{P}$

  then abort

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

$\boxed{\textbf{return } \pi_{\mathsf{sid}}}$ // $b = 0$

if $\pi_{\mathsf{sid}} \neq \bot$ **then**

  return $\pi \leftarrow\!\!\$\ \mathsf{Sim}_{\mathrm{Pf}}(\mathsf{td}, X, Y_{\mathsf{sid}})$

else abort // $b = 1$

Oracle $\mathrm{U}_1(\mathsf{sid}, Y_{\mathsf{sid}})$

if $\mathsf{sid} \in \mathcal{I}_1 \ \lor\ \mathtt{init} = 0 \ \lor\ (X, Y_{\mathsf{sid}}) \notin \mathcal{L}_{\mathsf{R}_{\mathsf{par}_g}}$ **then**

  abort

$\mathcal{I}_1 \leftarrow \mathcal{I}_1 \cup \{\mathsf{sid}\}$

$\boxed{(\mathsf{umsg}_1, \mathsf{st}_{\mathsf{sid}}) \leftarrow\!\!\$\ \mathsf{oNIP.U}_1(\mathsf{par}_{\mathsf{oNIP}}, X, Y_{\mathsf{sid}})}$ // $b = 0$

$\boxed{(\mathsf{umsg}_1, \mathsf{st}_{\mathsf{sid}}) \leftarrow\!\!\$\ \mathsf{Sim}_{\mathsf{U}}(\mathsf{td}, X)}$ // $b = 1$

return umsg_1

Oracle $\mathrm{U}_j(\mathsf{sid}, \mathsf{imsg}_j)$ // $j = 2, \ldots, r+1$

if $\mathsf{sid} \notin \mathcal{I}_1, \ldots, \mathcal{I}_{j-1} \ \lor\ \mathsf{sid} \in \mathcal{I}_j$

  then abort

$\mathcal{I}_j \leftarrow \mathcal{I}_j \cup \{\mathsf{sid}\}$

if $j < r + 1$ **then**

  $\boxed{(\mathsf{umsg}_j, \mathsf{st}_{\mathsf{sid}}) \leftarrow\!\!\$\ \mathsf{oNIP.U}_j(\mathsf{st}_{\mathsf{sid}}, \mathsf{imsg}_j)}$ // $b = 0$

  $\boxed{(\mathsf{umsg}_j, \mathsf{st}_{\mathsf{sid}}) \leftarrow\!\!\$\ \mathsf{Sim}_{\mathsf{U}}(\mathsf{st}_{\mathsf{sid}}, \mathsf{imsg}_j)}$ // $b = 1$

  return umsg_j

else

  $\boxed{\pi_{\mathsf{sid}} \leftarrow\!\!\$\ \mathsf{oNIP.U}_j(\mathsf{st}_{\mathsf{sid}}, \mathsf{imsg}_j)}$ // $b = 0$

  $\boxed{\pi_{\mathsf{sid}} \leftarrow\!\!\$\ \mathsf{Sim}_{\mathsf{U}}(\mathsf{st}_{\mathsf{sid}}, \mathsf{imsg}_j)}$ // $b = 1$

return $\mathtt{closed}$

Fig. 11. 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.

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 Figure 11 is defined as

$$\mathsf{Adv}^{\mathsf{oblv}}_{\mathsf{oNIP},\mathsf{Sim}_{\mathsf{Gen}},\mathsf{Sim}}(\mathcal{A}, \lambda) := |\Pr[\mathrm{OBLV}^{\mathcal{A}}_{\mathsf{oNIP},\mathsf{Sim}_{\mathsf{Gen}},\mathsf{Sim},0}(\lambda) = 1] - \Pr[\mathrm{OBLV}^{\mathcal{A}}_{\mathsf{oNIP},\mathsf{Sim}_{\mathsf{Gen}},\mathsf{Sim},1}(\lambda) = 1]| \,.$$

Our obliviousness definition is simulation-based instead of the definition in [OTZZ24]. Further, it only applies for statements in the language and not any statements. This is to achieve anonymity for our SAAC where the $\mathsf{SAAC.ObtHelp}$ algorithms in the game the helper protocol is simulated.

4.2 Construction

In this section, we construct a server-aided anonymous credential scheme $\mathsf{SAAC} = \mathsf{SAAC}[\mathsf{Gen}, \mathsf{KVAC}, \mathsf{oNIP}]$ for predicate family Φ and attribute space $\mathcal{M}$, using a $\mathsf{KVAC} = \mathsf{KVAC}[\mathsf{Gen}, \Phi, \mathcal{M}]$ scheme and an $\mathsf{oNIP} = \mathsf{oNIP}[\mathsf{Gen}, \mathsf{R_V}]$ protocol for the relation family $\mathsf{R_V}$ defined by the $\mathsf{KVAC.SVer_{key}}$ algorithm.

The high-level idea of our generic construction is to replace the key-dependent part $\mathsf{KVAC.SVer_{key}}$ of the keyed-verification credentials with oblivious proof issuance protocols. In particular, the key generation and issuance protocol remains that of the KVAC scheme, while the helper protocol starts by having the user runs $\mathsf{KVAC.Show_{key}}$ algorithm to obtain a state st and τ_{key} which is then used to run the oNIP protocol to produce a proof π_{V} of the statement $((\mathsf{par_{KVAC}}, \mathsf{pk}), \tau_{\mathsf{key}}) \in \mathcal{L}_{\mathsf{V},\mathsf{par}_g}$. To produce the showing message τ, the user would use the state to compute τ_{pub} by running $\mathsf{KVAC.Show_{pub}}$ with the specified predicate ϕ and the message $(\pi_{\mathsf{V}}, \mathsf{nonce})$. Then, the user returns $\tau = (\tau_{\mathsf{key}}, \tau_{\mathsf{pub}}, \pi_{\mathsf{V}})$. The generic construction of $\mathsf{SAAC} = \mathsf{SAAC}[\mathsf{Gen}, \mathsf{KVAC}, \mathsf{oNIP}]$ is given below.

Setup: $\mathsf{SAAC.Setup}(1^\lambda)$:

- Run $\mathsf{par}_g \leftarrow\$ \mathsf{Gen}(1^\lambda)$, $\mathsf{par_{KVAC}} \leftarrow\$ \mathsf{KVAC.Setup}(1^\ell, \mathsf{par}_g)$, and $\mathsf{par_{oNIP}} \leftarrow\$ \mathsf{oNIP.Setup}(\mathsf{par}_g)$
- Return $\mathsf{par} = (\mathsf{par_{KVAC}}, \mathsf{par_{oNIP}})$

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

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

- First, $\mathsf{SAAC.ObtHelp}$ runs $(\tau_{\mathsf{key}}, \mathsf{st}) \leftarrow\$ \mathsf{KVAC.Show_{key}}(\mathsf{par_{KVAC}}, \mathsf{pk}, \boldsymbol{m}, \sigma)$.
- Then, $\mathsf{SAAC.Helper}$ and $\mathsf{SAAC.ObtHelp}$ run the oNIP protocol $(\perp, \pi_{\mathsf{V}}) \leftarrow\$ \langle \mathsf{oNIP.Iss}(\mathsf{par_{oNIP}}, \mathsf{sk}, (\mathsf{par_{KVAC}}, \mathsf{pk})) \rightleftharpoons \mathsf{oNIP.U}(\mathsf{par_{oNIP}}, (\mathsf{par_{KVAC}}, \mathsf{pk}), \tau_{\mathsf{key}})\rangle$.
- Finally, $\mathsf{SAAC.ObtHelp}$ returns $\mathsf{aux} = (\tau_{\mathsf{key}}, \pi_{\mathsf{V}}, \mathsf{st})$.

Show: $\mathsf{SAAC.Show}(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}, \sigma, \mathsf{aux} = (\tau_{\mathsf{key}}, \pi_{\mathsf{V}}, \mathsf{st}), \phi, \mathsf{nonce})$:

- Compute $\tau_{\mathsf{pub}} \leftarrow\$ \mathsf{KVAC.Show_{pub}}(\mathsf{st}, \phi, (\pi_{\mathsf{V}}, \mathsf{nonce}))$
- Return $\pi = (\tau_{\mathsf{key}}, \tau_{\mathsf{pub}}, \pi_{\mathsf{V}})$

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

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

The following theorem then establishes the security properties of our generic SAAC construction.

Theorem 4.2. *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 $\mathsf{R_V}$ induced by $\mathsf{KVAC.SVer_{key}}$. 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 $\mathsf{Sim_{Gen}}, \mathsf{Sim_{oNIP}}, \mathsf{Sim_{KVAC}}$ such that oNIP is oblivious with respect to $\mathsf{Sim_{Gen}}$ and Sim_0, and KVAC satisfies anonymity and integrity with respect to $\mathsf{Sim_{Gen}}$ and $\mathsf{Sim_{KVAC}}$.*

Proof (of Theorem 4.2). Correctness easily follows from the correctness of the KVAC and the correctness of oNIP. In particular, if KVAC is η_{KVAC}-correct and oNIP is η_{oNIP}-correct, SAAC is η-correct for $\eta(\lambda) = \eta_{\mathsf{KVAC}}(\lambda) + \eta_{\mathsf{oNIP}}(\lambda)$ for all positive integers λ. Unforgeability and anonymity guarantees of SAAC, including the concrete security bounds, are stated in the two following lemmas, which are proved in Sections 4.3 and 4.4, respectively.

Lemma 4.3 (Unforgeability of SAAC). *Let $\mathcal{O}(\mathsf{par}_g, \mathsf{sk}, (\mathsf{par_{KVAC}}, \mathsf{pk}), \cdot)$ be an oracle, Sim be a simulator and Ext be an extractor, such that oNIP is $\mathcal{O}$-zero-knowledge with respect to Sim, and KVAC satisfies $\mathcal{O}$-unforgeability and validity of key generation with respect to Ext. There exists an extractor Ext' such that*

- *For any $\mathcal{A}$ running in time $t_\mathcal{A} = t_\mathcal{A}(\lambda)$, there exists an adversary $\mathcal{B}$ running in time roughly $t_\mathcal{A}$ such that*

$$\mathsf{Adv}^{\mathsf{par\text{-}indist}}_{\mathsf{SAAC},\mathsf{Ext}'}(\mathcal{A},\lambda) \leqslant \mathsf{Adv}^{\mathsf{par\text{-}indist}}_{\mathsf{KVAC},\mathsf{Ext}}(\mathcal{B},\lambda) \ .$$

- *For any $\mathcal{A}$ playing the game* UNF *of* SAAC, *running in time $t_\mathcal{A} = t_\mathcal{A}(\lambda)$, making at most $q_{\mathrm{Iss}}, q_{\mathrm{Help}}$ to* Iss *and* Help_1 *oracles (resp.), there exist adversaries $\mathcal{B}_{\mathsf{zk}}, \mathcal{B}_{\mathsf{sound}}, \mathcal{B}_{\mathsf{unf}}$, against the $\mathcal{O}$-zero-knowledge of* oNIP, *soundness of* oNIP, *and $\mathcal{O}$-unforgeability of* KVAC *(resp.), all running in time roughly $t_\mathcal{A}$ such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{unf}}_{\mathsf{SAAC},\mathsf{Ext}'}(\mathcal{A},\lambda) \leqslant{} & \mathsf{Adv}^{\mathsf{zk}}_{\mathsf{oNIP},\mathsf{Sim},\mathcal{O}}(\mathcal{B}_{\mathsf{zk}},\lambda) + \mathsf{Adv}^{\mathsf{sound}}_{\mathsf{oNIP}}(\mathcal{B}_{\mathsf{sound}},\lambda) \\ & + \mathsf{Adv}^{\mathsf{unf}}_{\mathsf{KVAC},\mathsf{Ext},\mathcal{O}}(\mathcal{B}_{\mathsf{unf}},\lambda) \ ,\end{aligned}$$

  Additionally, $\mathcal{B}_{\mathsf{zk}}$ starts at most q_{Help} sessions with the proof issuance oracle, and $\mathcal{B}_{\mathsf{unf}}$ makes at most q_{Iss} queries to its credential issuance oracle.

Unforgeability of our construction follows from $\mathcal{O}$-Unforgeability and validity of key-generation of KVAC and $\mathcal{O}$-Zero-Knowledge and soundness of oNIP. Note in particular that the oracle $\mathcal{O}$ needs to be the same for both security properties of KVAC and oNIP. At a high level, the proof would first apply soundness (along with validity of key-generation of KVAC) to restrict the forgery of the adversary to satisfy the keyed-verification algorithm $\mathsf{KVAC.SVer}$ with respect to the secret key sk that the game sampled. Then, we will simulate the helper protocol using the $\mathcal{O}$-Zero-Knowledge simulator. At this point, the game is still dependent on the secret key sk of the KVAC scheme, but only during the issuance protocol and to answer $\mathcal{O}$ queries from the simulator. This allows a simple reduction to $\mathcal{O}$-Unforgeability game of KVAC.

Lemma 4.4 (Anonymity of SAAC). *Let $\mathsf{Sim}_{\mathsf{Gen}}, \mathsf{Sim}_{\mathsf{oNIP}}$, and $\mathsf{Sim}_{\mathsf{KVAC}}$ be simulators 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 $\mathsf{Sim}_{\mathsf{KVAC}}$. Then, there exists a simulator Sim' such that*

- *For any $\mathcal{A}$ running in time $t_\mathcal{A} = t_\mathcal{A}(\lambda)$, there exists an adversary $\mathcal{B}_0, \mathcal{B}_1, \mathcal{B}_2$ running in time roughly $t_\mathcal{A}$ such that*

$$\mathsf{Adv}^{\mathsf{par\text{-}indist}}_{\mathsf{SAAC},\mathsf{Sim}}(\mathcal{A},\lambda) \leqslant \mathsf{Adv}^{\mathsf{par\text{-}indist}}_{\mathsf{oNIP},\mathsf{Sim}_{\mathsf{oNIP}}}(\mathcal{B}_0,\lambda) + \mathsf{Adv}^{\mathsf{par\text{-}indist}}_{\mathsf{KVAC},\mathsf{Sim}_{\mathsf{KVAC}}}(\mathcal{B}_1,\lambda) + \mathsf{Adv}^{\mathsf{par\text{-}indist}}_{\mathsf{Gen},\mathsf{Sim}_{\mathsf{Gen}}}(\mathcal{B}_2,\lambda) \ .$$

- *For any $\mathcal{A}$ playing the game* Anon *of* SAAC, *running in time $t_\mathcal{A} = t_\mathcal{A}(\lambda)$, making at most $q_{\mathrm{ObtH}}, q_{\mathrm{SH}}$ to* ObtH *and* SH *oracles (resp.), there exist adversaries $\mathcal{B}_{\mathsf{oblv}}, \mathcal{B}_{\mathsf{anon}}, \mathcal{B}_{\mathsf{integ}}$, against obliviousness of* oNIP, *anonymity of* KVAC, *and integrity of issued credentials of* KVAC *(resp.), all running in time roughly $t_\mathcal{A}$ such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{anon}}_{\mathsf{SAAC},\mathsf{Sim}'}(\mathcal{A},\lambda) \leqslant{} & \mathsf{Adv}^{\mathsf{oblv}}_{\mathsf{oNIP},\mathsf{Sim}_{\mathsf{Gen}},\mathsf{Sim}_{\mathsf{oNIP}}}(\mathcal{B}_{\mathsf{oblv}},\lambda) + \mathsf{Adv}^{\mathsf{anon}}_{\mathsf{KVAC},\mathsf{Sim}_{\mathsf{Gen}},\mathsf{Sim}_{\mathsf{KVAC}}}(\mathcal{B}_{\mathsf{anon}},\lambda) \\ & + q_{\mathrm{ObtH}} \cdot \mathsf{Adv}^{\mathsf{integ}}_{\mathsf{KVAC},\mathsf{Sim}_{\mathsf{Gen}},\mathsf{Sim}_{\mathsf{KVAC}}}(\mathcal{B}_{\mathsf{integ}},\lambda) \ ,\end{aligned}$$

  Additionally, $\mathcal{B}_{\mathsf{oNIP}}$ starts at most q_{ObtH} sessions with the user oracle of the obliviousness game, and $\mathcal{B}_{\mathsf{anon}}$ makes at most q_{SH} queries to its SH *oracle.*

Anonymity of our construction follows from anonymity and integrity of credential issuance of KVAC along with obliviousness of proofs for valid statements of oNIP. As a rough proof sketch, we first apply integrity of credential issuance to restrict τ_{key} so that the honest user generates to be a valid statement with high probability. Then, applying (a) obliviousness of oNIP for valid statements to simulate the user-side of the helper protocol and (b) anonymity of KVAC to simulate the issuance and showing concludes the proof. □

4.3 Proof of Lemma 4.3

We first give the description on the extractor Ext'.

- $\mathsf{Ext}'_{\mathsf{Setup}}(1^\lambda, 1^\ell)$: Run $\mathsf{par}_g \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)$, $(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{td}) \leftarrow\!\!\$\ \mathsf{Ext}_{\mathsf{Setup}}(1^\ell, \mathsf{par}_g)$ and $\mathsf{par}_{\mathsf{oNIP}} \leftarrow\!\!\$\ \mathsf{oNIP.Setup}(\mathsf{par}_g)$ and return $(\mathsf{par} = (\mathsf{par}_{\mathsf{KVAC}}, \mathsf{par}_{\mathsf{oNIP}}), \mathsf{td})$.

- $\mathsf{Ext}'_{\mathsf{Iss}}(\mathsf{td}, \mu, \phi)$: Run $\boldsymbol{m} \leftarrow \mathsf{Ext}_{\mathsf{Iss}}(\mathsf{td}, \mu, \phi)$ and return $\boldsymbol{m}$.

The public parameters sampled from $\mathsf{Ext}'_{\mathsf{Setup}}$ are indistinguishable from the one sampled from $\mathsf{SAAC.Setup}$. This is because $\mathsf{par}_{\mathsf{KVAC}}$ sampled from $\mathsf{Ext}_{\mathsf{Setup}}$ are indistinguishable from $\mathsf{KVAC.Setup}$, and the concrete bound follows easily.

Next, we want to show that no adversary can succeed in the unforgeability game. Hence, we consider an adversary $\mathcal{A}$ as described in the theorem statement. Now, we consider the following sequence of games.

Game $\mathbf{G}_0^{\mathcal{A}}(\lambda)$: This game is exactly the unforgeability game with respect to the extractor Ext'. The adversary $\mathcal{A}$ has access to a credential issuance oracle Iss, new user oracle NewUsr, showing oracle SH and the helper oracles $\mathrm{Help}_1, \ldots, \mathrm{Help}_r$. At the end of the game, it tries to output a valid forgery $(\phi^*, \mathsf{nonce}^*, \tau^* = (\tau^*_{\mathsf{key}}, \tau^*_{\mathsf{pub}}, \pi^*_{\mathsf{V}}))$. In particular, $\mathcal{A}$ succeeds if (a) the extractor fails or (b) $\phi^*(\boldsymbol{m}) = 0$ for all $\boldsymbol{m}$ extracted in the issuance oracle, $(\phi^*, \mathsf{nonce}^*, \tau^*)$ was not an output of the SH oracle, and

$$\mathsf{oNIP.Ver}(\mathsf{par}_{\mathsf{oNIP}}, ((\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}), \tau^*_{\mathsf{key}}), \pi^*_{\mathsf{V}}) = 1, \text{ and}$$
$$\mathsf{KVAC.SVer}_{\mathsf{pub}}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}, (\tau^*_{\mathsf{key}}, \tau^*_{\mathsf{pub}}), \phi^*, (\pi^*_{\mathsf{V}}, \mathsf{nonce}^*)) = 1 \ .$$

Game $\mathbf{G}_1^{\mathcal{A}}(\lambda)$: In this game, the simulation of the oracles are unchanged. However, the success event of the adversary $\mathcal{A}$ is now modified: in addition to checking the winning condition in $\mathbf{G}_0$, the game also checks that $\mathsf{KVAC.SVer}_{\mathsf{key}}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{sk}, (\tau^*_{\mathsf{key}}, \tau^*_{\mathsf{pub}}), \phi^*, (\pi^*_{\mathsf{V}}, \mathsf{nonce}^*)) = 1$. In particular, we can bound the the success probability of $\mathcal{A}$ in $\mathbf{G}_1$ as follows

$$\begin{aligned}
\Pr[\mathbf{G}_0^{\mathcal{A}}(\lambda) = 1] &= \Pr[\mathbf{G}_0^{\mathcal{A}}(\lambda) = 1 \ \wedge \ \mathsf{KVAC.SVer}_{\mathsf{key}}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{sk}, \mathsf{pk}, \tau^*_{\mathsf{key}}) = 0] \\
&\quad + \Pr[\mathbf{G}_0^{\mathcal{A}}(\lambda) = 1 \ \wedge \ \mathsf{KVAC.SVer}_{\mathsf{key}}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{sk}, \mathsf{pk}, \tau^*_{\mathsf{key}}) = 1] \\
&\leqslant \Pr[\mathsf{oNIP.Ver}(\mathsf{par}_{\mathsf{oNIP}}, ((\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}), \tau^*_{\mathsf{key}}), \pi^*_{\mathsf{V}}) = 1 \\
&\qquad \wedge \ \mathsf{KVAC.SVer}_{\mathsf{key}}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{sk}, \mathsf{pk}, \tau^*_{\mathsf{key}}) = 0] + \Pr[\mathbf{G}_1^{\mathcal{A}}(\lambda) = 1]
\end{aligned}$$

We will now analyze the first term on the right-hand side. By the validity of key generation property of KVAC, if $((\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}), \tau^*_{\mathsf{key}}) \in \mathcal{L}_{\mathsf{V}, \mathsf{par}_g}$, then $\mathsf{KVAC.SVer}_{\mathsf{key}}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{sk}, \mathsf{pk}, \tau^*_{\mathsf{key}}) = 1$. Hence, this particular event implies that the adversary outputs a valid π^*_{V} proof for a statement $((\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}), \tau^*_{\mathsf{key}})$ not in the language $\mathcal{L}_{\mathsf{V}, \mathsf{par}_g}$. Therefore, we can construct a reduction $\mathcal{B}_{\mathsf{sound}}$ breaking soundness of oNIP and running in time roughly $t_{\mathcal{A}}$ such that

$$\begin{aligned}
&\Pr[\mathsf{oNIP.Ver}(\mathsf{par}_{\mathsf{oNIP}}, ((\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}), \tau^*_{\mathsf{key}}), \pi^*_{\mathsf{V}}) = 1 \\
&\qquad \wedge \ \mathsf{KVAC.SVer}_{\mathsf{key}}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{sk}, \mathsf{pk}, \tau^*_{\mathsf{key}}) = 0] \leqslant \mathsf{Adv}^{\mathsf{sound}}_{\mathsf{oNIP}}(\mathcal{B}_{\mathsf{sound}}, \lambda) \ .
\end{aligned}$$

Game $\mathbf{G}_2^{\mathcal{A}}(\lambda)$: In this game, the simulation of the helper oracles are now done using the simulator Sim. In particular, (1) $\mathsf{par}_{\mathsf{oNIP}}$ is now generated with a trapdoor $\mathsf{td}_{\mathsf{oNIP}}$ using $\mathsf{Sim}_{\mathsf{Setup}}$ and (2) the helper oracle is run with $\mathsf{Sim}_{\mathsf{Iss}}$, which takes as input the trapdoor $\mathsf{td}_{\mathsf{oNIP}}$, the public key pk, and the protocol messages and has access to the oracle $\mathcal{O}(\mathsf{par}_g, \mathsf{sk}, (\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}), \cdot)$, and (3) the SH oracle now computes π_{V} by running the oNIP issuance protocol with the issuer replaced by the simulator $\mathsf{Sim}_{\mathsf{Iss}}$ as in the helper oracle. Note that since the game at this point still knows the secret key sk, it can simulate the oracle $\mathcal{O}$ efficiently to the simulator.

Then, we show the change in winning probability of $\mathcal{A}$ by giving a reduction $\mathcal{B}_{\mathsf{zk}}$ described as follows:

- Takes as input $\mathsf{par}_{\mathsf{oNIP}}$ (which implicitly contains par_g). Then, generate $(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{td}) \leftarrow_\$ \mathsf{Ext}_{\mathsf{Setup}}(1^{\ell}, \mathsf{par}_g)$.
- Generate the secret and public keys $(\mathsf{sk}, \mathsf{pk}) \leftarrow_\$ \mathsf{KVAC.KeyGen}(\mathsf{par}_{\mathsf{KVAC}})$ and call the Init using $(\mathsf{pk}, \mathsf{sk})$ as the witness and the partial statement. It then runs the adversary $\mathcal{A}$ on input $(\mathsf{par} = (\mathsf{par}_{\mathsf{KVAC}}, \mathsf{par}_{\mathsf{oNIP}}), \mathsf{pk})$.
- For credential issuance oracle Iss, it uses sk and td as in the game.
- For each query to helper oracle Help_j with session ID sid, the reduction forwards the user message to its proof issuance oracle Iss_j of the corresponding round and sid. The output from the issuance oracle is then the output of the helper oracle.
- For NewUsr oracle, it computes the credential σ using the secret key sk via the algorithm $\mathsf{KVAC.Iss}$.

- For SH oracle, it uses the credential σ_{cid} and the attributes $\boldsymbol{m}_{\mathsf{cid}}$ to compute τ_{key} and st via $\mathsf{KVAC.Show}_{\mathsf{key}}$. Then, it computes π_{V} by starting a new Iss session with its game while running the oNIP user-side algorithms with statement $((\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}), \tau_{\mathsf{key}})$ to obtain the proof π_{V}. Finally, it computes τ_{pub} via $\mathsf{KVAC.Show}_{\mathsf{pub}}(\mathsf{st}, \phi, (\pi_{\mathsf{V}}, M))$. Return $(\tau_{\mathsf{key}}, \tau_{\mathsf{pub}}, \pi_{\mathsf{V}})$ to the adversary.
- At the end of the game, the reduction checks, using its generated secret key sk, whether $\mathcal{A}$ wins the game, and if so it outputs 1. Otherwise, output 0.

We can easily see that if the ZK game uses an honest issuer, the view of $\mathcal{A}$ corresponds to its view in game $\mathbf{G}_1^{\mathcal{A}}(\lambda)$. Similarly, if the game uses a simulator, the view of $\mathcal{A}$ corresponds to its view in game $\mathbf{G}_2^{\mathcal{A}}(\lambda)$. Thus, proving that

$$|\Pr[\mathbf{G}_1^{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{G}_2^{\mathcal{A}}(\lambda) = 1]| \leqslant \mathsf{Adv}^{\mathsf{zk}}_{\mathsf{oNIP},\mathsf{Sim},\mathcal{O}}(\mathcal{B}_{\mathsf{zk}}, \lambda) .$$

Finally, we show that there exists an adversary $\mathcal{B}_{\mathsf{unf}}$ playing the unforgeability game of KVAC with respect to the extractor Ext and the oracle $\mathcal{O}$. In particular, $\mathcal{B}_{\mathsf{unf}}$ does the following

- It takes as input the public parameters $\mathsf{par}_{\mathsf{KVAC}}$ (containing par_g) and the public key pk, and samples $(\mathsf{par}_{\mathsf{oNIP}}, \mathsf{td}_{\mathsf{oNIP}}) \leftarrow\$ \mathsf{Sim}_{\mathsf{Setup}}(\mathsf{par}_g)$. It then runs $\mathcal{A}$ with $(\mathsf{par} = (\mathsf{par}_{\mathsf{KVAC}}, \mathsf{par}_{\mathsf{oNIP}}), \mathsf{pk})$. Note that $\mathcal{B}_{\mathsf{unf}}$ does not know the extraction trapdoor td.
- For credential issuance oracle, it forwards the input from $\mathcal{A}$ to its own issuance oracle.
- For the helper oracles, it runs the simulator $\mathsf{Sim}_{\mathsf{Iss}}$ using $\mathsf{td}_{\mathsf{oNIP}}$ and pk, and uses the access to oracle $\mathcal{O}$ to simulate the output of $\mathcal{O}$ without knowing sk.
- For NewUsr oracle, it forwards the query to the NewUsr oracle of its game.
- For SH oracle, it forwards the cid part of the query to the $\mathrm{SH}_{\mathsf{key}}$ oracle of its game, which returns $(\mathsf{sid}, \tau_{\mathsf{key}})$. Then, it computes the proof π_{V} as in $\mathbf{G}_2$. Then, it queries $\mathrm{SH}_{\mathsf{pub}}$ for τ_{pub} with input $(\mathsf{sid}, \phi, (\pi_{\mathsf{V}}, M))$, and returns $(\tau_{\mathsf{key}}, \tau_{\mathsf{pub}}, \pi_{\mathsf{V}})$.
- Finally, it outputs the forgery $(\phi^*, (\pi_{\mathsf{V}}^*, \mathsf{nonce}^*), (\tau_{\mathsf{key}}^*, \tau_{\mathsf{pub}}^*))$ returned from $\mathcal{A}$.

It is easy to see that the view of $\mathcal{A}$ within the reduction is identical to its view in $\mathbf{G}_2$. Now, if $\mathcal{A}$ wins in $\mathbf{G}_2$, then the extraction fails or we have that the forgery $(\phi^*, (\pi_{\mathsf{V}}^*, \mathsf{nonce}^*), (\tau_{\mathsf{key}}^*, \tau_{\mathsf{pub}}^*))$ does not correspond to any (ϕ, M, τ) tuples returned by the simulation of SH and ϕ^* is not satisfied by any extracted attributes during issuance. Therefore, in both cases $\mathcal{B}_{\mathsf{unf}}$ wins in the unforgeability game. Thus,

$$\Pr[\mathbf{G}_2^{\mathcal{A}}(\lambda) = 1] \leqslant \mathsf{Adv}^{\mathsf{unf}}_{\mathsf{KVAC},\mathsf{Ext},\mathcal{O}}(\mathcal{B}_{\mathsf{unf}}, \lambda) ,$$

concluding the proof for unforgeability. □

4.4 Proof of Lemma 4.4

We first give the description on the simulator Sim' which uses the simulators $\mathsf{Sim}_{\mathsf{Gen}}$, $\mathsf{Sim}_{\mathsf{oNIP}}$, and $\mathsf{Sim}_{\mathsf{KVAC}}$ as subroutines.

- $\mathsf{Sim}'_{\mathsf{Setup}}(1^\lambda, 1^\ell)$:
  - Run $(\mathsf{par}_g, \mathsf{td}_g) \leftarrow\$ \mathsf{Sim}_{\mathsf{Gen}}(1^\lambda)$
  - Run $(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{td}_{\mathsf{KVAC}}) \leftarrow\$ \mathsf{Sim}_{\mathsf{KVAC},\mathsf{Setup}}(1^\ell, \mathsf{par}_g)$ and $(\mathsf{par}_{\mathsf{oNIP}}, \mathsf{td}_{\mathsf{oNIP}}) \leftarrow\$ \mathsf{Sim}_{\mathsf{oNIP},\mathsf{Setup}}(\mathsf{par}_g)$
  - Return $(\mathsf{par} = (\mathsf{par}_{\mathsf{KVAC}}, \mathsf{par}_{\mathsf{oNIP}}), \mathsf{td} = (\mathsf{td}_g, \mathsf{td}_{\mathsf{KVAC}}, \mathsf{td}_{\mathsf{oNIP}}))$
- $\mathsf{Sim}'_{\mathsf{U}}$: This simulator runs $\mathsf{Sim}_{\mathsf{KVAC},\mathsf{U}}$ in both moves of the issuance protocol with inputs $\mathsf{td}_g, \mathsf{td}_{\mathsf{KVAC}}, \mathsf{pk}$ and the predicate ϕ.
- $\mathsf{Sim}'_{\mathsf{ObtH}}$: This simulator runs $\mathsf{Sim}_{\mathsf{oNIP},\mathsf{U}}$ in all rounds of the helper protocol with inputs $\mathsf{td}_g, \mathsf{td}_{\mathsf{oNIP}}$ and pk.
- $\mathsf{Sim}'_{\mathsf{Show}}(\mathsf{td}, \mathsf{pk}, \phi, M)$:
  - $(\tau_{\mathsf{key}}, \mathsf{st}) \leftarrow\$ \mathsf{Sim}_{\mathsf{KVAC},\mathsf{Show}}(\text{“}\mathsf{key}\text{”}, (\mathsf{td}_g, \mathsf{td}_{\mathsf{KVAC}}), \mathsf{pk})$
  - $\pi_{\mathsf{V}} \leftarrow\$ \mathsf{Sim}_{\mathsf{oNIP},\mathrm{Pf}}((\mathsf{td}_g, \mathsf{td}_{\mathsf{oNIP}}), \mathsf{pk}, \tau_{\mathsf{key}})$
  - $\tau_{\mathsf{pub}} \leftarrow\$ \mathsf{Sim}_{\mathsf{KVAC},\mathsf{Show}}(\text{“}\mathsf{pub}\text{”}, \mathsf{st}, \phi, (\pi_{\mathsf{V}}, M))$
  - Return $\tau = (\tau_{\mathsf{key}}, \tau_{\mathsf{pub}}, \pi_{\mathsf{V}})$

It is easy to see that the public parameters from Sim' are indistinguishable from $\mathsf{SAAC.Setup}$, and this follows from the indistinguishability of public parameters for $\mathsf{Sim}_{\mathsf{Gen}}$, $\mathsf{Sim}_{\mathsf{oNIP,Setup}}$ and $\mathsf{Sim}_{\mathsf{KVAC,Setup}}$.

Now, we need to show that no adversary $\mathcal{A}$ can distinguish between the game $\mathrm{Anon}_{\mathsf{SAAC,Sim'},0}$ and $\mathrm{Anon}_{\mathsf{SAAC,Sim'},1}$, and we do so by considering the following sequence of games.

Game $\mathbf{G}_0^{\mathcal{A}}(\lambda)$: This game is exactly the game $\mathrm{Anon}_{\mathsf{SAAC,Sim'},0}$ where $\mathcal{A}$ is interacting with honest users in all oracles. In particular, $\mathcal{A}$ interacts with the following:

- During the issuance phase, the game runs the user algorithms $\mathsf{SAAC.U}_1, \mathsf{SAAC.U}_2$.
- For the ObtH_j oracles, the game runs the user side of the protocol $\mathsf{SAAC.ObtHelp}_j$. Specifically, note that the first move of the user $\mathsf{SAAC.ObtHelp}_1$ involves computing τ_{key} and st using $\mathsf{KVAC.Show}_{\mathsf{key}}$ using the attributes $\boldsymbol{m}$ and the credential σ. At the end of sessions sid, the game obtains $\mathsf{aux}_{\mathsf{sid}}$, which contains $\tau_{\mathsf{key}}, \mathsf{st}$ (computed in the first move), π_{V} (obtained as a result of oNIP protocol), and
- For the SH oracle, $\mathcal{A}$ specifies an sid such that the game would run the $\mathsf{SAAC.Show}$ algorithm using $\mathsf{aux}_{\mathsf{sid}}$ obtained from the helper protocol in session sid.

Game $\mathbf{G}_1^{\mathcal{A}}(\lambda)$: In this game, the ObtH_j for $j \in [r+1]$ is now run using $\mathsf{Sim}'_{\mathsf{ObtH}}$, and the π_{V} part in SH is now computed using $\mathsf{Sim}_{\mathsf{oNIP},\mathrm{Pf}}$. More precisely, the simulation of the following oracles are modified.

- Oracle ObtH_j: The game runs the simulator $\mathsf{Sim}_{\mathsf{oNIP,U}}$ for oNIP in all moves of the helper protocol. Note that in the first move, the game does not compute τ_{key} and st using $\mathsf{KVAC.Show}_{\mathsf{key}}$ anymore, since the simulator $\mathsf{Sim}_{\mathsf{oNIP,U}}$ does not depend on the statement τ_{key}.
- $\mathrm{SH}(\mathsf{sid}, \phi, M)$: The game now computes $\tau = (\tau_{\mathsf{key}}, \tau_{\mathsf{pub}}, \pi_{\mathsf{V}})$ by
  - Running $(\tau_{\mathsf{key}}, \mathsf{st}) \leftarrow\!\!\$\ \mathsf{KVAC.Show}_{\mathsf{key}}(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}, \sigma)$,
  - Simulating $\pi_{\mathsf{V}} \leftarrow\!\!\$\ \mathsf{Sim}_{\mathsf{oNIP},\mathrm{Pf}}((\mathsf{td}_g, \mathsf{td}_{\mathsf{oNIP}}), \mathsf{pk}, \tau_{\mathsf{key}})$, and
  - Computing $\tau_{\mathsf{pub}} \leftarrow\!\!\$\ \mathsf{KVAC.Show}_{\mathsf{pub}}(\mathsf{st}, \phi, (\pi_{\mathsf{V}}, M))$.

Now, we show the change in the probability that $\mathcal{A}$ outputs 1 from $\mathbf{G}_0$ to $\mathbf{G}_1$. First, we consider the event Bad that there exists a session sid such that $((\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}), \tau_{\mathsf{key,sid}}) \notin \mathcal{L}_{\mathsf{V},\mathsf{par}_g}$. Then, denote $\mathsf{Pr}_i[\mathsf{Bad}]$ as the probability that Bad occurs in $\mathbf{G}_i$ for $i \in \{0,1\}$. But notice that this event only depends on the public parameters par, the trapdoor td, the public key pk, the issued credential σ, and the random coins of the $\mathsf{KVAC.Show}_{\mathsf{key}}$ algorithm. These are all independent of whether $\mathsf{Sim}_{\mathsf{oNIP}}$ is used in the helper oracles or not. Hence, $\mathsf{Pr}_0[\mathsf{Bad}] = \mathsf{Pr}_1[\mathsf{Bad}]$. Also, we have that

$$\mathsf{Pr}[\mathbf{G}_b^{\mathcal{A}}(\lambda) = 1] = \mathsf{Pr}[\mathbf{G}_b^{\mathcal{A}}(\lambda) = 1 | \mathsf{Bad}]\mathsf{Pr}_0[\mathsf{Bad}] + \mathsf{Pr}[\mathbf{G}_b^{\mathcal{A}}(\lambda) = 1 \ \wedge\ \neg\mathsf{Bad}] \ .$$

Then,

$$\begin{aligned}
&|\mathsf{Pr}[\mathbf{G}_1^{\mathcal{A}}(\lambda) = 1] - \mathsf{Pr}[\mathbf{G}_0^{\mathcal{A}}(\lambda) = 1]| \\
&\leqslant \mathsf{Pr}_0[\mathsf{Bad}] + |\mathsf{Pr}[\mathbf{G}_1^{\mathcal{A}}(\lambda) = 1 \ \wedge\ \neg\mathsf{Bad}] - \mathsf{Pr}[\mathbf{G}_0^{\mathcal{A}}(\lambda) = 1 \ \wedge\ \neg\mathsf{Bad}]| \ .
\end{aligned}$$

We will then bound the two terms above separately: (1) $\mathsf{Pr}_0[\mathsf{Bad}]$ will be bounded via a reduction $\mathcal{B}_{\mathsf{integ}}$ to the integrity property of KVAC, and (2) the second term will be bounded using a reduction $\mathcal{B}_{\mathsf{oblv}}$ to the obliviousness property of oNIP.

For $\mathsf{Pr}_0[\mathsf{Bad}]$, notice that

$$\mathsf{Pr}_0[\mathsf{Bad}] \leqslant q_{\mathrm{ObtH}} \mathsf{Pr}\left[\begin{array}{l|l} \sigma \neq \perp \wedge & (\mathsf{par}, \mathsf{td}) \leftarrow\!\!\$\ \mathsf{Sim}'_{\mathsf{Setup}}(1^\lambda, 1^\ell) \\ (\mathsf{pk}, \tau_{\mathsf{key}}) \notin \mathcal{L}_{\mathsf{V}} & (\mathsf{pk}, \boldsymbol{m}, \phi, \mathsf{st}) \leftarrow\!\!\$\ \mathcal{A}(\mathsf{par}, \mathsf{td}) \\ & \text{If } \phi(\boldsymbol{m}) = 0, \text{abort} \\ & (\perp, \sigma) \leftarrow\!\!\$\ \langle \mathcal{A}(\mathsf{st}) \rightleftharpoons \mathsf{KVAC.U}(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}, \phi) \rangle \\ & (\tau_{\mathsf{key}}, \mathsf{st}) \leftarrow\!\!\$\ \mathsf{KVAC.Show}_{\mathsf{key}}(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}, \sigma) \end{array}\right] .$$

Thus, there exists a reduction $\mathcal{B}_{\mathsf{integ}}$ such that $\mathsf{Pr}_0[\mathsf{Bad}] \leqslant q_{\mathrm{ObtH}} \cdot \mathsf{Adv}^{\mathsf{integ}}_{\mathsf{KVAC},\mathsf{Sim}_{\mathsf{KVAC}}}(\mathcal{B}_{\mathsf{integ}}, \lambda)$.

Next, consider the following reduction $\mathcal{B}_{\mathsf{oblv}}$ playing the game OBLV of oNIP:

- The reduction takes as input $\mathsf{par}_{\mathsf{oNIP}}, \mathsf{td}_g, \mathsf{td}_{\mathsf{oNIP}}$ and samples $(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{td}_{\mathsf{KVAC}}) \leftarrow\!\!\$\ \mathsf{Sim}_{\mathsf{KVAC,Setup}}(1^\ell, \mathsf{par}_g)$. (Again par_g is contained in $\mathsf{par}_{\mathsf{oNIP}}$).
- Then, the reduction runs $\mathcal{A}$ with $(\mathsf{par} = (\mathsf{par}_{\mathsf{KVAC}}, \mathsf{par}_{\mathsf{oNIP}}), \mathsf{td} = (\mathsf{td}_g, \mathsf{td}_{\mathsf{KVAC}}, \mathsf{td}_{\mathsf{oNIP}}))$, who outputs $(\mathsf{pk}, \boldsymbol{m}, \phi)$.
- The issuance protocol is run using the user algorithm SAAC.U and the reduction obtains a credential σ of attributes $\boldsymbol{m}$.
- For the ObtH oracles in sessions sid, the reduction runs $(\tau_{\mathsf{key,sid}}, \mathsf{st}_{\mathsf{sid}}) \leftarrow\!\!\$\ \mathsf{KVAC.Show}_{\mathsf{key}}(\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}, \boldsymbol{m}, \sigma)$, and opens a new oNIP protocol session sid using the statement $((\mathsf{par}_{\mathsf{KVAC}}, \mathsf{pk}), \tau_{\mathsf{key,sid}})$ with its OBLV game. If $(\mathsf{pk}, \tau_{\mathsf{key,sid}}) \notin \mathcal{L}_{\mathsf{V},\mathsf{par}_g}$, the game will return $\perp$ and the reduction would simply return a random guess $b' \leftarrow\!\!\$\ \{0,1\}$. Otherwise, it would forward the protocol messages back and forth between $\mathcal{A}$ and the OBLV game.
- For the SH oracle on input (sid, ϕ, M), the reduction first queries the Pf oracle with sid to get π_{V}. Then, it computes $\tau_{\mathsf{pub}} \leftarrow\!\!\$\ \mathsf{KVAC.Show}_{\mathsf{pub}}(\mathsf{st}_{\mathsf{sid}}, \phi, M)$ and returns $(\tau_{\mathsf{key,sid}}, \tau_{\mathsf{pub}}, \pi_{\mathsf{V}})$ to $\mathcal{A}$.
- Finally, it forwards the guess b' from $\mathcal{A}$ to its game.

Here, it is easy to see that if Bad occurs the probability that the reduction outputs 1 is 1/2 in both cases of OBLV game. Also, when Bad does not occur, the views of $\mathcal{A}$ within the reduction when OBLV is run with honest user and the simulator $\mathsf{Sim}_{\mathsf{oNIP}}$ are identical to its view in $\mathbf{G}_0$ and $\mathbf{G}_1$, respectively. Therefore,

$$|\Pr[\mathbf{G}_1^{\mathcal{A}}(\lambda) = 1 \ \wedge \ \neg\mathsf{Bad}] - \Pr[\mathbf{G}_0^{\mathcal{A}}(\lambda) = 1 \ \wedge \ \neg\mathsf{Bad}]| \leqslant \mathsf{Adv}^{\mathsf{oblv}}_{\mathsf{oNIP},\mathsf{Sim}_{\mathsf{oNIP}}}(\mathcal{B}_{\mathsf{integ}}, \lambda)\ .$$

Hence, $|\Pr[\mathbf{G}_1^{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{G}_0^{\mathcal{A}}(\lambda) = 1]| \leqslant \mathsf{Adv}^{\mathsf{oblv}}_{\mathsf{oNIP},\mathsf{Sim}_{\mathsf{oNIP}}}(\mathcal{B}_{\mathsf{integ}}, \lambda) + \mathsf{Adv}^{\mathsf{integ}}_{\mathsf{KVAC},\mathsf{Sim}_{\mathsf{KVAC}}}(\mathcal{B}_{\mathsf{integ}}, \lambda)$.

Game $\mathbf{G}_2^{\mathcal{A}}(\lambda)$: This game is exactly the game $\mathrm{Anon}_{\mathsf{SAAC},\mathsf{Sim}',1}$ where $\mathcal{A}$ is interacting with the simulator Sim' in all steps of the game. To show the change in probability that $\mathcal{A}$ returns 1, we can construct a reduction $\mathcal{B}_{\mathsf{anon}}$ to the anonymity game of KVAC. In particular, the reduction does the following:

1. To simulate the issuance protocol, it forwards the issuance protocol message and sends the outputs back to $\mathcal{A}$.
2. To simulate the helper protocol in $\mathrm{ObtH}_1, \ldots, \mathrm{ObtH}_{r+1}$, it runs $\mathsf{Sim}_{\mathsf{oNIP,U}}$.
3. to simulate SH queries of the form (sid, ϕ, M), it first queries $\mathrm{SH}_{\mathsf{key}}$ to get τ_{key}. Then, it uses $\mathsf{Sim}_{\mathsf{oNIP},\mathrm{Pf}}$ to compute the proof π_{V} for τ_{key}. Finally, it queries $\mathrm{SH}_{\mathsf{pub}}$ with $(\phi, (\pi_{\mathsf{V}}, M))$ to get τ_{pub} and returns $(\tau_{\mathsf{key}}, \tau_{\mathsf{pub}}, \pi_{\mathsf{V}})$.
4. It will then return the guess b' that $\mathcal{A}$ outputs.

Hence, the view of $\mathcal{A}$ corresponds to $\mathbf{G}_1$ and $\mathbf{G}_2$ when the Anon game is run with honest user and $\mathsf{Sim}_{\mathsf{KVAC}}$, respectively. Therefore,

$$|\Pr[\mathbf{G}_2^{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{G}_1^{\mathcal{A}}(\lambda) = 1]| \leqslant \mathsf{Adv}^{\mathsf{anon}}_{\mathsf{KVAC},\mathsf{Sim}_{\mathsf{KVAC}}}(\mathcal{B}_{\mathsf{anon}}, \lambda)\ ,$$

concluding the proof. □

5 Instantiation from BBS

In this section, we instantiate our generic SAAC construction with a KVAC based on the BBS MAC and a corresponding oNIP. We introduce the BBS MAC in Section 5.1, the KVAC in Section 5.2, and the oNIP in Section 5.3. Also, we discuss the final instantiation in Section 5.4.

<u>Global parameters generator.</u> Following the syntax in Section 4.1, we note that the *global parameters generator* for this instantiation is exactly the group generator GGen and the corresponding simulator $\mathsf{Sim}_{\mathsf{Gen}}(1^\lambda)$ simply outputs $\mathsf{par}_g = (p, G, \mathbb{G}) \leftarrow\!\!\$\ \mathsf{GGen}(1^\lambda)$ and does not output any trapdoor.

Algorithm $\mathsf{MAC}_{\mathsf{BBS}}.\mathsf{Setup}(1^\lambda)$:	Algorithm $\mathsf{MAC}_{\mathsf{BBS}}.\mathsf{M}(\mathsf{par}, \mathsf{sk} = x, \boldsymbol{m} \in \mathbb{Z}_p^\ell)$:
$(p, G, \mathbb{G}) \leftarrow\$ \mathsf{GGen}(1^\lambda); \boldsymbol{H} \leftarrow\$ \mathbb{G}^\ell$ $\mathsf{par} \leftarrow (p, G, \boldsymbol{H}, \mathbb{G})$ **return** par Algorithm $\mathsf{MAC}_{\mathsf{BBS}}.\mathsf{KG}(\mathsf{par})$: $x \leftarrow\$ \mathbb{Z}_p$ **return** $(\mathsf{sk} \leftarrow x, \mathsf{ipk} \leftarrow xG)$	$e \leftarrow\$ \mathbb{Z}_p; A \leftarrow (x+e)^{-1}(G + \sum_{i=1}^{\ell} \boldsymbol{m}[i]\boldsymbol{H}[i])$ **return** (A, e) Algorithm $\mathsf{MAC}_{\mathsf{BBS}}.\mathsf{Ver}(\mathsf{par}, \mathsf{sk}, \boldsymbol{m}, \sigma = (A, e))$: $C \leftarrow G + \sum_{i=1}^{\ell} \boldsymbol{m}[i]\boldsymbol{H}[i]$ **return** $((x+e)A = C)$

Fig. 12. Message Authentication Code from BBS Signatures.

5.1 BBS-based MAC

In this section, we give the $\mathsf{MAC}_{\mathsf{BBS}}$ scheme in Figure 12. The MAC tag for message $\boldsymbol{m} = (m_i)_{i=1}^{\ell}$ is computed as $(A := (x+e)^{-1}C, e \leftarrow\$ \mathbb{Z}_p)$ where $x \in \mathbb{Z}_p$ is the secret key, $C = G + \sum_{i=1}^{\ell} m_i H_i$, and $H_1 \ldots, H_\ell \in \mathbb{G}$ are parts of the public parameters. This scheme is similar to the one presented in Orrú's paper [Orr24], and Barki et al. [BBDT16] considered a variant of this scheme where the tag also includes a random scalar $s \in \mathbb{Z}_p$. The following theorem then establishes the unforgeability of $\mathsf{MAC}_{\mathsf{BBS}}$ in standard model.

Theorem 5.1 (Unforgeability of $\mathsf{MAC}_{\mathsf{BBS}}$). *Let* GGen *be a group generator that outputs groups of prime order* $p = p(\lambda)$*, and let* $\mathsf{MAC}_{\mathsf{BBS}} = \mathsf{MAC}_{\mathsf{BBS}}[\mathsf{GGen}]$*. For any adversary* $\mathcal{A}$ *playing the* rDDH-UFCMA *game of* $\mathsf{MAC}_{\mathsf{BBS}}$ *making at most* $q = q(\lambda)$ *queries to* MAC *and running in time* $t_{\mathcal{A}} = t_{\mathcal{A}}(\lambda)$*, there exist adversaries* $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$ *running in time roughly* $t_{\mathcal{A}}$ *such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{ufcma}}_{\mathsf{MAC}_{\mathsf{BBS}},\mathrm{rDDH}}(\mathcal{A}, \lambda) \leqslant q \cdot \mathsf{Adv}^{q\text{-SDH}}_{\mathsf{GGen},\mathrm{rDDH}}(\mathcal{B}_1, \lambda) + \mathsf{Adv}^{\mathrm{dlog}}_{\mathsf{GGen}}(\mathcal{B}_2, \lambda) \\ + \mathsf{Adv}^{q\text{-SDH}}_{\mathsf{GGen},\mathrm{rDDH}}(\mathcal{B}_3, \lambda) + \frac{q^2}{2p} + \frac{q+2}{p}\end{aligned}$$

Moreover, the same holds for UFCMA *without any of the* rDDH *oracles appearing anywhere in the statement.*

Proof. The result follows from a minor adaptation of Tessaro and Zhu's proof of security for BBS [TZ23a, Proof of Theorem 1]. For plain UFCMA, their proof does not rely on pairings and thus easily transfers to the MAC setting. For rDDH-UFCMA, it suffices to show that oracles V and rDDH can be simulated by the reduction. For verification, we use the fact that for $C = G + \sum_{i=1}^{\ell} m_i H_i$ we have $(x+e)^{-1}C$ if and only if $(G, xG, A, C - eA)$ is a DDH quadruple. This enables us to simulate verification with a restricted DDH oracle instead of knowledge of the secret key x. More precisely, the Tessaro-Zhu BBS SUF proof consists of three reductions:

1. Two reductions to q-SDH which simulate the game to $\mathcal{A}$ by signing using the secret key x from the q-SDH challenge.
2. A reduction to q-DL which simulates the game by signing using a randomly sampled secret key known to the reduction.

The first two reductions can simulate the verification oracle with their restricted DDH oracle as discussed, and obviously can simulate the restricted DDH oracle by passing queries to their restricted DDH oracle. The final reduction knows the secret key so verification and the restricted DDH oracle can be simulated canonically. □

Remark 5.2. Theorem 5.1 cannot be found in prior work, although similar results have been shown: Barki et al. showed that $\mathsf{MAC}_{\mathsf{BBS}+}$ is UFCMVA under the assumption that q-SDH is hard with a (unrestricted) DDH oracle [BBDT16]. Orrú proved that the slightly more efficient $\mathsf{MAC}_{\mathsf{BBS}}$ still achieves UFCMVA under the q-DL assumption in the AGM.

5.2 BBS-based KVAC

We first describe the $\mathsf{KVAC_{BBS}}$ scheme in Figure 13, which can be seen as a variant of the KVAC from [BBDT16]. The blind issuance starts by the user computing a Pedersen commitment $C = \sum_{i=1}^{\ell} m_i H_i + sH_{\ell+1}$ of its attributes $\boldsymbol{m}$ with randomness s, and the issuer signing this commitment by computing the $\mathsf{MAC_{BBS}}$ tag (A, e) where $A = (x + e)^{-1}(G + C)$. The credential for attributes $\boldsymbol{m}$ is then (A, e, s). We note that in contrast to [BBDT16], our issuer does not rerandomize the scalar s and thus saving one scalar in issuer's communication. 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 [CDL16]), and the issuer can check that $x\tilde{A} = \tilde{B}$.

<u>RELEVANT PROOF SYSTEMS.</u> 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:

$$\begin{aligned}
\mathsf{R_{com}} &:= \{((\boldsymbol{H}, C, \psi), (s, \boldsymbol{m})) : C = sH_{\ell+1} + \textstyle\sum_{i=1}^{\ell} m_i H_i \wedge \psi(\boldsymbol{m}) = 1\} \\
\mathsf{R_\sigma} &:= \{((X, A, B), x) : xG = X \wedge xA = B\} \\
\mathsf{R_{pub}} &:= \left\{ ((\tilde{A}, \tilde{B}, \tilde{C}, \boldsymbol{H}_{\mathrm{priv}}, Y), (e, r', r'', \hat{\boldsymbol{m}}, s) : \begin{array}{c} r''\tilde{C} + \langle \boldsymbol{H}_{\mathrm{priv}}, (\hat{\boldsymbol{m}} \| s) \rangle = Y \wedge \\ \tilde{B} = r'\tilde{C} - e\tilde{A} \end{array} \right\} .
\end{aligned}$$

The first proof system Π_{com} is used for the user to prove knowledge of openings to the commitment C during issuance. We require Π_{com} to be straightline-extractable for the relaxed relation $\widetilde{\mathsf{R}}_{\mathsf{com}}$ defined as

$$\widetilde{\mathsf{R}}_{\mathsf{com}} := \left\{ ((\boldsymbol{H}, C, \psi), (s, \boldsymbol{m})) : \begin{array}{c} (0_{\mathbb{G}} = \sum_{i=1}^{\ell} m_i H_i + sH_{\ell+1} \wedge \\ (s \| \boldsymbol{m}) \neq \boldsymbol{0}) \vee \\ ((\boldsymbol{H}, C, \psi), (s, \boldsymbol{m})) \in \mathsf{R_{com}} \end{array} \right\} ,$$

and it is instantiated using a variant of the Fischlin transform [Fis05, Ks22], which we describe in Appendix C. The proof systems Π_σ and Π_{pub} are used for proving validity of the issued credentials by the issuer and showing the credentials by the users, respectively. These proof systems are instantiated using the proof system Lin for linear relations on $\mathbb{G}$ (described in Section 2), with the corresponding linear maps for the relations R_σ and $\mathsf{R_{pub}}$ defined as follows:

$$M^{\sigma}_{G,A} := \begin{pmatrix} G \\ A \end{pmatrix}, \qquad M^{\mathsf{pub}}_{\tilde{C}, H_{\mathrm{priv},1}, \ldots, H_{\mathrm{priv},k}, \tilde{A}} := \begin{pmatrix} \tilde{C} & H_{\mathrm{priv},1} & \cdots & H_{\mathrm{priv},k} & 0 & 0 \\ 0 & 0 & \cdots & 0 & \tilde{C} & -\tilde{A} \end{pmatrix} .$$

We further note that *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.

<u>KEY-DEPENDENT VERIFICATION INDUCED-RELATION.</u> We point out that $\mathsf{SVer_{key}}$ 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.

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

Note that the augmented statement is $(\tilde{A}, \tilde{B})$ while the core relation $\mathsf{Core}(\mathsf{R_{dleq}})$ contains public-secret key pairs $(X = xG, x)$ defined by the key generation of $\mathsf{KVAC_{BBS}}$. We further note that checking if an augmented statement $(\tilde{A}, \tilde{B})$ is in the language can be done via the rDDH oracle, described in Figure 2.

<u>CORRECTNESS.</u> Correctness of $\mathsf{KVAC_{BBS}}$ follows from η-correctness of Π_{com}, perfect correctness of Π_σ and Π_{pub}, and that the honest user aborts with probability $1/p$ if the commitment $C = \sum_{i=1}^{\ell} m_i H_i + sH_{\ell+1} = -G$. In particular, the correctness error of the scheme is $\eta(\lambda) + \frac{1}{p}$.

<u>UNFORGEABILITY.</u> Unforgeability of $\mathsf{KVAC_{BBS}}$ against adversaries with access to the rDDH oracle, established in the following lemma, mainly follows from online-extractability of Π_{com} and a reduction to rDDH-UFCMA security of $\mathsf{MAC_{BBS}}$. Crucially, the reduction needs to (1) simulate the honest user showings and (2) rewind the adversary to extract a $\mathsf{MAC_{BBS}}$ forgery. To this end, our analysis, despite relying on standard techniques, is non-trivial, and we refer to Section 5.5 for the formal proof.

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

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 $(\tilde{A}, \tilde{B}) \leftarrow \tau_{\mathsf{key}}; (\tilde{C}, \pi_{\mathsf{pub}}) \leftarrow \tau_{\mathsf{pub}}$

$\boldsymbol{H}_{\mathrm{priv}} \leftarrow (H_i)_{i \in [\ell+1] \setminus \boldsymbol{I}}$

$Y \leftarrow G + \langle \boldsymbol{m}', (H_i)_{i \in \boldsymbol{I}} \rangle$

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. 13. 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, 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}.

Lemma 5.3. *Let* GGen *be a group generator that outputs groups of prime order* $p = p(\lambda)$, $\mathsf{Ext}_{\mathsf{com}}$ *be an extractor for the knowledge-soundness* Π_{com}, *and* Sim_σ *be a zero-knowledge simulator for* Π_σ. *Define* $\mathsf{Ext}_{\mathsf{BBS}} := (\mathsf{Ext}_{\mathsf{Setup}}, \mathsf{Ext}_{\mathsf{iss}})$ *as follows:*

- $\mathsf{Ext}_{\mathsf{Setup}}$ *on input* par_g *generates* $\boldsymbol{H}$ *as in* $\mathsf{KVAC}_{\mathsf{BBS}}.\mathsf{Setup}$ *and does not output any trapdoor.*
- $\mathsf{Ext}_{\mathsf{iss}}$ *on input* $(\mu = (C, \pi_{\mathsf{com}}), \psi)$ *outputs* $\boldsymbol{m} \leftarrow\!\!\$\ \mathsf{Ext}^{\mathsf{H}_0}_{\mathsf{com}}(\mathcal{Q}, (\boldsymbol{H}, C, \psi), \pi_{\mathsf{com}})$ *where* $\mathcal{Q}$ *is the set of* H_0 *queries the adversary has made so far.*

Then,

- *For any adversary* $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{par\text{-}indist}}_{\mathsf{KVAC}_{\mathsf{BBS}}, \mathsf{Ext}_{\mathsf{BBS}}}(\mathcal{A}, \lambda) = 0$.
- *Let* $\mathcal{A}$ *be an adversary against the* $(\mathsf{Ext}_{\mathsf{BBS}}, \mathrm{rDDH})$*-unforgeability of* $\mathsf{KVAC}_{\mathsf{BBS}} = \mathsf{KVAC}_{\mathsf{BBS}}[\mathsf{GGen}]$, *running in time* $t_{\mathcal{A}} = t_{\mathcal{A}}(\lambda)$ *making at most* $q_{h_0} = q_{h_0}(\lambda), q_{h_1} = q_{h_1}(\lambda), q_{h_2} = q_{h_2}(\lambda), q_{\mathsf{iss}} = q_{\mathsf{iss}}(\lambda), q_{\mathsf{Show}} = q_{\mathsf{Show}}(\lambda), q_{\mathrm{rDDH}} = q_{\mathrm{rDDH}}(\lambda)$ *queries to* $\mathsf{H}_0, \mathsf{H}_1$, H_2, Iss, SH, *and* rDDH *oracles, respectively. Let* $q = q_{\mathsf{iss}} + q_{h_2} + q_{\mathsf{Show}}$. *There exist adversaries* $\mathcal{B}_{\mathsf{ufcma}}$ *(playing* rDDH-UFCMA *game of* $\mathsf{MAC}_{\mathsf{BBS}}$*),* $\mathcal{B}_{\mathsf{com}}$ *(playing* KSND *game of* Π_{com}*),* $\mathcal{B}_{\mathrm{dlog}}, \mathcal{B}'_{\mathrm{dlog}}, \mathcal{B}''_{\mathrm{dlog}}$ *(playing* DL *game) and* $\mathcal{B}_\sigma$ *(playing the* ZK *game of* Π_σ*) such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{unf}}_{\mathsf{GGen}, \mathsf{Ext}_{\mathsf{BBS}}, \mathrm{rDDH}}(\mathcal{A}, \lambda) \leqslant{} & \sqrt{q \cdot \left(\mathsf{Adv}^{\mathsf{ufcma}}_{\mathsf{MAC}_{\mathsf{BBS}}, \mathrm{rDDH}}(\mathcal{B}_{\mathsf{ufcma}}, \lambda) + \mathsf{Adv}^{\mathrm{dlog}}_{\mathsf{GGen}}(\mathcal{B}''_{\mathrm{dlog}}, \lambda) + \frac{1}{p}\right)} \\ & + \mathsf{Adv}^{\mathrm{ksnd}}_{\Pi_{\mathsf{com}}, \mathsf{Ext}_{\mathsf{com}}, \tilde{\mathsf{R}}_{\mathsf{com}}}(\mathcal{B}_{\mathsf{com}}, \lambda) + \mathsf{Adv}^{\mathrm{dlog}}_{\mathsf{GGen}}(\mathcal{B}'_{\mathrm{dlog}}, \lambda) \\ & + \mathsf{Adv}^{\mathrm{dlog}}_{\mathsf{GGen}}(\mathcal{B}_{\mathrm{dlog}}, \lambda) + \mathsf{Adv}^{\mathsf{zk}}_{\Pi_\sigma, \mathsf{Sim}_\sigma}(\mathcal{B}_\sigma, \lambda) + \frac{q^2 + q + 2}{p}\ .\end{aligned}$$

$\mathsf{Sim}_{\mathsf{Setup}}(1^\ell, \mathsf{par}_g = (p, G, \mathbb{G}))$:

$\boldsymbol{H} = (H_i)_{i=1}^{\ell+1} \leftarrow\!\!\$\ \mathbb{G}^{\ell+1}$
return $\mathsf{par} = (p, G, \mathbb{G}, \boldsymbol{H})$

$\mathsf{Sim}_{\mathsf{Show}}(\text{“key”}, \mathsf{td} = \perp, X)$:

$\alpha \leftarrow\!\!\$\ \mathbb{Z}_p^*; \tilde{A} \leftarrow \alpha G; \tilde{B} \leftarrow \alpha X$
return $((\tilde{A}, \tilde{B}), \mathsf{st} = X)$

$\mathsf{Sim}_{\mathsf{Show}}(\text{“pub”}, \mathsf{st}, \phi_{\boldsymbol{I},\boldsymbol{a}}, \mathsf{nonce})$:

$\tilde{C} \leftarrow\!\!\$\ \mathbb{G}^*; \boldsymbol{H}_{\mathrm{priv}} \leftarrow (H_i)_{i \in [\ell+1] \setminus \boldsymbol{I}}$
$Y \leftarrow G + \sum_{i \in \boldsymbol{I}} a_i H_i$
// Sim programs H_2
$\pi_{\mathsf{pub}} \leftarrow \mathsf{Sim}_{\mathsf{pub}}^{\mathsf{H}_2}((M^{\mathsf{pub}}_{\tilde{C}, \boldsymbol{H}_{\mathrm{priv}}, \tilde{A}}, (Y, \tilde{B})), (\phi_{\boldsymbol{I},\boldsymbol{a}}, \mathsf{nonce}))$
return $(\tilde{C}, \pi_{\mathsf{pub}})$

$\mathsf{Sim}_{\mathsf{U}_1}(\mathsf{td} = \perp, \mathsf{pk}, \psi)$:

$C \leftarrow\!\!\$\ \mathbb{G}$
if $C + G = 0_{\mathbb{G}}$ **then abort**
// Sim programs H_0
$\pi_{\mathsf{com}} \leftarrow \mathsf{Sim}_{\mathsf{com}}^{\mathsf{H}_0}(\boldsymbol{H}, C, \psi)$
return $(\mu \leftarrow (C, \pi_{\mathsf{com}}), \mathsf{st}_{\mathsf{Sim}} \leftarrow C)$

$\mathsf{Sim}_{\mathsf{U}_2}(\mathsf{st}_{\mathsf{Sim}}, \mathsf{imsg})$:

$C \leftarrow \mathsf{st}_{\mathsf{Sim}}; (A, e, \pi_\sigma) \leftarrow \mathsf{imsg}$
$B \leftarrow G + C - eA$
if $\Pi_\sigma.\mathsf{Ver}^{\mathsf{H}_1}(((G, A), (X, B)), \pi_\sigma) = 0$ **then**
  return $\perp$
$\sigma \leftarrow 1$

Fig. 14. Simulator $\mathsf{Sim}_{\mathsf{BBS}} = \mathsf{Sim}_{\mathsf{BBS}}[\mathsf{Sim}_{\mathsf{com}}, \mathsf{Sim}_{\mathsf{pub}}]$

Additionally, $\mathcal{B}_{\mathrm{dlog}}, \mathcal{B}'_{\mathrm{dlog}}$ runs in time roughly $t_{\mathcal{A}}$, while $\mathcal{B}_{\mathsf{ufcma}}, \mathcal{B}''_{\mathrm{dlog}}$ runs in time roughly $2t_{\mathcal{A}}$. Also, $\mathcal{B}_{\mathsf{ufcma}}$ makes at most $2q_{\mathsf{iss}}$ and $2q_{\mathrm{rDDH}}$ queries to its MAC *and* rDDH *oracles, respectively. Finally, $\mathcal{B}_{\mathsf{com}}$ makes at most q_{h_0} queries to H_0 and q_{iss} queries to $\mathcal{O}_{\mathsf{Ext}}$.*

<u>ANONYMITY.</u> The following lemma establishes anonymity of $\mathsf{KVAC}_{\mathsf{BBS}}$ which follows from zero-knowledge properties of $\Pi_{\mathsf{com}}, \Pi_{\mathsf{pub}}$, soundness of Π_σ (to ensure that the maliciously issued credential is valid), and the rerandomization of the credential during showing as described earlier. The formal proof is given in Section 5.6.

Lemma 5.4 (Anonymity of $\mathsf{KVAC}_{\mathsf{BBS}}$). *Let* GGen *be a group generator that outputs groups of prime order $p = p(\lambda)$ and $\mathsf{Sim}_{\mathsf{Gen}}$ be the simulator for the global parameters generator (note again that it does not output any trapdoor). Let $\mathsf{Sim}_{\mathsf{com}}$ and $\mathsf{Sim}_{\mathsf{pub}}$ be zero-knowledge simulators for Π_{com} and Π_{pub}, and define $\mathsf{Sim}_{\mathsf{BBS}} = \mathsf{Sim}_{\mathsf{KVAC}_{\mathsf{BBS}}}[\mathsf{Sim}_{\mathsf{com}}, \mathsf{Sim}_{\mathsf{pub}}]$ as in Figure 14. Then,*

- *For any adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{par\text{-}indist}}_{\mathsf{KVAC}_{\mathsf{BBS}}, \mathsf{Sim}_{\mathsf{BBS}}}(\mathcal{A}, \lambda) = 0$.*
- *For any adversary $\mathcal{A}$ against the anonymity of $\mathsf{KVAC}_{\mathsf{BBS}}$ making at most $q_{h_0} = q_{h_0}(\lambda), q_{h_1} = q_{h_1}(\lambda), q_{h_2} = q_{h_2}(\lambda)$ queries to $\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2$, there exist adversaries $\mathcal{B}_{\mathsf{com}}$ playing the* ZK *game of Π_{com} and making at most q_{h_0} queries to H_0, $\mathcal{B}_\sigma$ playing the soundness game of Π_σ making at most q_{h_1} queries to H_1, and $\mathcal{B}_{\mathsf{pub}}$ playing the* ZK *game of Π_{pub} making at most q_{h_2} queries to H_2 such that:*

$$\mathsf{Adv}^{\mathsf{anon}}_{\mathsf{KVAC}_{\mathsf{BBS}}, \mathsf{Sim}_{\mathsf{Gen}}, \mathsf{Sim}_{\mathsf{BBS}}}(\mathcal{A}, \lambda) \leqslant \mathsf{Adv}^{\mathsf{zk}}_{\Pi_{\mathsf{com}}, \mathsf{Sim}_{\mathsf{com}}}(\mathcal{B}_{\mathsf{com}}) + \mathsf{Adv}^{\mathsf{zk}}_{\Pi_{\mathsf{pub}}, \mathsf{Sim}_{\mathsf{pub}}}(\mathcal{B}_{\mathsf{pub}}) + 2\mathsf{Adv}^{\mathsf{sound}}_{\Pi_\sigma}(\mathcal{B}_\sigma) .$$

<u>INTEGRITY AND VALIDITY OF KEY GENERATION.</u> The following two lemmas establish the integrity (with respect to the simulators $\mathsf{Sim}_{\mathsf{Gen}}, \mathsf{Sim}_{\mathsf{BBS}}$ defined in Lemma 5.4) and validity of key generation (with respect to the extractor $\mathsf{Ext}_{\mathsf{BBS}}$ defined in Lemma 5.3) for $\mathsf{KVAC}_{\mathsf{BBS}}$.

Lemma 5.5 (Validity of Key Generation of $\mathsf{KVAC}_{\mathsf{BBS}}$). *Let* GGen *and $\mathsf{Ext}_{\mathsf{BBS}}$ be as defined in Lemma 5.3. $\mathsf{KVAC}_{\mathsf{BBS}}$ satisfies validity of key generation with respect to $\mathsf{Ext}_{\mathsf{BBS}}$ defined in Lemma 5.3.*

Proof. The relation $\mathsf{R}_{\mathsf{dleq}}$ induced from the definition of $\mathsf{KVAC}_{\mathsf{BBS}}.\mathsf{SVer}_{\mathsf{key}}$ is defined in Equation (1). The lemma follows from the fact that, since $\mathbb{G}$ is prime-order, the public key $X \in \mathbb{G}$ fixes a unique underlying secret key $x \in \mathbb{Z}_p$.

Algorithm $\mathsf{oNIP.Setup}(\mathsf{par}_g = (p, G, \mathbb{G}))$:

$W \leftarrow\$ \mathbb{G}$

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

return $\mathsf{par}_{\mathsf{oNIP}} = (p, G, \mathbb{G}, W, \mathsf{H}_c)$

Algorithm $\mathsf{oNIP.Iss}_1(\mathsf{par}_{\mathsf{oNIP}}, x, (A, B))$:

if $xA \neq B$ **then abort**

$r_0, s_1, c_1 \leftarrow\$ \mathbb{Z}_p$

$R_{0,G} \leftarrow r_0 G; R_1 \leftarrow s_1 G - c_1 W$

$R_{0,A} \leftarrow r_0 A$

return $(R_{0,G}, R_{0,A}, R_1)$

Algorithm $\mathsf{oNIP.Iss}_2(c)$:

$c_0 \leftarrow c - c_1; s_0 \leftarrow r_0 + c_0 \cdot x$

return (c_0, s_0, s_1)

Algorithm $\mathsf{oNIP.Ver}(\mathsf{par}_{\mathsf{oNIP}}, (X, A, B), \pi)$:

parse $(c_0, c_1, s_0, s_1) \leftarrow \pi$

$R_{0,G} \leftarrow s_0 G - c_0 X$

$R_{0,A} \leftarrow s_0 A - c_0 B$

$R_1 \leftarrow s_1 G - c_1 W$

$c \leftarrow \mathsf{H}_c(X, A, B, R_{0,G}, R_{0,A}, R_1)$

return $(c_0 + c_1 = c)$

Algorithm $\mathsf{oNIP.U}_1(\mathsf{par}_{\mathsf{oNIP}}, (X, \tilde{A}, \tilde{B}))$:

$\beta \leftarrow\$ \mathbb{Z}_p$

$(A, B) \leftarrow (\tilde{A} + \beta G, \tilde{B} + \beta X)$

return (A, B)

Algorithm $\mathsf{oNIP.U}_2(R_{0,G}, R_{0,A}, R_1)$:

$\delta_0, \delta_1, \gamma_0, \gamma_1 \leftarrow\$ \mathbb{Z}_p$

$R'_{0,G} \leftarrow R_{0,G} + \delta_0 G - \gamma_0 X$

$R'_{0,A} \leftarrow R_{0,A} - \beta R_{0,G} + \delta_0 \tilde{A} - \gamma_0 \tilde{B}$

$R'_1 \leftarrow R_1 + \delta_1 G - \gamma_1 W$

$c' \leftarrow\$ \mathsf{H}_c(X, \tilde{A}, \tilde{B}, R'_{0,G}, R'_{0,A}, R'_1)$

return $c \leftarrow c' - \gamma_0 - \gamma_1$

Algorithm $\mathsf{oNIP.U}_3(c_0, s_0, s_1)$:

$c_1 \leftarrow c - c_0$

if $R_{0,G} + c_0 X \neq s_0 G \vee$

$\quad R_{0,A} + c_0 B \neq s_0 A \vee$

$\quad R_1 + c_1 W \neq s_1 G$ **then abort**

$c'_0 \leftarrow c_0 + \gamma_0; s'_0 \leftarrow s_0 + \delta_0$

$c'_1 \leftarrow c_1 + \gamma_1; s'_1 \leftarrow s_1 + \delta_1$

return $\pi \leftarrow (c'_0, c'_1, s'_0, s'_1)$

Fig. 15. Oblivious proof issuance $\mathsf{oNIP} = \mathsf{oNIP}[\mathsf{GGen}, \mathsf{R}_{\mathsf{dleq}}]$ for the DLEQ relation. We omitted the user and issuer's states and assume that any variable defined in the previous round is accessible in the next round.

Lemma 5.6 (Integrity of $\mathsf{KVAC}_{\mathsf{BBS}}$). *Let $\mathsf{GGen}, \mathsf{Sim}_{\mathsf{Gen}}$ and $\mathsf{Sim}_{\mathsf{BBS}}$ be as defined in Lemma 5.4. Let $\mathcal{A}$ be an adversary playing the integrity of issued credentials game of $\mathsf{KVAC}_{\mathsf{BBS}}$ with respect to the simulators $\mathsf{Sim}_{\mathsf{Gen}}$ and $\mathsf{Sim}_{\mathsf{BBS}}$ defined in Lemma 5.4 and making at most $q_{h_1} = q_{h_1}(\lambda)$ queries to H_1. There exists an adversary $\mathcal{B}$ against the soundness of Π_σ and making at most q_{h_1} queries to H_1 such that*

$$\mathsf{Adv}^{\mathsf{integ}}_{\mathsf{KVAC}_{\mathsf{BBS}}, \mathsf{Sim}_{\mathsf{Gen}}, \mathsf{Sim}_{\mathsf{BBS}}}(\mathcal{A}, \lambda) \leqslant \mathsf{Adv}^{\mathsf{sound}}_{\Pi_\sigma}(\mathcal{B}, \lambda) .$$

Proof. The reduction $\mathcal{B}$ simulates the integrity of issuance game to $\mathcal{A}$ and outputs the statement $(A, (B := C - eA))$ along with proof π_σ. Winning the integrity of issuance game implies $x\tilde{A} \neq \tilde{B}$, which can occur for an honest user only if $xA' \neq B'$, and π_σ must have been valid otherwise the honest user would have aborted.

5.3 oNIP for BBS-based instantiation

In this section, we give the $\mathsf{oNIP}_{\mathsf{BBS}} = \mathsf{oNIP}[\mathsf{GGen}, \mathsf{R}_{\mathsf{dleq}}]$ protocol (described in Figure 15) for the family of relations $\mathsf{R}_{\mathsf{dleq}}$, defined in Equation (1). 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 $(\tilde{A}, \tilde{B})$. We remark that this protocol is similar to a recent blind signature scheme [CATZ24] and the oNIP for $\mathsf{R}_{\mathsf{dleq}}$ in [OTZZ24], except that in their cases the issuer computes $B = xA$ for the user who sends A.

The following theorem then establishes the security properties of $\mathsf{oNIP}_{\mathsf{BBS}}$ with the proof in Section 5.7. Note that $\mathsf{oNIP}_{\mathsf{BBS}}$ is zero-knowledge with respect to rDDH, because the simulator needs to check that $B = xA$ without x. On the technical side, the proofs for both zero-knowledge and obliviousness utilize the structure of OR-proofs in that they generate the public parameters $W \leftarrow wG$ with a trapdoor $w \leftarrow \mathbb{Z}_p$ and use w to simulate the issuance protocol and the non-interactive proof.

Theorem 5.7. *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* $\mathsf{Sim}_{\mathsf{Gen}}$*.*

5.4 BBS-based SAAC

The following corollary establishes the security of $\mathsf{SAAC}_{\mathsf{BBS}}$, a BBS-based instantiation of our generic SAAC construction from Section 4.2. The corollary immediately follows from Theorems 4.2 and 5.7 and Lemmas 5.3 to 5.6.

Corollary 5.8. *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 4.2. Then,* $\mathsf{SAAC}_{\mathsf{BBS}}$ *satisfies correctness, unforgeability in the ROM assuming* (q, rDDH)*-SDH, and anonymity in the ROM.*

<u>INTEGRITY.</u> Although we do not formally show this, strong integrity of $\mathsf{SAAC}_{\mathsf{BBS}}$ (defined in Figure 9) follows from (1) the public key X fixing a unique underlying secret key x and (2) soundness of Π_σ ensuring that the issued credential is valid.

5.5 Unforgeability proof of $\mathsf{KVAC}_{\mathsf{BBS}}$

Proof (of Lemma 5.3). Since $\mathsf{Ext}_{\mathsf{Setup}}$ is exactly $\mathsf{KVAC}_{\mathsf{BBS}}.\mathsf{Setup}$, parameter indistinguishability follows immediately.

To show the advantage of $\mathcal{A}$ in the unforgeability game, consider the following sequence of games:

$\mathbf{G}_1(\lambda)$**:** $(\mathsf{Ext}_{\mathsf{BBS}}, \text{rDDH})$-unforgeability of $\mathsf{KVAC}_{\mathsf{BBS}}$.

$\mathbf{G}_2(\lambda)$**:** The oracle Iss is modified so that after checking validity of C, π_{com} it runs the extractor $(s, \boldsymbol{m}) \leftarrow \mathsf{Ext}^{\mathsf{H}_0}_{\mathsf{com}}(\mathcal{Q}, (\boldsymbol{H}, C, \psi), \pi_{\mathsf{com}})$. If $((\boldsymbol{H}, C, \psi), (s, \boldsymbol{m})) \notin \tilde{\mathsf{R}}_{\mathsf{com}}$, call this event BadCom, then abort.
We now construct a reduction $\mathcal{B}_{\mathsf{com}}$ to knowledge soundness of Π_{com} with oracle access to $\mathcal{O}_{\mathsf{Ext}}$. The reduction $\mathcal{B}_{\mathsf{com}}$ simulates $\mathbf{G}_1$ to $\mathcal{A}$ and on every Iss query, queries its oracle $\mathcal{O}_{\mathsf{Ext}}$ with $(\boldsymbol{H}, C, \psi), \pi_{\mathsf{com}}$. By definition of the straight-line extractable knowledge-soundness game, $\mathcal{B}_{\mathsf{com}}$ wins if BadCom occurs. Hence,

$$\Pr[\mathbf{G}_2{}^{\mathcal{A}}(\lambda) = 1] \geqslant \Pr[\mathbf{G}_1{}^{\mathcal{A}}(\lambda) = 1] - \mathsf{Adv}^{\mathrm{ksnd}}_{\Pi_{\mathsf{com}}, \mathsf{Ext}_{\mathsf{com}}, \tilde{\mathsf{R}}_{\mathsf{com}}}(\mathcal{B}_{\mathsf{com}}, \lambda) .$$

$\mathbf{G}_3(\lambda)$**:** In this game, we abort if the extracted attributes $\boldsymbol{m}$ and randomness s satisfies $0_{\mathbb{G}} = \sum_{i=1}^{\ell} m_i H_i + sH_{\ell+1}$, denote this event as BadExt. This is to ensure that in each session $((\boldsymbol{H}, C, \psi), (s, \boldsymbol{m})) \in \mathsf{R}_{\mathsf{com}}$ and $\psi(\boldsymbol{m}) = 1$. Note that the event BadExt implies breaking rel-DL on $\boldsymbol{H}$. Thus, by Lemma 2.1, there exists a reduction $\mathcal{B}_{\mathrm{dlog}}$ running in time roughly $t_{\mathcal{A}}$ such that

$$\Pr[\mathbf{G}_3{}^{\mathcal{A}}(\lambda) = 1] \geqslant \Pr[\mathbf{G}_2{}^{\mathcal{A}}(\lambda) = 1] - \mathsf{Adv}^{\mathrm{dlog}}_{\mathsf{GGen}}(\mathcal{B}_{\mathrm{dlog}}, \lambda) - \frac{1}{p} .$$

$\mathbf{G}_4(\lambda)$**:** In this game, we simulate the proof π_σ in the issuance oracle using Sim_σ, which programs H_1. The reduction $\mathcal{B}_\sigma$ simulates the entire game $\mathcal{A}$, in one case using the real NIZK protocol and in the other case using the simulator.

$$\Pr[\mathbf{G}_4{}^{\mathcal{A}}(\lambda) = 1] \geqslant \Pr[\mathbf{G}_3{}^{\mathcal{A}}(\lambda) = 1] - \mathsf{Adv}^{\mathsf{zk}}_{\Pi_\sigma, \mathsf{Sim}_\sigma}(\mathcal{B}_\sigma, \lambda)$$

$\mathbf{G}_5(\lambda)$**:** The game now simulates the new user oracle and the showing oracles as follows:

- At the start of the game we initialize table $T_2 \leftarrow (\,)$ and use it for H_2 lazy-sampling.
- Oracle NewUsr is modified so that it just sets $\sigma_{\mathsf{cid}} \leftarrow \bot$ instead of using $\mathsf{KVAC}_{\mathsf{BBS}}.\mathsf{Iss}$.

- Oracle $\mathrm{SH}_{\mathsf{key}}$ does the following instead of running $\mathsf{KVAC}_{\mathsf{BBS}}.\mathsf{Show}$: sample $r, r' \leftarrow_{\$} \mathbb{Z}_p^*$, compute $\tau_{\mathsf{key}} \leftarrow (\tilde{A} := rG, \tilde{B} := rX)$. Then, $\mathrm{SH}_{\mathsf{pub}}$ computes $\tilde{C} \leftarrow r'G$, output $\perp$ if $\phi_{\boldsymbol{I},\boldsymbol{a}}(\boldsymbol{m}_{\mathsf{cid}}) \neq 1$, otherwise simulate the proof π_{pub} by programming values into T_2. Explicitly, the reduction sets $k := \ell - |\boldsymbol{I}|$, sets $\boldsymbol{H}_{\mathsf{priv}} \leftarrow (H_i)_{i \in [\ell] \setminus \boldsymbol{I}}$, computes $Y \leftarrow G + \langle (m_i)_{i \in \boldsymbol{I}}, (H_i)_{i \in \boldsymbol{I}} \rangle$, samples $c \leftarrow_{\$} \mathbb{Z}_p$ and $s \leftarrow_{\$} \mathbb{Z}_p^{k+3}$, then computes $R \leftarrow M^{\mathsf{pub}}_{\tilde{C}, H_{\mathsf{priv},1}, \ldots, H_{\mathsf{priv},k}, \tilde{A}} \boldsymbol{s} - c(Y, \tilde{B})^T$. It then sets $T_2(M^{\mathsf{pub}}_{\tilde{C}, H_{\mathsf{priv},1}, \ldots, H_{\mathsf{priv},k}, \tilde{A}}, (Y, \tilde{B})^T, R, M) \leftarrow c$, unless that value has already been set in which case the reduction aborts – call this event $\mathsf{CollShow}$.

Considering the distribution of $(\tilde{A}, \tilde{B}, \tilde{C})$ in the prior game, we start with $A, B, C = G + sH_{\ell+1} + \sum_{i=1}^{\ell} m_i H_i$ such that $xA = B = C - eA$, then sample $r, r' \leftarrow_{\$} \mathbb{Z}_p^*$, and then compute $\tilde{C} \leftarrow rC$, $\tilde{A} \leftarrow r'rA$, and $\tilde{B} \leftarrow r'\tilde{C} - e\tilde{A} = r'rC - er'rA = r'r(C - eA) = r'rB$. By inspection $\tilde{C}$ is uniform in $\mathbb{G}^*$ and independent of $(\tilde{A}, \tilde{B})$ which are uniform in the set of DH tuples. Thus, the distribution is exactly the same assuming that $\mathsf{CollShow}$ does not occur. Note that the input being programmed contains group elements which are uniform in $\mathbb{G}$. A collision occurs if the value was already set which could occur during lazy sampling or programming for simulation. By the union bound,

$$\Pr[\mathbf{G_5}^{\mathcal{A}}(\lambda) = 1] \geqslant \Pr[\mathbf{G_4}^{\mathcal{A}}(\lambda) = 1] - \frac{q_{\mathsf{Show}}(q_{\mathsf{Show}} + q_{h_2})}{p}$$
$$\geqslant \Pr[\mathbf{G_4}^{\mathcal{A}}(\lambda) = 1] - \frac{q^2}{p} \; .$$

$\mathbf{G}_6(\lambda)$: This game aborts if the forgery corresponds to a programmed random oracle input. In particular, when $\mathcal{A}$ outputs the forgery $(\phi^*, \mathsf{nonce}^*, \tau^*)$, the game checks the validity of the forgery and does the following:

- Parse τ^* as $((\tilde{A}, \tilde{B}), (\tilde{C}, (c, \boldsymbol{s})))$.
- The game computes $Y = G + \sum_{i \in \boldsymbol{I}} m_i H_i$ and $R \leftarrow M^{\mathsf{pub}}_{\tilde{C}, H_{\mathsf{priv},1}, \ldots, H_{\mathsf{priv},k}, \tilde{A}} \boldsymbol{s} - c(Y, \tilde{B})^T$.
- The game aborts if the hash query $\mathsf{H}_2(M^{\mathsf{pub}}_{\tilde{C}, H_{\mathsf{priv},1}, \ldots, H_{\mathsf{priv},k}, \tilde{A}}, (Y, \tilde{B})^T, R, \mathsf{nonce}^*)$ was programmed in $\mathrm{SH}_{\mathsf{pub}}$.

Note that since $(\phi^*, \mathsf{nonce}^*, \tau^*)$ is not the same as any output from SH (by the winning condition) and the hash query contains $\tilde{A}, \tilde{B}, \tilde{C}, \mathsf{nonce}^*, \phi^*$, it can only be the case that $\boldsymbol{s} \neq \tilde{\boldsymbol{s}}$, where $\tilde{\boldsymbol{s}}$ are contained in the output of $\mathrm{SH}_{\mathsf{pub}}$ query which programs H_2 at the same input. By how R is computed we have $M^{\mathsf{pub}}_{\tilde{C}, H_{\mathsf{priv},1}, \ldots, H_{\mathsf{priv},k}, \tilde{A}}(\boldsymbol{s} - \tilde{\boldsymbol{s}}) = 0$. Let $\boldsymbol{s}_1$ be the first $k+1$ elements of $\boldsymbol{s}$, and $\boldsymbol{s}_2$ be the last two elements. Let $\tilde{\boldsymbol{s}}_1, \tilde{\boldsymbol{s}}_2$ analogously relative to $\tilde{\boldsymbol{s}}$. We break $\boldsymbol{s} \neq \tilde{\boldsymbol{s}}$ into two cases: (a) $\boldsymbol{s}_1 \neq \tilde{\boldsymbol{s}}_1$ and (b) $\boldsymbol{s}_2 \neq \tilde{\boldsymbol{s}}_2$. In both cases, we have a non-trivial linear equation over $\tilde{C}, \boldsymbol{H}_{\text{priv}}$ or $\tilde{C}, \tilde{A}$, which allows us to break rel-DL.

The reduction, on a rel-DL instance $(p, G, \mathbb{G})$ with $2q_{\mathsf{Show}} + \ell + 1$ group element challenges which we parse to the form $(\tilde{A}_i, \tilde{C}_i)_{i \in [q_{\mathsf{Show}}]}$ and $\boldsymbol{H} \in \mathbb{G}^{\ell+1}$. The reduction then samples the secret key $\mathsf{sk} = x \leftarrow_{\$} \mathbb{Z}_p$ and runs the game as in $\mathbf{G_6}$ with an exception that each SH query (indexed with i), use $\tilde{A}_i$ and $\tilde{C}_i$, and computes $\tilde{B} = x\tilde{A}$. Note that the view of $\mathcal{A}$ remains as in $\mathbf{G_6}$. Now, when the added abort is supposed to occur, we have a non-trivial linear equation over the challenges. Hence, by Lemma 2.1, there exists an adversary $\mathcal{B}'_{\text{dlog}}$ running in time roughly $t_{\mathcal{A}}$ such that

$$\Pr[\mathbf{G_6}^{\mathcal{A}}(\lambda) = 1] \geqslant \Pr[\mathbf{G_5}^{\mathcal{A}}(\lambda) = 1] - \frac{1}{p} - \mathsf{Adv}^{\text{dlog}}_{\mathsf{GGen}}(\mathcal{B}'_{\text{dlog}}, \lambda) \; .$$

$\mathbf{G}_7(\lambda)$: At the start of the game we sample $(h_1, e_1), \ldots, (h_q, e_q) \leftarrow_{\$} \mathbb{Z}_p \times \mathbb{Z}_p$ and initialize a counter $\mathsf{cnt} \leftarrow 0$. Whenever we need to program an RO value for T_2, we use h_{cnt} and then set $\mathsf{cnt} \leftarrow \mathsf{cnt} + 1$. Similarly, in Iss, instead of $e \leftarrow_{\$} \mathbb{Z}_p$ we do $e \leftarrow e_{\mathsf{cnt}}$ and set $\mathsf{cnt} \leftarrow \mathsf{cnt} + 1$. We then sample a set $\rho = (\rho', \rho_{\mathcal{A}})$ of coins chosen uniformly at random, where ρ' is used for the game to run other components not associated with the issuance, H_1, or H_2, (i.e., these contains the random coins for H_0 and $\mathrm{SH}_{\mathsf{key}}$ simulations) and $\rho_{\mathcal{A}}$ will be the random coins for $\mathcal{A}$. We run $\mathcal{A}$ with random coins $\rho_{\mathcal{A}}$, simulating the game to them as described. When $\mathcal{A}$ outputs $(\phi^*, \mathsf{nonce}^*, \tau^*)$, first check the winning conditions and abort if they are not satisfied. After $\mathcal{A}$ outputs their forgery $(\tau^*, \phi^*, \mathsf{nonce}^*)$, we parse $((\tilde{A}, \tilde{B}), (\tilde{C}, (c, \boldsymbol{s}))) \leftarrow \tau$ and $(\boldsymbol{I}, \boldsymbol{m}') \leftarrow \phi^*$. Then, compute Y, R as in $\mathbf{G}_6$. Note that even if the hash query $\mathsf{H}_2(M^{\mathsf{pub}}_{\tilde{C},H_{\mathsf{priv},1},\ldots,H_{\mathsf{priv},k},\tilde{A}}, (Y,\tilde{B})^T, R, \mathsf{nonce}^*)$ was not made after $\mathcal{A}$ stopped, in that case the reduction makes it on its own while checking if τ is valid, so the index of that hash query exists–let it be J. Now $(h'_J, e'_J), \ldots, (h'_q, e'_q) \leftarrow\!\!\$\ \mathbb{Z}_p \times \mathbb{Z}_p$, clear T_2 and set $\mathsf{cnt} \leftarrow 0$. We run the game again with the same random coins ρ and do everything exactly the same up until $\mathsf{cnt} \geqslant J$, at which point we start using h'_{cnt} and e'_{cnt} instead of h_{cnt} and e_{cnt}. Again we check $\mathcal{A}$'s winning conditions and if they are satisfied then look up the index of the forgery hash query at the end of the game, let it be J'. If $J \neq J'$ or $h_J = h'_J$ then abort. Note that due to the change in $\mathbf{G}_6$, J and J' do not correspond to an RO query programmed in SH. Finally, use the two different RO responses to extract a witness $(e, r', r'', s^*, \hat{\boldsymbol{m}})$ for $\mathsf{R}_{\mathsf{pub}}$ using special soundness of the underlying sigma protocol of Π_{pub}. Reconstruct $\boldsymbol{m}^*$ from $\hat{\boldsymbol{m}}$ (undisclosed attributes), $\boldsymbol{I}$ (indices of disclosed attributes), and $(m_i)'_{i\in\boldsymbol{I}}$ (disclosed attributes). By the generalized forking lemma,[6]

$$\Pr[\mathbf{G}_6{}^{\mathcal{A}}(\lambda) = 1] \leqslant \sqrt{q \cdot \Pr[\mathbf{G}_7{}^{\mathcal{A}}(\lambda) = 1]} + \frac{q}{p}$$

$\mathbf{G}_8(\lambda)$: We add the winning condition that $r'' \neq 0$. If $\mathcal{A}$ wins with $r'' = 0$ then $0 = G + \langle \boldsymbol{H}, (\boldsymbol{m}^* \| s^*)\rangle$, which allows us to break rel-DL on $(G\|\boldsymbol{H})$. We have that there exists $\mathcal{B}''_{\mathrm{dlog}}$ with running time roughly $2t_{\mathcal{A}}$ such that

$$\Pr[\mathbf{G}_8{}^{\mathcal{A}}(\lambda) = 1] \geqslant \Pr[\mathbf{G}_7{}^{\mathcal{A}}(\lambda) = 1] - \mathsf{Adv}^{\mathrm{dlog}}_{\mathsf{GGen}}(\mathcal{B}''_{\mathrm{dlog}}, \lambda) - \frac{1}{p} \ .$$

Finally, let $\mathcal{B}_{\mathsf{ufcma}}$ be the algorithm playing the rDDH-UFCMA game for $\mathsf{MAC}_{\mathsf{BBS}}$, which on input $(p, G, \mathbb{G}, H), \mathsf{ipk}$ and with access to oracles MAC and rDDH$'$ simulates $\mathbf{G}_8$ with the following changes the first time we run $\mathcal{A}$:

1. Set $\sigma_1, \ldots, \sigma_q \leftarrow \perp$
2. Instead of $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KVAC}_{\mathsf{BBS}}.\mathsf{KeyGen}(1^\lambda)$, do $\mathsf{pk} \leftarrow \mathsf{ipk}$
3. To form A', e without knowledge of x in an Iss query, query $(A, e) \leftarrow \mathrm{MAC}((\boldsymbol{m}\|s))$ where $\boldsymbol{m}$ and s are extracted from π_{com}. Additionally, record $\sigma_{\mathsf{cnt}} \leftarrow (A, e)$.
4. Forward any rDDH oracle queries to the rDDH$'$ oracle

This yields $(\tau_{\mathsf{key},1} = (\tilde{A}_1, \tilde{B}_1), \tau_{\mathsf{pub},1} = (\tilde{C}_1, \pi_{\mathsf{pub},1}))$ and $\phi_{\boldsymbol{I}_1, \boldsymbol{m}'_1}$. Also, $\mathcal{B}_{\mathsf{ufcma}}$ aborts if $\pi_{\mathsf{pub},1}$ is not valid. When we run $\mathcal{A}$ for the second time, we still set up the key in the same way and forward rDDH oracles queries. For all Iss queries where $\mathsf{cnt} < J$ we respond with σ_{cnt}, and after that point we query $(A, e) \leftarrow \mathrm{MAC}(\boldsymbol{m})$ and set $A' \leftarrow sA$. Running $\mathcal{A}$ for the second time yields $(\tau_{\mathsf{key},2} = (\tilde{A}_2, \tilde{B}_2), \tau_{\mathsf{pub},2} = (\tilde{C}_2, \pi_{\mathsf{pub},2}))$ and $\phi_{\boldsymbol{I}_2, \boldsymbol{m}'_2}$ described as $(\boldsymbol{I}_2, \boldsymbol{m}'_2)$. Also, $\mathcal{B}_{\mathsf{ufcma}}$ aborts if $\pi_{\mathsf{pub},2}$ is not valid. Since $\mathcal{A}$ is run with the same randomness and inputs up to the point where $\mathsf{cnt} < J$, they will make the exact same oracle queries to H_2 and Iss up to that point, so our simulation of the last game is perfect.

Once we extract $\boldsymbol{m}^*, s^*, \tilde{A}, r'', e$, we first check if $r' = 0$. This implies $\tilde{B} = -e\tilde{A} = x\tilde{A}$, thus $x = e$, and we can forge a MAC using the secret key. Otherwise, we compute $A \leftarrow (r'')^{-1}(r')^{-1}\tilde{A}$ and output (A, e). If $x\tilde{A} = \tilde{B}$, which is part of $\mathcal{A}$'s winning condition in the last game, then rearranging $\tilde{B} = r'\tilde{C} - e\tilde{A} = x\tilde{A}$ yields $(x + e)\tilde{A} = r'(r'')(G + \langle \boldsymbol{H}, (\boldsymbol{m}^* \| s^*)\rangle)$ and thus $A = (x+e)^{-1}(G + \langle \boldsymbol{H}, (\boldsymbol{m}^* \| s^*)\rangle)$, so (A, e) is a valid BBS signature on $(\boldsymbol{m}^* \| s)$. Moreover, we have by the winning condition of $\mathcal{A}$ in the last game that $\boldsymbol{m}^*$ satisfies the selective disclosure predicate $\phi_{\boldsymbol{I}_1, \boldsymbol{m}'_1}$, meaning that $\boldsymbol{m}^*_i = \boldsymbol{m}'_{1,i}$ for all $i \in \boldsymbol{I}_1$, whereas this does not hold true for any of the messages extracted during issuance (due to the winning condition of $\mathcal{A}$). This guarantees that $\boldsymbol{m}^*$ is a fresh forgery with regard to the MAC queries in the first run of $\mathcal{A}$, and the analogous argument guarantees that it is a fresh forgery with regard to the MAC queries in the second run, and thus fresh overall. We conclude that

$$\mathsf{Adv}^{\mathsf{ufcma}}_{\mathsf{BBS}}(\mathcal{B}_{\mathsf{ufcma}}, \lambda) \geqslant \mathsf{Adv}^{\mathbf{G}_8}(\mathcal{A}, \lambda) \ .$$

□

[6] The generalized forking lemma applied to our setting only guarantees $(h_J, e_J) \neq (h'_J, e'_J)$ as tuples, which is not sufficient for our proof. This is merely a technicality of the theorem statement, and it is not hard to see how the proof can be modified so that we may expect $h_J \neq h'_J$ with the probability shown.

5.6 Anonymity proof of $\mathsf{KVAC}_{\mathsf{BBS}}$

Proof (of Lemma 5.4). Parameter indistinguishability for $\mathsf{Sim}_{\mathsf{BBS}}$ follows because $\mathsf{Sim}_{\mathsf{Setup}}$ is identical to the Setup algorithm.

We assume without loss of generality that the queries made to H_1 when the game verifies π_σ are already made by $\mathcal{A}$. (To be more precise, this increases the query count by 1). We proceed via a sequence of games.

$\mathbf{G}_1(\lambda)$: This is the game $\mathrm{Anon}_{\mathsf{KVAC}_{\mathsf{BBS}},\mathsf{Sim}_{\mathsf{Gen}},\mathsf{Sim}_{\mathsf{BBS}},0}$.

$\mathbf{G}_2(\lambda)$: We simulate π_{com} as in $\mathsf{Sim}_{\mathsf{U}_1}$ instead of generating it honestly. There exists $\mathcal{B}_{\mathsf{com}}$ making at most q_{h_0} queries to H_0 such that $\left|\Pr[\mathbf{G}_2{}^{\mathcal{A}}(\lambda)=1]-\Pr[\mathbf{G}_1{}^{\mathcal{A}}(\lambda)=1]\right| \leqslant \mathsf{Adv}^{\mathsf{zk}}_{\Pi_{\mathsf{com}},\mathsf{Sim}_{\mathsf{com}}}(\mathcal{B}_{\mathsf{com}})$.

$\mathbf{G}_3(\lambda)$: We simulate π_{pub} making at most q_{h_2} queries to H_2 as in $\mathsf{Sim}_{\mathsf{Show}}$ instead of generating it honestly. There exists $\mathcal{B}_{\mathsf{pub}}$ such that $\left|\Pr[\mathbf{G}_3{}^{\mathcal{A}}(\lambda)=1]-\Pr[\mathbf{G}_2{}^{\mathcal{A}}(\lambda)=1]\right| \leqslant \mathsf{Adv}^{\mathsf{zk}}_{\Pi_{\mathsf{pub}},\mathsf{Sim}_{\mathsf{pub}}}(\mathcal{B}_{\mathsf{pub}})$.

$\mathbf{G}_4(\lambda)$: We add a condition in $\mathsf{KVAC}_{\mathsf{BBS}}.\mathsf{U}_2$ that if $(C,A,e)\notin \mathsf{R}_\sigma$ then the game aborts. There exists $\mathcal{B}_\sigma$ making at most q_{h_1} queries to H_1 such that $\left|\Pr[\mathbf{G}_4{}^{\mathcal{A}}(\lambda)=1]-\Pr[\mathbf{G}_3{}^{\mathcal{A}}(\lambda)=1]\right| \leqslant \mathsf{Adv}^{\mathsf{sound}}_{\Pi_\sigma}(\mathcal{B}_\sigma)$.

$\mathbf{G}_5(\lambda)$: We use $\mathsf{Sim}_{\mathsf{U}_1}$ and $\mathsf{Sim}_{\mathsf{U}_2}$ instead of $\mathsf{KVAC}_{\mathsf{BBS}}.\mathsf{U}_1$ and $\mathsf{KVAC}_{\mathsf{BBS}}.\mathsf{U}_2$, but keep the relation check as part of the winning condition, so we have that $\sigma=(A,e)$ is such that $xA=C-eA$ where $C=G+\langle \boldsymbol{H},(\boldsymbol{m}\|s)\rangle$. Consider the following equal distributions:

$$
\begin{aligned}
\{(rC, r'rA, r'rC - er'rA) : r, r' \leftarrow\!\!\$\ \mathbb{Z}_p^*\} &\equiv \{(rC, r'A, r'(C-eA)) : r, r' \leftarrow\!\!\$\ \mathbb{Z}_p^*\} \\
&\equiv \{(rC, r'A, r'(xA) : r, r' \leftarrow\!\!\$\ \mathbb{Z}_p^*\} \\
&\equiv \{(\tilde{C}, \alpha G, \alpha X) : \tilde{C} \leftarrow\!\!\$\ \mathbb{G}^*, \alpha \leftarrow\!\!\$\ \mathbb{Z}_p^*\}\ .
\end{aligned}
$$

Also $\{sH_{\ell+1}+(G+\sum_{i=1}^{\ell} m_iH_i) : s \leftarrow\!\!\$\ \mathbb{Z}_p\} \equiv \{C : C \leftarrow\!\!\$\ \mathbb{G}\}$ (the above equations then follows due to the abort introduced in U_1 and $\mathsf{Sim}_{\mathsf{U}_1}$ that ensures $C+G\neq 0_{\mathbb{G}}$), so $\Pr[\mathbf{G}_5{}^{\mathcal{A}}(\lambda)=1]=\Pr[\mathbf{G}_4{}^{\mathcal{A}}(\lambda)=1]$

$\mathbf{G}_6(\lambda)$: We remove the inefficient check that $(C,A,e)\notin \mathsf{R}_\sigma$, which yields the game $\mathrm{Anon}_{\mathsf{KVAC}_{\mathsf{BBS}},\mathsf{Sim}_{\mathsf{Gen}},\mathsf{Sim}_{\mathsf{BBS}},1}$. We have $\left|\Pr[\mathbf{G}_6{}^{\mathcal{A}}(\lambda)=1]-\Pr[\mathbf{G}_5{}^{\mathcal{A}}(\lambda)=1]\right| \leqslant \mathsf{Adv}^{\mathsf{sound}}_{\Pi_\sigma}(\mathcal{B}_\sigma)$, which yields the claim.

□

5.7 Security Proof of $\mathsf{oNIP}_{\mathsf{BBS}}$

In this section, we prove Theorem 5.7. Correctness of the protocol follows easily from the algebra. The following lemmas then establish soundness, zero-knowledge, and obliviousness for valid statements.

Lemma 5.9 (Soundness of $\mathsf{oNIP}_{\mathsf{BBS}}$). *For any adversary $\mathcal{A}$ making at most $q_{\mathsf{H}}=q_{\mathsf{H}}(\lambda)$ queries to H_c modeled as a random oracle and running in time $t_{\mathcal{A}}=t_{\mathcal{A}}(\lambda)$, there exists an adversary $\mathcal{B}$ playing the* DL *game and running in time roughly $2t_{\mathcal{A}}$ such that*

$$
\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{oNIP}_{\mathsf{BBS}}}(\mathcal{A},\lambda) \leqslant \sqrt{(q_{\mathsf{H}}+1)\mathsf{Adv}^{\mathrm{dlog}}_{\mathsf{GGen}}(\mathcal{B},\lambda)} + \frac{q_{\mathsf{H}}+1}{p}\ .
$$

Proof. Let $\mathcal{A}$ be an adversary as described in the lemma statement and w.l.o.g. assume that $\mathcal{A}$ already made the RO query corresponding to the verification of its output $((X,A,B),\pi)$ (This increases q_{H} by 1).

Consider an adversary $\mathcal{B}$ playing the DL game such that it sets $\mathsf{par}_{\mathsf{oNIP}}$ as its input $(p,G,\mathbb{G},W)$ and runs $(X,A,B,\pi) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{H}_c}(\mathsf{par}_{\mathsf{oNIP}})$. Note that $\mathcal{B}$ answers RO queries with uniformly random $h_1,\ldots,h_{q_{\mathsf{H}}+1} \leftarrow\!\!\$\ \mathbb{Z}_p$. Let I be the index of the RO queries which corresponds to the verification of (X,A,B,π). Then, $\mathcal{B}$ rewinds $\mathcal{A}$ to when the I-th query is made and from that point on uses uniformly random $h'_I,\ldots,h'_{q_{\mathsf{H}}+1} \leftarrow\!\!\$\ \mathbb{Z}_p$ to answer the RO queries. Finally, $\mathcal{A}$ outputs (X',A',B',π'). If the verification of this output does not correspond to the I-th RO query, $\mathcal{B}$ aborts. Otherwise, it parses

$$
\pi=(c_0,c_1,s_0,s_1)\ ,\pi'=(c'_0,c'_1,s'_0,s'_1)\ ,
$$

and if $c_1\neq c'_1$, it returns $(s'_1-s_1)/(c'_1-c_1)$.

First, let Succ be the event that $h_I \neq h'_I$, and $\mathcal{A}$ successfully outputs (X, A, B, π) and (X', A', B', π') such that $(\mathrm{dlog}_G X) \cdot A \neq B$ and $(\mathrm{dlog}_G X') \cdot A' \neq B'$ but the proofs verifies and they corresponds to the same RO query. Then, by the forking lemma,

$$\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{oNIP}_{\mathsf{BBS}}}(\mathcal{A}, \lambda) \leqslant \sqrt{(q_{\mathsf{H}}+1)\mathsf{Pr}[\mathsf{Succ}]} + \frac{q_{\mathsf{H}}+1}{p} .$$

Now, notice that when Succ occurs, π and π' corresponds to the same hash query which implies that:

(a) $(X, A, B) = (X', A', B')$
(b) $s_0 G - c_0 X = s'_0 G - c'_0 X$ and $s_0 A - c_0 B = s'_0 A - c'_0 B$
(c) $s_1 G - c_1 W = s'_1 G - c'_1 W$

Since $B \neq (\mathrm{dlog}_G X) \cdot A$, by (b), it is only the case that $c_0 = c'_0$. Hence, by (c) and with $c_0 + c_1 = h_I \neq h'_I = c'_0 + c'_1$, we have $c_1 \neq c'_1$, so $\mathcal{B}$ extracts the discrete log of W. Therefore, $\mathsf{Pr}[\mathsf{Succ}] \leqslant \mathsf{Adv}^{\mathrm{dlog}}_{\mathsf{GGen}}(\mathcal{B}, \lambda)$, proving the lemma. □

Lemma 5.10 (Zero-Knowledge of $\mathsf{oNIP}_{\mathsf{BBS}}$). *For the restricted DDH oracle* $\mathrm{rDDH}((p, G, \mathbb{G}), x, X, \cdot)$, *there exists a simulator* $\mathsf{Sim} = (\mathsf{Sim}_{\mathsf{Setup}}, \mathsf{Sim}_{\mathsf{Iss}})$ *and such that for any adversary* $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{zk}}_{\mathsf{oNIP}_{\mathsf{BBS}}, \mathsf{Sim}, \mathrm{rDDH}}(\mathcal{A}, \lambda) = 0$.

Proof. Consider the following simulator Sim:

- $\mathsf{Sim}_{\mathsf{Setup}}(p, G, \mathbb{G})$: Sample $w \in \mathbb{Z}_p$ and return $(\mathsf{par}_{\mathsf{oNIP}} = (p, G, \mathbb{G}, W), \mathsf{td} = w)$.
- $\mathsf{Sim}^{\mathrm{rDDH}}_{\mathsf{Iss}}(\mathsf{td}, X, \mathsf{umsg}_1 = (A, B))$: Query $\mathrm{rDDH}((p, G, \mathbb{G}), x, X, (A, B))$ and if the oracle outputs 0, abort. Otherwise, sample $s_0, c_0, r_1 \leftarrow_{\$} \mathbb{Z}_p$ and set $(R_{0,G}, R_{0,A}) \leftarrow (s_0 G - c_0 X, s_0 A - c_0 B)$ and $R_1 \leftarrow r_1 G$ and return these elements. On the next round with $\mathsf{umsg}_2 = c$, return $c_0, c_1 = c - c_0, s_0, s_1 = r_1 + c_1 \cdot w$. (For simplicity, we assume c_0, c_1 are both send – but in the protocol, only one can be derived from the other.)

To see that the distribution of the view of $\mathcal{A}$ is identical in ZK_0 and ZK_1 games, we consider the following:

- The distribution on $\mathsf{par}_{\mathsf{oNIP}}$ is identical to $\mathsf{oNIP.Setup}$, since W is still uniformly random.
- Next, because the simulator aborts correctly with the help of the oracle rDDH, we only have to consider the case when $xA = B$. Now, it is easy to see that the distributions of $(R_{0,G}, R_{0,A}, R_1, c_0, c_1, s_0, s_1)$ conditioned on (A, B, c) are identical between the two games.

□

Lemma 5.11 (Obliviousness of $\mathsf{oNIP}_{\mathsf{BBS}}$). *Let* $\mathsf{Sim}_{\mathsf{Gen}}$ *be the simulator for global parameters generator* GGen. *There exists a simulator* $\mathsf{Sim} = (\mathsf{Sim}_{\mathsf{Setup}}, \mathsf{Sim}_{\mathsf{U}}, \mathsf{Sim}_{\mathrm{Pf}})$ *such that*

- *For any adversary* $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{par\text{-}indist}}_{\mathsf{oNIP}_{\mathsf{BBS}}, \mathsf{Sim}}(\mathcal{A}, \lambda) = 0$.
- *For any adversary* $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{zk}}_{\mathsf{oNIP}_{\mathsf{BBS}}, \mathsf{Sim}_{\mathsf{Gen}}, \mathsf{Sim}}(\mathcal{A}, \lambda) = 0$.

Proof. As a reminder, $\mathsf{Sim}_{\mathsf{Gen}}$ does not output any trapdoor and samples $(p, G, \mathbb{G})$ as in GGen. Consider the following simulator Sim:

- $\mathsf{Sim}_{\mathsf{Setup}}(p, G, \mathbb{G})$: Sample $w \in \mathbb{Z}_p$ and return $(\mathsf{par}_{\mathsf{oNIP}} = (p, G, \mathbb{G}, W), \mathsf{td} = w)$.
- $\mathsf{Sim}_{\mathsf{U}}(\mathsf{td}, X)$: For the first move, return $(A, B) = (\beta G, \beta X)$ for $\beta \leftarrow_{\$} \mathbb{Z}_p$. For the second move, return $c \leftarrow_{\$} \mathbb{Z}_p$. At the end of the protocol, the simulator checks if the transcript $((A, B), (R_{0,G}, R_{0,H}, R_1), c, (c_0, c_1, s_0, s_1))$ satisfies the check identical to the one in $\mathsf{oNIP.U}_3$.
- $\mathsf{Sim}_{\mathrm{Pf}}(\mathsf{td}, X, (\tilde{A}, \tilde{B}))$: Compute the proof π by (1) sampling $s'_0, c'_0, r'_1 \leftarrow_{\$} \mathbb{Z}_p$ and set $(R'_{0,G}, R'_{0,A}) \leftarrow (s'_0 G - c'_0 X, s'_0 A - c'_0 B)$ and $R'_1 \leftarrow r'_1 G$, (2) computing $\mathsf{H}_c(X, \tilde{A}, \tilde{B}, R'_{0,G}, R'_{0,A}, R'_1)$, (3) Return $(c'_0, c'_1 = c - c'_0, s'_0, s'_1 = r'_1 + c'_1 \cdot w)$.

First, the distribution of $\mathsf{par}_{\mathsf{oNIP}}$ stays identical to that of $\mathsf{oNIP.Setup}$. Next, to show the advantage of $\mathcal{A}$ in the obliviousness game, we first only consider the game where $\mathcal{A}$ only *starts* 1 *session*. Note that we can easily extend this to Q sessions via standard hybrid argument, since the reduction could use the trapdoor (in the OBLV game the adversary knows the trapdoor) to simulate other sessions. In the following, we follow similar proof strategy from [OTZZ24].

To show indistinguishability, we first w.l.o.g. assume that $\mathcal{A}$'s randomness is fixed and it finishes the proof issuance session and sees the proof π. Also, we remark again that the game only starts the issuance protocol if a valid statement is given i.e., $\tilde{B} = x\tilde{A}$. We define the view of $\mathcal{A}$ after its execution as $V_{\mathcal{A}} = (W, X, (\tilde{A}, \tilde{B}), T, \pi)$ where T is the transcript of the protocol and π is the proof from Pf defined as $T := (A, B, R_{0,G}, R_{0,A}, R_1, c, c_0, c_1, s_0, s_1)$ and $\pi := (c'_0, c'_1, s'_0, s'_1)$. For simplicity, we assume c_0, c_1 are both sent. Since the randomness of $\mathcal{A}$ is fixed, we only consider the randomness of the honest user (i.e., $\mathrm{U}_1, \mathrm{U}_2$) and the simulator $\mathsf{Sim}_{\mathsf{U}}, \mathsf{Sim}_{\mathrm{Pf}}$. Denote η_b as the randomness of the honest user/simulator in the OBLV_b game, which are of the form

$$\eta_0 = (\beta, \gamma_0, \gamma_1, \delta_0, \delta_1)\ , \eta_1 = (\beta, \bar{c}, \bar{c}'_0, \bar{s}'_0, \bar{r}'_1)\ .$$

Note that $\bar{(\cdot)}$ is used to distinct the value in the transcript and the randomness of the simulator. Now, we only need to show that the distribution of $V_{\mathcal{A}}$ is identical in both cases of $b = 0, b = 1$, which we do so by showing that for any fixed view $\varDelta$ where $\mathsf{Pr}[V_{\mathcal{A}} = \varDelta | b = 1] > 0$, there is a unique randomness η_0, η_1 which results in $V_{\mathcal{A}} = \varDelta$ for both cases. Thus, proving that the probability of $V_{\mathcal{A}} = \varDelta$ are $1/p^5$ in both cases. (We note some abuse of notations here, and denote values in $\varDelta$ using the corresponding letters for the random variables in $V_{\mathcal{A}}$.)

For $b = 0$, $V_{\mathcal{A}} = \varDelta$ if and only if

$$\beta = \mathrm{dlog}_G(A - \tilde{A}),\ \forall i \in \{0, 1\} : \delta_i = s'_i - s_i,\ \gamma_i = c'_i - c_i\ .$$

The if direction ($\Rightarrow$) follows easily from the equations. The only-if direction ($\Leftarrow$) follows similarly from the blindness proof in [CATZ24]. In particular, β fixes the first message of the user to be (A, B) since $\tilde{B} = \mathrm{dlog}_G X\tilde{A}$. Then, by inspection and the fact that π is valid, the user sense $c = c_0 + c_1$ in the second move and the final proof is π.

For $b = 1$, $V_{\mathcal{A}} = \varDelta$ if and only if

$$\beta = \mathrm{dlog}_G(A),\ \bar{c} = c,\ \bar{c}'_0 = c'_0,\ \bar{s}'_0 = s'_0,\ \bar{r}'_1 = s'_1 - c'_1\mathrm{dlog}_G(W)\ .$$

The if direction ($\Rightarrow$) follows easily from the equations and the fact that the final proof π verifies. For the only-if direction, β ensures that $(\beta G, \beta X) = (A, B)$, $\bar{c}$ ensures that the second user message is c. Finally, because the final proof is valid, $c'_0 + c'_1 = \mathsf{H}_c(X, \tilde{A}, \tilde{B}, R'_{0,G}, R'_{0,A}, R'_1)$ where $R'_{0,G}, R'_{0,A}, R'_1$ are defined as in the verification algorithm. Then, the values of $\bar{c}'_0, \bar{s}'_0, \bar{r}'_1$ ensures that the proof π is exactly what is in the transcript $\varDelta$. $\square$

6 Instantiation from DDH

In this section, we instantiate our generic construction with a DDH-based KVAC by Chase, Meiklejohn, and Zaverucha's [CMZ14] and a corresponding oNIP scheme. We first introduce the underlying algebraic MAC in Section 6.1. Then, we discuss the DDH-based KVAC in Section 6.2, and the oNIP in Section 6.3. Finally, we discuss the SAAC instantiation in Section 5.4.

Global parameters generator. Following the syntax in Section 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 returns $\mathsf{par}_g = (p, G, \mathbb{G}, H)$. For security of both KVAC and oNIP, we define the simulator $\mathsf{Sim}_{\mathsf{Gen}}$ which samples $(p, G, \mathbb{G})$ from $\mathsf{GGen}(1^\lambda)$ and $H = vG$ with a trapdoor $v \leftarrow_{\$} \mathbb{Z}_p^*$. It is easy to see that the security of

6.1 DDH-based MAC

In Figure 16, we describe a variant of the DDH-based MAC introduced by Chase, Meiklejohn, and Zaverucha [CMZ14]. Everything is roughly the same, with the only difference being that zH is included in ipk, which we justify later in Section 6.2. A tag for message $\boldsymbol{m} = (m_i)_{i=1}^{\ell}$ is

$$(S_w, S_x, S_y, S_z) := (U \leftarrow\$\ \mathbb{G}, (x_0 + \textstyle\sum_{i=1}^{\ell} x_i m_i)U, (y_0 + \sum_{i=1}^{\ell} y_i m_i)U, zU)$$

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. The following theorem, proved in Section 6.5, establishes the UFCMA security of $\mathsf{MAC}_{\mathsf{DDH}}$ against any adversary with access to the $\mathcal{O}_{\mathsf{SVerDDH}}$ oracle (defined in Figure 16). The verification of $\mathsf{MAC}_{\mathsf{DDH}}$ can also be simulated by $\mathcal{O}_{\mathsf{SVerDDH}}$, so in some sense we have shown a stronger security notion for this scheme than prior works. An outline of the security proof for this scheme follows:

1. We generate the parameters and ipk in an indistinguishable way which allows us to simulate the $\mathcal{O}_{\mathsf{SVerDDH}}$ oracle, and the winning condition at the end of the game, using the twin Diffie-Hellman technique [CKS08]. In this step, we deviate from [CMZ14] in how we generate ipk to be able to simulate $\mathcal{O}_{\mathsf{SVerDDH}}$ instead of the verification algorithm $\mathsf{MAC}_{\mathsf{DDH}}.\mathsf{Ver}$
2. We show one-by-one that each MAC oracle query reveals nothing about $\boldsymbol{x}$. To do this, we use DDH to introduce noise into how we compute S_x which allows us to argue that each S_x is uniformly random.
3. After all of these transitions, the verification equation uses a value (essentially x_0) which is information-theoretically hidden. At this point, a forgery can be valid with only negligible probability.

Theorem 6.1. *Let* GGen *be a group generator that outputs groups of prime order* $p = p(\lambda)$*, and let* $\mathsf{MAC}_{\mathsf{DDH}} = \mathsf{MAC}_{\mathsf{DDH}}[\mathsf{GGen}]$*. Additionally, let* $\mathcal{O}_{\mathsf{SVerDDH}}$ *be as described in Figure 16. For any adversary* $\mathcal{A}$ *making at most* $q_{\mathcal{O}_{\mathsf{SVerDDH}}} = q_{\mathcal{O}_{\mathsf{SVerDDH}}}(\lambda)$ *queries to* $\mathcal{O}_{\mathsf{SVerDDH}}$ *and* $q_m = q_m(\lambda)$ *queries to* $\mathsf{MAC}_{\mathsf{DDH}}.\mathsf{M}$ *and running in time* $t_{\mathcal{A}} = t_{\mathcal{A}}(\lambda)$*, there exists an adversary* $\mathcal{B}_{\mathsf{DDH}}$ *(technically* q_m *different ones) running in time roughly* $t_{\mathcal{A}}$ *such that*

$$\mathsf{Adv}^{\mathsf{ufcma}}_{\mathsf{MAC}_{\mathsf{DDH}},\mathcal{O}_{\mathsf{SVerDDH}}}(\mathcal{A},\lambda) \leqslant q_m \cdot \mathsf{Adv}^{\mathrm{ddh}}_{\mathsf{GGen}}(\mathcal{B}_{\mathsf{DDH}},\lambda) + \frac{3q_{\mathcal{O}_{\mathsf{SVerDDH}}} + 3}{p}$$

6.2 DDH-based KVAC

We first discuss the DDH-based KVAC in [CMZ14], building on top of $\mathsf{MAC}_{\mathrm{DDH}}$. The credential for $\boldsymbol{m}$ is exactly a $\mathsf{MAC}_{\mathrm{DDH}}$ tag. For blind issuance in [CMZ14], the 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 $(S'_w = rS_w, C_x = rS_x + r_x H, C_y = rS_y + r_y H, S'_z = rS_z)$ for $r \leftarrow\$\ \mathbb{Z}_p^*, r_x, r_y \leftarrow\$\ \mathbb{Z}_p$. Then, the user computes commitments $C_i = m_i U' + r_i G$ to their attributes. With U' and $(C_i)_{i=1}^{\ell}$, the issuer can use their secret key to compute (for example) $\tilde{V}_x = x_0 U' + \sum_{i=1}^{\ell} x_i C_i = (x_0 + \sum_{i=1}^{\ell} x_i m_i)U' + \sum_{i=1}^{\ell} r_i X_i$ which is close to C_x, but with added randomness from the blinding. Hence, the user also sends $\Gamma_x := \sum_{i=1}^{\ell} r_i X_i - r_x H$ (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 Figure 17, then made the following changes to their scheme:

1. **Public key:** In [CMZ14], 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 uniquely determined (binding is computational), which is insufficient for our SAAC compiler. We (a) instead include ElGamal ciphertexts of x_0, y_0 (security is not affected), and (b) 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.[7]

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

$\mathsf{MAC}_{\mathsf{DDH}}.\mathsf{Setup}(1^\lambda)$:

$(p, G, \mathbb{G}) \leftarrow \mathsf{GGen}(1^\lambda)$
$H \leftarrow\$ \mathbb{G}$
return $(p, G, \mathbb{G}, H)$

$\mathsf{MAC}_{\mathsf{DDH}}.\mathsf{KeyGen}(p, G, \mathbb{G}, H)$:

$z \leftarrow\$ \mathbb{Z}_p$
$\boldsymbol{x} := (x_i)_{i=0}^{\ell} \leftarrow\$ \mathbb{Z}_p^{\ell+1}$
$\boldsymbol{y} := (y_i)_{i=0}^{\ell} \leftarrow\$ \mathbb{Z}_p^{\ell+1}$
for $i \in [\ell]$ **do**
  $X_i \leftarrow x_i H; Y_i \leftarrow y_i H$
$Z \leftarrow zH$
return $(\mathsf{sk} := (z, \boldsymbol{x}, \boldsymbol{y}), \mathsf{ipk} := ((X_i)_{i=1}^{\ell}, (Y_i)_{i=1}^{\ell}, Z))$

$\mathsf{MAC}_{\mathsf{DDH}}.\mathsf{M}(\mathsf{par}, \mathsf{sk}, \boldsymbol{m} \in \mathbb{Z}_p^{\ell})$

$r \leftarrow\$ \mathbb{Z}_p$
$S_w \leftarrow rG; S_z \leftarrow rzG$
$S_x \leftarrow r(x_0 + \sum_{i=1}^{\ell} m_i x_i)G$
$S_y \leftarrow r(y_0 + \sum_{i=1}^{\ell} m_i y_i)G$
return (S_w, S_x, S_y, S_z)

$\mathsf{MAC}_{\mathsf{DDH}}.\mathsf{Ver}(\mathsf{par}, \mathsf{sk}, \boldsymbol{m} \in \mathbb{Z}_p^{\ell}, \sigma)$

$(S_w, S_x, S_y, S_z) \leftarrow \sigma$
return $(zS_w = S_z) \wedge$
  $((x_0 + \sum_{i=1}^{\ell} m_i x_i)S_w = S_x) \wedge$
  $((y_0 + \sum_{i=1}^{\ell} m_i y_i)S_w = S_y) \wedge S_w \neq 0_{\mathbb{G}}$

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

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

Fig. 16. $\mathsf{MAC}_{\mathsf{DDH}} = \mathsf{MAC}_{\mathsf{DDH}}[\mathsf{GGen}]$ Scheme and Oracle $\mathcal{O}_{\mathsf{SVerDDH}}$.

2. **Blind Issuance:** In [CMZ14], users individually encrypt each m_i, and let the issuer computes and sends ciphertexts of S_x, S_y. Observe that pk contains $X_i = x_i H$, $Y_i = y_i H$ for $i \in [\ell]$, so the user can compute ciphertexts of $\sum_{i=1}^{\ell} m_i X_i$ and $\sum_{i=1}^{\ell} m_i Y_i$, 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.

<u>RELEVANT PROOF SYSTEMS.</u> 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.

$$\mathsf{R}_{\mathsf{com}} := \left\{ \begin{matrix} ((\widetilde{E}_x, \widetilde{E}_y, D, (X_i)_{i=1}^{\ell}, (Y_i)_{i=1}^{\ell}, \psi), \\ (u_x, u_y, \boldsymbol{m} = (m_i)_{i=1}^{\ell})) \end{matrix} : \begin{matrix} \widetilde{E}_x = (u_x G, u_x D + \sum_{i=1}^{\ell} m_i X_i) \\ \widetilde{E}_y = (u_y G, u_y D + \sum_{i=1}^{\ell} m_i Y_i), \psi(\boldsymbol{m}) = 1 \end{matrix} \right\}$$

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

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

The first proof system Π_{com} is used for the user to prove knowledge of openings to the ciphertexts E_x, E_y during issuance. We require Π_{com} to be straightline-extractable for a relaxed relation $\widetilde{\mathsf{R}}_{\mathsf{com}} \supseteq \mathsf{R}_{\mathsf{com}}$ defined as

$$\widetilde{\mathsf{R}}_{\mathsf{com}} := \left\{ \begin{matrix} ((E_x, E_y, D, (X_i)_{i=1}^{\ell}, (Y_i)_{i=1}^{\ell}, \psi), \\ (u_x, u_y, \boldsymbol{m} = (m_i)_{i=1}^{\ell})) \end{matrix} : \begin{matrix} (\sum_{i=1}^{\ell} m_i X_i = \sum_{i=1}^{\ell} m_i Y_i = 0_{\mathbb{G}} \wedge \\ \boldsymbol{m} \neq \boldsymbol{0}) \vee \\ (E_x = (u_x G, u_x D + \sum_{i=1}^{\ell} m_i X_i) \wedge \\ E_y = (u_y G, u_y D + \sum_{i=1}^{\ell} m_i Y_i) \wedge \\ \psi(\boldsymbol{m}) = 1) \end{matrix} \right\}.$$

and it is instantiated using a variant of the Fischlin transform [Fis05, Ks22], which we describe in Appendix C. The proof systems Π_σ and Π_{pub} are used for proving validity of the issued credentials by the issuer and

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

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

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

$\boldsymbol{x}, \boldsymbol{y} \leftarrow_\$ \mathbb{Z}_p^{\ell+1}; z, t_x, t_y \leftarrow_\$ \mathbb{Z}_p$

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

$\mathsf{sk} \leftarrow (\boldsymbol{x}, \boldsymbol{y}, z, t_x, t_y)$

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

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

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

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

 then abort

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

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

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

$\pi_\sigma \leftarrow \Pi_\sigma.\mathsf{Prove}^{\mathsf{H}_1}((M^\sigma_{G,H,S_w,D,E_x,E_y},$

 $(\tilde{E}_x, \tilde{E}_y, Z, \mathsf{ct}_x, \mathsf{ct}_y)), (z, x_0, y_0, r^{-1}, t_x, t_y, \gamma_x, \gamma_y))$

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 $\Pi_{\mathsf{pub}}.\mathsf{Ver}^{\mathsf{H}_2}((M^{\mathsf{pub}}_{G,H,S'_w,\boldsymbol{X},\boldsymbol{Y}}, ((C_i)_{i\in[\ell]\setminus\boldsymbol{I}},$

 $(C_i - a_i S'_w)_{i\in\boldsymbol{I}}, \Gamma_x, \Gamma_y)), \pi_{\mathsf{pub}}, (\phi_{\boldsymbol{I},\boldsymbol{a}}, \mathsf{nonce}))$

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

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

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

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

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

 $(u_x, u_y, \boldsymbol{m}))$

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

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

if $\Pi_\sigma.\mathsf{Ver}^{\mathsf{H}_1}((M^\sigma_{G,H,S_w,D,E_x,E_y},$

 $(\tilde{E}_x, \tilde{E}_y, Z, \mathsf{ct}_x, \mathsf{ct}_y)), \pi_\sigma) = 0$

 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. 17. Scheme $\mathsf{KVAC}_{\mathsf{DDH}} = \mathsf{KVAC}_{\mathsf{DDH}}[\mathsf{Gen}_{\mathsf{DDH}}]$. $\Pi_{\mathsf{com}}, \Pi_\sigma, \Pi_{\mathsf{pub}}$ are NIZKs for $\mathsf{R}_{\mathsf{com}}, \mathsf{R}_\sigma, \mathsf{R}_{\mathsf{pub}}$ defined in Section 6, 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}.

showing the credentials by the users, respectively. These proof systems are instantiated using the proof system Lin for linear relations on $\mathbb{G}$ (described in Section 2), with the corresponding linear maps $M^\sigma_{G,H,S_w,D,E_x,E_y}$ and $M^{\mathsf{pub}}_{G,H,S_w,\boldsymbol{X},\boldsymbol{Y}}$ for the relations R_σ and $\mathsf{R}_{\mathsf{pub}}$, analogously defined to what was done in Section 5.2 (omitting the explicit representation for brevity).

<u>Key-depedent verification induced-relation.</u> The algorithm $\mathsf{SVer}_{\mathsf{key}}$ induces the relation family $\mathsf{R}_{\mathsf{DDH}}$ (defined below), parameterized by $\mathsf{par}_g = (p, G, \mathbb{G}, H)$ (which we omit in the subscript), for which we give a corresponding oNIP protocol.

$$\mathsf{R}_{\mathsf{DDH}} := \left\{ \begin{array}{l} ((\mathsf{pk} = ((X_i)_{i=1}^\ell, (Y_i)_{i=1}^\ell, Z, \mathsf{ct}_x, \mathsf{ct}_y), \\ \tau_{\mathsf{key}} = (S_w, (C_i)_{i\in[\ell]}, \zeta_x, \zeta_y, S_z)) \\ \mathsf{sk} = ((x_i)_{i=0}^\ell, (y_i)_{i=0}^\ell, z, t_x, t_y)) \end{array} : \begin{array}{l} Z = zH, S_w \neq 0_{\mathbb{G}}, S_z = zS_w \\ \forall i \in [\ell] : X_i = x_i H, Y_i = y_i H \\ \zeta_x = x_0 S_w + \sum_{i=1}^\ell x_i C_i, \zeta_y = y_0 S_w + \sum_{i=1}^\ell y_i C_i \\ \mathsf{ct}_x = (t_x G, t_x H + x_0 G), \mathsf{ct}_y = (t_y G, t_y H + y_0 G) \end{array} \right\}. \quad (2)$$

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 $\mathsf{Show}_{\mathsf{key}}$. We further note that checking if the augmented statement $\tau_{\mathsf{key}} = (S_w, (C_i)_{i\in[\ell]}, \zeta_x, \zeta_y, S_z)$ can be done using the oracle $\mathcal{O}_{\mathsf{SVerDDH}}$ described in Figure 17.

<u>Correctness.</u> Correctness of $\mathsf{KVAC}_{\mathsf{DDH}}$ follows from η-correctness of Π_{com}, perfect correctness of Π_σ and Π_{pub}, and inspecting the algebra. In particular, the correctness error of the scheme is $\eta(\lambda)$.

UNFORGEABILITY. The following lemma establishes the unforgeability of $\mathsf{KVAC}_{\mathsf{DDH}}$ against adversaries with access to the $\mathcal{O}_{\mathsf{SVerDDH}}$ oracle (described in Figure 17). Then, we give a reduction from unforgeability of $\mathsf{MAC}_{\mathsf{DDH}}$ (established in Theorem 6.1) to that of $\mathsf{KVAC}_{\mathsf{DDH}}$'s. We remark that with our stronger unforgeability requirement of KVAC, there are several non-trivial steps in the proof:

(1) We need to take into account the attributes extracted that is in the relaxed relation $\widetilde{\mathsf{R}}_{\mathsf{com}}$ but not in $\mathsf{R}_{\mathsf{com}}$. To rule out this event, we give a reduction to the security of $\mathsf{MAC}_{\mathsf{DDH}}$ using the structure of $\widetilde{\mathsf{R}}_{\mathsf{com}}$.
(2) We give a careful rewinding argument to extract a MAC forgery from the KVAC forgery. Our reduction simulates the showings honest users by querying for a tag on uniformly random attributes. Crucially, these attributes need to be hidden from the view of the adversary, in order for the extracted forgery to be fresh with high probability.

Lemma 6.2 (Unforgeability of $\mathsf{KVAC}_{\mathsf{DDH}}$). *Let $\mathsf{Gen}_{\mathsf{DDH}}$ be a global parameters generator defined in Section 6 which outputs a group of prime order $p = p(\lambda)$ and a generator H, $\mathsf{Ext}_{\mathsf{com}}$ be an extractor for knowledge soundness of Π_{com}, and Sim_σ be a ZK simulator for Π_σ. Define $\mathsf{Ext}_{\mathsf{DDH}} := (\mathsf{Ext}_{\mathsf{Setup}}, \mathsf{Ext}_{\mathsf{iss}})$ as follows:*

- $\mathsf{Ext}_{\mathsf{Setup}}$ *on input* $\mathsf{par}_g = (p, G, \mathbb{G}, H), 1^\ell$ *returns* $\mathsf{par} = (p, G, \mathbb{G}, H, \ell)$ *without any trapdoor.*
- $\mathsf{Ext}_{\mathsf{iss}}$ *on input* $(\mu = (\widetilde{E}_x, \widetilde{E}_y, D, \pi_{\mathsf{com}}), \psi)$ *returns* $(u_x, u_y, \boldsymbol{m}) \leftarrow \mathsf{Ext}_{\mathsf{com}}^{\mathsf{H}_0}(\mathcal{Q}, (\boldsymbol{X}, \boldsymbol{Y}, \widetilde{E}_x, \widetilde{E}_y, D, \psi), \pi_{\mathsf{com}})$.

Then,

- *For any adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{par\text{-}indist}}_{\mathsf{KVAC}_{\mathsf{DDH}},\mathsf{Ext}_{\mathsf{DDH}}}(\mathcal{A}, \lambda) = 0$.*
- *Let $\mathcal{A}$ be an adversary against the $(\mathsf{Ext}_{\mathsf{DDH}}, \mathcal{O}_{\mathsf{SVerDDH}})$-unforgeability of $\mathsf{KVAC}_{\mathsf{DDH}} = \mathsf{KVAC}_{\mathsf{DDH}}[\mathsf{GGen}]$, running in time $t_{\mathcal{A}} = t_{\mathcal{A}}(\lambda)$ making at most $q_{h_0} = q_{h_0}(\lambda), q_{h_1} = q_{h_1}(\lambda), q_{h_2} = q_{h_2}(\lambda), q_{\mathsf{iss}} = q_{\mathsf{iss}}(\lambda), q_{\mathsf{Show}} = q_{\mathsf{Show}}(\lambda), q_{\mathcal{O}_{\mathsf{SVerDDH}}} = q_{\mathcal{O}_{\mathsf{SVerDDH}}}(\lambda)$ queries to $\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2$, Iss, $\mathrm{SH}_{\mathsf{key}}$, and $\mathcal{O}_{\mathsf{SVerDDH}}$ respectively. Let $q = q_{h_2} + q_{\mathsf{iss}} + 2q_{\mathsf{Show}}$. There exist adversaries $\mathcal{B}_{\mathsf{ufcma}}$, $\mathcal{B}'_{\mathsf{ufcma}}$ (playing the $\mathcal{O}_{\mathsf{SVerDDH}}$-UFCMA game of $\mathsf{MAC}_{\mathrm{DDH}}$), $\mathcal{B}_{\mathsf{com}}$ (playing the KSND game of Π_{com}), $\mathcal{B}_{\mathsf{DDH}}$ (playing the DDH game), $\mathcal{B}_{\mathrm{dlog}}$, $\mathcal{B}'_{\mathrm{dlog}}$ (playing the DL game), and $\mathcal{B}_\sigma$ (playing the ZK game of Π_σ) such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{unf}}_{\mathsf{KVAC}_{\mathsf{DDH}},\mathsf{Ext}_{\mathsf{DDH}},\mathcal{O}_{\mathsf{SVerDDH}}}(\mathcal{A}, \lambda) \leqslant & \sqrt{q \cdot \left(\mathsf{Adv}^{\mathsf{ufcma}}_{\mathsf{MAC}_{\mathsf{DDH}},\mathcal{O}_{\mathsf{SVerDDH}}}(\mathcal{B}'_{\mathsf{ufcma}}, \lambda) + \frac{1}{p^\ell}\right)} \\ & + \mathsf{Adv}^{\mathrm{dlog}}_{\mathsf{GGen}}(\mathcal{B}_{\mathrm{dlog}}, \lambda) + \mathsf{Adv}^{\mathrm{ddh}}_{\mathsf{GGen}}(\mathcal{B}_{\mathsf{DDH}}, \lambda) \\ & + \mathsf{Adv}^{\mathrm{ksnd}}_{\Pi_{\mathsf{com}},\mathsf{Ext}_{\mathsf{com}}}(\mathcal{B}_{\mathsf{com}}, \lambda) + \mathsf{Adv}^{\mathsf{ufcma}}_{\mathsf{GGen},\mathcal{O}_{\mathsf{SVerDDH}}}(\mathcal{B}_{\mathsf{ufcma}}, \lambda) \\ & + \mathsf{Adv}^{\mathsf{zk}}_{\Pi_\sigma,\mathsf{Sim}_\sigma}(\mathcal{B}_\sigma, \lambda) + \frac{q^2 + q + 3}{p} \ .\end{aligned}$$

Also, $\mathcal{B}_{\mathsf{ufcma}}, \mathcal{B}'_{\mathrm{dlog}}$ run in time roughly $t_{\mathcal{A}}$, and $\mathcal{B}'_{\mathsf{ufcma}}, \mathcal{B}_{\mathrm{dlog}}$ run in time roughly $2t_{\mathcal{A}}$. Moreover, $\mathcal{B}_{\mathsf{com}}$ makes at most q_{h_0} queries to H_0 and q_{iss} queries to $\mathcal{O}_{\mathsf{Ext}}$, while $\mathcal{B}_\sigma$ makes at most q_{h_1} queries to H_1. Additionally, $\mathcal{B}_{\mathsf{ufcma}}$ makes at most q_{iss} and $q_{\mathcal{O}_{\mathsf{SVerDDH}}}$ to its Iss and $\mathcal{O}_{\mathsf{SVerDDH}}$, respectively, and $\mathcal{B}'_{\mathsf{ufcma}}$ makes at most $2q_{\mathsf{iss}}$ and $2q_{\mathcal{O}_{\mathsf{SVerDDH}}}$ to its Iss and $\mathcal{O}_{\mathsf{SVerDDH}}$, respectively.

ANONYMITY. The following lemma establishes anonymity of $\mathsf{KVAC}_{\mathsf{BBS}}$ which follows from zero-knowledge properties of $\Pi_{\mathsf{com}}, \Pi_{\mathsf{pub}}$, soundness of Π_σ (to ensure that the maliciously issued credential is valid), and the DDH assumption (which comes into play when arguing that the ciphertexts E_x, E_y sent by the user during issuance hide the underlying attributes $\boldsymbol{m}$). The formal proof is given in Section 6.7.

Lemma 6.3 (Anonymity of $\mathsf{KVAC}_{\mathsf{DDH}}$). *Let $\mathsf{Gen}_{\mathsf{DDH}}$ be a global parameters generator defined in Section 6 which outputs a group of prime order $p = p(\lambda)$ and a generator H. Let $\mathsf{Sim}_{\mathsf{Gen}}$ be the simulator for the global parameters generator $\mathsf{Gen}_{\mathrm{DDH}}$ and $\mathsf{Sim}_{\mathsf{com}}, \mathsf{Sim}_{\mathsf{pub}}$ be the simulators for the zero-knowledge properties of $\Pi_{\mathsf{com}}, \Pi_{\mathsf{pub}}$. There exists a simulator $\mathsf{Sim}_{\mathsf{DDH}} = \mathsf{Sim}[\mathsf{Sim}_{\mathsf{com}}, \mathsf{Sim}_{\mathsf{pub}}]$, described in Figure 18, such that*

- *For any adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{par\text{-}indist}}_{\mathsf{KVAC}_{\mathsf{DDH}},\mathsf{Sim}_{\mathsf{DDH}}}(\mathcal{A}, \lambda) = 0$.*

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

$\mathsf{par} \leftarrow (p, G, \mathbb{G}, H)$
return $(\mathsf{par}, \mathsf{td}_{\mathsf{KVAC}} = \bot)$

$\mathsf{Sim}_{\mathsf{U}_1}(\mathsf{td} = v, \mathsf{pk}, \psi)$:

parse $(\boldsymbol{X}, \boldsymbol{Y}, Z, \mathsf{ct}_x, \mathsf{ct}_y) \leftarrow \mathsf{pk}$
$u_x, u'_x, u_y, u'_y \leftarrow\$ \mathbb{Z}_p$
$D \leftarrow\$ \mathbb{G}$
$\tilde{E}_x \leftarrow (u_x G, u'_x D), \tilde{E}_y \leftarrow (u_y G, u'_y G)$
// $\mathsf{Sim}_{\mathsf{com}}$ programs H_0
$\pi_{\mathsf{com}} \leftarrow\$ \mathsf{Sim}_{\mathsf{com}}^{\mathsf{H}_0}(D, E_x, E_y, \boldsymbol{X}, \boldsymbol{Y}, \psi)$
return $(\mu \leftarrow (D, \tilde{E}_x, \tilde{E}_y, \pi_{\mathsf{com}}),$
$\quad \mathsf{st}_{\mathsf{Sim}} \leftarrow (D, \tilde{E}_x, \tilde{E}_y))$

$\mathsf{Sim}_{\mathsf{U}_2}(\mathsf{st}_{\mathsf{Sim}}, \mathsf{imsg})$:

parse $(S_w, E_x, E_y, S_z, \pi_\sigma) \leftarrow \mathsf{imsg}$
if $\Pi_\sigma.\mathsf{Ver}^{\mathsf{H}_1}((\mathsf{pk}, D, \tilde{E}_x, \tilde{E}_y,$
$\quad S_w, E_x, E_y, S_z), \pi_\sigma) = 0$ **then**
$\quad$ **return** $\bot$
return 1

$\mathsf{Sim}_{\mathsf{Show}}(\text{“key”}, \mathsf{td} = (\mathsf{td}_g = v, \mathsf{td}_{\mathsf{KVAC}} = \bot), \mathsf{pk})$:

parse $(\boldsymbol{X}, \boldsymbol{Y}, Z, \mathsf{ct}_x, \mathsf{ct}_y) \leftarrow \mathsf{pk}$
$X_0 \leftarrow \mathsf{ct}_{x,1} - v\mathsf{ct}_{x,0}; Y_0 \leftarrow \mathsf{ct}_{y,1} - v\mathsf{ct}_{y,0}$
$r_w \leftarrow\$ \mathbb{Z}_p^*; r_1, \ldots, r_\ell \leftarrow\$ \mathbb{Z}_p; C_x, C_y \leftarrow\$ \mathbb{G}$
$S'_w \leftarrow r_w G; S'_z \leftarrow r v^{-1} Z$
for $i \in [\ell] : C_i \leftarrow r_i H$
$\Gamma_x \leftarrow r_w X_0 + \sum_{i=1}^{\ell} r_i X_i - C_x$
$\Gamma_y \leftarrow r_w Y_0 + \sum_{i=1}^{\ell} r_i Y_i - C_y$
$\tau_{\mathsf{key}} \leftarrow (S'_w, (C_i)_{i\in[\ell]}, C_x, C_y, \Gamma_x, \Gamma_y, S'_z)$
return $(\tau_{\mathsf{key}}, \mathsf{st} = (\mathsf{td}, \mathsf{pk}, \tau_{\mathsf{key}}))$

$\mathsf{Sim}_{\mathsf{Show}}(\text{“pub”}, \mathsf{st}, \phi_{\boldsymbol{I},\boldsymbol{a}}, \mathsf{nonce})$:

parse $(S'_w, (C_i)_{i\in[\ell]}, C_x, C_y, \Gamma_x, \Gamma_y, S'_z) \leftarrow \tau_{\mathsf{key}}$
// $\mathsf{Sim}_{\mathsf{pub}}$ programs H_2
$\pi_{\mathsf{pub}} \leftarrow \mathsf{Sim}_{\mathsf{pub}}^{\mathsf{H}_2}((M^{\mathsf{pub}}_{G,H,S'_w,\boldsymbol{X},\boldsymbol{Y}}, ((C_i)_{i\in[\ell]\setminus I}, \Gamma_x, \Gamma_y)), (\phi_{\boldsymbol{I},\boldsymbol{a}}, \mathsf{nonce}))$
return π_{pub}

Fig. 18. Simulator $\mathsf{Sim}_{\mathsf{DDH}} = \mathsf{Sim}[\mathsf{Sim}_{\mathsf{com}}, \mathsf{Sim}_{\mathsf{pub}}]$

- *For any adversary $\mathcal{A}$ playing the* Anon *game of* $\mathsf{KVAC}_{\mathsf{DDH}}$ *making at most* $q_{\mathsf{Show}} = q_{\mathsf{Show}}(\lambda), q_{h_0} = q_{h_0}(\lambda), q_{h_1} = q_{h_1}(\lambda), q_{h_2} = q_{h_2}(\lambda)$ *to the oracles* $\mathrm{SH}_{\mathsf{key}}, \mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2$*, respectively, and running in time* $t_{\mathcal{A}} = t_{\mathcal{A}}(\lambda)$*, there exist adversaries* $\mathcal{B}_{\mathsf{com}}, \mathcal{B}_{\mathsf{pub}}$ *(playing the* ZK *game of* Π_{com} *and* Π_{pub}*, resp.),* $\mathcal{B}_\sigma$ *(playing the soundness game of* Π_σ*), and* $\mathcal{B}_{\mathrm{DDH}}$ *(playing the* DDH *game) such that*

$$\mathsf{Adv}^{\mathsf{anon}}_{\mathsf{KVAC}_{\mathsf{DDH}},\mathsf{Sim}_{\mathsf{Gen}},\mathsf{Sim}_{\mathsf{DDH}}}(\mathcal{A}, \lambda) \leqslant \mathsf{Adv}^{\mathsf{zk}}_{\Pi_{\mathsf{com}},\mathsf{Sim}_{\mathsf{com}}}(\mathcal{B}_{\mathsf{com}}, \lambda) + \mathsf{Adv}^{\mathsf{zk}}_{\Pi_{\mathsf{pub}},\mathsf{Sim}_{\mathsf{pub}}}(\mathcal{B}_{\mathsf{pub}}, \lambda) + \mathsf{Adv}^{\mathrm{ddh}}_{\mathsf{GGen}}(\mathcal{B}_{\mathrm{DDH}}, \lambda) + 2\mathsf{Adv}^{\mathsf{sound}}_{\Pi_\sigma}(\mathcal{B}_\sigma, \lambda) + \frac{1}{p-1} .$$

  Additionally, $\mathcal{B}_{\mathsf{com}}$ *makes at most* q_{h_0} *queries to* H_0*,* $\mathcal{B}_\sigma$ *makes at most* q_{h_1} *queries to* H_1*, and* $\mathcal{B}_{\mathsf{pub}}$ *makes at most* q_{h_2} *queries to* H_2 *and* q_{Show} *queries to its prover oracle. Moreover,* $\mathcal{B}_{\mathrm{DDH}}$ *runs in time roughly* $t_{\mathcal{A}}$*.*

Integrity and validity of key generation. The following two lemmas establish the integrity (with respect to the simulators $\mathsf{Sim}_{\mathsf{Gen}}, \mathsf{Sim}_{\mathsf{DDH}}$ defined in Lemma 6.3) and validity of key generation (with respect to the extractor $\mathsf{Ext}_{\mathsf{DDH}}$ defined in Lemma 6.2) for $\mathsf{KVAC}_{\mathsf{DDH}}$.

Lemma 6.4 (Validity of Key Generation of $\mathsf{KVAC}_{\mathsf{DDH}}$). *Let* $\mathsf{Gen}_{\mathsf{DDH}}$ *and* $\mathsf{Ext}_{\mathsf{DDH}}$ *be as defined in Lemma 6.2. Then,* $\mathsf{KVAC}_{\mathsf{DDH}}$ *satisfies validity of key generation with respect to* Ext*.*

Proof. Note that since $\mathsf{Ext}_{\mathsf{DDH}}$ generates the public parameters as in Setup, we will consider any public keys pk generated honestly. Recall that the public keys are of the form $(\boldsymbol{X}, \boldsymbol{Y}, Z, \mathsf{ct}_x, \mathsf{ct}_y)$. Then, since G, H are generators of $\mathbb{G}$, we have that there exists *a unique secret key* $\mathsf{sk} = (\boldsymbol{x}, \boldsymbol{y}, z, t_x, t_y)$ such that $X_i = x_i H, Y_i = y_i H$ for $i \in [\ell]$, $Z = zH$, and $\mathsf{ct}_x = (t_x G, t_x H + x_0 G), \mathsf{ct}_y = (t_y G, t_y H + y_0 G)$. Therefore, the validity of key generation follows immediately. □

Lemma 6.5 (Integrity of $\mathsf{KVAC}_{\mathsf{DDH}}$). *Let* $\mathsf{Gen}_{\mathsf{DDH}}, \mathsf{Sim}_{\mathsf{Gen}}$*, and* $\mathsf{Sim}_{\mathsf{DDH}}$ *be as defined in Lemma 6.3. Let $\mathcal{A}$ be an adversary playing the integrity of issued credentials game of* $\mathsf{KVAC}_{\mathsf{BBS}}$ *with respect to the simulators*

$\mathsf{Sim}_{\mathsf{Gen}}$ *and* $\mathsf{Sim}_{\mathsf{DDH}}$ *making at most* $q_{h_1} = q_{h_1}(\lambda)$ *queries to* H_1*. There exists an adversary* $\mathcal{B}$ *against the soundness of* Π_σ *making at most* q_{h_1} *queries to* H_1 *such that*

$$\mathsf{Adv}^{\mathsf{integ}}_{\mathsf{KVAC}_{\mathsf{DDH}},\mathsf{Sim}_{\mathsf{Gen}},\mathsf{Sim}_{\mathsf{DDH}}}(\mathcal{A},\lambda) \leqslant \mathsf{Adv}^{\mathsf{sound}}_{\Pi_\sigma}(\mathcal{B},\lambda) .$$

Proof. First, we note that $\mathsf{Sim}_{\mathsf{Gen}}$ returns $(p, G, \mathbb{G}, H)$ which is identically distributed to Gen. Now, consider interaction with a malicious issuer as in the integrity game such that

- The adversary on input $\mathsf{par}_{\mathsf{KVAC}}, \mathsf{td} = (\mathsf{td}_g, \mathsf{td}_{\mathsf{KVAC}})$, generated from $\mathsf{Sim}_{\mathsf{Setup}}$, picks its own public key pk, a vector of attributes $\boldsymbol{m} \in \mathbb{Z}_p^\ell$ and a predicate ϕ such that $\phi(\boldsymbol{m}) = 1$.
- The honest user computes $D \leftarrow dG$ for $d \leftarrow\!\!\$\, \mathbb{Z}_p$ and $\widetilde{E}_x \leftarrow (s_xG, s_xD + \sum_{i=1}^{\ell} m_iX_i), \widetilde{E}_y \leftarrow (s_yG, s_yD + \sum_{i=1}^{\ell} m_iY_i)$ for $s_x, s_y \leftarrow\!\!\$\, \mathbb{Z}_p$ along with a proof of knowledge π_{com}.
- The adversary replies with $(S_w, E_x, E_y, S_z, \pi_\sigma)$. Then, the user checks π_σ uses d to compute $S_x \leftarrow E'_{x,1} - dE'_{x,0}, S_y \leftarrow E'_{y,1} - dE'_{y,0}$.

Then, consider the public key pk and $\tau_{\mathsf{key}} = (S'_w, (C_i)_{i\in[\ell]}, C_x, C_y, \Gamma_x, \Gamma_y, S'_z)$ such that $(\mathsf{pk}, \tau_{\mathsf{key}}) \notin \mathcal{L}_{V,\mathsf{par}_g}$. Since the public key $\mathsf{pk} = (\boldsymbol{X}, \boldsymbol{Y}, Z, \mathsf{ct}_x, \mathsf{ct}_y)$ fixes the underlying secret key $(\boldsymbol{x}, \boldsymbol{y}, z, t_x, t_y)$, $(\mathsf{pk}, \tau_{\mathsf{key}}) \notin \mathcal{L}_{V,\mathsf{par}_g}$ implies that one of the following is true:

$$C_x + \Gamma_x \neq x_0S'_w + \sum_{i=1}^{\ell} x_iC_i \;\;\vee\;\; C_y + \Gamma_y \neq y_0S'_w + \sum_{i=1}^{\ell} y_iC_i \;\;\vee\;\; S'_z \neq zS'_w . \tag{3}$$

Next, suppose that S_w, E_x, E_y, S_z which the issuer sends during the issuance protocol is such that $S_w = rH, E_x = (\gamma_xG, \gamma_xD + x_0S_w) + r\widetilde{E}_x, E_y = (\gamma_yG, \gamma_yD + x_0S_w) + r\widetilde{E}_y$, and $S_z = rzH$ for some $r \in \mathbb{Z}_p^*$, then $S_x = r(x_0 + \sum_{i=1}^{\ell} m_ix_i)H, S_y = r(y_0 + \sum_{i=1}^{\ell} m_iy_i)H$. With a similar argument from the anonymity proof, we have that this contradicts Equation (3).

Therefore, if $(\mathsf{pk}, \tau_{\mathsf{key}}) \notin \mathcal{L}_{V,\mathsf{par}_g}$, we have that (S_w, E_x, E_y, S_z) does not satisfy the equations defined by R_σ, and $\mathcal{A}$ breaks the soundness of π_σ, since the proof verifies. Hence, this implies the lemma. □

6.3 oNIP for DDH-based instantiation

In this section, we give the protocol $\mathsf{oNIP}_{\mathsf{DDH}} = \mathsf{oNIP}[\mathsf{Gen}_{\mathsf{DDH}}, \mathsf{R}_{\mathsf{DDH}}]$, in Figure 19 for the family of relations $\mathsf{R}_{\mathsf{DDH}}$ described in Equation (2), containing a statement pk, an augmented statement τ_{key} and witness sk.

We explicitly note that the relation induces the linear maps M_{Core} and $M_{\mathsf{Aug}} = M_{\mathsf{Aug},S_w,(C_i)_{i=1}^{\ell}}$, such that $M_{\mathsf{Core}}\mathsf{sk} = \mathsf{pk}, M_{\mathsf{Aug}}\mathsf{sk} = (\zeta_x \| \zeta_y \| S_z)$. More specifically, M_{Core} and M_{Aug} map elements from $\mathbb{Z}_p^{2\ell+5}$ to $\mathbb{G}^{2\ell+5}$ and $\mathbb{G}^3$, respectively, such that

$$\begin{aligned} M_{\mathsf{Core}}(\boldsymbol{x}\|\boldsymbol{y}\|z\|t_x\|t_y) &= ((x_iH)_{i\in[\ell]} \| (y_iH)_{i\in[\ell]} \| zH \| t_xG \| t_xH + x_0G \| t_yG \| t_yH + y_0G) \\ M_{\mathsf{Aug}}(\boldsymbol{x}\|\boldsymbol{y}\|z\|t_x\|t_y) &= (x_0S_w + \textstyle\sum_{i\in[\ell]} x_iC_i \| y_0S_w + \sum_{i\in[\ell]} y_iC_i \| zS_w) . \end{aligned}$$

Note that M_{Core} is a bijection since G and H are generators of $\mathbb{G}$ and the public key has unique underlying secret key.

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 $S_w = \alpha S'_w, C_i = \alpha C'_i + \beta_iH$ with random scalars $\alpha, \beta_1, \ldots, \beta_\ell$ and uses $\boldsymbol{X}, \boldsymbol{Y}$ in the public key to compute $\zeta_x = \alpha\zeta'_x + \sum_{i=1}^{\ell} \beta_iX_i, \zeta_y = \alpha\zeta'_y + \sum_{i=1}^{\ell} \beta_iY_i, S_z = \alpha S'_z$, which still preserves the membership of the language. The issuer then checks whether the rerandomized statement is in the language.

The following theorem then establishes the security properties of $\mathsf{oNIP}_{\mathsf{DDH}}$ with the proof given in Section 6.8. Most of the proofs follow from standard techniques as with $\mathsf{oNIP}_{\mathsf{BBS}}$, with an exception of obliviousness where we *inherently requires* the global trapdoor v to efficiently simulate honest users without knowing the augmented statement τ_{key}.

Algorithm $\mathsf{oNIP}_{\mathsf{DDH}}.\mathsf{Iss}_1(\mathsf{par}_{\mathsf{oNIP}}, \mathsf{sk}, \mathsf{umsg}_1)$:

$\textbf{parse } (S_w, (C_i)_{i=1}^{\ell}, \zeta_x, \zeta_y, S_z) \leftarrow \mathsf{umsg}_1$

$\textbf{if } M_{\mathsf{Aug}, S_w, (C_i)_{i=1}^{\ell}} \mathsf{sk} \neq (\zeta_x \| \zeta_y \| S_z)$

$\quad \textbf{then abort}$

$s_1, c_1 \leftarrow\$ \mathbb{Z}_p; r_0 \leftarrow\$ \mathbb{Z}_p^{2\ell+5}$

$\boldsymbol{R}_{0,\mathsf{Core}} \leftarrow M_{\mathsf{Core}} \boldsymbol{r}_0$

$\boldsymbol{R}_{0,\mathsf{Aug}} \leftarrow M_{\mathsf{Aug}, S_w, (C_i)_{i=1}^{\ell}} \boldsymbol{r}_0$

$R_1 \leftarrow s_1 G - c_1 W$

$\textbf{return } (\boldsymbol{R}_{0,\mathsf{Core}}, \boldsymbol{R}_{0,\mathsf{Aug}}, R_1)$

Algorithm $\mathsf{oNIP}_{\mathsf{DDH}}.\mathsf{Iss}_2(c)$:

$c_0 \leftarrow c - c_1; \boldsymbol{s}_0 \leftarrow \boldsymbol{r}_0 + c_0 \cdot \mathsf{sk}$

$\textbf{return } (c_0, \boldsymbol{s}_0, s_1)$

Algorithm $\mathsf{oNIP}_{\mathsf{DDH}}.\mathsf{U}_3(c_0, \boldsymbol{s}_0, s_1)$:

$c_1 \leftarrow c - c_0$

$\textbf{if } (\boldsymbol{R}_{0,\mathsf{Core}} + c_0 \cdot \mathsf{pk} \neq M_{\mathsf{Core}} \boldsymbol{s}_0) \vee$

$\quad (\boldsymbol{R}_{0,\mathsf{Aug}} + c_0 \cdot (\zeta_x \| \zeta_y \| S_z) \neq M_{\mathsf{Aug}, S_w, (C_i)_{i=1}^{\ell}} \boldsymbol{s}_0) \vee$

$\quad (R_1 + c_1 W \neq s_1 G) \textbf{ then abort}$

$c_0' \leftarrow c_0 + \gamma_0; \boldsymbol{s}_0' \leftarrow \boldsymbol{s}_0 + \boldsymbol{\delta}_0$

$c_1' \leftarrow c_1 + \gamma_1; s_1' \leftarrow s_1 + \delta_1$

$\textbf{return } \pi = (c_0', c_1', \boldsymbol{s}_0', s_1')$

Algorithm $\mathsf{oNIP}_{\mathsf{DDH}}.\mathsf{Ver}(\mathsf{par}_{\mathsf{oNIP}}, (\mathsf{pk}, \tau_{\mathsf{key}}), \pi)$:

$\textbf{parse } (S_w, (C_i)_{i=1}^{\ell}, \zeta_x, \zeta_y, S_z) \leftarrow \tau_{\mathsf{key}}$

$\textbf{parse } (c_0, c_1, \boldsymbol{s}_0, s_1) \leftarrow \pi$

$\boldsymbol{R}_{0,\mathsf{Core}} \leftarrow M_{\mathsf{Core}} \boldsymbol{s}_0 - c_0 \mathsf{pk}$

$\boldsymbol{R}_{0,\mathsf{Aug}} \leftarrow M_{\mathsf{Aug}, S_w, (C_i)_{i=1}^{\ell}} \boldsymbol{s}_0 - c_0 (\zeta_x \| \zeta_y \| S_z)$

$R_1 \leftarrow s_1 G - c_1 W$

$c \leftarrow \mathsf{H}_c(H, \mathsf{pk}, \tau_{\mathsf{key}}, \boldsymbol{R}_{0,\mathsf{Core}}, \boldsymbol{R}_{0,\mathsf{Aug}}, R_1)$

$\textbf{return } (c_0 + c_1 = c)$

Algorithm $\mathsf{oNIP}_{\mathsf{DDH}}.\mathsf{Setup}(\mathsf{par}_g = (p, G, \mathbb{G}, H))$:

$W \leftarrow\$ \mathbb{G}$

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

$\textbf{return } \mathsf{par}_{\mathsf{oNIP}} = (p, G, \mathbb{G}, H, W, \mathsf{H}_c)$

Algorithm $\mathsf{oNIP}_{\mathsf{DDH}}.\mathsf{U}_1(\mathsf{par}_{\mathsf{oNIP}}, (\mathsf{pk}, \tau_{\mathsf{key}}))$:

$\textbf{parse } ((X_i)_{i=1}^{\ell}, (Y_i)_{i=1}^{\ell}, Z, \mathsf{ct}_x, \mathsf{ct}_y) \leftarrow \mathsf{pk}$

$\textbf{parse } (S_w', (C_i')_{i=1}^{\ell}, \zeta_x', \zeta_y', S_z') \leftarrow \tau_{\mathsf{key}}$

$\textbf{parse } M_{\mathsf{Aug}} \leftarrow M_{\mathsf{Aug}, S_w', (C_i')_{i=1}^{\ell}}$

// Randomize the augmented statement.

$\textbf{if } S_w' = 0_{\mathbb{G}} \textbf{ then abort}$

$\alpha \leftarrow\$ \mathbb{Z}_p^*, \boldsymbol{\beta} \leftarrow\$ \mathbb{Z}_p^{\ell}$

$S_w \leftarrow \alpha S_w'; S_z \leftarrow \alpha S_z'$

$\textbf{for } i \in [\ell] \textbf{ do } C_i \leftarrow \alpha C_i' + \beta_i H$

$\zeta_x \leftarrow \alpha \zeta_x' + \sum_{i=1}^{\ell} \beta_i X_i$

$\zeta_y \leftarrow \alpha \zeta_y' + \sum_{i=1}^{\ell} \beta_i Y_i$

$\textbf{return } (S_w, S_z, (C_i)_{i=1}^{\ell}, \zeta_x, \zeta_y)$

Algorithm $\mathsf{oNIP}_{\mathsf{DDH}}.\mathsf{U}_2(\boldsymbol{R}_{0,\mathsf{Core}}, \boldsymbol{R}_{0,\mathsf{Aug}}, R_1)$:

// Derandomize $\boldsymbol{R}_{0,\mathsf{Aug}}$.

$\textbf{parse } ((R_{x,i}, R_{y,i})_{i=1}^{\ell}, R_z,$

$\quad R_{\mathsf{ct},x}, R_{\mathsf{ct},y}) \leftarrow \boldsymbol{R}_{0,\mathsf{Core}}$

$\textbf{parse } (R_{\zeta,x}, R_{\zeta,y}, R_{S,z}) \leftarrow \boldsymbol{R}_{0,\mathsf{Aug}}$

$\bar{R}_{\zeta,x} \leftarrow \alpha^{-1}(R_{\zeta,x} - \sum_{i=1}^{\ell} \beta_i R_{x,i})$

$\bar{R}_{\zeta,y} \leftarrow \alpha^{-1}(R_{\zeta,y} - \sum_{i=1}^{\ell} \beta_i R_{y,i})$

$\bar{\boldsymbol{R}}_{0,\mathsf{Aug}} \leftarrow (\bar{R}_{\zeta,x} \| \bar{R}_{\zeta,y} \| \alpha^{-1} R_{S,z})$

// Blind $\boldsymbol{R}_0, R_1$.

$\delta_1, \gamma_0, \gamma_1 \leftarrow\$ \mathbb{Z}_p; \boldsymbol{\delta}_0 \leftarrow\$ \mathbb{Z}_p^{2\ell+6}$

$\boldsymbol{R}_{0,\mathsf{Core}}' \leftarrow \boldsymbol{R}_{0,\mathsf{Core}} + M_{\mathsf{Core}} \boldsymbol{\delta}_0 - \gamma_0 \mathsf{pk}$

$\boldsymbol{R}_{0,\mathsf{Aug}}' \leftarrow \bar{\boldsymbol{R}}_{0,\mathsf{Aug}} + M_{\mathsf{Aug}} \boldsymbol{\delta}_0 - \gamma_0 (\zeta_x' \| \zeta_y' \| S_z')$

$R_1' \leftarrow R_1 + \delta_1 G - \gamma_1 W$

$c' \leftarrow\$ \mathsf{H}_c(H, \mathsf{pk}, \tau_{\mathsf{key}}, \boldsymbol{R}_{0,\mathsf{Core}}', \boldsymbol{R}_{0,\mathsf{Aug}}', R_1')$

$\textbf{return } c = c' - \gamma_0 - \gamma_1$

Fig. 19. Oblivious proof issuance $\mathsf{oNIP}_{\mathsf{DDH}} = \mathsf{oNIP}[\mathsf{Gen}_{\mathsf{DDH}}, \mathsf{R}_{\mathsf{DDH}}]$. We omitted the user and issuer's states and assume that any variable defined in the previous round is accessible in the next round.

Theorem 6.6. *Let* $\mathsf{Gen}_{\mathsf{DDH}}$ *be a global parameters generator defined in Section 6 and* $\mathcal{O}_{\mathsf{SVerDDH}}$ *be the oracle in Figure 17. 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* $\mathsf{Sim}_{\mathsf{Gen}}$.

6.4 DDH-based SAAC

The following corollary establishes the security of $\mathsf{SAAC}_{\mathsf{DDH}}$, a DDH-based instantiation of our generic SAAC construction from Section 4.2. The corollary immediately follows from Theorems 4.2 and 6.6 and Lemmas 6.2 to 6.5.

Corollary 6.7. *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 4.2. Then,* $\mathsf{SAAC}_{\mathsf{DDH}}$ *satisfies correctness, unforgeability, and anonymity (both in the ROM and assuming* DDH*).*

Integrity. Similar to $\mathsf{SAAC}_{\mathsf{BBS}}$, although we do not give a formal proof, strong integrity of $\mathsf{SAAC}_{\mathsf{DDH}}$ follows from the structure of $\mathsf{KVAC}_{\mathsf{DDH}}$'s public key, which uniquely fixes the secret key, and the soundness of Π_σ, which ensures validity of the (possibly maliciously) issued credentials.

6.5 Unforgeability Proof of $\mathsf{MAC}_{\mathsf{DDH}}$

Proof (Theorem 6.1). The proof is similar to Chase, Meiklejohn, and Zaverucha's UFCMVA proof [CMZ14], except our version of the scheme is slightly different since we publish zH, we consider a stronger security notion (UFCMA in the presence of $\mathcal{O}_{\mathsf{SVerDDH}}$), and we go about certain steps of the proof differently. Consider the following sequence of games.

$\mathbf{G}_1(\lambda)$: This is exactly $\mathcal{O}_{\mathsf{SVerDDH}}$-UFCMA for $\mathsf{MAC}_{\mathsf{DDH}}$.

$\mathbf{G}_2(\lambda)$: We modify Setup to trapdoor H: do $\beta \leftarrow\!\!\$\ \mathbb{Z}_p$ and set $H \leftarrow \beta G$. If $\beta = 0$, then abort. Also, modify KeyGen to do the following:

1. $(x'_i)_{i=0}^{\ell}, (y'_i)_{i=0}^{\ell}, (v_i)_{i=0}^{\ell} \leftarrow\!\!\$\ \mathbb{Z}_p^{\ell+1}$ and $z, s, t \leftarrow\!\!\$\ \mathbb{Z}_p$
2. Set $x_i \leftarrow \frac{x'_i}{\beta} + v_i$ and $y_i \leftarrow y'_i - s x_i$ for all $i \in [\ell]$. Set $y_0 \leftarrow \frac{y'_0}{\beta} - s x_0$ and $z \leftarrow \frac{z'}{\beta} - t$. Set $X_i \leftarrow x'_i G + v_i H$, $Y_i \leftarrow y'_i H - s X_i$, and $Z \leftarrow z' G - tH$.

Lastly, in $\mathsf{MAC}_{\mathsf{DDH}}.\mathsf{M}$, compute everything relative to H instead of G, i.e., do $S_w \leftarrow rH$, $S_z \leftarrow rzH$, $S_x \leftarrow (x_0 + \sum_{i=1}^{\ell} x_i m_i) S_w$ and $S_y \leftarrow (y_0 + \sum_{i=1}^{\ell} y_i m_i) S_w$. Everything is distributed exactly the same assuming that we do not abort due to $\beta = 0$, which occurs with probability $1/p$, so

$$\Pr[\mathbf{G_2}^{\mathcal{A}}(\lambda) = 1] \geqslant \Pr[\mathbf{G_1}^{\mathcal{A}}(\lambda) = 1] - \frac{1}{p}\ .$$

$\mathbf{G}_3(\lambda)$: We handle queries to $\mathcal{O}_{\mathsf{SVerDDH}}$ like so: on input $(\mathsf{par}, \mathsf{sk}, \zeta_x, \zeta_y, (C_i)_{i=1}^{\ell}, S_w, S_z)$ compute $b' \leftarrow z'(\zeta_y + s\zeta_x - \sum_{i=1}^{\ell} y'_i C_i) = y'_0(S_z + tS_w)$ and $b \leftarrow (zS_w = S_z \wedge \zeta_x = x_0 S_w + \sum_{i=1}^{\ell} x_i C_i \wedge \zeta_y = y_0 S_w + \sum_{i=1}^{\ell} y_i C_i$. If $b \neq b'$, then abort. Otherwise, output b. This change is perfect unless an abort happens. Call E_i the event that the game aborts on the i-th query to $\mathcal{O}_{\mathsf{SVerDDH}}$, and fix $i \in [q_{\mathcal{O}_{\mathsf{SVerDDH}}}]$. We will show that E_i occurs with negligible probability. Event E_i occurs only if the game has not aborted in a previous step, which means that, up to the i-th query, the game is perfectly indistinguishable from $\mathbf{G_3}$. Suppose that $b = 1$, which implies $S_z = zS_w$, $\zeta_x = \langle \boldsymbol{x}, S_w \| \boldsymbol{C} \rangle$, and $\zeta_y = \langle \boldsymbol{y}, S_w \| \boldsymbol{C} \rangle$. Observe that

$$\begin{aligned} \zeta_y = \langle \boldsymbol{y}, S_w \| \boldsymbol{C} \rangle &= \langle \boldsymbol{y'}, \frac{1}{\beta} S_w \| \boldsymbol{C} \rangle - s \langle \boldsymbol{x}, S_w \| \boldsymbol{C} \rangle \\ &= \langle \boldsymbol{y'}, \frac{1}{\beta} S_w \| \boldsymbol{C} \rangle - s\zeta_x\ , \end{aligned}$$

hence $z'(\zeta_y + s\zeta_x) = \beta(z + t)(\langle y', \frac{1}{\beta} S_w \| \boldsymbol{C} \rangle) = y'_0(S_z + tS_w) + z'(\sum_{i=1}^{\ell} y'_i C_i)$, so it must be the case that $b' = 1$ as well. On the other hand, suppose that $b' = 1$, meaning that $z'(\zeta_y + s\zeta_x - \sum_{i=1}^{\ell} y'_i C_i) = y'_0(S_z + tS_w)$. Define $\varDelta_x := \zeta_x - \langle \boldsymbol{x}, S_w \| \boldsymbol{C} \rangle$, $\varDelta_y := \zeta_y - \langle \boldsymbol{y}, S_w \| \boldsymbol{C} \rangle$, and $\varDelta_z := zS_w - S_z$, and note that $b = 1$ if only if $\varDelta_x, \varDelta_y, \varDelta_z$ are all zero. We have

$$\begin{aligned} z'(\zeta_y + s\zeta_x - \sum_{i=1}^{\ell} y'_i C_i) &= y'_0(S_z + tS_w) \\ z'\left(\varDelta_y + \langle \boldsymbol{y}, S_w \| \boldsymbol{C} \rangle + s\left(\varDelta_x + \langle \boldsymbol{x}, S_w \| \boldsymbol{C} \rangle\right) - \sum_{i=1}^{\ell} y'_i C_i\right) &= y'_0(zS_w - \varDelta_z + tS_w) \\ z'\left(\varDelta_y + \langle \boldsymbol{y} \rangle, S_w \| \boldsymbol{C} + s\left(\varDelta_x + \langle \boldsymbol{x}, S_w \| \boldsymbol{C} \rangle\right) - \sum_{i=1}^{\ell} (y_i + s x_i) C_i\right) &= y'_0(zS_w - \varDelta_z + tS_w) \\ z'\left(\varDelta_y + \langle \boldsymbol{y}, S_w \| \boldsymbol{C} \rangle + s\left(\varDelta_x + \langle \boldsymbol{x}, S_w \| \boldsymbol{C} \rangle\right) - \sum_{i=1}^{\ell} (y_i + s x_i) C_i\right) &= y'_0(zS_w - \varDelta_z + tS_w) \end{aligned}$$

$$
\begin{aligned}
z'\left(\Delta_y + s\Delta_x + (y_0 + sx_0)S_w\right) &= y_0'(zS_w - \Delta_z + tS_w) \\
z'\left(\Delta_y + s\Delta_x + \frac{y_0'}{\beta}S_w\right) &= y_0'(zS_w - \Delta_z + tS_w) \\
\beta(z+t)\left(\Delta_y + s\Delta_x + \frac{y_0'}{\beta}S_w\right) &= y_0'(zS_w - \Delta_z + tS_w) \\
\beta(z+t)\left(\Delta_y + s\Delta_x + \frac{y_0'}{\beta}S_w\right) &= y_0'((z+t)S_w - \Delta_z) \\
\beta(z+t)\left(\Delta_y + s\Delta_x\right) &= y_0'(-\Delta_z) \\
(z+t)\left(\Delta_y + s\Delta_x\right) &= (y_0 + sx_0)(-\Delta_z)\ .
\end{aligned}
$$

Recall that, up until the i-th query to $\mathcal{O}_{\mathsf{SVerDDH}}$, everything is exactly the same as in $\mathbf{G}_3$. This means that s and t are information-theoretically hidden from $\mathcal{A}$'s view as none of the values y_i', y_0', or z' were used in $\mathbf{G}_3$. We have $y_0 + sx_0 \neq 0$ with probability $1 - 1/p$, and in this case

$$
\frac{(z+t)(\Delta_y + s\Delta_x)}{y_0 + sx_0} = -\Delta_z\ . \tag{4}
$$

Additionally, $\Delta_y + s\Delta_x \neq 0$ with probability $1 - 1/p$ due to the fact that s is perfectly hidden from $\mathcal{A}$ and uniform in $\mathbb{Z}_p$. Lastly, since t is also hidden from $\mathcal{A}$ and uniform in $\mathbb{Z}_p$, so is the left-hand side of Equation (4). Thus $-\Delta_z = 0$ with probability $1/p$. In total, we have $\mathsf{Pr}[E_i] \leqslant \frac{1}{p} + \left(1 - \frac{1}{p}\right)\frac{1}{p} + \left(1 - \frac{1}{p}\right)^2 \frac{1}{p} \leqslant \frac{3}{p}$. Then

$$
\begin{aligned}
\left|\mathsf{Pr}[\mathbf{G}_3{}^{\mathcal{A}}(\lambda) = 1] - \mathsf{Pr}[\mathbf{G}_2{}^{\mathcal{A}}(\lambda) = 1]\right| &\leqslant \mathsf{Pr}[E_1 \vee E_2 \vee \ldots \vee E_{q_{\mathcal{O}_{\mathsf{SVerDDH}}}}] \\
&= \sum_{i=1}^{q_{\mathcal{O}_{\mathsf{SVerDDH}}}} \mathsf{Pr}[E_i] \leqslant \frac{3q_{\mathcal{O}_{\mathsf{SVerDDH}}}}{p}\ .
\end{aligned}
$$

$\mathbf{G}_4(\lambda)$: We handle queries to $\mathcal{O}_{\mathsf{SVerDDH}}$ like so: on input $(\zeta_x, \zeta_y, (C_i)_{i=1}^{\ell}, S_w, S_z)$, output 1 if and only if $z'(\zeta_y + s\zeta_x - \sum_{i=1}^{\ell} y_i'C_i) = y_0'(S_z + tS_w)$. We have

$$
\mathsf{Pr}[\mathbf{G}_4{}^{\mathcal{A}}(\lambda) = 1] \geqslant \mathsf{Pr}[\mathbf{G}_3{}^{\mathcal{A}}(\lambda) = 1]\ ,
$$

as the only way the games can differ is that the adversary can cause the game to abort in $\mathbf{G}_3$, and if this does not happen then everything is exactly the same, so an adversary that wins in the previous game necessarily wins in this game.

$\mathbf{G}_5(\lambda)$: Instead of $\mathsf{MAC}_{\mathsf{DDH}}.\mathsf{Ver}$ at the end of the game, we check

$$
y_0'(S_x^* - \langle \boldsymbol{v}, 1\|\boldsymbol{m}^*\rangle S_w^*) = \langle \boldsymbol{x}', 1\|\boldsymbol{m}^*\rangle (S_y^* + sS_x^* - \langle \boldsymbol{y}', 0\|\boldsymbol{m}^*\rangle)\ .
$$

This condition is implied by the previous winning condition. We can see this by plugging in definitions and winning conditions to the left-hand side as

$$
\begin{aligned}
y_0'(S_x^* - \langle \boldsymbol{v}, 1\|\boldsymbol{m}^*\rangle S_w^*) &= y_0'(\langle \boldsymbol{x}, 1\|\boldsymbol{m}^*\rangle S_w - \langle \boldsymbol{v}, 1\|\boldsymbol{m}^*\rangle S_w^*) \\
&= y_0'((\langle \boldsymbol{x}', 1\|\boldsymbol{m}^*\rangle/\beta + \langle \boldsymbol{v}, 1\|\boldsymbol{m}^*\rangle)S_w - \langle \boldsymbol{v}, 1\|\boldsymbol{m}^*\rangle S_w) \\
&= \langle \boldsymbol{x}', 1\|\boldsymbol{m}^*\rangle (y_0'/\beta) S_w^* \\
&= \langle \boldsymbol{x}', \boldsymbol{m}^*\rangle (S_y^* + sS_x^* - \langle \boldsymbol{y}', 0\|\boldsymbol{m}^*\rangle S_w)\ .
\end{aligned}
$$

Hence, an adversary that wins in the prior game must win in this game, meaning

$$
\mathsf{Pr}[\mathbf{G}_5{}^{\mathcal{A}}(\lambda) = 1] \geqslant \mathsf{Pr}[\mathbf{G}_4{}^{\mathcal{A}}(\lambda) = 1]\ .
$$

$\mathbf{G}_6(\lambda)$: We now revert to $H \leftarrow\!\!\$\, \mathbb{G}$ instead of generating a trapdoor. The only difference is that previously the game would abort if $\beta = 0$. Note that nowhere in the game do we use the values $(x_i)_{i=0}^{\ell}$, $(y_i)_{i=0}^{\ell}$ or β anymore, this is because we compute

$$\begin{aligned}
S_w &= rH \\
S_z &= rzH = r(z'/\beta - t)H = r(z'G - tH) \\
S_x &= r(x_0 + \textstyle\sum_{i=1}^{\ell} m_i x_i)H = r(x_0'G + v_0 H + \textstyle\sum_{i=1}^{\ell} m_i X_i) \\
S_y &= r(y_0 + \textstyle\sum_{i=1}^{\ell} m_i y_i)H = r(y_0'G - s(x_0'G + v_0 H) + \textstyle\sum_{i=1}^{\ell} m_i Y_i) \\
&= r(y_0'G + \textstyle\sum_{i=1}^{\ell} y_i' m_i H) - sS_x
\end{aligned}$$

Hence,

$$\Pr[\mathbf{G}_6{}^{\mathcal{A}}(\lambda) = 1] \geqslant \Pr[\mathbf{G}_5{}^{\mathcal{A}}(\lambda) = 1] \;,$$

$\mathbf{G}_7(\lambda)$: Consider a sequence of sub-games $\mathbf{G}_6 = \mathbf{G}_{7,1}, \ldots, \mathbf{G}_{7,q_m} = \mathbf{G}_7$ where $\mathbf{G}_{7,i}$ is such that the first $i-1$ queries to $\mathsf{MAC}_{\mathsf{DDH}}.\mathsf{M}$ are computed in the following manner:
1. $r, \omega, \chi \leftarrow\!\!\$\, \mathbb{Z}_p$
2. $S_w \leftarrow \omega H; S_z \leftarrow rz'G - t\omega H$
3. $S_x \leftarrow \chi H,$
4. $S_y \leftarrow ry_0'G + \omega \sum_{i=1}^{\ell} y_i' m_i H - sS_x$

and the rest are computed as in $\mathbf{G}_6$. To argue that $\mathcal{A}$ has roughly the same advantage in $\mathbf{G}_{7,i}$ and $\mathbf{G}_{7,i+1}$ for all $i \in [q_m - 1]$, we need a few hybrids for each step. Fix $i \in [q_m - 1]$. Let $\mathbf{G}_{7,i,\star}$ be $\mathbf{G}_{7,i}$ with the change that on the i-th query a tag is computed as:
1. $r, \omega \leftarrow\!\!\$\, \mathbb{Z}_p$
2. $S_w \leftarrow \omega H; S_z \leftarrow rz'G - t\omega H$
3. $S_x \leftarrow r\langle \boldsymbol{x}', 1\|\boldsymbol{m}\rangle G + \omega\langle \boldsymbol{v}, 1\|\boldsymbol{m}\rangle H$
4. $S_y \leftarrow ry_0'G + \omega \sum_{i=1}^{\ell} y_i' m_i H - sS_x$

We'll first show that $\mathbf{G}_{7,i} \approx \mathbf{G}_{7,i,\star}$, and then show that $\mathbf{G}_{7,i,\star} \approx \mathbf{G}_{7,i+1}$. The only difference between $\mathbf{G}_{7,i,\star}$ and $\mathbf{G}_{7,i}$ is this tag for the i-th query; in particular, in $\mathbf{G}_{7,i}$ the tag for the i-th query was computed as:
1. $r \leftarrow\!\!\$\, \mathbb{Z}_p$
2. $S_w \leftarrow rH; S_z \leftarrow r(z'G - tH)$
3. $S_x \leftarrow r\langle \boldsymbol{x}, 1\|\boldsymbol{m}\rangle H = r(\langle \boldsymbol{x}', 1\|\boldsymbol{m}\rangle G + \langle \boldsymbol{v}, 1\|\boldsymbol{m}\rangle H)$
4. $S_y \leftarrow r(y_0'G + \sum_{i=1}^{\ell} y_i' m_i H) - sS_x$

Consider the reduction $\mathcal{B}_{\mathsf{DDH}}$ playing the DDH game, which on challenge $(p, G, \mathbb{G}, A, B, C)$ simulates the entire game to $\mathcal{A}$ with $H \leftarrow A$ and the following for the i-th tag oracle query:
1. $S_w \leftarrow C; S_z \leftarrow z'B - tC$
2. $S_x \leftarrow \langle \boldsymbol{x}', 1\|\boldsymbol{m}\rangle B + \langle \boldsymbol{v}, 1\|\boldsymbol{m}\rangle C$
3. $S_y \leftarrow y_0'B + \sum_{i=1}^{\ell} y_i' m_i C - sS_x$

If (A, B, C) is a DDH triple then the above perfectly simulates $\mathbf{G}_{7,i}$. On the other hand, if A, B, C are all sampled independently and uniformly at random from $\mathbb{Z}_p$ then the above perfectly simulates $\mathbf{G}_{7,i+1}$. We may conclude that

$$\left|\Pr[\mathbf{G}_{7,i}{}^{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{G}_{7,i,\star}{}^{\mathcal{A}}(\lambda) = 1]\right| \leqslant \mathsf{Adv}^{\mathrm{ddh}}_{\mathsf{GGen}}(\mathcal{B}_{\mathsf{DDH}}, \lambda) \;.$$

We now argue that $\mathcal{A}$ behaves roughly the same in $\mathbf{G}_{7,\star}$ and $\mathbf{G}_{7,i+1}$. It suffices to show that $\langle \boldsymbol{v}, 1\|\boldsymbol{m}\rangle$ is uniform in $\mathbb{Z}_p$ and independent from all values in the game. The value $\boldsymbol{v}$ does not appear at any point prior to the i-th MAC query. After this point, it is only used in two places: (1) future ($i+1, \ldots, q_m$-th) MAC queries, and (2) at the end of the game as part of the winning condition. Regarding (1), $\boldsymbol{v}$ is information-theoretically hidden by $\boldsymbol{x}$, and the tag oracle only uses $\boldsymbol{x}$, not $\boldsymbol{x}'$ or $\boldsymbol{v}$. For (2), when $\mathcal{A}$ outputs $(\boldsymbol{m}^* = (m_1^*, \ldots, m_\ell^*), \sigma^* = (S_w^*, S_x^*, S_y^*, S_w^*))$ they win if $\boldsymbol{m}^* \notin \mathsf{MsgQ}$ and $S_x^* = S_w^*(\langle \boldsymbol{x}', 1\|\boldsymbol{m}^*\rangle G + \langle \boldsymbol{v}, 1\|\boldsymbol{m}^*\rangle H)$

among other conditions not involving $\boldsymbol{v}$. As $\boldsymbol{m} \neq \boldsymbol{m}^*$, there exists $j \in [\ell]$ such that $m_j \neq m_j^*$. For any $\alpha_1, \alpha_2 \in \mathbb{Z}_p$, we have

$$\begin{aligned}
&\Pr[\langle \boldsymbol{v}, 1\|\boldsymbol{m}\rangle = \alpha_1 \wedge \langle \boldsymbol{v}, 1\|\boldsymbol{m}^*\rangle = \alpha_2] \\
&\quad = \Pr[\langle \boldsymbol{v}, 1\|\boldsymbol{m}\rangle = \alpha_1 \ \mid\ \langle \boldsymbol{v}, 1\|\boldsymbol{m}^*\rangle = \alpha_2] \cdot \Pr[\langle \boldsymbol{v}, 1\|\boldsymbol{m}^*\rangle = \alpha_2] \\
&\quad = \Pr[\langle \boldsymbol{v}, 1\|\boldsymbol{m}\rangle - \langle \boldsymbol{v}, 1\|\boldsymbol{m}^*\rangle = \alpha_1 - \alpha_2] \cdot \frac{1}{p} \\
&\quad = \Pr\left[\sum_{i=1}^{\ell} v_i(m_i - m_i^*) = \alpha_1 - \alpha_2\right] \cdot \frac{1}{p} \\
&\quad = \Pr\left[v_j(m_j - m_j^*) = \alpha_1 - \alpha_2 - \sum_{i \in [\ell]\setminus\{j\}} v_i(m_i - m^*)\right] \cdot \frac{1}{p} = \frac{1}{p^2}\ .
\end{aligned}$$

Where the final equality can be seen by viewing $(v_i)_{i\in[\ell]\setminus\{j\}}$ as fixed and taking the probability over the random choice of v_j; the left-hand side is uniform in $\mathbb{Z}_p$ (since $m_j - m_j^* \neq 0$) and equal to a fixed value. This means that $\langle \boldsymbol{v}, 1\|\boldsymbol{m}\rangle$ and $\langle \boldsymbol{v}, 1\|\boldsymbol{m}^*\rangle$ are independent. We have

$$\left|\Pr[\mathbf{G}_7{}^{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{G}_6{}^{\mathcal{A}}(\lambda) = 1]\right| \leqslant q_m \cdot \mathsf{Adv}^{\mathrm{ddh}}_{\mathsf{GGen}}(\mathcal{B}_{\mathsf{DDH}}, \lambda)\ .$$

$\mathbf{G}_8(\lambda)$: Finally, we have that $\mathcal{A}$'s forgery $(\boldsymbol{m}^* = (m_1^*, \ldots, m_\ell^*), \sigma^* = (S_w^*, S_x^*, S_y^*, S_w^*))$ at the end of the game has to satisfy

$$y_0'(S_x^* - \langle \boldsymbol{v}, 1\|\boldsymbol{m}^*\rangle S_w^*) = \langle \boldsymbol{x}', 1\|\boldsymbol{m}^*\rangle(S_y^* + sS_x^* - \langle \boldsymbol{y}', 0\|\boldsymbol{m}^*\rangle S_w)$$

Assuming that $y_0' \neq 0$, which occurs with probability $1 - 1/p$, since v_0 is information-theoretically hidden due to the fact that v_0 and x_0 are never used in any value given to $\mathcal{A}$ and $S_w^* \neq 0$, their output can only satisfy this equation with probability $1/p$. Therefore

$$\Pr[\mathbf{G}_8{}^{\mathcal{A}}(\lambda) = 1] \leqslant \frac{1}{p} + \left(1 - \frac{1}{p}\right)\frac{1}{p} \leqslant \frac{2}{p}.$$

□

6.6 Unforgeability Proof of $\mathsf{KVAC}_{\mathsf{DDH}}$

Proof (of Lemma 6.2). Parameter indistinguishability follows from $\mathsf{Ext}_{\mathsf{Setup}}$ generating par as in Setup.

Now, we show the advantage of $\mathcal{A}$ in the unforgeability game. We assume without loss of generality that any RO query (except for programming) the game has to make in the verification of some proofs or showing messages is already made by $\mathcal{A}$. (To be more precise, this increases the number of queries to $\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2$ by at most q.)

$\mathbf{G}_1(\lambda)$: $(\mathsf{Ext}_{\mathsf{DDH}}, \mathcal{O}_{\mathsf{SVerDDH}})$-unforgeability of $\mathsf{KVAC}_{\mathsf{DDH}}$.

$\mathbf{G}_2(\lambda)$: The oracle Iss is modified so that after checking validity of π_{com} and running $(u_x, u_y, \boldsymbol{m}) \leftarrow \mathsf{Ext}^{\mathsf{H}_0}_{\mathsf{com}}(\mathcal{Q}, (\boldsymbol{X}, \boldsymbol{Y}, \widetilde{E}_x, \widetilde{E}_y, D, \psi), \pi_{\mathsf{com}})$, it aborts if $((\boldsymbol{X}, \boldsymbol{Y}, \widetilde{E}_x, \widetilde{E}_y, D, \psi), (u_x, u_y, \boldsymbol{m})) \notin \widetilde{\mathsf{R}}_{\mathsf{com}}$. We call this event BadCom.

We now define a reduction $\mathcal{B}_{\mathsf{com}}$ playing the KSND game for $\mathit{\Pi}_{\mathsf{com}}$ with respect to the extractor $\mathsf{Ext}_{\mathsf{com}}$. With oracle access to $\mathcal{O}_{\mathsf{Ext}}$, it simulates $\mathbf{G}_1$ to $\mathcal{A}$ on every Iss query, queries its oracle $\mathcal{O}_{\mathsf{Ext}}$ with $(\boldsymbol{X}, \boldsymbol{Y}, \widetilde{E}_x, \widetilde{E}_y, D, \psi), \pi_{\mathsf{com}}$. By definition of the straight-line extractable knowledge soundness game, $\mathcal{B}_{\mathsf{com}}$ wins if BadCom ever occurs. Hence,

$$\Pr[\mathbf{G}_2{}^{\mathcal{A}}(\lambda) = 1] \geqslant \Pr[\mathbf{G}_1{}^{\mathcal{A}}(\lambda) = 1] - \mathsf{Adv}^{\mathrm{ksnd}}_{\mathit{\Pi}_{\mathsf{com}}, \mathsf{Ext}_{\mathsf{com}}}(\mathcal{B}_{\mathsf{com}}, \lambda)\ .$$

$\mathbf{G}_3(\lambda)$**:** In this game, we simulate the proof π_σ in the issuance oracle using Sim_σ, which programs H_1. To argue the change in advantage, we construct a straightforward reduction $\mathcal{B}_\sigma$ to the ZK game of Π_σ, such that

$$\Pr[\mathbf{G_3}^{\mathcal{A}}(\lambda) = 1] \geqslant \Pr[\mathbf{G_2}^{\mathcal{A}}(\lambda) = 1] - \mathsf{Adv}^{\mathsf{zk}}_{\Pi_\sigma, \mathsf{Sim}_\sigma}(\mathcal{B}_\sigma, \lambda) \; .$$

$\mathbf{G}_4(\lambda)$**:** At the start of the game we initialize a table $T_2 \leftarrow (\;)$ and use it to lazy-sample values for H_2. Then $\mathrm{SH}_{\mathsf{pub}}$ simulates the proof π_{pub} by programming values into T_2. Explicitly, in $\mathrm{SH}_{\mathsf{key}}$, the reduction computes $S'_w, (C_i)_{i=1}^{\ell}, C_x, \Gamma_x, C_y, \Gamma_y$, and S'_z as an honest user would. Then, the reduction first samples $c \leftarrow\!\!\$\; \mathbb{Z}_p$ and $\boldsymbol{s} \leftarrow\!\!\$\; \mathbb{Z}_p^{4\ell+2}$ sets $\boldsymbol{Y}_{\mathsf{pub}} := (C_i)_{i=1}^{\ell} \| \Gamma_x \| \Gamma_y$ and computes $\boldsymbol{R} = M^{\mathsf{pub}}_{G,H,S'_w,\boldsymbol{X},\boldsymbol{Y}_{\mathsf{pub}}} \boldsymbol{s} - c\boldsymbol{Y}_{\mathsf{pub}}$, and then sets $T_2(M^{\mathsf{pub}}_{G,H,S'_w,\boldsymbol{X},\boldsymbol{Y}_{\mathsf{pub}}}, \boldsymbol{Y}_{\mathsf{pub}}, \boldsymbol{R}, \phi, \mathsf{nonce}) \leftarrow c$, or aborts if it is already set. Since the hash query contains elements uniform in $\mathbb{G}$, and queries to H_2 happen either on a query to $\mathrm{SH}_{\mathsf{pub}}$ or directly (which in total is less than q queries), we have the following by the union bound:

$$\Pr[\mathbf{G_4}^{\mathcal{A}}(\lambda) = 1] \geqslant \Pr[\mathbf{G_3}^{\mathcal{A}}(\lambda) = 1] - \frac{q^2}{p} \; .$$

$\mathbf{G}_5(\lambda)$**:** We modify KeyGen so that ct_x and ct_y are each sampled uniformly at random from $\mathbb{G}^2$. In this game, the public keys are now independent of x_0 and y_0. By Lemma 2.2,

$$\Pr[\mathbf{G_5}^{\mathcal{A}}(\lambda) = 1] \geqslant \Pr[\mathbf{G_4}^{\mathcal{A}}(\lambda) = 1] - \mathsf{Adv}^{\mathrm{ddh}}_{\mathsf{GGen}}(\mathcal{B}_{\mathsf{DDH}}, \lambda) - \frac{1}{p-1} \; .$$

$\mathbf{G}_6(\lambda)$**:** This game aborts if during issuance the extracted witness $(u_x, u_y, \boldsymbol{m})$ is such that $\boldsymbol{m} \neq 0$ and $\sum_{i=1}^{\ell} m_i X_i = \sum_{i=1}^{\ell} m_i Y_i = 0_{\mathbb{G}}$. Denote this event as BadExt. This is to rule out the case that straightline-extraction outputs $\boldsymbol{m}$ that does not correspond to the openings of $\widetilde{E}_x, \widetilde{E}_y$.
Notice that this breaks rel-DL with respect to bases $\boldsymbol{X}$ and $\boldsymbol{Y}$, but we cannot directly reduce to rel-DL, since the game needs the discrete log of X_i, Y_i's to simulate. Hence, we will reduce to the security of $\mathsf{MAC}_{\mathsf{DDH}}$ instead. In particular, we construct the following reduction $\mathcal{B}_{\mathsf{ufcma}}$ playing the UFCMA game for $\mathsf{MAC}_{\mathsf{DDH}}$ with access to the oracle $\mathcal{O}$. It takes as input the public parameters $\mathsf{par} = (p, G, \mathbb{G}, H)$ and $\mathsf{ipk} = (\boldsymbol{X}, \boldsymbol{Y}, Z)$ and samples $\mathsf{ct}_x, \mathsf{ct}_y \leftarrow\!\!\$\; \mathbb{G}^2$ as in the previous game. It then runs the adversary $\mathcal{A}$ on par and $\mathsf{pk} = (\boldsymbol{X}, \boldsymbol{Y}, Z, \mathsf{ct}_x, \mathsf{ct}_y)$. Then, it simulates the oracles as follows:

- On issuance queries, it runs the extractor to extract $(u_x, u_y, \boldsymbol{m})$. If BadExt occurs, i.e., $\boldsymbol{m} \neq \boldsymbol{0}$ and $\sum_{i=1}^{\ell} m_i X_i = \sum_{i=1}^{\ell} m_i Y_i = 0_{\mathbb{G}}$. The reduction queries its MAC oracle to get a tag (S_w, S_x, S_y, S_z) on message $\boldsymbol{0}$ and return (S_w, S_x, S_y, S_z) as its forgery for $\boldsymbol{m}$.
  Otherwise, it queries the MAC oracle on message $\boldsymbol{m}$ for a tag (S_w, S_x, S_y, S_z). Then, it returns $S_w, E_x = (\gamma_x G, \gamma_x D + S_x), E_y = (\gamma_y G, \gamma_y D + S_y), S_z$ and a simulated proof π_σ.
- The NewUsr oracle on input $\boldsymbol{m}$ and ϕ (for $\phi(\boldsymbol{m}) = 1$) is simulated honestly: $\mathcal{B}_{\mathsf{ufcma}}$ queries its MAC oracle on $\boldsymbol{m}$ to get the credential. Note that if $\boldsymbol{m} \neq \boldsymbol{0}$ is such that $\sum_{i=1}^{\ell} m_i X_i = \sum_{i=1}^{\ell} m_i Y_i = 0_{\mathbb{G}}$, we compute the forgery as when BadExt occurs.
- The $\mathrm{SH}_{\mathsf{key}}$ and $\mathrm{SH}_{\mathsf{pub}}$ are simulated as in the previous game, and this can be done since the game knows the credential and the attributes.
- Queries to $\mathcal{O}$ are forwarded to its oracle $\mathcal{O}$.

Note that the view of $\mathcal{A}$ is identical to its view in $\mathbf{G}_5$. Moreover, if BadExt occurs, then $\mathcal{B}_{\mathsf{ufcma}}$ wins the game. Hence,

$$\Pr[\mathbf{G_6}^{\mathcal{A}}(\lambda) = 1] \geqslant \Pr[\mathbf{G_5}^{\mathcal{A}}(\lambda) = 1] - \mathsf{Adv}^{\mathsf{ufcma}}_{\mathsf{MAC}_{\mathsf{DDH}}, \mathcal{O}_{\mathsf{SVerDDH}}}(\mathcal{B}_{\mathsf{ufcma}}, \lambda) \; .$$

$\mathbf{G}_7(\lambda)$**:** The oracle NewUsr is modified so that it just sets $\sigma_{\mathsf{cid}} \leftarrow \perp$ instead of using $\mathsf{KVAC}_{\mathsf{BBS}}.\mathsf{Iss}$. We modify $\mathrm{SH}_{\mathsf{key}}$ to do the following instead of running $\mathsf{KVAC}_{\mathsf{BBS}}.\mathsf{Show}_{\mathsf{key}}$:

1. $r \leftarrow\!\!\$\; \mathbb{Z}_p^*$
2. $S_w \leftarrow rG; S_z \leftarrow zS_w$
3. $(C_i)_{i=1}^{\ell} \leftarrow\!\!\$\; \mathbb{G}^{\ell}; C_x, C_y \leftarrow\!\!\$\; \mathbb{G}$
4. $\Gamma_x \leftarrow x_0 S'_w + (\sum_{i=1}^{\ell} x_i C_i) - C_x$; $\Gamma_y \leftarrow y_0 S'_w + (\sum_{i=1}^{\ell} y_i C_i) - C_y$

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

This makes no external change as $S_w, (C_i)_{i=1}^{\ell}$ are still random, and $C_x + \Gamma_x$ and $C_y + \Gamma_y$ still satisfies $\mathsf{SVer}_{\mathsf{key}}$, so

$$\Pr[\mathbf{G}_7{}^{\mathcal{A}}(\lambda) = 1] \geqslant \Pr[\mathbf{G}_6{}^{\mathcal{A}}(\lambda) = 1] .$$

$\mathbf{G}_8(\lambda)$**:** The game aborts if the forgery $(\phi^*, \mathsf{nonce}^*, \tau^*)$ corresponds to an RO value which was programmed via simulation in $\mathrm{SH}_{\mathsf{pub}}$. More specifically, at the end of the game:

1. Parse $((S'_w, S'_z, (C_i)_{i=1}^{\ell}, C_x, C_y, \Gamma_x, \Gamma_y), (\boldsymbol{r}', \pi_{\mathsf{pub}}) \leftarrow \tau$ and $(\boldsymbol{I}, \boldsymbol{m}') \leftarrow \phi^*$.
2. Parse $(c, \boldsymbol{s}) \leftarrow \pi_{\mathsf{pub}}$, define $M := M^{\mathsf{pub}}_{G,H,S'_w,\boldsymbol{X},\boldsymbol{Y}_{\mathsf{pub}}}$ and $\boldsymbol{Y}_{\mathsf{pub}} := (C_i)_{i=1}^{\ell} \| \Gamma_x \| \Gamma_y$. Compute $\boldsymbol{R} = M\boldsymbol{s} - c\boldsymbol{Y}_{\mathsf{pub}}$.
3. Abort if $H(M, \boldsymbol{Y}', \boldsymbol{R}, \phi^*, \mathsf{nonce}^*)$ was programmed in the act of simulating a π_{pub} proof (rather than via lazy sampling).

Note that, as part of $\mathcal{A}$'s winning condition, $(\phi^*, \mathsf{nonce}^*, \tau^*) \notin \mathsf{PfQ}$. However, as we know that the hash query input is the same as one that was simulated, and the hash query contains $G, H, S'_w, \boldsymbol{X}, \boldsymbol{Y}$ and $(C_i)_{i=1}^{\ell}$, Γ_x, Γ_y, as well as nonce^*, and ϕ^*, there must a simulated $(\phi, \mathsf{nonce}, \tau)$ which is exactly the same as the forgery except $\boldsymbol{s} \neq \tilde{\boldsymbol{s}}$, where $\tilde{\boldsymbol{s}}$ corresponds to the simulated proof. Since $(\phi^*, \mathsf{nonce}^*, \tau^*) \notin \mathsf{PfQ}$, the only way this can occur is if $\boldsymbol{s} \neq \tilde{\boldsymbol{s}}$, where $\tilde{\boldsymbol{s}}$ is part of the simulated proof. Unpacking $\boldsymbol{s}$ into $(s_{m_i})_{i=1}^{\ell}, (s_{r_i})_{i=1}^{\ell}, s_{r_x}, s_{r_y})$ and doing the same for $\tilde{\boldsymbol{s}}$, we have the following system of equations (by $\boldsymbol{M}\boldsymbol{s} = \boldsymbol{M}\tilde{\boldsymbol{s}}$):

$$\begin{aligned}
&(s_{m_i} - \tilde{s}_{m_i})S_w + (s_{r_i} - \tilde{s}_{r_i})H = 0 \text{ for } i \in [\ell] \\
&\sum_{i=1}^{\ell}(s_{r_i} - \tilde{s}_{r_i})X_i - (s_{r_x} - \tilde{s}_{r_x})H = 0 \\
&\sum_{i=1}^{\ell}(s_{r_i} - \tilde{s}_{r_i})Y_i - (s_{r_y} - \tilde{s}_{r_y})H = 0
\end{aligned}$$

Using $\boldsymbol{s} \neq \tilde{\boldsymbol{s}}$, at first glance there are roughly four cases to consider. However, if $H \neq 0$, which occurs with probability $1 - \frac{1}{p}$, then $s_{m_i} - \tilde{s}_{m_i} = 0$ for all $i \in [\ell]$ would imply $s_{r_i} - \tilde{s}_{r_i} = 0$ for all $i \in [\ell]$, and that in turn would imply $s_{r_x} - \tilde{s}_{r_x} = 0$ and $s_{r_y} - \tilde{s}_{r_y} = 0$. Thus, it suffices to consider only the case that $s_{m_i} - \tilde{s}_{m_i}$ for some $i \in [\ell]$. Consider the reduction $\mathcal{B}_{\mathrm{dlog}}$ which on challenge $P \in \mathbb{G}$ samples $\beta \leftarrow \mathbb{Z}_p$ and sets $H = \beta G$. It then simulates proofs by computing S_w as $a_i P$ for $a_i \leftarrow\!\!\$\ \mathbb{Z}_p$. When the adversary forges, we obtain an equation of the form $(s_{m_i} - \tilde{s}_{m_i})a_i P + (s_{r_i} - \tilde{s}_{r_i})\beta G = 0$ from which we can recover $\log_G P$ assuming that $a_i \neq 0$. We can conclude that

$$\Pr[\mathbf{G}_8{}^{\mathcal{A}}(\lambda) = 1] \geqslant \Pr[\mathbf{G}_7{}^{\mathcal{A}}(\lambda) = 1] - \frac{1}{p} - \mathsf{Adv}^{\mathrm{dlog}}_{\mathsf{GGen}}(\mathcal{B}_{\mathrm{dlog}}, \lambda) .$$

$\mathbf{G}_9(\lambda)$**:** At the start of the game we sample $(h_1, r_1), \ldots, (h_q, r_q) \leftarrow\!\!\$\ \mathbb{Z}_p \times \mathbb{Z}_p^*$ and initialize a counter $\mathsf{cnt} \leftarrow 0$. Whenever we need to program an RO value for T_2, we use h_{cnt} and then set $\mathsf{cnt} \leftarrow \mathsf{cnt} + 1$. Similarly, in Iss, instead of $r \leftarrow\!\!\$\ \mathbb{Z}_p$, we use $r \leftarrow r_{\mathsf{cnt}}$ and set $\mathsf{cnt} \leftarrow \mathsf{cnt} + 1$. For other oracles ($\mathsf{H}_0, \mathsf{H}_1$ and $\mathrm{SH}_{\mathsf{key}}$) and the adversary $\mathcal{A}$, the game samples random coins $\rho = (\rho', \rho_{\mathcal{A}})$ where ρ' is used to program $\mathsf{H}_0, \mathsf{H}_1$ and $\mathrm{SH}_{\mathsf{key}}$, while $\rho_{\mathcal{A}}$ is the random coins for $\mathcal{A}$. Via an essentially identical rewinding argument to our proof of $\mathsf{KVAC}_{\mathsf{BBS}}$ unforgeability, we can extract a witness $((m_i, r_i)_{i \in [\ell] \setminus \boldsymbol{I}}, r_x, r_y)$ corresponding to the forgery $(\phi^*, \mathsf{nonce}^*, \tau^*)$ with high probability. Concretely, by the forking lemma,

$$\Pr[\mathbf{G}_8{}^{\mathcal{A}}(\lambda) = 1] \leqslant \sqrt{q \cdot \Pr[\mathbf{G}_9{}^{\mathcal{A}}(\lambda) = 1]} + \frac{q}{p} .$$

We now describe the reduction $\mathcal{B}'_{\mathsf{ufcma}}$ playing the $\mathcal{O}_{\mathsf{SVerDDH}}$-UFCMA game for $\mathsf{MAC}_{\mathsf{DDH}}$, which on input $(p, G, \mathbb{G}, H), \mathsf{ipk}$ and with access to oracle MAC simulates $\mathbf{G}_9$ to $\mathcal{A}$. At the start of the game, sample $\boldsymbol{m}_{\mathrm{SH}} = (m_{i,\mathrm{SH}})_{i=1}^{\ell} \leftarrow\!\!\$\ \mathbb{Z}_p^{\ell}$ and query $\sigma_{\mathrm{SH}} \leftarrow \mathrm{MAC}(\boldsymbol{m}_{\mathrm{SH}})$. Sample $(\alpha_i)_{i=1}^{q_{\mathrm{SH}}} \leftarrow\!\!\$\ (\mathbb{Z}_p^* \times \mathbb{Z}_p^{\ell+2})^{q_{\mathrm{SH}}}$. The first run of $\mathcal{A}$, we make the following changes:

1. Set $\sigma_1, \dots, \sigma_q \leftarrow \perp$
2. Instead of $(\mathsf{sk}', \mathsf{ipk}') \leftarrow \mathsf{MAC_{DDH}}.\mathsf{KeyGen}(\mathsf{par})$, do $(\mathsf{sk}', \mathsf{ipk}') \leftarrow (\mathsf{sk}, \mathsf{ipk})$.
3. In Iss, query $\sigma := (S_w, S_x, S_y, S_z) \leftarrow \mathrm{MAC}(\boldsymbol{m})$ (where $\boldsymbol{m}$ is extracted from Π_{com}), set $E_x \leftarrow (\gamma_x G, \gamma_x H + S_x)$ and $E_y \leftarrow (\gamma_y G, \gamma_y H + S_y)$. Record $\sigma_{\mathsf{cnt}} \leftarrow \sigma$ and increment cnt by 1. Simulation is perfect since $\widetilde{E_x}$ is an encryption of $\sum_{i=1}^{\ell} m_i X_i$, $\widetilde{E_y}$ is an encryption of $\sum_{i=1}^{\ell} m_i Y_i$.
4. On the query to SH, do $(r', (r_i)_{i=1}^{\ell}, r_x, r_y) \leftarrow \boldsymbol{\alpha}_j$. Compute $(S'_w, S'_x, S'_y, S'_z) \leftarrow r'\sigma_{\mathrm{SH}}$, then $C_i \leftarrow m_{i,\mathrm{SH}} S'_w + r_i H$, $C_x \leftarrow S'_x + r_x H$, and $C_y \leftarrow S'_y + r_y H$. Compute $\Gamma_x \leftarrow \sum_{i=1}^{\ell} r_i X_i - r_x H$ and Γ_y similarly. These outputs have the same distribution as in $\mathbf{G}_9$ since the MAC tag σ_{SH} is valid for $(m_{i,\mathrm{SH}})$. Note that due to the change in game $\mathbf{G}_7$ and the rewinding in $\mathbf{G}_9$, the key-dependent showing message τ_{key} will be the same in both runs, identically distributed to the ones in this reduction.

When the reduction runs $\mathcal{A}$ a second time, it does everything the same except for $\mathsf{cnt} < J$ it return σ_{cnt} in Iss instead of querying $\mathrm{MAC}(\boldsymbol{m})$. Since $\mathcal{A}$ is run with the same randomness and inputs for the entire period of the game when $\mathsf{cnt} < J$, they will make the same queries to Iss up to that point, so our simulation is perfect. For the queries after $\mathsf{cnt} \geqslant J$, it runs the game using the newly sampled $h'_J, \dots, h'_q \leftarrow\$ \mathbb{Z}_p$ instead as described for the first run.

At the end of the game $\mathcal{A}$ outputs $(\phi^*, \mathsf{nonce}^*, \tau^*)$ and we parse $((S_w, S_z, (C_i)_{i=1}^{\ell}, C_x, C_y, \Gamma_x, \Gamma_y), (\boldsymbol{r}', \pi_{\mathsf{pub}})) \leftarrow \tau^*$ and $(\boldsymbol{I}, \boldsymbol{m}') \leftarrow \phi^*$. We also have the extracted $((m_i)_{i \in [\ell] \setminus \boldsymbol{I}}, (r_i)_{i=1}^{\ell}, r_x, r_y)$. Reconstruct $\boldsymbol{m}^* = (m_1^*, \dots, m_\ell^*)$ from $\boldsymbol{m}'$ and $(m_i)_{i \in [\ell] \setminus \boldsymbol{I}}$. If the forgery verifies,

$$
\begin{aligned}
\Gamma_x + C_x &= x_0 S'_w + \sum_{i=1}^{\ell} x_i C_i \\
&= x_0 S'_w + \sum_{i=1}^{\ell} x_i (m_i^* S'_w + r_i G) \\
&= (x_0 + \sum_{i=1}^{\ell} x_i m_i^*) S'_w + \sum_{i=1}^{\ell} r_i X_i
\end{aligned}
$$

and $\Gamma_x = (\sum_{i=1}^{\ell} r_i X_i) - r_x H$, so $C_x - r_x H = (x_0 + \sum_{i=1}^{\ell} x_i m_i^*) S'_w$, and analogously $C_y - r_y H = (y_0 + \sum_{i=1}^{\ell} y_i m_i^*) S'_w$. We can make the same argument for E_y. At the end, we obtain a valid MAC tag $(S'_w, C_x - r_x H, C_y - r_y H, S'_z)$. Finally, note that since the combined view of $\mathcal{A}$ in both runs is identical to that in $\mathbf{G}_9$, $\boldsymbol{m}_{\mathrm{SH}}$ is information-theoretically hidden in all values given to $\mathcal{A}$. Thus, $\boldsymbol{m}^* = \boldsymbol{m}_{\mathrm{SH}}$ with probability at most $\frac{1}{p^\ell}$, and otherwise, $\mathcal{B}'_{\mathsf{ufcma}}$ wins. Therefore,

$$
\mathsf{Adv}^{\mathsf{ufcma}}_{\mathsf{MAC_{DDH}}, \mathcal{O}_{\mathsf{SVerDDH}}}(\mathcal{B}'_{\mathsf{ufcma}}, \lambda) \geqslant \Pr[\mathbf{G}_9{}^{\mathcal{A}}(\lambda) = 1] - \frac{1}{p^\ell} \, . \square
$$

6.7 Anonymity Proof of $\mathsf{KVAC_{DDH}}$

Proof (of Lemma 6.3). We note first that the global parameters generator $\mathsf{Gen}(1^\lambda)$ outputs $\mathsf{par}_g = (p, G, \mathbb{G}, H)$ and $\mathsf{Sim_{Gen}}$ additionally outputs a trapdoor $v \in \mathbb{Z}_p^*$ such that $vG = H$. Note that v will also be given to the simulator Sim

We assume without loss of generality that the queries made to H_1 when the game verifies π_σ are already made by $\mathcal{A}$. (This includes the query count by 1). To show security, we consider the following sequence of games:

$\mathbf{G}_1(\lambda)$**:** This is the game $\mathrm{Anon}_{\mathsf{KVAC_{DDH}}, \mathsf{Sim_{Gen}}, \mathsf{Sim_{DDH}}, 0}$.
$\mathbf{G}_2(\lambda)$**:** We simulate π_{com} as in $\mathsf{Sim}_{\mathsf{U}_1}$ and π_{pub} as in $\mathsf{Sim_{Show}}$ instead of generating it honestly. There exists $\mathcal{B}_{\mathsf{com}}$ and $\mathcal{B}_{\mathsf{pub}}$ where $\mathcal{B}_{\mathsf{pub}}$ makes at most q_{Show} queries to its prover oracle such that

$$
\left| \Pr[\mathbf{G}_2{}^{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{G}_1{}^{\mathcal{A}}(\lambda) = 1] \right| \leqslant \mathsf{Adv}^{\mathsf{zk}}_{\Pi_{\mathsf{com}}, \mathsf{Sim_{com}}}(\mathcal{B}_{\mathsf{com}}, \lambda) + \mathsf{Adv}^{\mathsf{zk}}_{\Pi_{\mathsf{pub}}, \mathsf{Sim_{pub}}}(\mathcal{B}_{\mathsf{pub}}, \lambda) \, .
$$

The RO query count follows as in the lemma statement.

$\mathbf{G}_3(\lambda)$: We add an inefficient check in U_2 that checks whether the issuer's message (S_w, E_x, E_y, S_z), the public key pk, and the user's first message $(D, \widetilde{E}_x, \widetilde{E}_y)$ is in the induced language of R_σ. If not, abort the game. By soundness of Π_σ, we have that

$$\left|\Pr[\mathbf{G}_3{}^{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{G}_2{}^{\mathcal{A}}(\lambda) = 1]\right| \leqslant \mathsf{Adv}^{\mathsf{sound}}_{\Pi_\sigma}(\mathcal{B}_\sigma, \lambda) .$$

The RO query count follows as in the lemma statement.

$\mathbf{G}_4(\lambda)$: This game simulates $\mathrm{SH}_{\mathsf{key}}$ as in $\mathsf{Sim}_{\mathsf{Show}}$. At this point, the showing oracles are all independent of the attributes $\boldsymbol{m}$ (except for when checking validity of $\phi(\boldsymbol{m}) = 1$ in $\mathrm{SH}_{\mathsf{pub}}$).

Now, we argue the indistinguishability. First, we consider τ_{key} as in $\mathbf{G}_3$. Let $\mathsf{sk} = (\boldsymbol{x}, \boldsymbol{y}, z, t_x, t_y)$ be the underlying secret key fixed by pk. By the introduced check in the previous game and with how an honest user compute $\widetilde{E}_x, \widetilde{E}_y$, we have that for some $r \in \mathbb{Z}_p$

$$S_w = rH \; , S_z = rzH$$
$$S_x = r(x_0 + \sum_{i=1}^{\ell} m_i x_i)H \; , S_y = r(y_0 + \sum_{i=1}^{\ell} m_i y_i)H \; . \tag{5}$$

By how $\mathsf{KVAC}_{\mathsf{DDH}}.\mathsf{Show}_{\mathsf{key}}$ is defined,

$$\begin{aligned} S'_w = r' S_w = rr'H \; &, S'_z = r' S_z = rr'zH \\ C_x = r' S_x + r_x H \; &, \Gamma_x = \sum_{i=1}^{\ell} r_i X_i - r_x H \\ C_y = r' S_y + r_y H \; &, \Gamma_y = \sum_{i=1}^{\ell} r_i Y_i - r_y H \\ C_i = m_i S'_w + r_i H \; &, \forall i \in [\ell] \end{aligned}$$

where $r' \leftarrow\!\!\$\, \mathbb{Z}_p^*, r_1, \ldots, r_\ell, r_x, r_y \leftarrow\!\!\$\, \mathbb{Z}_p$. Next, notice that

$$\begin{aligned} C_x + \Gamma_x &= r' S_x + \sum_{i=1}^{\ell} r_i X_i = r' r x_0 H + \sum_{i=1}^{\ell} (r' r m_i x_i H + r_i X_i) \\ &= x_0 S'_w + \sum_{i=1}^{\ell} x_i (m_i S'_w + r_i H) = x_0 S'_w + \sum_{i=1}^{\ell} x_i C_i \\ C_y + \Gamma_y &= r' S_y + \sum_{i=1}^{\ell} r_i Y_i = r' r y_0 H + \sum_{i=1}^{\ell} (r' r m_i y_i H + r_i Y_i) \\ &= y_0 S'_w + \sum_{i=1}^{\ell} y_i (m_i S'_w + r_i H) = y_0 S'_w + \sum_{i=1}^{\ell} y_i C_i \; . \end{aligned}$$

Since $r' \leftarrow\!\!\$\, \mathbb{Z}_p^*, r_1, \ldots, r_\ell \leftarrow\!\!\$\, \mathbb{Z}_p$, we have that $S'_w, (C_i)_{i \in [\ell]}$ are uniformly random. Moreover, they determine $C_x + \Gamma_x, C_y + \Gamma_y, S'_z$. Hence, with $r_x, r_y \leftarrow \mathbb{Z}_p$, we have that $C_x, C_y, \Gamma_x, \Gamma_y$ can be sampled by sampling $C_x, C_y \leftarrow\!\!\$\, \mathbb{G}$ and computing $\Gamma_x \leftarrow x_0 S'_w + \sum_{i=1}^{\ell} x_i C_i - C_x, \Gamma_y \leftarrow y_0 S'_w + \sum_{i=1}^{\ell} y_i C_i - C_y$. Note that with the simulator sampling $S'_w, (C_i)_{i \in [\ell]}$ while knowing their discrete logarithms, it computes Γ_x, Γ_y efficiently using the elements in the public key and the trapdoor v. Hence, the distributions of τ_{key} from $\mathsf{KVAC}_{\mathsf{DDH}}.\mathsf{Show}_{\mathsf{key}}$ and $\mathsf{Sim}_{\mathsf{Show}}$ are identical. Thus, $\Pr[\mathbf{G}_4{}^{\mathcal{A}}(\lambda) = 1] = \Pr[\mathbf{G}_3{}^{\mathcal{A}}(\lambda) = 1]$.

$\mathbf{G}_5(\lambda)$: This game removes the check introduced in $\mathbf{G}_3$ and also does not compute S_x, S_y in U_2 anymore. With a similar argument as in $\mathbf{G}_3$, we have that there exists $\mathcal{B}_\sigma$ such that

$$\left|\Pr[\mathbf{G}_5{}^{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{G}_4{}^{\mathcal{A}}(\lambda) = 1]\right| \leqslant \mathsf{Adv}^{\mathsf{sound}}_{\Pi_\sigma}(\mathcal{B}_\sigma, \lambda) .$$

$\mathbf{G}_6(\lambda)$: This game simulates U_1 by computing $E_x \leftarrow (u_x G, u'_x D), E_y \leftarrow (u_y G, u'_y D)$ with $u_x, u_y, u'_x, u'_y \leftarrow_\$ \mathbb{Z}_p$ for $i \in [\ell]$. This game hop follows by a reduction $\mathcal{B}_{\mathrm{DDH}}$ to n-DDH. Hence, by Lemma 2.2,

$$\left|\Pr[\mathbf{G_6}^{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{G_5}^{\mathcal{A}}(\lambda) = 1]\right| \leqslant \mathsf{Adv}^{\mathrm{ddh}}_{\mathsf{GGen}}(\mathcal{B}_{\mathrm{DDH}}, \lambda) + \frac{1}{p-1} .$$

Since $\mathbf{G}_6$ is exactly $\mathrm{Anon}_{\mathsf{KVAC}_{\mathsf{DDH}},\mathsf{Sim}_{\mathsf{Gen}},\mathsf{Sim}_{\mathsf{DDH}},1}$, this concludes the proof. ◻

6.8 Security Proof of $\mathsf{oNIP}_{\mathsf{DDH}}$

In this section, we give the proof of Theorem 6.6. Correctness follows easily by inspection. The following lemmas then establish soundness, zero-knowledge, and obliviousness for valid statements.

Lemma 6.8 (Soundness of $\mathsf{oNIP}_{\mathsf{DDH}}$.). *For any adversary $\mathcal{A}$ making at most $q_{\mathsf{H}} = q_{\mathsf{H}}(\lambda)$ queries to H_c modeled as a random oracle and running in time $t_{\mathcal{A}} = t_{\mathcal{A}}(\lambda)$, there exists an adversary $\mathcal{B}$ playing the* DL *game such that*

$$\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{oNIP}_{\mathsf{DDH}}}(\mathcal{A}, \lambda) \leqslant \sqrt{(q_{\mathsf{H}} + 1)\mathsf{Adv}^{\mathrm{dlog}}_{\mathsf{GGen}}(\mathcal{B}, \lambda)} + \frac{q_{\mathsf{H}} + 1}{p} .$$

Proof. The proof for this lemma follows similarly from the rewinding reduction in Lemma 5.9, except that in the event that the adversary outputs a statement $(\mathsf{pk}, \tau_{\mathsf{key}}) \notin \mathcal{L}_{\mathsf{R}_{\mathrm{DDH}}}$ and a valid proof π, we have to show that there exists only one bad challenge c_0 which allows the adversary to find $\boldsymbol{s}_0$ which satisfies the verification equation.

To see this, consider $(\mathsf{pk}, \tau_{\mathsf{key}}) \notin \mathcal{L}_{\mathsf{R}_{\mathrm{DDH}}}$, $\boldsymbol{R}_{0,\mathsf{Core}}$, $\boldsymbol{R}_{0,\mathsf{Aug}}$, and two tuples $(c_0, \boldsymbol{s}_0)$ and $(c'_0, \boldsymbol{s}'_0)$ such that

(a) $\tau_{\mathsf{key}} = (S_w, S_z, (C_i)_{i=1}^{\ell}, \zeta_x, \zeta_y)$ with $S_w \neq 0_{\mathbb{G}}$.
(b) $\boldsymbol{R}_{0,\mathsf{Core}} = M_{\mathsf{Core}}\boldsymbol{s}_0 - c_0\mathsf{pk} = M_{\mathsf{Core}}\boldsymbol{s}'_0 - c'_0\mathsf{pk}$.
(c) $\boldsymbol{R}_{0,\mathsf{Aug}} = M_{\mathsf{Aug},S_w,(C_i)_{i=1}^{\ell}}\boldsymbol{s}_0 - c_0(\zeta_x\|\zeta_y\|S_z) = M_{\mathsf{Aug},S_w,(C_i)_{i=1}^{\ell}}\boldsymbol{s}'_0 - c'_0(\zeta_x\|\zeta_y\|S_z)$

Suppose $c_0 \neq c'_0$. Then, by (b) and (c), we have that for $\mathsf{sk}' = (c'_0 - c_0)^{-1}(\boldsymbol{s}'_0 - \boldsymbol{s}_0)$, $M_{\mathsf{Core}}\mathsf{sk}' = \mathsf{pk}$ and $M_{\mathsf{Aug},S_w,(C_i)_{i=1}^{\ell}}\mathsf{sk}' = (\zeta_x\|\zeta_y\|S_z)$, which contradicts with the fact that $(\mathsf{pk}, \tau_{\mathsf{key}})$ is not in the language. Therefore, $c_0 = c'_0$. Hence, a similar rewinding reduction strategy from the proof of Lemma 5.9 solves the DL problem when $\mathcal{A}$ wins in the soundness game in both runs. ◻

Lemma 6.9 (Zero-Knowledge of $\mathsf{oNIP}_{\mathsf{DDH}}$.). *For the oracle $\mathcal{O}_{\mathsf{SVerDDH}}$ as described in Figure 17, there exists a simulator $\mathsf{Sim} = (\mathsf{Sim}_{\mathsf{Setup}}, \mathsf{Sim}_{\mathsf{Iss}})$ such that for any adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{zk}}_{\mathsf{oNIP}_{\mathsf{DDH}},\mathsf{Sim},\mathcal{O}_{\mathsf{SVerDDH}}}(\mathcal{A}, \lambda) = 0$.*

Proof. Consider the following simulator Sim:

- $\mathsf{Sim}_{\mathsf{Setup}}(p, G, \mathbb{G}, H)$: Sample $w \in \mathbb{Z}_p$ and return $(\mathsf{par}_{\mathsf{oNIP}} = (p, G, \mathbb{G}, W, H), \mathsf{td} = w)$.
- $\mathsf{Sim}_{\mathsf{Iss}}^{\mathcal{O}_{\mathsf{SVerDDH}}}(\mathsf{td}, \mathsf{pk}, \mathsf{umsg}_1 = (S_w, S_z, (C_i)_{i=1}^{\ell}, \zeta_x, \zeta_y))$: Query $\mathcal{O}_{\mathsf{SVerDDH}}((p, G, \mathbb{G}, H), \mathsf{sk}, \mathsf{pk}, \cdot)$ with umsg_1 and if the oracle outputs 0, abort. Otherwise, sample $\boldsymbol{s}_0 \leftarrow_\$ \mathbb{Z}_p^{2\ell+6}, c_0, r_1 \leftarrow_\$ \mathbb{Z}_p$ and set
  - $\boldsymbol{R}_{0,\mathsf{Core}} \leftarrow M_{\mathsf{Core}}\boldsymbol{s}_0 - c_0\mathsf{pk}$
  - $\boldsymbol{R}_{0,\mathsf{Aug}} \leftarrow M_{\mathsf{Aug},S_w,(C_i)_{i=1}^{\ell}}\boldsymbol{s}_0 - c_0(\zeta_x\|\zeta_y\|S_z)$
  - $R_1 \leftarrow r_1 G$

  Then, it returns these elements to the adversary.
  On the next round with $\mathsf{umsg}_2 = c$, return $c_0, c_1 = c - c_0, \boldsymbol{s}_0, s_1 = r_1 + c_1 \cdot w$. (For simplicity, we assume c_0, c_1 are both send – but in the protocol, only one can be derived from the other.)

To see that the distribution of the view of $\mathcal{A}$ is identical in ZK_0 and ZK_1 games, we consider the following:

- The distribution on $\mathsf{par}_{\mathsf{oNIP}}$ is identical to $\mathsf{oNIP.Setup}$, since W is still uniformly random.
- Next, because the simulator aborts correctly with the help of the oracle $\mathcal{O}_{\mathsf{SVerDDH}}$, we only have to consider the case when $(\zeta_x\|\zeta_y\|S_z) = M_{\mathsf{Aug},S_w,(C_i)_{i=1}^{\ell}}\mathsf{sk}$. Then, it is easy to see that the joint distribution of $(\boldsymbol{R}_{0,\mathsf{Core}}, \boldsymbol{R}_{0,\mathsf{Aug}}, R_1, c_0, c_1, \boldsymbol{s}_0, \boldsymbol{s}_1)$ conditioned on (umsg_1, c) are identical regardless of whether the issuer uses sk or w to run the protocol. ◻

Lemma 6.10 (Obliviousness of $\mathsf{oNIP}_{\mathsf{DDH}}$). *Let* $\mathsf{Sim}_{\mathsf{Gen}}$ *be the global parameters simulator for* $\mathsf{Gen}_{\mathsf{DDH}}$*. There exists a simulator* $\mathsf{Sim} = (\mathsf{Sim}_{\mathsf{Setup}}, \mathsf{Sim}_{\mathsf{U}}, \mathsf{Sim}_{\mathrm{Pf}})$ *such that*

- *For any adversary* $\mathcal{A}$*,* $\mathsf{Adv}^{\mathsf{par\text{-}indist}}_{\mathsf{oNIP}_{\mathsf{DDH}},\mathsf{Sim}}(\mathcal{A}, \lambda) = 0$*.*
- *For any adversary* $\mathcal{A}$*,* $\mathsf{Adv}^{\mathsf{zk}}_{\mathsf{oNIP}_{\mathsf{DDH}},\mathsf{Sim}_{\mathsf{Gen}},\mathsf{Sim}}(\mathcal{A}, \lambda) = 0$*.*

Proof. First, we note again that the simulator $\mathsf{Sim}_{\mathsf{Gen}}(1^\lambda)$ for the global parameters generator returns $\mathsf{par}_g = (p, G, \mathbb{G}, H)$ and $\mathsf{td}_g = v \leftarrow_\$ \mathbb{Z}_p^*$ such that $vG = H$. Now, consider the following simulator Sim:

- $\mathsf{Sim}_{\mathsf{Setup}}(p, G, \mathbb{G}, H)$: Sample $w \in \mathbb{Z}_p$ and return $(\mathsf{par}_{\mathsf{oNIP}} = (p, G, \mathbb{G}, W), \mathsf{td} = w)$.
- $\mathsf{Sim}_{\mathsf{U}}(\mathsf{td} = (v, w), \mathsf{pk})$:
  - First, parse $\mathsf{pk} = ((X_i)_{i=1}^{\ell}, (Y_i)_{i=1}^{\ell}, Z, \mathsf{ct}_x = (\mathsf{ct}_{x,0}, \mathsf{ct}_{x,1}), \mathsf{ct}_y = (\mathsf{ct}_{y,0}, \mathsf{ct}_{y,1}))$
  - The simulator then uses v to compute $X_0 \leftarrow \mathsf{ct}_{x,1} - v\mathsf{ct}_{x,0}, Y_0 \leftarrow \mathsf{ct}_{y,1} - v\mathsf{ct}_{y,0}$.
  - For the first move, sample $a \leftarrow_\$ \mathbb{Z}_p^*, \boldsymbol{\beta} \leftarrow_\$ \mathbb{Z}_p^{\ell}$. Compute $S_w \leftarrow \alpha G, C_i \leftarrow \beta_i H$ and $\zeta_x \leftarrow \alpha X_0 + \sum_{i=1}^{\ell} \beta_i X_i, \zeta_y \leftarrow \alpha Y_0 + \sum_{i=1}^{\ell} \beta_i Y_i, S_z \leftarrow \alpha Z$.
  - For the second move, return $c \leftarrow_\$ \mathbb{Z}_p$.
  - At the end of the protocol, the simulator checks if the transcript satisfies the check in $\mathsf{oNIP}.\mathsf{U}_3$.
- $\mathsf{Sim}_{\mathrm{Pf}}(\mathsf{td} = (v, w), \mathsf{pk}, \tau_{\mathsf{key}} = (S'_w, S'_z, (C'_i)_{i=1}^{\ell}, \zeta'_x, \zeta'_y))$: Sample $\boldsymbol{s}_0 \leftarrow_\$ \mathbb{Z}_p^{2\ell+6}, c_0, r_1 \leftarrow_\$ \mathbb{Z}_p$ and set
  - $\boldsymbol{R}_{0,\mathsf{Core}} \leftarrow M_{\mathsf{Core}}\boldsymbol{s}_0 - c_0\mathsf{pk}$
  - $\boldsymbol{R}_{0,\mathsf{Aug}} \leftarrow M_{\mathsf{Aug},S_w,(C_i)_{i=1}^{\ell}}\boldsymbol{s}_0 - c_0(\zeta_x \| \zeta_y \| S_z)$
  - $R_1 \leftarrow r_1 G$.

  Compute $c \leftarrow \mathsf{H}_c(\mathsf{pk}, \tau_{\mathsf{key}}, \boldsymbol{R}_{0,\mathsf{Core}}, \boldsymbol{R}_{0,\mathsf{Aug}}, R_1)$ and return $(c_0, c_1 = c - c_0, \boldsymbol{s}_0, s_1 = r_1 + c_1 \cdot w)$.

The distribution of $\mathsf{par}_{\mathsf{oNIP}}$ stays identical to that of $\mathsf{oNIP.Setup}$. Next, to show the advantage of $\mathcal{A}$ in the obliviousness game, we only consider the game where $\mathcal{A}$ only *starts* 1 *session.* Then, we can easily extend this to Q sessions via standard hybrid argument, since the reduction could use the trapdoor (in the OBLV game the adversary knows the trapdoor) to simulate other sessions which was changed to using a simulator.

To show indistinguishability, we first w.l.o.g. assume that $\mathcal{A}$'s randomness is fixed and it finishes the proof issuance session and sees the proof π. Also, we remark again that the game only consider issuance protocol for valid statements. We define the view of $\mathcal{A}$ after its execution as $V_{\mathcal{A}} = (H, W, \mathsf{pk}, \tau_{\mathsf{key}}, T, \pi)$ where $(\mathsf{pk}, \tau_{\mathsf{key}})$ is the statement the adversary selected, T is the transcript of the protocol, and π is the proof from Pf defined as

$$\begin{aligned} \mathsf{pk} &:= ((X_i)_{i=1}^{\ell}, (Y_i)_{i=1}^{\ell}, Z, \mathsf{ct}_x = (\mathsf{ct}_{x,0}, \mathsf{ct}_{x,1}), \mathsf{ct}_y = (\mathsf{ct}_{y,0}, \mathsf{ct}_{y,1})) \\ \tau_{\mathsf{key}} &:= (S'_w, S'_z, (C'_i)_{i=1}^{\ell}, \zeta'_x, \zeta'_y)\ , \\ T &:= ((S_w, S_z, (C_i)_{i=1}^{\ell}, \zeta_x, \zeta_y), \boldsymbol{R}_{0,\mathsf{Core}}, \boldsymbol{R}_{0,\mathsf{Aug}}, R_1, c, c_0, c_1, \boldsymbol{s}_0, s_1)\ , \\ \pi &:= (c'_0, c'_1, \boldsymbol{s}'_0, s'_1)\ . \end{aligned} \tag{6}$$

For simplicity, we assume c_0, c_1 are both sent. Since the randomness of $\mathcal{A}$ is fixed, we only consider the randomness of the honest user (i.e., $\mathrm{U}_1, \mathrm{U}_2$) and the simulator $\mathsf{Sim}_{\mathsf{U}}, \mathsf{Sim}_{\mathrm{Pf}}$. Denote η_b as the randomness of the honest user/simulator in the OBLV_b game, which are of the form

$$\eta_0 = (\alpha, \boldsymbol{\beta}, \gamma_0, \gamma_1, \boldsymbol{\delta}_0, \delta_1)\ , \eta_1 = (\alpha, \boldsymbol{\beta}, \bar{c}, \bar{c}'_0, \bar{\boldsymbol{s}}'_0, \bar{r}'_1)\ .$$

Note that $\bar{(\cdot)}$ is used to distinct the value in the transcript and the randomness of the simulator. Now, we only need to show that the distribution of $V_{\mathcal{A}}$ is identical in both cases of $b = 0, b = 1$, which we do so by showing that for any fixed view Δ where $\Pr[V_{\mathcal{A}} = \Delta | b = 1] > 0$, there is a unique randomness η_0, η_1 which results in $V_{\mathcal{A}} = \Delta$ for both cases. Since both η_0, η_1 consist of the same number of scalars ($1\mathbb{Z}_p^* + (2\ell + 3)\mathbb{Z}_p$ elements), this concludes the proof.

Now, we show that the claim above is true. (We note some abuse of notations here, and denote values in Δ using the corresponding letters for the random variables in $V_{\mathcal{A}}$.)

For $b = 0$, $V_\mathcal{A} = \Delta$ if and only if

$$\alpha = \mathrm{dlog}_G(S_w)/\mathrm{dlog}_G(S'_w)\ , \boldsymbol{\beta} = (\mathrm{dlog}_G(C_i - C'_i))_{i\in[\ell]}\ ,$$
$$\boldsymbol{\delta}_0 = \boldsymbol{s}'_0 - \boldsymbol{s}_0\ , \delta_1 = s'_1 - s_1, \gamma_0 = c'_0 - c_0, \gamma_1 = c'_1 - c_1\ .$$

The if part ($\Rightarrow$) follows easily from how the user algorithm is defined. To show the only-if direction, we have to show that with the defined randomness the protocol messages are as in Δ, i.e., as given in Equation (6). This follows from inspection, but due to the complexity of the protocol, we show the implication below.

First, we note that the statement $(\mathsf{pk}, \tau_{\mathsf{key}})$ in Δ needs to be in the language, meaning there exists $\mathsf{sk} = (\boldsymbol{x}, \boldsymbol{y}, z, t_x, t_y)$ such that $M_{\mathsf{Core}}\mathsf{sk} = \mathsf{pk}$ and $M_{\mathsf{Aug},S'_w,(C'_i)_{i=1}^\ell} = (\zeta'_x \| \zeta'_y \| S'_z)$. By how $\alpha, \boldsymbol{\beta}$ is defined, the user outputs $S_w, (C_i)_{i=1}^\ell$ as in Δ, and for (ζ_x, ζ_y, S_z),

$$\begin{aligned}
\alpha\zeta'_x + \sum_{i\in[\ell]} \beta_i X_i &= \alpha(x_0 S'_w + \sum_{i\in[\ell]} x_i C'_i) + \sum_{i\in[\ell]} \beta_i X_i \\
&= x_0 S_w + \sum_{i\in[\ell]} x_i C_i = \zeta_x\ , \\
\alpha\zeta'_y + \sum_{i\in[\ell]} \beta_i Y_i &= \alpha(y_0 S'_w + \sum_{i\in[\ell]} y_i C'_i) + \sum_{i\in[\ell]} \beta_i Y_i\ , \\
&= y_0 S_w + \sum_{i\in[\ell]} y_i C_i = \zeta_y\ , \\
\alpha S'_z &= \alpha z S'_w = z S_w = S_z\ .
\end{aligned}$$

Next, we have to show that the honest user sends c as in Δ. For the equations below, we additionally let $\boldsymbol{s}_0$ contains $((s_{0,x_i})_{i\in[\ell]}, (s_{0,y_i})_{i\in[\ell]}, s_{0,z}, s_{0,t_x}, s_{0,t_y})$. To see this, we consider the blinded values $\boldsymbol{R}'_{0,\mathsf{Core}}, \boldsymbol{R}'_{0,\mathsf{Aug}}, \boldsymbol{R}'_1$

$$\begin{aligned}
\boldsymbol{R}'_{0,\mathsf{Core}} &= \boldsymbol{R}_{0,\mathsf{Core}} + M_{\mathsf{Core}}\boldsymbol{\delta}_0 - \gamma_0\mathsf{pk} \\
&= M_{\mathsf{Core}}\boldsymbol{s}_0 - c_0\mathsf{pk} M_{\mathsf{Core}}\boldsymbol{\delta}_0 - \gamma_0\mathsf{pk} && \text{By } \mathsf{oNIP.U}_3 \text{ checks} \\
&= M_{\mathsf{Core}}\boldsymbol{s}'_0 - c'_0 \cdot \mathsf{pk} && \text{Def of } \boldsymbol{\delta}_0, \gamma_0 \\
\bar{\boldsymbol{R}}_{0,\mathsf{Aug}} &= \alpha^{-1}(\boldsymbol{R}_{0,\mathsf{Aug}} - \sum_{i\in[\ell]} \beta_i (R_{x,i} \| R_{y,i} \| 0)) \\
&= \alpha^{-1}(M_{\mathsf{Aug},S_w,(C_i)_{i=1}^\ell}\boldsymbol{s}_0 - c_0(\zeta_x \| \zeta_y \| S_z) - \sum_{i\in[\ell]} \beta_i (R_{x,i} \| R_{y,i} \| 0)) && \text{By } \mathsf{oNIP.U}_3 \text{ checks} \\
&= \alpha^{-1} \begin{bmatrix} s_{0,x_0} S_w - c_0\zeta_x - \sum_{i=1}^\ell s_{0,x_i} C_i - \beta_i(s_{0,x_i} H - c_0 X_i) \\ s_{0,y_0} S_w - c_0\zeta_y - \sum_{i=1}^\ell s_{0,y_i} C_i - \beta_i(s_{0,y_i} H - c_0 Y_i) \\ s_{0,z} S_w - c_0 S_z \end{bmatrix} && \text{By } \mathsf{oNIP.U}_3 \text{ checks} \\
&= \begin{bmatrix} s_{0,x_0} S'_w - c_0\zeta'_x - \sum_{i=1}^\ell s_{0,x_i} C'_i \\ s_{0,y_0} S'_w - c_0\zeta'_y - \sum_{i=1}^\ell s_{0,y_i} C'_i \\ s_{0,z} S'_w - c_0 S'_z \end{bmatrix} && \text{Def of } \boldsymbol{\beta}, \alpha \\
&= M_{\mathsf{Aug},S'_w,(C'_i)_{i=1}^\ell}\boldsymbol{s}_0 - c_0(\zeta'_x \| \zeta'_y \| S'_z) \\
\boldsymbol{R}'_{0,\mathsf{Aug}} &= \bar{\boldsymbol{R}}_{0,\mathsf{Aug}} + M_{\mathsf{Aug}}\boldsymbol{\delta}_0 - \gamma_0(\zeta'_x \| \zeta'_y \| S'_z) \\
&= M_{\mathsf{Aug},S'_w,(C'_i)_{i=1}^\ell}\boldsymbol{s}'_0 - c'_0(\zeta'_x \| \zeta'_y \| S'_z) && \text{Def of } \boldsymbol{\delta}_0, \gamma_0 \\
R'_1 &= R_1 + \delta_1 G - \gamma_1 W \\
&= s_1 G - c_1 W + \delta_1 G - \gamma_1 W && \text{By } \mathsf{oNIP.U}_3 \text{ checks} \\
&= s'_1 G - c'_1 W && \text{Def of } \delta_1, \gamma_1
\end{aligned}$$

Hence, because π verifies, the user sends $\mathsf{H}_c(\mathsf{pk}, \tau'_{\mathsf{key}}, \boldsymbol{R}'_{0,\mathsf{Core}}, \boldsymbol{R}'_{0,\mathsf{Aug}}, R_1) - \gamma_0 - \gamma_1 = c'_0 + c'_1 - \gamma_0 - \gamma_1 = c_0 + c_1 = c$. Finally, it is clear from the equations above and how $\gamma_0, \gamma_1, \boldsymbol{\delta}_0, \delta_1$ are defined that the output of the oracle Pf is (c'_0, c'_1, s'_0, s'_1).

For $b = 1$, $V_{\mathcal{A}} = \Delta$ if and only if

$$\alpha = \mathrm{dlog}_G(S_w)\ , \boldsymbol{\beta} = (\mathrm{dlog}_G(C_i))_{i \in [\ell]}\ ,$$
$$\bar{c} = c\ , \bar{c}'_0 = c'_0\ , \bar{\boldsymbol{s}}'_0 = \boldsymbol{s}'_0\ , \bar{r}'_1 = s'_1 - c'_1 \mathrm{dlog}_G W\ .$$

The if direction ($\Rightarrow$) follows easily from the equations and the fact that the final proof π verifies. For the only-if direction, $\alpha, \boldsymbol{\beta}$ ensures that $(S_w, S_z, (C_i)_{i=1}^{\ell}, \zeta_x, \zeta_y)$ as in Δ is sent, and $\bar{c}$ ensures that the second user message is c. Finally, because the final proof verifies, $c'_0 + c'_1 = \mathsf{H}_c(\mathsf{pk}, \tau'_{\mathsf{key}}, R'_{0,\mathsf{Core}}, R'_{0,\mathsf{Aug}}, R'_1)$ where $R'_{0,\mathsf{Core}}, R'_{0,\mathsf{Aug}}, R'_1$ are defined as in the verification algorithm. Then, the values of $\bar{c}'_0, \bar{s}'_0, \bar{r}'_1$ ensures that the proof π is exactly what is in the transcript Δ. $\square$

Acknowledgements

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

ARF24. The european digital identity wallet architecture and reference framework, 2024. Accessed: 2025-02-13.

ASM06. Man Ho Au, Willy Susilo, and Yi Mu. Constant-size dynamic k-TAA. In Roberto De Prisco and Moti Yung, editors, *SCN 06*, volume 4116 of *LNCS*, pages 111–125. Springer, Berlin, Heidelberg, September 2006.

BB08. Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of Cryptology*, 21(2):149–177, April 2008.

BBC+24. Carsten Baum, Olivier Blazy, Jan Camenisch, Jaap-Henk Hoepman, Eysa Lee, Anja Lehmann, Anna Lysyanskaya, René Mayrhofer, Hart Montgomery, Ngoc Khanh Nguyen, Bart Praneel, abhi shelat, Daniel Slamanig, Stefano Tessaro, Søren Eller Thomsen, and Carmela Troncoso. Cryptographers' feedback on the eu digital identity's ARF. https://github.com/user-attachments/files/15904122/cryptographers-feedback.pdf, 2024.

BBDT16. Amira Barki, Solenn Brunet, Nicolas Desmoulins, and Jacques Traoré. Improved algebraic MACs and practical keyed-verification anonymous credentials. In Roberto Avanzi and Howard M. Heys, editors, *SAC 2016*, volume 10532 of *LNCS*, pages 360–380. Springer, Cham, August 2016.

BBS04. Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 41–55. Springer, Berlin, Heidelberg, August 2004.

BCR+. Elaine Barker, Lily Chen, Allen Roginsky, Apostol Vassilev, and Richard Davis. Recommendation for pair-wise key-establishment schemes using discrete logarithm cryptography. Technical Report NIST Special Publication 800-56A, National Institute of Standards (NIST). Accessed: 2025-02-13.

BL. Daniel J. Bernstein and Tanja Lange. Safecurves: choosing safe curves for elliptic-curve cryptography. Accessed: 2025-02-13.

BL13. Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 1087–1098. ACM Press, November 2013.

BLL+21. Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. On the (in)security of ROS. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 33–53. Springer, Cham, October 2021.

BR93. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

Bra99. Stefan Brands. *Rethinking Public Key Infrastructure and Digital Certificates— Building in Privacy*. PhD thesis, Eindhoven Inst. of Tech. The Netherlands, 1999.

BS20. Dan Boneh and Victor Shoup. A graduate course in applied cryptography (2020). *A book in preparation, v0*, 5:80, 2020.

CATZ24. Rutchathon Chairattana-Apirom, Stefano Tessaro, and Chenzhi Zhu. Pairing-free blind signatures from CDH assumptions. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part I*, volume 14920 of *LNCS*, pages 174–209. Springer, Cham, August 2024.

CDDH19. Jan Camenisch, Manu Drijvers, Petr Dzurenda, and Jan Hajny. Fast keyed-verification anonymous credentials on standard smart cards. In *ICT Systems Security and Privacy Protection: 34th IFIP TC 11 International Conference, SEC 2019, Lisbon, Portugal, June 25-27, 2019, Proceedings 34*, pages 286–298. Springer, 2019.

CDL16. Jan Camenisch, Manu Drijvers, and Anja Lehmann. Anonymous attestation using the strong diffie hellman assumption revisited. In *Trust and Trustworthy Computing: 9th International Conference, TRUST 2016, Vienna, Austria, August 29-30, 2016, Proceedings 9*, pages 1–20. Springer, 2016.

Cha82. David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82*, pages 199–203. Plenum Press, New York, USA, 1982.

CKL^{+}16. Jan Camenisch, Stephan Krenn, Anja Lehmann, Gert Læssøe Mikkelsen, Gregory Neven, and Michael Østergaard Pedersen. Formal treatment of privacy-enhancing credential systems. In Orr Dunkelman and Liam Keliher, editors, *SAC 2015*, volume 9566 of *LNCS*, pages 3–24. Springer, Cham, August 2016.

CKS08. David Cash, Eike Kiltz, and Victor Shoup. The twin Diffie-Hellman problem and applications. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 127–145. Springer, Berlin, Heidelberg, April 2008.

CL01. Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer, Berlin, Heidelberg, May 2001.

CL03. Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN 02*, volume 2576 of *LNCS*, pages 268–289. Springer, Berlin, Heidelberg, September 2003.

CL04. Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72. Springer, Berlin, Heidelberg, August 2004.

CMZ14. Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. Algebraic MACs and keyed-verification anonymous credentials. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 1205–1216. ACM Press, November 2014.

CPZ20. Melissa Chase, Trevor Perrin, and Greg Zaverucha. The Signal private group system and anonymous credentials supporting efficient verifiable encryption. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1445–1459. ACM Press, November 2020.

CV02. Jan Camenisch and Els Van Herreweghen. Design and implementation of the idemix anonymous credential system. In Vijayalakshmi Atluri, editor, *ACM CCS 2002*, pages 21–30. ACM Press, November 2002.

DHH^{+}21. Nico Döttling, Dominik Hartmann, Dennis Hofheinz, Eike Kiltz, Sven Schäge, and Bogdan Ursu. On the impossibility of purely algebraic signatures. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part III*, volume 13044 of *LNCS*, pages 317–349. Springer, Cham, November 2021.

EHK^{+}13. Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Villar. An algebraic framework for Diffie-Hellman assumptions. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 129–147. Springer, Berlin, Heidelberg, August 2013.

Fis05. Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 152–168. Springer, Berlin, Heidelberg, August 2005.

Ger24. Architecture proposal for the german eIDAS implementation, 2024. Accessed: 2025-02-13.

Har12. Dick Hardt. The OAuth 2.0 Authorization Framework. RFC 6749, October 2012.

JT20. Joseph Jaeger and Stefano Tessaro. Expected-time cryptography: Generic techniques and applications to concrete soundness. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part III*, volume 12552 of *LNCS*, pages 414–443. Springer, Cham, November 2020.

KLR23. Julia Kastner, Julian Loss, and Omar Renawi. Concurrent security of anonymous credentials light, revisited. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 45–59. ACM Press, November 2023.

KRW24. Michael Klooß, Michael Reichle, and Benedikt Wagner. Practical blind signatures in pairing-free groups. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part I*, volume 15484 of *LNCS*, pages 363–395. Springer, Singapore, December 2024.

Ks22. Yashvanth Kondi and abhi shelat. Improved straight-line extraction in the random oracle model with applications to signature aggregation. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part II*, volume 13792 of *LNCS*, pages 279–309. Springer, Cham, December 2022.

LKWL24. Tobias Looker, Vasilis Kalos, Andrew Whitehead, and Mike Lodder. The BBS Signature Scheme. Internet-Draft draft-irtf-cfrg-bbs-signatures-07, Internet Engineering Task Force, September 2024. Work in Progress.

LRSW99. Anna Lysyanskaya, Ron Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In Howard Heys and Carlisle Adams, editors, *Selected Areas in Cryptography*, volume 1758 of *LNCS*, 1999.

Lys02. Anna Lysyanskaya. *Signature schemes and applications to cryptographic protocol design.* PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 2002.

Mau15. Ueli Maurer. Zero-knowledge proofs of knowledge for group homomorphisms. *DCC*, 77(2-3):663–676, 2015.

MBS^{+}25. Omid Mirzamohammadi, Jan Bobolz, Mahdi Sedaghat, Emad Heydari Beni, Aysajan Abidin, Dave Singelee, and Bart Preneel. Keyed-verification anonymous credentials with highly efficient partial disclosure. Cryptology ePrint Archive, Paper 2025/041, 2025.

Ora. Orange Innovation. The BBS# protocol: technical details. Accessed: 2025-02-13.

Orr24. Michele Orrù. Revisiting keyed-verification anonymous credentials. Cryptology ePrint Archive, Paper 2024/1552, 2024.

OTZZ24. Michele Orrù, Stefano Tessaro, Greg Zaverucha, and Chenzhi Zhu. Oblivious issuance of proofs. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part IX*, volume 14928 of *LNCS*, pages 254–287. Springer, Cham, August 2024.

PZ13. Christian Paquin and Greg Zaverucha. U-prove cryptographic specification v1.1 (revision 3), December 2013. Released under the Open Specification Promise (http://www.microsoft.com/openspecifications/en/us/programs/osp/default.aspx).

TD. Jacques Traoré and Antoine Dumanois. BBS# and eIDAS 2.0.: Making BBS anonymous credentials eIDAS 2.0 compliant. Accessed: 2025-02-13.

TZ23a. Stefano Tessaro and Chenzhi Zhu. Revisiting BBS signatures. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 691–721. Springer, Cham, April 2023.

TZ23b. Stefano Tessaro and Chenzhi Zhu. Threshold and multi-signature schemes from linear hash functions. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 628–658. Springer, Cham, April 2023.

A Multi-User Anonymity of SAAC

The multi-user anonymity game of SAAC is defined in Figure 20. The game is similar to the single-user case, except that the adversary is allowed to issue more than one credential through the $\mathrm{U}_1, \mathrm{U}_2$ oracles. Note that the adversary is also allowed specify which user/credential is being shown (through specifying the credential ID cid) when the user requests the helper proof and during showing in the SH oracle. The corresponding advantage of $\mathcal{A}$ is

$$\mathsf{Adv}^{\text{mu-anon}}_{\mathsf{SAAC},\mathsf{Sim}}(\mathcal{A}, \lambda) := |\mathsf{Pr}[\text{MU-Anon}^{\mathcal{A}}_{\mathsf{SAAC},\mathsf{Sim},0} = 1] - \mathsf{Pr}[\text{MU-Anon}^{\mathcal{A}}_{\mathsf{SAAC},\mathsf{Sim},1} = 1]| \ .$$

For readability, we give more details on our multi-user anonymity game here. The adversary will first receive the public parameters par and the trapdoor td generated by the simulator. It will then have access to the following oracles.

- **Initialization oracle Init:** This oracle allows the adversary to initialize its own issuer's public key.
- **User oracles $\mathbf{U}_1, \mathbf{U}_2$:** The adversary (as a malicious issuer) can specify the attributes $\boldsymbol{m}$ and the predicate ϕ. For these oracles, the adversary would interact with either an honest user requesting a credential of $\boldsymbol{m}$ or a simulator which *does not know* $\boldsymbol{m}$. Note that each attributes vector $\boldsymbol{m}_{\mathsf{cid}}$ and credential σ_{cid} obtained by the honest user is indexed with a credential ID cid.
- **Obtain/Request help oracle** $\mathrm{ObtH}_1, \ldots \mathrm{ObtH}_{r+1}$**:** The adversary is allowed to specify a credential ID cid to force a user holding σ_{cid} to request a helper information. In these oracles, the adversary would interact with either an honest user, who knows the attributes $\boldsymbol{m}_{\mathsf{cid}}$ and the credential σ_{cid}, or the simulator, who does not know either of those values. At the end, the user would receive a helper information $\mathsf{aux}_{\mathsf{sid}}$ tied to the session ID sid.
- **Credential showing oracle** SH**:** The adversary is allowed to specify a helper information (via sid) owned by a honest user, a predicate ϕ, and an additional value nonce, such that the honest user uses the helper information $\mathsf{aux}_{\mathsf{sid}}$ to show a credential σ_{cid} for attributes satisfying ϕ. Note that each helper information is restricted to only be used once. On the other hand, the simulator only needs the trapdoor td, the public key pk, and the specified predicate ϕ to simulate.

The following lemma shows that single-user anonymity (defined in Section 3.2) implies multi-user anonymity.

Lemma A.1 (Multi-User Anonymity). *Let* SAAC *be a server-aided anonymous credentials scheme which is single-user anonymous with respect to a simulator* Sim*. For any adversary* $\mathcal{A}$ *playing the* MU-Anon *game with respect to the simulator* Sim *making at most* $q = q(\lambda), q_{\mathrm{ObtH}} = q_{\mathrm{ObtH}}(\lambda), q_{\mathrm{SH}} = q_{\mathrm{SH}}(\lambda)$ *queries to oracles* $\mathrm{U}_1, \mathrm{ObtH}_1, \mathrm{SH}$ *respectively, there exists an adversary* $\mathcal{B}$ *playing the* Anon *game with respect to the simulator* Sim *such that*

$$\mathsf{Adv}^{\text{mu-anon}}_{\mathsf{SAAC},\mathsf{Sim}}(\mathcal{A}, \lambda) \leqslant q \cdot \mathsf{Adv}^{\text{anon}}_{\mathsf{SAAC},\mathsf{Sim}}(\mathcal{A}, \lambda) \ .$$

Additionally, $\mathcal{B}$ *makes at most* $q_{\mathrm{ObtH}}, q_{\mathrm{SH}}$ *queries to its* $\mathrm{ObtH}_1, \mathrm{SH}$ *oracles.*

Proof. Let $\mathcal{A}$ be the adversary playing MU-Anon game making $q, q_{\mathrm{ObtH}}, q_{\mathrm{SH}}$ queries to oracles $\mathrm{U}_1, \mathrm{ObtH}_1, \mathrm{SH}$ respectively. We then consider the following sequence of games $\mathbf{G}^{\mathcal{A}}_0(\lambda), \ldots, \mathbf{G}^{\mathcal{A}}_Q(\lambda)$.

For $i = 0, \ldots, q$, the game $\mathbf{G}^{\mathcal{A}}_i(\lambda)$ is defined as follows:

- The public parameters and trapdoor $(\mathsf{par}, \mathsf{td})$ are generated from the simulator $\mathsf{Sim}_{\mathsf{Setup}}$. The oracle Init stays the same.
- For the j-th query to U_1 for $j \in [q]$:
  - Denote $\mathsf{cid}^{(j)}$ as the corresponding credential ID cid of this session (only if the oracle does not abort after checking the validity of the inputs).
  - If $1 \leqslant j \leqslant i$: Compute μ using the simulator $\mathsf{Sim}_{\mathsf{U}}$ (as in the game $\text{MU-Anon}_{\mathsf{SAAC},\mathsf{Sim},1}$).
  - Else $i < j \leqslant q$: Compute μ using the user algorithm $\mathsf{SAAC}.\mathsf{U}_1$ (as in the game $\text{MU-Anon}_{\mathsf{SAAC},\mathsf{Sim},1}$).

Game $\text{MU-Anon}^{\mathcal{A}}_{\mathsf{SAAC},\mathsf{Sim},b}(\lambda)$:

$\mathtt{init} \leftarrow 0; \mathcal{I}_1, \ldots, \mathcal{I}_r, \mathcal{HP}, \mathcal{C}_1, \mathcal{C}_2 \leftarrow \varnothing$

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

$b' \leftarrow\$ \mathcal{A}^{\text{INIT},\text{U}_1,\text{U}_2,\text{ObtH}_1,\ldots,\text{ObtH}_{r+1},\text{SH}}(\mathsf{par}, \mathsf{td})$

return b'

Oracle $\text{ObtH}_1(\mathsf{cid}, \mathsf{sid})$:

if $\mathsf{sid} \in \mathcal{I}_1 \ \vee\ \mathsf{cid} \notin \mathcal{C}_2$
  then abort

$\mathcal{I}_1 \leftarrow \mathcal{I}_1 \cup \{\mathsf{sid}\}; \mathsf{cid}_{\mathsf{sid}} \leftarrow \mathsf{cid}$

if $j = 1$ **then** $/\!\!/\ b = 0$
  $(\mathsf{umsg}_1, \mathsf{st}_{\mathsf{sid}}) \leftarrow\$$
    $\mathsf{SAAC.ObtHelp}_1(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}_{\mathsf{cid}}, \sigma_{\mathsf{cid}})$
  return umsg_1

if $j = 1$ **then** $/\!\!/\ b = 1$
  $(\mathsf{umsg}_1, \mathsf{st}_{\mathsf{sid}}) \leftarrow\$ \mathsf{Sim}_{\mathsf{ObtH}}(\mathsf{td}, \mathsf{pk})$

return umsg_1

Oracle $\text{ObtH}_j(\mathsf{sid}, \mathsf{hmsg}_{j-1})$: $/\!\!/\ j = 2, \ldots, r+1$

if $\mathsf{sid} \notin \mathcal{I}_1, \ldots, \mathcal{I}_{j-1} \ \vee\ \mathsf{sid} \in \mathcal{I}_j$
  then abort

$\mathcal{I}_j \leftarrow \mathcal{I}_j \cup \{\mathsf{sid}\}$

if $1 < j \leqslant r$ **then** $/\!\!/\ b = 0$
  $(\mathsf{umsg}_j, \mathsf{st}_{\mathsf{sid}}) \leftarrow\$ \mathsf{SAAC.ObtHelp}_j(\mathsf{st}_{\mathsf{sid}}, \mathsf{hmsg}_{j-1})$
  return umsg_j
if $j = r+1$ **then**
  $\mathsf{aux}_{\mathsf{sid}} \leftarrow\$ \mathsf{SAAC.ObtHelp}_{r+1}(\mathsf{st}_{\mathsf{sid}}, \mathsf{hmsg}_r)$
  if $\mathsf{aux}_{\mathsf{sid}} = \perp$ **then abort**
  $\mathcal{HP} \leftarrow \mathcal{HP} \cup \{\mathsf{sid}\}$

if $1 < j \leqslant r$ **then** $/\!\!/\ b = 1$
  $(\mathsf{umsg}_j, \mathsf{st}_{\mathsf{sid}}) \leftarrow\$ \mathsf{Sim}_{\mathsf{ObtH}}(\mathsf{st}_{\mathsf{sid}}, \mathsf{hmsg}_{j-1})$
  return umsg_j
if $j = r+1$ **then**
  $b_{\mathsf{sid}} \leftarrow\$ \mathsf{Sim}_{\mathsf{ObtH}}(\mathsf{st}_{\mathsf{sid}}, \mathsf{hmsg}_{j-1})$
  if $b_{\mathsf{sid}} = 0$ **then abort**
  $\mathcal{HP} \leftarrow \mathcal{HP} \cup \{\mathsf{sid}\}$

return $\mathtt{closed}$

Oracle $\text{INIT}(\tilde{\mathsf{pk}})$:

if $\mathtt{init} = 1$ **then abort**

$\mathtt{init} \leftarrow 1; \mathsf{pk} \leftarrow \tilde{\mathsf{pk}}$

return $\mathtt{closed}$

Oracle $\text{U}_1(\mathsf{cid}, \boldsymbol{m}, \phi)$:

if $\phi(\boldsymbol{m}) = 0 \ \vee\ \mathtt{init} = 0 \ \vee\ \mathsf{cid} \in \mathcal{C}_1$ **then**
  abort

$\mathcal{C}_1 \leftarrow \mathcal{C}_1 \cup \{\mathsf{cid}\}$

$\boldsymbol{m}_{\mathsf{cid}} \leftarrow \boldsymbol{m}$

$(\mu, \mathsf{st}^u_{\mathsf{cid}}) \leftarrow\$ \mathsf{SAAC.U}_1(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}, \phi)$
  $/\!\!/\ b = 0$

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

return μ

Oracle $\text{U}_2(\mathsf{cid}, \mathsf{imsg})$:

if $\mathsf{cid} \notin \mathcal{C}_1 \ \vee\ \mathsf{cid} \in \mathcal{C}_2$ **then** $\perp$

$\mathcal{C}_2 \leftarrow \mathcal{C}_2 \cup \{\mathsf{cid}\}$

$\sigma_{\mathsf{cid}} \leftarrow\$ \mathsf{SAAC.U}_2(\mathsf{st}^u_{\mathsf{cid}}, \mathsf{imsg})$
if $\sigma_{\mathsf{cid}} = \perp$ **then abort** $/\!\!/\ b = 0$

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

return $\mathtt{closed}$

Oracle $\text{SH}(\mathsf{sid}, \phi, \mathsf{nonce})$:

$\mathsf{cid} \leftarrow \mathsf{cid}_{\mathsf{sid}}$

if $\mathsf{cid} = \perp \ \vee\ \phi(\boldsymbol{m}_{\mathsf{cid}}) = 0 \ \vee\ \mathsf{sid} \notin \mathcal{HP}$
  then abort

$\mathcal{HP} \leftarrow \mathcal{HP} \setminus \{\mathsf{sid}\}$

$\pi \leftarrow\$$ $/\!\!/\ b = 0$
  $\mathsf{SAAC.Show}(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}_{\mathsf{cid}}, \sigma_{\mathsf{cid}}, \mathsf{aux}_{\mathsf{sid}}, \phi, \mathsf{nonce})$

$\pi \leftarrow\$ \mathsf{Sim}_{\mathsf{Show}}(\mathsf{td}, \mathsf{pk}, \phi, \mathsf{nonce})$ $/\!\!/\ b = 1$

return π

Fig. 20. Anonymity game for SAAC for multi-user and single-user (defined including the dotted boxes). The game is parameterized with a simulator Sim and the goal of the adversary $\mathcal{A}$ is to guess whether it is interacting with honest users (case $b = 0$, denoted in the dashed boxes) or the simulator (case $b = 1$, denoted in the dashed and highlighted boxes). We note that when querying the oracle SH, the adversary can specify the session ID corresponding to a helper information aux which the user will use in the showing algorithm.

- For query to U_2 corresponding to $\mathsf{cid}^{(j)}$ for some $j \in [q]$: If the oracle does not abort while checking the validity of the input $(\mathsf{cid}, \mathsf{imsg})$, it uses the simulator as in the game MU-Anon$_{\mathsf{SAAC},\mathsf{Sim},1}$ if $j \leqslant i$; otherwise, it runs the user algorithm as in the game MU-Anon$_{\mathsf{SAAC},\mathsf{Sim},0}$.
- For oracles $\mathrm{ObtH}_1, \ldots, \mathrm{ObtH}_{r+1}$: Let j be such that $\mathsf{cid}^{(j)}$ corresponds to $\mathsf{cid}_{\mathsf{sid}}$. (Assuming the input to the oracles do not force the validity checks to abort.) Then, these oracles are run with the simulator $\mathsf{Sim}_{\mathsf{ObtH}}$ if $j \leqslant i$; otherwise, they are run with the $\mathsf{SAAC.ObtHelp}$ algorithms while knowing the corresponding attributes $\boldsymbol{m}_{\mathsf{cid}_{\mathsf{sid}}}$ and credential $\sigma_{\mathsf{cid}_{\mathsf{sid}}}$.
- For oracle SH on input $(\mathsf{sid}, \phi, \mathsf{nonce})$: Let j be such that $\mathsf{cid}^{(j)}$ corresponds to $\mathsf{cid}_{\mathsf{sid}}$. (Again, assuming no input-check aborts.) Then, these oracles are run with the simulator $\mathsf{Sim}_{\mathsf{Show}}$ if $j \leqslant i$; otherwise, they are run with the $\mathsf{SAAC.ObtHelp}$ algorithms while knowing the corresponding attributes $\boldsymbol{m}_{\mathsf{cid}_{\mathsf{sid}}}$, the credential $\sigma_{\mathsf{cid}_{\mathsf{sid}}}$, and the helper proof π.
- The output b' of $\mathcal{A}$ is returned by the game.

Notice that $\mathbf{G}_0^{\mathcal{A}}(\lambda)$ and $\mathbf{G}_q^{\mathcal{A}}(\lambda)$ are exactly MU-Anon$_{\mathsf{SAAC},\mathsf{Sim},0}$ and MU-Anon$_{\mathsf{SAAC},\mathsf{Sim},1}$ games, respectively. Moreover, there exists a reduction $\mathcal{B}$ playing the Anon game with respect to the same simulator Sim such that the bound in the lemma is satisfied. The reduction $\mathcal{B}$ is defined as follows:

- It takes as input $(\mathsf{par}, \mathsf{td})$ and samples $i^* \in [q]$. It then runs $\mathcal{A}$ on input $(\mathsf{par}, \mathsf{td})$. Note that for the Init call by $\mathcal{A}$, the reduction simply saves the public key pk into its state.
- Simulates the oracles as in the game $\mathbf{G}_{i*}$ with the exception that in the oracles where the corresponding index j (added in the description of the game) is i^*, the reduction forwards the values to its own game as follows:
  - For U_1 on input $(\mathsf{cid}, \boldsymbol{m}, \boldsymbol{\phi})$, it returns $(\mathsf{pk}, \boldsymbol{m}, \boldsymbol{\phi})$ along with its state and receives μ which is forwarded to $\mathcal{A}$

    For U_2 on input $(\mathsf{cid}, \mathsf{imsg})$, it returns imsg along with its state.
  - For oracles ObtH_k with $k \in [r+1]$ and SH, the inputs are forwarded to its ObtH_k oracle and the returned values are forwarded to $\mathcal{A}$.
- The output b' of $\mathcal{A}$ is returned to its game.

It is easy to see that if the Anon game uses honest user and $i^* = i$, the view of $\mathcal{A}$ is identical to its view in $\mathbf{G}_{i-1}$. Similarly, the view of $\mathcal{A}$ when the Anon game uses the simulator is identical to its view in $\mathbf{G}_i$. Hence, it follows that

$$\mathsf{Adv}^{\mathsf{mu\text{-}anon}}_{\mathsf{SAAC},\mathsf{Sim}}(\mathcal{A}, \lambda) \leqslant \sum_{i=1}^{q} \left|\Pr[\mathbf{G}_i^{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{G}_{i-1}^{\mathcal{A}}(\lambda) = 1]\right| \leqslant q \cdot \mathsf{Adv}^{\mathsf{anon}}_{\mathsf{SAAC},\mathsf{Sim}}(\mathcal{A}, \lambda) \,.$$

B Integrity of SAAC

In this section we prove Theorem 3.1, which states that weak integrity is implied by anonymity and correctness. We remark that this result relies on the fact that our η-correctness definition states that the correctness experiment should succeed with probability $1-\eta$ for any fixed key pair which can possibly be output by the key generation algorithm. If our definition instead stated that the probability should be $1-\eta$ taken over a set of random coins used to generate a key pair (in addition to the random coins used to run the algorithms), then Theorem 3.1 would be false.

Proof (Theorem 3.1). Suppose that SAAC satisfies anonymity with respect to some simulator Sim. This immediately implies that SAAC satisfies an even harder (for an adversary to win) version of the anonymity game where the adversary does not get to see the trapdoor, and they have to output ρ which will be used as randomness for key generation, i.e., $\mathsf{pk} \leftarrow \mathsf{SAAC.KeyGen}(\mathsf{par}; \rho)$. We consider a version of that anonymity game parameterized by a bit b where, after outputting the randomness for the key, the adversary first outputs two message-predicate pairs $(\boldsymbol{m}_0, \tilde{\phi}_0), (\boldsymbol{m}_1, \tilde{\phi}_1)$. The adversary interacts with two honest users both using pk in separately identifiable sessions, user A using $\boldsymbol{m}_0$ and $\tilde{\phi}_0$, and user B using $\boldsymbol{m}_1$ and $\tilde{\phi}_1$, for one run of the issuance protocol and one run of the helper protocol. The respective credentials and pieces of helper information are not revealed to the adversary, and if either honest user outputs $\perp$ for their credential or showing then the game aborts, i.e. outputs 1. At the end of the game, the adversary gets to output a predicate ϕ and a nonce nonce, and is given $\tau \leftarrow \mathsf{SAAC.Show}(\mathsf{par}, \mathsf{pk}, \boldsymbol{m}_b, \sigma_b, \mathsf{aux}_b, \phi, \mathsf{nonce})$. Additionally, the game aborts (outputs 1) if any of $\tilde{\phi}_0(\boldsymbol{m}_0)$, $\tilde{\phi}_1(\boldsymbol{m}_1)$, or $\phi(\boldsymbol{m}_b)$ are zero. The adversary outputs a bit representing a guess for whether it got a credential produced by user A or user B. If we call this game $\mathbf{G}_b$, then one can show via a hybrid argument that

$$\left|\Pr[\mathbf{G}_0^{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{G}_1^{\mathcal{A}}(\lambda) = 1]\right| \leqslant 2\mathsf{Adv}^{\mathsf{anon}}_{\mathsf{SAAC},\mathsf{Sim}}(\mathcal{A}, \lambda).$$

Also, suppose that $\mathbf{G}_b$ does $\mathsf{par} \leftarrow \mathsf{SAAC.Setup}(1^\lambda, 1^\ell)$ instead of $(\mathsf{par}, \mathsf{td}) \leftarrow_{\$} \mathsf{Sim}_{\mathsf{Setup}}(1^\lambda, 1^\ell)$, which we can argue by using parameter indistinguishability of Sim twice.

Let $\mathcal{A}$ be an adversary against the integrity of SAAC. On input par, the reduction $\mathcal{B}$ runs $(\rho, \boldsymbol{m}, \tilde{\phi}, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(\mathsf{par})$ and outputs randomness ρ and message-signature pairs $(\boldsymbol{m}, \tilde{\phi})$ and $(\boldsymbol{m}, \tilde{\phi})$. The reduction $\mathcal{B}$ interacts with the challenger, in the first session using the honest issuer protocols, and in the second session using $\mathcal{A}$. After the interactions, $\mathcal{B}$ runs $(\phi, \mathsf{nonce}) \leftarrow \mathcal{A}(\mathsf{st}''_{\mathcal{A}})$. To finish, $\mathcal{B}$ requests a showing τ for (ϕ, nonce) and outputs $\mathsf{SAAC.SVer}(\mathsf{par}, \mathsf{pk}, \tau, \phi, \mathsf{nonce})$. If the showing is relative to the first session, then the showing is valid by correctness with probability at least $1 - \eta$. On the other hand, if the showing is relative to the second session, then the showing is valid with probability that $\mathcal{A}$ loses the integrity game. More formally, if SAAC has η-correctness, then

$$\left|\Pr[\mathbf{G}_0^{\mathcal{B}}(\lambda) = 1] - \Pr[\mathbf{G}_1^{\mathcal{B}}(\lambda) = 1]\right| \geqslant \left|(1 - \eta) - (1 - \mathsf{Adv}^{\mathsf{integ}}_{\mathsf{SAAC}}(\mathcal{A}, \lambda))\right|.$$

Thus $\mathsf{Adv}^{\mathsf{integ}}_{\mathsf{SAAC}}(\mathcal{A}, \lambda) \leqslant \left|\Pr[\mathbf{G}_0^{\mathcal{B}}(\lambda) = 1] - \Pr[\mathbf{G}_1^{\mathcal{B}}(\lambda) = 1]\right| + \eta$. □

C Construction of Straight-line Extractable Proofs

In this section, we recall a variant of the (randomized) Fischlin transform [Fis05, Ks22] for Σ-protocols with super-polynomial challenge space which was given in [KRW24]. The transformation require that the Σ-protocol $\Sigma = (\mathsf{Init}, \mathsf{Resp}, \mathsf{Verify})$ for a relation R satisfies the following property in addition to correctness, HVZK, high min-entropy, and special soundness:

(Relaxed) Strong Special Soundness. For a relaxed relation $\widetilde{\mathsf{R}} \supseteq \mathsf{R}$, there exists an efficient deterministic extractor Ext such that for any statement x and valid transcripts $(R, c, z) \neq (R, c', z')$, $w \leftarrow \mathsf{Ext}(x, R, c, c', z, z')$ is such that $(x, w) \in \widetilde{\mathsf{R}}$.[8]

Although we do not recall the transformation, we remark that the simplified randomized Fischlin transform gives an NIZK in the random oracle model where the construction depends on the following parameters:

- Challenge space: $k = \log(|\mathcal{CH}|) \geqslant 4$ where $\mathcal{CH}$ is the challenge space of Σ.
- Random oracle output bit-size: $b = b(\lambda)$ such that $\mathsf{H} : \{0,1\}^* \to \{0,1\}^b$.
- Parallel repetition: $r = r(\lambda) \in \mathbb{N}$.
- Iterations: $t = t(\lambda) \in \mathbb{N}$ denoting the maximum restart 2^t. Note that we require $2^t = \mathsf{poly}(\lambda)$.

Now, we restate the results given in [KRW24, Appendix C.].

Theorem C.1. *Let Σ be a Σ-protocol for a relation R that also satisfies high min-entropy and strong special soundness for a relaxed relation $\widetilde{\mathsf{R}}$. Then, the proof system NIZK obtained from compiling Σ via the simplified randomized Fischlin transform satisfies the following properties.*

Correctness. NIZK *has correctness error $r \cdot e^{-2^{t-b}}$ and the prover runs in time $\mathsf{poly}(2^t)$.*

(Relaxed) Knowledge Soundness. *There exists a straight-line extractor Ext can observe the adversary's random oracle queries such that for any $\mathcal{A}$ making at most $Q = Q(\lambda)$ queries to H,*

$$\mathsf{Adv}^{\mathrm{ksnd}}_{\mathsf{NIZK},\mathsf{Ext}}(\mathcal{A}, \lambda) \leqslant Q \cdot 2^{-r \cdot b} .$$

[8] In contrast to special soundness, this does not require $c \neq c'$.

Algorithm $\Sigma_{\mathsf{com,BBS}}.\mathsf{Init}((\boldsymbol{H}, C', \phi_{\boldsymbol{I},\boldsymbol{a}}), (s, \boldsymbol{m}))$:	Algorithm $\Sigma_{\mathsf{com,BBS}}.\mathsf{Resp}(\mathsf{st}, c)$:
$C = C' - \sum_{i \in \boldsymbol{I}} a_i H_i$ $\boldsymbol{H}_{\text{priv}} \leftarrow (H_i)_{i \in [\ell+1] \setminus \boldsymbol{I}}$ $(\boldsymbol{R}, \mathsf{st}) \leftarrow\$ \Sigma_{\mathsf{Lin}}.\mathsf{Init}((\boldsymbol{H}_{\text{priv}}, C), ((m_i)_{i \in [\ell] \setminus \boldsymbol{I}}, s))$ **return** $(\boldsymbol{R}, \mathsf{st})$	**return** $\Sigma_{\mathsf{Lin}}.\mathsf{Resp}(\mathsf{st}, c)$ Algorithm $\Sigma_{\mathsf{com,BBS}}.\mathsf{Verify}((\boldsymbol{H}, C', \phi_{\boldsymbol{I},\boldsymbol{a}}), (\boldsymbol{R}, c, \boldsymbol{z}))$: $C = C' - \sum_{i \in \boldsymbol{I}} a_i H_i$ $\boldsymbol{H}_{\text{priv}} \leftarrow (H_i)_{i \in [\ell+1] \setminus \boldsymbol{I}}$ **return** $\Sigma_{\mathsf{Lin}}.\mathsf{Verify}((\boldsymbol{H}_{\text{priv}}, C), (\boldsymbol{R}, c, \boldsymbol{z}))$

Fig. 21. Σ-Protocol for $\mathsf{R}_{\mathsf{com}}$ of BBS-based scheme.

Zero-Knowledge. *There exists a simulator* Sim *which can program the random oracle* H *such that for any adversary* $\mathcal{A}$ *making at most* $Q = Q(\lambda)$ *queries to* H

$$\mathsf{Adv}^{\mathsf{zk}}_{\mathsf{NIZK,Sim}}(\mathcal{A}, \lambda) \leqslant Q \cdot 2^{-\mathsf{H}_{\mathsf{min}}(\Sigma)} + 3r \cdot 2^{-(k-b)/2} .$$

Now, to obtain a straight-line extractable proof for our KVAC constructions, we show that the Σ-protocols for the linear relations $\mathsf{R}_{\mathsf{com}}$ induced by the constructions of $\mathsf{KVAC}_{\mathsf{BBS}}$ and $\mathsf{KVAC}_{\mathsf{DDH}}$ satisfies High Min-Entropy and Relaxed Strong Special Soundness for the relaxed relations described in Sections 5 and 6, respectively. Note again that for our instantiations we only consider selective disclosure predicates. For simplicity, let Σ_{Lin} be a Σ-protocol for general linear relation, described in Section 2, which we can easily that the min-entropy is $\mathsf{H}_{\mathsf{min}}(\Sigma) = \log p$.

<u>Relation and proof system for BBS-based scheme.</u> Recall from Section 5 the description of $\mathsf{R}_{\mathsf{com}}$ and $\widetilde{\mathsf{R}}_{\mathsf{com}}$ (omitting par in the subscript).

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

$$\widetilde{\mathsf{R}}_{\mathsf{com}} := \left\{ ((\boldsymbol{H}, C', \psi), (s, \boldsymbol{m})) : \begin{array}{c} (0_{\mathbb{G}} = \sum_{i=1}^{\ell} m_i H_i + sH_{\ell+1} \wedge \\ (s \| \boldsymbol{m}) \neq \boldsymbol{0}) \vee \\ ((\boldsymbol{H}, C', \psi), (s, \boldsymbol{m})) \in \mathsf{R}_{\mathsf{com}} \end{array} \right\} .$$

For a selective disclosure predicate $\psi_{I,\boldsymbol{a}}$ for $I \subseteq [\ell]$, the linear relation being proved by Σ_{Lin} becomes $s + H_{\ell+1} + \sum_{i \notin \boldsymbol{I}, i \leqslant \ell} m_i H_i = C' - \sum_{i \in \boldsymbol{I}} a_i H_i$.

Now, fix a statement $(\boldsymbol{H}, C', \phi_{\boldsymbol{I},\boldsymbol{a}})$, and consider any two different valid transcripts $(R, c, \boldsymbol{z}), (R, c', \boldsymbol{z}')$. If $c \neq c'$, we simply rely on the special soundness of Σ_{Lin} and extract $(s, \boldsymbol{m}_{[\ell] \setminus \boldsymbol{I}})$ such that $C' = sH_{\ell+1} + \sum_{i \in \boldsymbol{I}} a_i H_i + \sum_{i \notin \boldsymbol{I}} m_i H_i$, which is a witness for $\mathsf{R}_{\mathsf{com}}$.

Otherwise, $c = c'$. Then, by the validity of $(R, c, \boldsymbol{z}), (R, c', \boldsymbol{z}')$,

$$R + c \left(C' - \sum_{i \in \boldsymbol{I}} a_i H_i \right) = \sum_{i \notin \boldsymbol{I}} z_i H_i = \sum_{i \notin I} z_i H_i .$$

Therefore, with $\boldsymbol{z} \neq \boldsymbol{z}'$, we have a $\widetilde{\mathsf{R}}_{\mathsf{com}}$-witness $\boldsymbol{m}' = \boldsymbol{z} - \boldsymbol{z}' \neq \boldsymbol{0}$ and $\sum_{i \notin \boldsymbol{I}} m_i' H_i = 0_{\mathbb{G}}$.

<u>Relation and proof system for DDH-based scheme.</u> Recall from Section 5 the description of $\mathsf{R}_{\mathsf{com}}$ and $\widetilde{\mathsf{R}}_{\mathsf{com}}$ (omitting par in the subscript).

$$\mathsf{R}_{\mathsf{com}} := \left\{ \begin{array}{c} ((E_x, E_y, D, (X_i)_{i=1}^{\ell}, (Y_i)_{i=1}^{\ell}, \psi), \\ (u_x, u_y, \boldsymbol{m} = (m_i)_{i=1}^{\ell})) \end{array} : \begin{array}{c} E_x = (u_x G, u_x D + \sum_{i=1}^{\ell} m_i X_i) \\ E_y = (u_y G, u_y D + \sum_{i=1}^{\ell} m_i Y_i) \\ \psi(\boldsymbol{m}) = 1 \end{array} \right\}$$

$$\widetilde{\mathsf{R}}_{\mathsf{com}} := \left\{ \begin{array}{c} ((E_x, E_y, D, (X_i)_{i=1}^{\ell}, (Y_i)_{i=1}^{\ell}, \psi), \\ (u_x, u_y, \boldsymbol{m} = (m_i)_{i=1}^{\ell})) \end{array} : \begin{array}{c} (\sum_{i=1}^{\ell} m_i X_i = \sum_{i=1}^{\ell} m_i Y_i = 0_{\mathbb{G}} \wedge \\ \boldsymbol{m} \neq \boldsymbol{0}) \vee \\ (E_x = (u_x G, u_x D + \sum_{i=1}^{\ell} m_i X_i) \wedge \\ E_y = (u_y G, u_y D + \sum_{i=1}^{\ell} m_i Y_i) \wedge \\ \psi(\boldsymbol{m}) = 1) \end{array} \right\} .$$

Algorithm $\Sigma_{\mathsf{com},\mathsf{DDH}}.\mathsf{Init}(x, w)$:	Algorithm $\Sigma_{\mathsf{com},\mathsf{DDH}}.\mathsf{Resp}(\mathsf{st}, c)$:
parse $(E_x, E_y, D, (X_i)_{i=1}^{\ell}, (Y_i)_{i=1}^{\ell}, \phi_{\boldsymbol{I},\boldsymbol{a}}) \leftarrow x$ **parse** $(u_x, u_y, (m_i)_{i=1}^{\ell}) \leftarrow w$ $E'_x = E_x - (0, \sum_{i \in \boldsymbol{I}} a_i X_i)$ $E'_y = E_y - (0, \sum_{i \in \boldsymbol{I}} a_i Y_i)$ $(\boldsymbol{R}, \mathsf{st}) \leftarrow\$ \Sigma_{\mathsf{Lin}}.\mathsf{Init}((M_{\boldsymbol{I},D,\boldsymbol{X},\boldsymbol{Y}}, (E'_x \\| E'_y)),$ $((m_i)_{i \in [\ell] \setminus \boldsymbol{I}}))$ **return** $(\boldsymbol{R}, \mathsf{st})$	**return** $\Sigma_{\mathsf{Lin}}.\mathsf{Resp}(\mathsf{st}, c)$ Algorithm $\Sigma_{\mathsf{com},\mathsf{DDH}}.\mathsf{Verify}(x, (\boldsymbol{R}, c, \boldsymbol{z}))$: **parse** $(E_x, E_y, D, (X_i)_{i=1}^{\ell}, (Y_i)_{i=1}^{\ell}, \phi_{\boldsymbol{I},\boldsymbol{a}}) \leftarrow x$ $E'_x = E_x - (0, \sum_{i \in \boldsymbol{I}} a_i X_i)$ $E'_y = E_y - (0, \sum_{i \in \boldsymbol{I}} a_i Y_i)$ **return** $\Sigma_{\mathsf{Lin}}.\mathsf{Verify}((M_{\boldsymbol{I},D,\boldsymbol{X},\boldsymbol{Y}}, (E'_x \\| E'_y)),$ $(\boldsymbol{R}, c, \boldsymbol{z}))$

Fig. 22. Σ-Protocol for $\mathsf{R}_{\mathsf{com}}$ of DDH-based scheme.

For a selective disclosure predicate $\psi_{\boldsymbol{I},\boldsymbol{a}}$ for $\boldsymbol{I} \subseteq [\ell]$, the linear relation being proved by Σ_{Lin} becomes $(u_x G, u_x D + \sum_{i \notin \boldsymbol{I}} m_i X_i) = E_x - (0, \sum_{i \in \boldsymbol{I}} a_i X_i)$ and $(u_y G, u_y D + \sum_{i \notin \boldsymbol{I}} m_i Y_i) = E_y - (0, \sum_{i \in \boldsymbol{I}} a_i Y_i)$. In particular, this corresponds to the following linear map for $\boldsymbol{I} = (i_1, \ldots, i_k)$

$$M_{\boldsymbol{I},D,\boldsymbol{X},\boldsymbol{Y}} = \begin{bmatrix} G & 0 & 0 & \ldots & 0 \\ D & 0 & X_{i_1} & \ldots & X_{i_k} \\ 0 & G & 0 & \ldots & 0 \\ 0 & D & Y_{i_1} & \ldots & Y_{i_k} \end{bmatrix}$$

Again, fix a statement $(E_x, E_y, D, (X_i)_{i=1}^{\ell}, (Y_i)_{i=1}^{\ell}, \psi_{I,\boldsymbol{a}})$ and two transcripts $(\boldsymbol{R}, c, \boldsymbol{z}) \neq (\boldsymbol{R}, c', \boldsymbol{z}')$. We additionally denote $\boldsymbol{z} = (z_x, z_y, (z_i)_{i \notin \boldsymbol{I}})$ and $\boldsymbol{z}' = (z'_x, z'_y, (z'_i)_{i \notin \boldsymbol{I}})$. Now, consider the two cases: $c \neq c'$ and $c = c'$. For the former, special soundness allows us to extract the witness corresponding to $\mathsf{R}_{\mathsf{com}}$, so we are done. For the latter, we have that $\boldsymbol{z} \neq \boldsymbol{z}'$ and by the validity of $(\boldsymbol{R}, c, \boldsymbol{z}), (\boldsymbol{R}, c', \boldsymbol{z}')$,

$$R + c \begin{bmatrix} E_{x,0} \\ E_{x,1} - \sum_{i \in \boldsymbol{I}} a_i X_i \\ E_{y,0} \\ E_{y,1} - \sum_{i \in \boldsymbol{I}} a_i Y_i \end{bmatrix} = \begin{bmatrix} z_x G \\ z_x D - \sum_{i \notin \boldsymbol{I}} z_i X_i \\ z_y G \\ z_y D - \sum_{i \notin \boldsymbol{I}} z_i Y_i \end{bmatrix} = \begin{bmatrix} z'_x G \\ z'_x D - \sum_{i \notin \boldsymbol{I}} z'_i X_i \\ z'_y G \\ z'_y D - \sum_{i \notin \boldsymbol{I}} z'_i Y_i \end{bmatrix} .$$

Subtracting the equations on $\boldsymbol{z}$ and $\boldsymbol{z}'$, we have that $z_x = z'_x, z_y = z'_y$ and $\sum_{i \notin \boldsymbol{I}} (z_i - z'_i) X_i = \sum_{i \notin \boldsymbol{I}} (z_i - z'_i) Y_i = 0_{\mathbb{G}}$, which gives us a witness for $\widetilde{\mathsf{R}}_{\mathsf{com}}$.