Aggregate Signatures with Versatile Randomization and Issuer-Hiding Multi-Authority Anonymous Credentials

Omid Mir[1⋆], Balthazar Bauer[2], Scott Griffy[3], Anna Lysyanskaya[3], and Daniel Slamanig[4]

[1] Johannes Kepler University Linz
LIT Secure and Correct Systems Lab, Linz, Austria
mir@ins.jku.at
[2] IRIF, CNRS, Paris, France
Balthazar.Bauer@ens.fr
[3] Brown University, Providence, USA
{firstname_lastname}@brown.edu
[4] AIT Austrian Institute of Technology, Vienna, Austria
daniel.slamanig@ait.ac.at

Abstract. Anonymous credentials (AC) have emerged as a promising privacy-preserving solution for user-centric identity management. They allow users to authenticate in an anonymous and unlinkable way such that only required information (i.e., attributes) from their credentials are revealed. With the increasing push towards decentralized systems and identity, e.g., self-sovereign identity (SSI) and the concept of verifiable credentials, this also necessitates the need for suitable AC systems. For instance, when relying on existing AC systems, obtaining credentials from different issuers requires the presentation of independent credentials, which can become cumbersome. Consequently, it is desirable for AC systems to support the so-called multi-authority (MA) feature. It allows a compact and efficient showing of multiple credentials from different issuers. Another important property is called issuer hiding (IH). This means that showing a set of credentials is not revealed which issuer has issued which credentials but only whether a verifier-defined policy on the acceptable set of issuers is satisfied. This issue becomes particularly acute in the context of MA, where a user could be uniquely identified by the combination of issuers in their showing. Unfortunately, there are no AC schemes that satisfy both these properties simultaneously.

To close this gap, we introduce the concept of Issuer-Hiding Multi-Authority Anonymous Credentials (IhMA). Our proposed solution involves the development of two new signature primitives with versatile randomization features which are independent of interest: 1) Aggregate Signatures with Randomizable Tags and Public Keys (AtoSa) and 2) Aggregate Mercurial Signatures (ATMS), which extend the functionality of AtoSa to additionally support the randomization of messages and yield the first instance of an aggregate (equivalence-class) structure-preserving signature. These primitives can be elegantly used to obtain IhMA with different trade-offs but have applications beyond.

We formalize all notations and provide rigorous security definitions for our proposed primitives. We present provably secure and efficient instantiations of the two primitives as well as corresponding IhMA systems. Finally, we provide benchmarks based on an implementation to demonstrate the practical efficiency of our constructions.

1 Introduction

Authentication and authorization are essential and security-critical tasks in a digital world. They are aimed to ensure that the communication partner is the one it claims to be and to enforce access control to digital resources such as services. A central concept is that of a digital identity, which can be seen as a collection of attributes (e.g., name, age, nationality, gender, etc.) representing a (real-world) entity in the digital realm.

On the Internet, a widely adopted practice is to have centralized identity providers (IdP), e.g., Google or Meta, to maintain the digital identity of users. Other services can then simply rely on the identity provided by the IdP. From a privacy perspective, however, this is problematic as users lose control over their digital identity (all their attributes reside at the IdP), and the IdP learns all the services a user consumes on the Internet (and data related to the use).

⋆ First and corresponding author; remaining authors in alphabetical order.

Already in the 1980s, Chaum [26, 27] envisioned cryptographic techniques for creating more privacy-friendly and user-centric solutions to authentication and authorization. They put users in control of their identity and allow users to selectively reveal information (i.e., attributes) about their digital identities in an unlinkable and thus untraceable way. Such techniques are commonly known as anonymous credentials (ACs), and there is a vast body of research into different approaches to construct such AC systems [14, 20, 21, 22, 3, 5, 19, 57, 34, 39, 59, 44, 29].

While early AC systems such as U-Prove [56] and Idemix [24] did not see a widespread adoption, nowadays related techniques such as direct anonymous attestation (DAA) [15, 17] and Enhanced Privacy ID (EPID) [16] are deployed in billions of devices. Most recently, ACs have seen adoption within the popular Signal messenger to realize private groups [25]. They also see increasing popularity in the form of anonymous tokens (with private or public metadata bit) [33, 49, 62]. Among the applications are private browsing with DDoS protection being standardized by the IETF[5] (Privacy Pass [33] and Private Access Tokens [52]) or the PrivateStats proposal by Facebook[6] to privately collect client-side telemetry from WhatsApp.

Decentralized identity. Like with centralized IdPs, all AC solutions mentioned so far are in a centralized setting, i.e., a single party called the issuer is issuing credentials to users. Today we however see a trend to move away from this centralized setting towards a decentralized identity. A popular concept in the decentralized identity space is that of self-sovereign identity (SSI) with Sovrin[7] being a prominent example. In SSI users are collecting certified attributes (called verifiable credentials) from *different sources* and then presenting (subsets of) verifiable credentials from this collection. There is an increasing push towards standardization of this verifiable credentials concept within W3C[8] and large efforts such as the future European data infrastructure (Gaia-X)[9] or the European Blockchain Services Infrastructure (EBSI)[10] are adopting this approach.

Within the verifiable credential initiative in W3C, it is also observed that privacy related features are important. In particular well-known features from AC systems such as supporting selective disclosure and proving predicates about attributes[11]. To realize this functionality within W3C it is intended to base this upon the BBS+ signature scheme[12], a well-known building block for ACs currently being standardized as the BBS variant [64] within the IETF[13].

Privacy in a decentralized setting. The aforementioned approach allows to preserve privacy in a setting where a user wants to show a single verifiable credential issued by a single party. However, for a decentralized setting, where typically a subset of a collection of verifiable credentials from different issuers needs to be shown, the problem of how to efficiently realize this arises. A naive way is to conduct a parallel credential showing with all the required verifiable credentials. However, apart from reduced efficiency, this also has privacy implications. In particular, every verifiable credential reveals the exact issuer providing a lot of contextual partial information, e.g., a passport issued from a certain country or a driving license issued by a certain state reveals geographic information. This can be highly privacy intrusive in many settings and undermining the very objective of SSI systems [12]. Consequently, it would be desirable to be able to show a credential in a way that it is only revealed that it comes from one of a larger set of issuers acceptable by a verifier. A set of recent independent works introduced a property providing this features for AC systems, which is called issuer-hiding [8, 29, 12]. While this is a step towards countering the above privacy issues, these works only consider single issuers and are thus not yet suitable for a decentralized setting with multiple issuers.

ACs in a decentralized setting. Before discussing the different approaches to tackle a decentralized setting within ACs, it is important to recall that, generically, AC systems can be built from signature schemes and non-interactive zero-knowledge (NIZK) proofs. Loosely speaking, an issuer produces a signature on a list of attributes and a public-key of the user. The showing of a credential is repre-

[5] https://datatracker.ietf.org/wg/privacypass/about/
[6] https://research.fb.com/privatestats
[7] https://sovrin.org/
[8] https://www.w3.org/TR/vc-data-model/
[9] https://gaia-x.eu/
[10] https://ec.europa.eu/digital-building-blocks/wikis/display/EBSI/Home
[11] https://www.w3.org/TR/vc-data-model/#privacy-considerations
[12] https://w3c-ccg.github.io/ldp-bbs2020/
[13] https://datatracker.ietf.org/doc/draft-irtf-cfrg-bbs-signatures/

sented by a NIZK proof that the user possesses a valid signature from the issuer on a public-key for which it knows the corresponding secret key and that a certain predicate over the signed attributes is valid, e.g., for the attribute age it holds that it is above 18.

Looking at existing AC solutions, Garman et al. [40] first introduced a solution with no credential issuers and thus no signatures. Users make claims about their identity attributes in the form of commitments, which are submitted to a public transaction ledger, i.e., a blockchain. These registered commitments can then be used as a basis to compute NIZK proofs, representing showings. While interesting, this cannot be considered a general solution as for many types of common credentials, e.g., passports, driving licenses, and academic degrees; there is the need for explicit issuers.

Secondly, there is the concept of threshold issuance anonymous credentials, e.g., Coconut by Sonnino et al. [63] or ones based on threshold BBS+ by Doerner et al. [35]. Such a system thresholdizes a single issuer among a set of parties. While this helps to make AC systems more robust, it does not efficiently support multiple issuers and also does not support issuer-hiding.

Thirdly, Rosenberg et al. [58] present a framework to build ACs from existing identity documents, e.g., passports, and driving licenses, whose contents (attributes) are registered in lists of "issuers". In particular, they are represented as commitments organized in Merkle trees, and users obtain the authentication paths to their credentials. Then, users can use succinct NIZK proofs (zk-SNARKs) to prove statements about the attributes encoded in potentially multiple credentials. Here the zk-SNARK proofs for the single credentials are linked via NIZK proofs. This approach is very generic and avoids the lack of issuers in [40]. However, it is very complex, and their credentials showing due to computing Merkle membership proofs and linking of zk-SNARKS via NIZK can add significant costs (showing times in the order of seconds). Moreover, they require a dedicated infrastructure such as a transparency log, a Byzantine system, or a blockchain.

Finally, and most related, we want to discuss the work by Hébant and Pointcheval [45]. The authors introduced the concept of (traceable) Multi-Authority Anonymous Credentials (MA-ACs). Loosely speaking, their approach to realize MA-ACs is based on so called aggregate signatures with randomizable tags and allows to aggregate showings of credentials of different issuers (but with respect to the same tag) into one compact showing. Due to randomizability of signatures and tags, it is possible to produce unlinkable showings. Moreover, the tag component has a secret part representing the user secret. While this is an interesting concept, it does not provide an efficient way of providing the issuer-hiding (IH) feature [8, 29, 12]. There is an obvious generic way to use a succinct NIZK (i.e., a zk-SNARK) and prove that the aggregated signature verifies for the given attributes under a subset of issuer keys without revealing which ones. While this can lead to an asymptotically compact solution, the prover will concretely be very expensive due to the size of the verification keys (they are of size $\mathbb{G}_2^{3+2n}$ each with n being the maximum number (types) of attributes) and the complexity of the verification equation in [45] which is proven with a zk-SNARK. Switching to non-succinct Schnorr-type NIZK obtained via Fiat-Shamir as done in [8] (in Construction 2), however, will result in a non-compact showing of size $O(n \cdot K)$ with K being the number of issuers used in the aggregated showing (even when ignoring the size of the proof corresponding to the non-shown attributes).

In this paper, our goal is to efficiently combine these features and propose the first AC system that is specifically designed to provide multi-authority and issuer-hiding features at the same time.

Aggregate signatures. Aggregate signatures, introduced by Boneh et al. in [11], allow to combine multiple signatures σ_i for messages m_i and associated public keys vk_i into a single signature σ, that authenticates the entire set of messages w.r.t the set of public keys. Ideally, the aggregated signatures is of length identical to a single signatures and thus allows to compress a set of signatures into a single one.

This primitive is valuable in optimizing storage and bandwidth as well as minimizing cryptographic overhead in scenarios such as compressing certificate chains or aggregating signatures in blockchains. Many different variants have been proposed [55, 6, 10, 42] and we will briefly mention some relevant schemes. Sequential aggregation, studied in [51], requires signers to interact sequentially. Synchronized aggregation, examined in [2], assumes synchronization among signers such that in every time period t each signer only contributes one signature at most. Indexed or tag-based aggregated signatures, introduced in [45], allow aggregation of signatures for different messages under different public keys if they share the same tag or index. These signatures are useful for constructing an AC system.

Unfortunately, existing aggregate signature schemes do not explicitly possess properties to make them amenable for the design of efficient decentralized AC systems with advanced properties. We will close this gap by introducing aggregate (structure-preserving) signatures with the ability to randomize signatures, tags, (messages,) and verification keys.

1.1 Our Contribution

Our contribution in this paper is twofold:

Aggregate signatures with randomization features. The key technique to achieve our goal is to introduce tag-based aggregate signatures with randomizable tags and public keys. We further extend them to additionally support randomization of messages resembling the functionality of equivalence class signatures (SPSEQ) [39]. For both of these types of schemes we provide rigorous formal security models as well as instantiations that are provably secure in this model. More precisely, we introduce:

Aggregate signatures with randomizable keys and tags (AtoSa[14] for short) where signatures are associated to tags (consisting of a private and a public part) and signatures with respect to the *same* tag can be aggregated. Aside from signatures, erification keys and tags can be randomized. Tags and verification keys are defined with respect to equivalence classes and randomization switches between representatives of these classes.[15] Then existing signatures can be adapted to ones that verify under the randomized public keys and tags. We provide an AtoSa scheme based on the well-known Pointcheval-Sanders (PS) signatures [57]. PS signatures have already served as a basis for various privacy-preserving primitives such as group signatures and anonymous credentials [57], redactable [59, 60] or dynamically malleable signatures [7]. They are very efficient and have interesting features such as support for blind signing, i.e., signing of committed (hidden) messages, and efficient ways of proving their knowledge.

Aggregate Mercurial Signatures with Randomizable Tags (ATMS) extend the functionality of AtoSa to support the randomization of messages, i.e., equivalence classes of messages similar to (SPSEQ). This means that in addition to AtoSa existing signatures can be adapted to verify under randomized messages (i.e., other representatives of the message class). Consequently, we obtain a version of mercurial signatures [32] that is both aggregatable and has randomizable tags. To the best of our knowledge, this is the first instance of an aggregate structure-preserving signature (and, subsequently, SPSEQ). We provide an ATMS construction inspired by the message-indexed SPS in [31], which on itself is a variant of Ghadafi's SPS [41] scheme.

Restrictions of our Constructions. We should mention that in contrast to standard aggregate signatures, our constructions 1) either require that all aggregated messages and corresponding verification keys are known before requesting the first signature or 2) to make the same assumption as within synchronized aggregate signatures [2, 46]. In particular, adapted to our setting, latter means that every issuer ensures that for each tag only a single signature is issued. We will present our results based on the first approach and discuss adaptions for the second (which do not change any of the interfaces or security definitions and proofs). Since our main application is anonymous credentials, depending on the concrete application scenario either the first or the second approach can be chosen. It remains an interesting open question to get fully dynamic signatures without any of the above assumptions.

Like other types of signatures with randomization features, we also expect that our schemes will find applications beyond the one presented here.

Issuer-Hiding Multi-Authority Anonymous Credentials. We present a rigorous formal model for issuer-hiding multi-authority anonymous credentials (IhMA). Then we present two constructions based on AtoSa (called $\mathsf{IhMA}_{\mathsf{AtoSa}}$) and ATMS (called $\mathsf{IhMA}_{\mathsf{ATMS}}$) respectively, where both are concretely very efficient but offer some trade-offs (as discussed below). Thus this represents an important contribution to the field of ACs in that it provides a solution that addresses the challenges of

[14] The (ancient) Greek transliteration of the old Persian name Utau` a. Atossa means "bestowing very richly" or "well trickling" or "well granting". It refers to an Achaemenid empress who was the daughter of Cyrus the Great, and the wife of Darius the Great.

[15] This can be seen as aggregate signatures with randomizable tags as introduced in [45] with the additional features of randomizable keys with appropriate signature adaption.

user privacy and scalability in multi-authority (decentralizing) settings. In our constructions, obtaining a credential amounts to obtaining signatures on desired attributes from a set of issuers on different attributes, but under the same tag (which can be thought of as the user's identity in credential schemes). Showing simply amounts to randomizing signatures from issuers that should be shown as well as the tags and aggregating them. Finally, one provides the aggregated signature and either opens (subsets of) attributes or proves predicates over them along with proof of knowledge of the secret tag part.

Supporting the issuer-hiding feature [29, 9] works roughly as follows: Each verifier generates a so-called *key-policy*, which defines a set of issuers (via their verification keys) that the verifier would accept an (aggregated) credential from. This policy is a collection of SPSEQ signatures on verification keys of the AtoSa or ATMS scheme. Since the equivalence classes of the SPSEQ (the message space) match with the key equivalence class of AtoSa and ATMS, showing a credential then works as above, but all verification keys of the AtoSa or ATMS are randomized, and the respective SPSEQ signatures in the key-policy are adapted accordingly.

For the $\mathsf{IhMA}_{\mathsf{ATMS}}$ scheme, instead of directly signing attributes, we use the framework of Fuchsbauer et al. [39]. Here the signature scheme is used to sign set commitments to attribute sets. Moreover, in order to prove the anonymity of this construction as an additional contribution we introduce a generalization of the decisional uber assumption family by Boyen [13] along with an interactive version. Using this approach is however not straightforward as we have to make set commitments compatible with the message space of our ATMS. While $\mathsf{IhMA}_{\mathsf{AtoSa}}$ and $\mathsf{IhMA}_{\mathsf{ATMS}}$ share a common aim, the differences in the constructions entail certain trade-offs in terms of functionality and efficiency:

- Credential size: The $\mathsf{IhMA}_{\mathsf{ATMS}}$ scheme can yield a fixed-sized credential, while the $\mathsf{IhMA}_{\mathsf{AtoSa}}$ scheme does not achieve this without utilizing Zero Knowledge Proof of Knowledge (ZKPOK) of signatures.
- Efficiency: The $\mathsf{IhMA}_{\mathsf{ATMS}}$ scheme is more efficient at showing and verifying credentials compared to the $\mathsf{IhMA}_{\mathsf{AtoSa}}$ scheme.
- Need for a trusted party: The $\mathsf{IhMA}_{\mathsf{ATMS}}$ scheme requires a trusted party, while the $\mathsf{IhMA}_{\mathsf{AtoSa}}$ scheme does not. This is because $\mathsf{IhMA}_{\mathsf{ATMS}}$ relies on a trusted party to hold a trapdoor to generate set commitments, whereas $\mathsf{IhMA}_{\mathsf{AtoSa}}$ does not require such a trusted party.
- Expressiveness: The $\mathsf{IhMA}_{\mathsf{ATMS}}$ supports revealing a subset of attributes from a set of attributes per issuer, i.e., selective disclosure per issuer. The $\mathsf{IhMA}_{\mathsf{AtoSa}}$ scheme only supports a single attribute for each credential. Consequently, it only supports selective disclosure over all issuers. However, both schemes allow for proving arbitrary predicates over signed messages.

Overall, the choice of the concrete construction depends on the specifics of the use case or application and priorities set in the overall system.

1.2 Comparison of IhMA with Previous Work

We have already discussed that there is only one dedicated MA-AC scheme [45]. This is however not issuer-hiding (IH) and as mentioned, adding IH comes with a significant overhead. In Table 1, we compare our IhMA approaches to other schemes in the literature that provide the IH feature [8, 12, 29] and for comparison we use the naive approach to achieve MA, i.e., parallel showings of single credentials, which we indicate by $\approx$. We compare them in terms of the size of credential $|\mathbf{Cred}|$, communication cost of showing $|\mathbf{Show}|$, and computational cost of showing **Show** for user **(P)** and verifier **(V)**. We provide concrete analysis for our schemes' communication cost in Appendix A.

To ensure a fair comparison between the schemes, we consider a typical case where k out of n attributes come from K out of N issuers where n is the total number of attributes given to the user by N issuers, and k is the number of attributes involved in the showing (and K the number of issuers indicated in the showing).

With respect to credential size $|\mathbf{Cred}|$, the naive approach to **MA** leads to $O(K)$ complexity. Our $\mathsf{IhMA}_{\mathsf{ATMS}}$ scheme maintains a constant credential size even when there are $K > 1$ issuers, while our $\mathsf{IhMA}_{\mathsf{AtoSa}}$ scheme has $O(K)$ credentials. However, we can aggregate credentials and then during showing apply a ZKPOK of a PS signature, which allows us to reduce the credential size to a constant size. In contrast, others have a credential size linear in the number of issuers K.

Table 1: Comparison of AC schemes in **MA** setting (n: Attributes; k: Disclosed attributes, u: Undisclosed attributes, N: Total issuers in policy, K: issuers in showing)

	[29] ‡	[12]**	[8]**	IhMA$_{\mathsf{AtoSa}}$	IhMA$_{\mathsf{ATMS}}$
IH	✓	✓	✓	✓	✓
MA	≈	≈	≈	✓	✓
|**Cred**|	$O(N)$	$O(N)$	$O(N)$	$O(N)^\star$	$O(N)^\star$
|**Show**|	$O(K \cdot N)$	$O(k \cdot K)$	$O(k \cdot 2K)$	$O(K)$	$O(K)$
Show (P)	$O(KuN)$	$O(k \cdot K)$	$O(k \cdot 2K)$	$O(K)^\dagger$	$O(u \cdot K)$
Show (V)	$O(KkN)$	$O(k \cdot K)$	$O(k \cdot 2K)$	$O(k)$	$O(k \cdot K)$

* We present the scheme in a way that supports ad-hoc attribute/issuer aggregation, but for fixed signatures, a constant size credential is achievable. For ATMS we will show how to achieve this in Section 5.3.
** K refers to proving knowledge of K credentials and K signatures of key policy in Showing.
† Since the ad-hoc aggregation cost is negligible, it is skipped here. Also, without considering IH, it becomes $O(1)$.
‡ This scheme uses standard assumptions in the ROM while other schemes use the GGM.

In terms of communication cost in showing (|**Show**|), our schemes require sending the randomized vks of the K issuers, along with two signatures (one for the credential and one for the key policy), overall giving $O(K)$. In [8], the communication size is based on sending K blinded credentials and K blinded signatures in the key policy and provide a ZKPOK of having correctly done so. The scheme in [12] is similar to [8], but the size of the policy is fixed. Finally, in the scheme described in [29], one needs to prove knowledge of K out of N verification keys (a linear sized OR statement) and sends them along with K credentials. Note that the size of ZKPOK includes many group elements and significantly more than only transferring K verification keys, as it is the case for our constructions.

When it comes to the computational cost of showing, i.e., **Show (P)** and **Show (V)**, our IhMA$_{\mathsf{AtoSa}}$ scheme has a minimal computational cost for provers as they only need to perform a small/constant number of operations for aggregation, along with K exponentiations for randomizing the verification keys vk. Our IhMA$_{\mathsf{ATMS}}$ scheme involves additional computation in the creation of a witness for set commitments corresponding to undisclosed attributes (a multi-exponentiation of $O(u)$). In [8], this cost includes proving knowledge of k signatures (in the key policy), K credentials, and k disclosed attributes. Similarly, [12] requires the computation of generating witness for their aggregator (accumulator) on K credentials, proving knowledge of k credential, but it does not need to prove knowledge of signatures in the policy. Moreover, in [29], proving knowledge of K-out-of-N verification keys is necessary, along with the computation of generating witness on undisclosed attributes for set commitments on K credentials. Again, the cost of ZKPOK for credentials or committed attributes is significantly more expensive than in our case, which is needed only to prove a secret key and some multi-exponentiation for creating witness. We should mention here that by leveraging ZKPOK, arbitrary relationships can be proved on attributes.

In summary, while the efficiency of different schemes may appear to be close asymptotically, our IhMA approaches are significantly more efficient than existing approaches while providing both properties simultaneously. Indeed, we only need group operations on $\mathbb{G}_i$ at the cost of $O(k)$. In contrast, other schemes require proving knowledge of signatures or keys, which is significantly more expensive.

2 Preliminaries

Notation. We use $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P, \hat{P}) \leftarrow \mathsf{BGGen}(1^\lambda)$ to denote a bilinear group generator for asymmetric type 3 groups, where p is a prime of bitlength λ. When applying a scalar a componentwise to a vector $\mathbf{T} \in \mathbb{G}_1^n$ we write $\mathbf{T}^a = (T_1^a, T_2^a, \ldots, T_n^a)$. We write $[x]_{\mathcal{R}}$ to denotes denote representative x of the equivalence class for given relation $\mathcal{R}$. Given a finite set S, we denote by $x \leftarrow S$ or $x \xleftarrow{\$} S$ the sampling of an element uniformly at random from S. For an algorithm A, let $y \leftarrow \mathsf{A}(x)$ be the process of running A on input x with access to uniformly random coins and assigning the result to y. With $\mathcal{A}^{\mathsf{B}}$ we denote that $\mathcal{A}$ has oracle access to B. We use $\langle \mathcal{O} \rangle$ to denote oracles defined in games and use ϵ to indicate a negligible function. We assume all algorithms are polynomial-time (PPT) unless otherwise specified and public parameters are an implicit input to all algorithms in a scheme.

Diffie-Hellman Message Space. Over an asymmetric bilinear group, a pair $(M, N) \in \mathbb{G}_1 \times \mathbb{G}_2$ is called a Diffie-Hellman (DH) message $\mathcal{M}_{\mathsf{DH}}$ [1] if there exists $m \in \mathbb{Z}_p$ s.t. $M = P^m$ and $N = \hat{P}^m$. One can efficiently verify whether $(M, N) \in \mathcal{M}_{\mathsf{DH}}$ by checking $e(M, \hat{P}) = e(P, N)$. Thus, the message space is a vector of Diffie-Hellman pairs $(\mathbf{M}, \mathbf{N}) = (M_1, \ldots, M_n, N_1, \ldots, N_n)$ s.t. for all $i \in [n]$, $(M_i, N_i) = (P^{m_i}, \hat{P}^{m_i}) \in \mathcal{M}_{\mathsf{DH}}$ for $m_i \in \mathbb{Z}_p$. Crites et al. [31] adapt the DH message space $\mathcal{M}_{\mathsf{DH}}$ to a tuple $(id, M, N) \in I \times \mathbb{G}_1 \times \mathbb{G}_2$ called Indexed Diffie-Hellman message space $\mathcal{M}^H_{\mathsf{iDH}}$, which uses a random basis $h \in \mathbb{G}_1$ computed using a random oracle H instead of P, as follows:

Definition 1 (Indexed Diffie-Hellman Message Space $\mathcal{M}^H_{\mathsf{iDH}}$ [31]). *Given a bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, g, \hat{g}) \leftarrow \mathsf{BGGen}(1^\lambda)$, an index set $\mathcal{I}$, and a random oracle $H : \mathcal{I} \to \mathbb{G}_1$, $\mathcal{M}^H_{\mathsf{iDH}}$ is an indexed Diffie-Hellman (DH) message space if $\mathcal{M}^H_{\mathsf{iDH}} \subset \{(id, \tilde{M}) \mid id \in \mathcal{I}, m \in \mathbb{Z}_p, \tilde{M} = (H(id)^m, \hat{g}^m) \in \mathbb{G}_1 \times \mathbb{G}_2\}$ and the following index uniqueness property holds: for all $(id, \tilde{M}) \in \mathcal{M}^H_{\mathsf{iDH}}$, $(id', \tilde{M}') \in \mathcal{M}^H_{\mathsf{iDH}}$, $id = id' \Rightarrow \tilde{M} = \tilde{M}'$. One can define the equivalence class for each message $\tilde{M} = (M, N) \in \tilde{\mathcal{M}}^H_{\mathsf{iDH}}$, as $\mathsf{EQ}_{\mathsf{iDH}}(M, N) = \{(M^r, N) \mid \exists\, r \in \mathbb{Z}_p\}$.*

Note that one can efficiently decide subset membership by checking $e(M, \hat{P}) = e(h, N)$. The uniqueness property guarantees that no two messages use the same index, which needs to be ensured by signers.

Camenisch and Stadler Notation. We use the common notation due to Camenisch and Stadler [23] for ZKPOK (or NIZK) as follows:

$$\mathsf{ZKPOK}\left\{(\alpha, \beta) : y = P^\alpha \wedge z = P^\beta \cdot h^\alpha\right\},$$

which denotes an (non-) interactive proof of knowledge of discrete logarithms (α, β) (the witness) satisfying the right-hand side statement about the public values y, P, z, h.

2.1 Digital Signatures

Pointcheval-Sanders (PS) Signatures. The PS signature [57] works on an asymmetric bilinear groups. It is EUF-CMA-secure under the PS assumption (cf. Def. 31). For a single scalar message it is defined as follows:

- $\mathsf{Setup}(1^\lambda)$: Take the security parameter λ as input and return the public parameters $\mathsf{pp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P, \hat{P})$, where P and $\hat{P}$ are randomly picked generators.
- $\mathsf{KeyGen}(\mathsf{pp})$: Take pp as input, sample two randoms $(x, y) \xleftarrow{\$} \mathbb{Z}_p^*$, and return the verification key $\mathsf{vk} = (\hat{X} = \hat{P}^x, \hat{Y} = \hat{P}^y)$ and the secret key $\mathsf{sk} = (x, y)$.
- $\mathsf{Sign}(\mathsf{sk}, m)$: Take the secret key sk and a message $m \in \mathbb{Z}_p$ as input. Sample $r \xleftarrow{\$} \mathbb{Z}_p^*$ uniformly at random and then compute $\sigma = (h, s) = (P^r, h^{x+my})$ and return the signature σ as output.
- $\mathsf{Verify}(\mathsf{vk}, \sigma, m)$: To verify a signature σ, this algorithm takes the verification key vk and message m as input. If $h \neq 1$ and the pairing product equation $e(h, \hat{X} \cdot \hat{Y}^m) = e(s, \hat{P})$ holds, then it returns 1 (accept), otherwise 0 (reject).

Ghadafi SPS. Structure-preserving signatures (SPSs) [1] are signatures where public keys and the signatures are source group elements of a bilinear group, and verification will be done only using group-membership tests and pairing-product equations. We recall the SPS scheme by Ghadafi [41] which is a structure-preserving variant of PS signatures [57]. Ghadafi's SPS construction over a Diffie-Hellman message space $\mathcal{M}_{\mathsf{DH}}$ is defined as follows:

- $\mathsf{Setup}(1^\lambda)$: Generates a bilinear group $\mathsf{pp} = (p, \hat{P}, P, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e)$ and outputs parameters pp.
- $\mathsf{KeyGen}(\mathsf{pp})$: Takes pp as input, chooses two randoms $(x, y) \xleftarrow{\$} \mathbb{Z}_p^*$, and returns the verification key $\mathsf{vk} = (\hat{P}^x, \hat{P}^y)$ and the secret key $\mathsf{sk} = (x, y)$.
- $\mathsf{Sign}(\mathsf{sk}, (M, N))$: Takes the sk and DH message $(M, N) \in \mathcal{M}_{\mathsf{DH}}$ such that $e(M, \hat{P}) = e(P, N)$ as input. Samples $r \xleftarrow{\$} \mathbb{Z}_p^*$ and computes the signature as

$$\sigma = (R, S, T) = (R = P^r, S = M^r, T = R^x \cdot S^y)$$

- $\mathsf{Verify}(\mathsf{vk}, \sigma, (M, N))$: Takes pp, vk, a signature $\sigma = (R, S, T)$ and a message $(M, N) \in \mathcal{M}_{\mathsf{DH}}$. If the following equations hold, returns 1 and 0 otherwise:

$$e(R, N) = e(S, \hat{P}) \wedge e(T, \hat{P}) = e(R, \hat{X}) e(S, \hat{Y}) \wedge h \neq 1$$

Message-Indexed Ghadafi SPS. To construct our schemes, we use the version that is presented in [31] for signing elements of an Indexed Diffie-Hellman message space $\mathcal{M}^H_{\mathsf{iDH}}$ (cf. Def. 1):

$\mathsf{Setup}(1^\lambda)$: Let $\mathsf{pp} = (p, \hat{P}, P, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e)$. Output parameters pp.

$\mathsf{KeyGen}(\mathsf{pp})$: Choose two random $(x, y) \xleftarrow{\$} \mathbb{Z}_p^*$, returns the verification key as $\mathsf{vk} = (\hat{P}^x, \hat{P}^y)$ and the secret signing key $\mathsf{sk} = (x, y)$.

$\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, (id, M, N))$: On input sk and $(id, M, N) \in \mathcal{M}^H_{\mathsf{iDH}}$ such that $e(M, \hat{P}) = e(h, N)$, where $h = H(id) = P^r$. Computes the signature as $\sigma = (h, s) = (h, s = h^x \cdot M^y)$.

$\mathsf{Verify}(\mathsf{pp}, \mathsf{vk}, \sigma, (M, N))$: Check if the following equations hold, returns 1 (verify a signature σ), otherwise it returns 0 as: $e(h, N) = e(M, \hat{P}) \wedge e(s, \hat{P}) = e(h, \hat{X})e(M, \hat{Y}) \wedge h \neq 1$.

Equivalence Class Signatures. Structure-preserving signatures on equivalence classes SPSEQ [39, 43] allows to efficiently and jointly randomize messages and signatures in public, where the message space consists of group-element vectors. Indeed, messages with representatives of projective equivalence classes defined on the projective space underlying $\mathbb{G}^\ell$ (for $\ell > 1$ and some prime-order group p). Based on such classes, a signature allows the randomization of both messages and signatures via a change of representatives and a matching signature update. More precisely, it is used the following equivalence relation to partition $(\mathbb{G}^*)^\ell$ into classes:

$$\mathcal{R} = \left\{(\mathbf{M}, \mathbf{N}) \in (\mathbb{G}^*)^\ell \times (\mathbb{G}^*)^\ell \;\middle|\; \exists\ \in \mathbb{Z}_p^* : \mathbf{N} = s \cdot \mathbf{M}\right\} \subseteq (\mathbb{G}^*)^{2\ell}$$

We recall the SPS-EQ scheme from [39]:

$\mathsf{BG}_{\mathcal{R}}(1^\lambda)$: This algorithm on input of a security parameter λ outputs a bilinear group BG.

$\mathsf{KeyGen}(\mathsf{BG}, \ell)$: This algorithm on input of a bilinear group BG and a vector length $\ell > 1$ outputs a key pair $(\mathsf{sk}, \mathsf{vk})$ as $\mathsf{sk} \leftarrow (x_i)_{i\in[\ell]}$, and the public key $\mathsf{vk} \leftarrow (\hat{X}_i)_{i\in[\ell]} = (\hat{P}^{x_i})_{i\in[\ell]}$.

$\mathsf{Sign}(\mathsf{sk}, \mathbf{M})$: This algorithm on input a representative $\mathbf{M} \in (\mathbb{G}_i^*)^\ell$and a secret key sk outputs a signature σ for the equivalence class $[\mathbf{M}]_{\mathcal{R}}$ as $\sigma = (Z \leftarrow (\prod_{i\in[\ell]} M_i^{x_i})^y, Y \leftarrow P^{\frac{1}{y}}, \hat{Y} \leftarrow \hat{P}^{\frac{1}{y}})$.

$\mathsf{ChangRep}(\mathbf{M}, \sigma, \mu, \mathsf{vk})$: This algorithm on input of a representative $\mathbf{M} \in (\mathbb{G}_i^*)^\ell$ of class $[\mathbf{M}]_{\mathcal{R}}$, a signature σ for $\mathbf{M}$, a scalar μ and a public key vk returns an updated message-signature pair $(\mathbf{M}', \sigma')$, where $\mathbf{M}' = \mathbf{M}^\mu$ is the new representative and σ' its updated signature as follow pick $r \xleftarrow{\$} \mathbb{Z}_p^*$ and return $\sigma' \leftarrow (Z^{r\mu}, Y^{\frac{1}{r}}, \hat{Y}^{\frac{1}{r}})$.

$\mathsf{Verify}(\mathbf{M}, \sigma, \mathsf{vk})$: This algorithm on input of a representative $\mathbf{M} \in (\mathbb{G}_i^*)^\ell$, a signature σ and a public key vk outputs a bit $b \in \{0, 1\}$ if $\prod_{i\in[\ell]} e(M_i, \hat{X}_i) = e(Z, \hat{Y}) \wedge e(Y, \hat{P}) = e(P, \hat{Y})$.

Mercurial Signatures. Mercurial signatures [32] (which is an extensions of SPSEQ) and signatures with flexible public keys (SFPK) [4], receptively that allow signatures to be adapted not only to multiples of the signed message but also to multiples of the verification key (provide relations $[\mathsf{vk}]_{\mathcal{R}}$ on public keys). Mercurial signatures additionally support randomization of the messages such as SPSEQ and we use the formalization from mercurial signatures throughout the paper. The combination of randomizing message spaces and public keys allows an anonymous credential scheme to delegate signing to intermediate signers such that, if a user receives a credential from an intermediate signer, a verifier determine which intermediate signer issued the credential [32]. The additional property compare to SPSEQ is called public key class-hiding and states that it is hard to distinguish if a random public key is in relation to a different public key. The space of public keys, similar to the message space in [39], consist of vectors of group elements from $\mathbb{G}_2^*$ and so we can define the following equivalence relations as follows:

$$\mathcal{R}_{\mathsf{vk}} = \{(\mathsf{vk}', \mathsf{vk}) \in (\mathbb{G}_2^*)^\ell \times (\mathbb{G}_2^*)^\ell | \exists \omega \in \mathbb{Z}_p^*\ st.\ \mathsf{vk}' = \mathsf{vk}^\omega\}$$

$$\mathcal{R}_{\mathsf{sk}} = \{(\mathsf{sk}, \mathsf{sk}') \in (\mathbb{Z}_p^*)^\ell \times (\mathbb{Z}_p^*)^\ell | \exists \omega \in \mathbb{Z}_p^*\ st.\ \mathsf{sk}' = \omega \cdot \mathsf{sk}\}$$

Formally, in addition to the SPSEQ algorithms, we also require the following algorithms, which for the scheme in [32] (which extends the SPSEQ scheme from [39] presented above) are as follows:

$\mathsf{ConvertSK}(\mathsf{sk}, \omega) \rightarrow \mathsf{sk}'$: On input sk and a key converter $\omega \in \mathbb{Z}_p^*$, output a new secret key $\mathsf{sk}' = \omega \cdot \mathsf{sk}$.

$\mathsf{ConvertVK}(\mathsf{vk}, \omega) \to \mathsf{vk}'$: On input $\mathsf{vk} = (\hat{X}_i)_{i\in[\ell]}$ and a key converter $\omega \in \mathbb{Z}_p^*$, output a new public key $\mathsf{vk}' \in [\mathsf{vk}]_{\mathcal{R}_{\mathsf{vk}}}$ as $\mathsf{vk}' = \mathsf{vk}^\omega$ (guarantee if vk corresponds to sk, then vk′ corresponds to sk′).

$\mathsf{ConvertSig}(\mathsf{vk}, m, \sigma, \omega) \to \sigma'$: On input vk, a message $m \in \mathbb{Z}_p^*$, a signature $\sigma = (Z, Y, \hat{Y})$, and key converter $\omega \in \mathbb{Z}_p$, pick $r \xleftarrow{\$} \mathbb{Z}^* p$, this probabilistic algorithm returns a new signature σ' such that $\mathsf{Verify}(\mathsf{vk}', m, \sigma') = 1$ as: $\sigma' \leftarrow (Z^{r\omega}, Y^{\frac{1}{r}}, \hat{Y}^{\frac{1}{r}})$.

A mercurial signature is origin-hiding if in addition to the origin-hiding of ChangRep (cf. Def. 15) the following property holds:

Definition 2 (Origin-hiding of ConvertSig [32]). *For all λ, for all $\mathsf{pp} \in \mathsf{Setup}(1^\lambda)$, for all vk, for all M, σ, if $\mathsf{Verify}(\mathsf{vk}, M, \sigma) = 1$, if $\omega \leftarrow \mathbb{Z}_p$, then $\mathsf{ConvertSig}(\mathsf{vk}, M, \sigma, \omega)$ outputs a uniformly random σ and $\mathsf{ConvertVK}(\mathsf{vk}, \omega)$ outputs a uniformly random element of $[\mathsf{vk}]_{\mathcal{R}}$.*

2.2 Set Commitments

The notion of a set commitment SC with subset openings has been introduced in [39]. SC allows committing to a set $S \subset \mathbb{Z}_p$ by committing to a monic polynomial whose roots are the elements of S and supports openings for sets $T \subseteq S$.

Set Commitment Construction of [39]. To simplify our description, we ignore the case that a set S contains the trapdoor α. For a non-empty set S, [39] defines the polynomials $f_S(X) := \prod_{s\in S}(X - s) = \sum_{i=0}^{|S|} f_i \cdot X^i$. Note that for P, since $P^{f_S(\alpha)} = \prod_{i=0}^{|S|} P^{(f_i \cdot \alpha^i)}$, one can efficiently compute $P^{f_S(\alpha)}$ when given $\left(P^{\alpha^i}\right)_{i=0}^{|S|}$. We follow the definition given in [54], which makes the algorithm to randomize set commitments and opening information RndmzC explicit.

$\mathsf{SC.Setup}(1^\lambda, 1^t) \to \mathsf{pp_{SC}}$: On input a security parameter λ and a maximum set cardinality t, run $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, \hat{P}, e) \leftarrow \mathsf{BGGen}(1^\lambda)$, pick $\alpha \leftarrow \mathbb{Z}_p$ and output $\mathsf{pp_{SC}} \leftarrow (\mathsf{BG}, (P^{\alpha^i}, \hat{P}^{\alpha^i})_{i\in[t]})$, which defines message space $S_{\mathsf{pp_{SC}}} = \{S \subset \mathbb{Z}_p | 0 < |S| \leq t\}$. $\mathsf{pp_{SC}}$ will be an implicit input to all algorithms.

$\mathsf{SC.Commit}(S) \to (C, O)$: On input a set $S \in S_{\mathsf{pp_{SC}}}$, pick $\rho \leftarrow \mathbb{Z}_p^*$, compute $C \leftarrow (P^{f_S(\alpha)})^\rho \in \mathbb{G}_1^*$ and output (C, O) with $O \leftarrow \rho$.

$\mathsf{SC.Commit_2}(S_j, \alpha, P^{\rho_j}) \to (C_j, O_j)$: On input a set $S_j \in S_{\mathsf{pp_{SC}}}$, α, and P^{ρ_j}: compute a commitment $C_j = (P^{\rho_j})^{f_{S_j}(\alpha)} \in \mathbb{G}_1^*$ and output (C_j, O_j) with $O_j \leftarrow \perp$.

$\mathsf{SC.Open}(C, S, O) \to 0/1$: On input a commitment C, a set S, and an opening information $O = \rho$: if $C \notin \mathbb{G}_1^*$ or $\rho \notin \mathbb{Z}_p^*$ or $S \notin S_{\mathsf{pp_{SC}}}$ then return $\perp$. Otherwise if $O = \rho$ and $C = (P^{f_S(\alpha)})^\rho$, return 1; else return 0.

$\mathsf{SC.OpenSubset}(C, S, O, T) \to W$: On input a commitment C, a set S, an opening information O and a set T, if $0 \leftarrow \mathsf{SC.Open}(C, S, O)$ or $T \not\subseteq S$ or $T = \emptyset$ then return $\perp$. If $O = \rho$, output $W \leftarrow (P^{f_{S\setminus T}(\alpha)})^\rho$.

$\mathsf{SC.VerifySubset}(C, T, W) \to 0/1$: On input a commitment C, a set T and a witness W: if $C \notin \mathbb{G}_1^*$ or $T \notin S_{\mathsf{pp_{SC}}}$, return 0. Else if $W \in \mathbb{G}_1^* \wedge e(W, \hat{P}^{f_T(\alpha)}) = e(C, \hat{P})$, return 1; else 0.

$\mathsf{SC.RndmzC}(C, O, \mu) \to (C', O')$: On input a set commitment C, an opening information O and a randomness $\mu \in \mathbb{Z}_p$, output $C' = C^\mu$ and $O' = \mu \cdot O$.

In [54] it is shown that one can batch different subset opening witnesses into one and to improve the efficiency of the verification operation called as cross-set commitment aggregation. Therefore, two additional algorithms to aggregate witnesses across k commitments and verify them are added:

$\mathsf{SC.AggregateAcross}(\{C_j, T_j, W_j\}_{j\in[k]}) \to \pi$. Takes as input a collection $(\{C_j, T_j\}_{j\in[k]})$ along with the corresponding subset opening witnesses $\{W_j\}_{j\in[k]}$ and outputs an aggregated proof π as follows:

$$\pi := \prod_{j\in[k]} W_j^{t_j}, \text{ where } t_j = H(j, \{C_j, T_j\}_{j\in[k]}).$$

$\mathsf{SC.VerifyAcross}(\{C_j, T_j\}_{j\in[k]}, \pi) \to b$. Checks that the following equation holds:

$$\prod_{j\in[k]} e(C_j, \hat{P}^{t_j \cdot Z_{S\setminus T_j}(\alpha)}) = e(\pi, \hat{P}^{Z_S(\alpha)})$$

where $S = \bigcup_j T_j$, and $Z_S(\alpha) = \prod_{i\in S}(\alpha - i)$.

3 Aggregate Signatures with Randomizable Keys and Tags

Now we introduce a novel primitive named AtoSa where one can aggregate signatures of different messages under different keys only if they are associated with the same tag (consisting of a private and a public part). Moreover, apart from allowing randomizing signatures, verification keys as well as tags can be randomized. Unlike mercurial signatures, our ATMS scheme does not allow for randomization of messages. Tags and verification keys are defined with respect to equivalence classes and randomization switches between representatives of these classes. We introduce a comprehensive formal model and a construction which as a starting point takes PS signatures [57]. For our AtoSa scheme we show how to integrate tags into PS signatures, use the above discussed features to make them aggregatable, and show that the key-randomization features of PS signatures (cf. [28] with $\Delta_2 = 0$) applies to our modification.

3.1 Formal Definitions

The public key randomization is similar to that of mercurial signatures [32], which allow to define equivalence classes on the key space $[\mathsf{vk}]_{\mathcal{R}_{\mathsf{vk}}}$, $[\mathsf{sk}]_{\mathcal{R}_{\mathsf{sk}}}$ (cf. Section 2.1). Let a tag be $(\tau, \mathbf{T})$, where τ and $\mathbf{T}$ are the secret and public parts of tag respectively. For the tag randomization, we define equivalence classes $[\mathbf{T}]_{\mathcal{R}_\tau}$ ($[\tau]_{\mathcal{R}_\tau}$ for secret parts) on the tag space $\mathcal{T}$ similar to $[\mathsf{vk}]_{\mathcal{R}_{\mathsf{vk}}}$ and $[\mathsf{sk}]_{\mathcal{R}_{\mathsf{sk}}}$ as:

$$\mathcal{R}_\tau = \begin{Bmatrix} (\mathbf{T}', \mathbf{T}) \in (\mathbb{G}_1^*)^\ell \times (\mathbb{G}_1^*)^\ell \mid \exists \mu \in \mathbb{Z}_p^* : \ \mathbf{T}' = \mathbf{T}^\mu \\ (\tau', \tau) \in (\mathbb{Z}_p^*)^\ell \times (\mathbb{Z}_p^*)^\ell \mid \exists \mu \in \mathbb{Z}_p^* : \ \tau' = \tau \cdot \mu \end{Bmatrix}$$

We denote the space of all tags as $\mathcal{T}$ and the messages space is $\mathbb{Z}_p$. In contrast to SPSEQ (and mercurial) signatures, we do not consider equivalence classes on the message space for AtoSa.

Definition 3 (Aggregate Signatures with Randomizable Public Keys and Tag (AtoSa)). *An* AtoSa *for parameterized equivalence relations $\mathcal{R}_\tau$, $\mathcal{R}_{\mathsf{sk}}$ and $\mathcal{R}_{\mathsf{vk}}$, consists of the following algorithms:*

$\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: On input the security parameter λ, output the public parameters pp.
$\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{vk})$: On input the public parameters pp, output a key pair $(\mathsf{sk}, \mathsf{vk})$.
$\mathsf{VKeyGen}(\mathsf{sk})$: On input a secret key sk, output a verification key vk.
$\mathsf{GenAuxTag}(S) \to (\{\mathsf{aux}_j\}_{j\in[n]}, (\tau, \mathbf{T}))$: Given a message-key set $S = \{(m_j, \mathsf{vk}_j)_{j\in[n]}\}$, output auxiliary data $\{\mathsf{aux}_j\}_{j\in[n]}$ correlated to (vk_j, m_j) and a tag pair $(\tau, \mathbf{T})$, where all vk_j should be distinct.
$\mathsf{Sign}(\mathsf{sk}_j, \tau, \mathsf{aux}_j, m_j) \to \sigma_j$: On input a secret key sk_j, tag's secret τ, auxiliary data aux_j and message $m_j \in \mathbb{Z}_p$, output a signature σ_j for $(\tau, \mathbf{T})$ and m_j under the verification key vk_j.
$\mathsf{Verify}(\mathsf{vk}_j, \mathbf{T}, m_j, \sigma_j) \to \{0, 1\}$: Given a verification key vk_j, tag's public $\mathbf{T}$, message m_j and signature σ_j, output 1 if σ_j is valid relative to vk_j, m_j and $\mathbf{T}$, and 0 otherwise.
$\mathsf{AggrSign}(\mathbf{T}, \{(\mathsf{vk}_j, m_j, \sigma_j)\}_{j=1}^\ell) \to \sigma$: Given ℓ signatures, $(\sigma_j)_{j\in[\ell]}$ for messages $(m_j)_{j\in[\ell]}$ under verification keys, $(\mathsf{vk}_j)_{j\in[\ell]}$ on the same tag $\mathbf{T}$, output an aggregate signature σ on all messages $\mathbb{M} = (m_j)_{j\in[\ell]}$ under the tag $\mathbf{T}$ and aggregated verification key $\mathsf{avk} = (\mathsf{vk}_j)_{j\in[\ell]}$.
$\mathsf{VerifyAggr}(\mathsf{avk}, \mathbf{T}, \mathbb{M}, \sigma) \to \{0, 1\}$: Given an aggregated verification key avk, tag $\mathbf{T}$, messages $\mathbb{M}$ and signature σ, output 1 if σ is valid relative to avk, $\mathbb{M}$ and $\mathbf{T}$, and 0 otherwise.
$\mathsf{ConvertTag}(\mathbf{T}, \mu) \to \mathbf{T}'$: On input a tag $\mathbf{T}$ and randomness μ, output a new randomized tag $\mathbf{T}' \in [\mathbf{T}]_{\mathcal{R}_\tau}$.
$\mathsf{RndSigTag}(\mathsf{vk}, \mathbf{T}, m, \sigma, \mu) \to (\sigma', \mathbf{T}')$: (Randomize Signature and Tag together) Given a signature σ on a message m under tag $\mathbf{T}$ and vk, and randomness μ. Return a randomized signature and tag $(\sigma', \mathbf{T}')$ s.t $\mathsf{Verify}(\mathsf{vk}, \mathbf{T}', m, \sigma') = 1$, where $\mathbf{T}' \leftarrow \mathsf{ConvertTag}(\mathbf{T}, \mu)$.
$\mathsf{ConvertSK}(\mathsf{sk}, \omega) \to \mathsf{sk}'$: On input a sk and key converter ω, output a new secret key sk'.
$\mathsf{ConvertVK}(\mathsf{vk}, \omega) \to \mathsf{vk}'$: On input a vk and key converter ω, output a new public key vk'.
$\mathsf{ConvertSig}(\mathsf{vk}, m, \mathbf{T}, \sigma, \omega) \to \sigma'$: On input a vk, message m, tag $\mathbf{T}$, signature σ, and key converter ω, return a new signature σ' s.t $\mathsf{Verify}(\mathsf{vk}', \mathbf{T}, m, \sigma') = 1$, where $\mathsf{vk}' \leftarrow \mathsf{ConvertVK}(\mathsf{vk}, \omega)$.

We note that VKeyGen is only required in the security definition and is never used in the construction. Although the signer receives the tag secret key τ, we replace this with a ZKP in our IhMA scheme.

3.2 Security Definitions

Correctness. As usual we require that honest signatures verify as expected, but need to consider all the randomizations as well as the aggregation. We fomalize this in Appendix B.1.

Unforgeability. We model unforgeability following the ideas in the chosen-key model [11, 53], where the adversary $\mathcal{A}$ is given a single public key vk' and access to a signing oracle on the challenge key. The adversary wins if the aggregate signature, σ, is a valid aggregate signature on a vector of messages $\mathbb{M} = (m_1, \ldots, m_n)$ under keys $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$, and σ is nontrivial, i.e., the adversary did not request a signature on a m_j for $\mathsf{vk}_j = \mathsf{vk}'$ or more precisely where vk_j is in the same equivalence class as the challenge key vk'. $\mathcal{A}$ has the power to choose all public keys except the challenger's public key vk'. For our instantiation, however, we have to work in a slightly weakened model which is equivalent to the certified-keys model [50, 51]. In this setting the $\mathcal{A}$ registers pairs of $(\mathsf{vk}, \mathsf{sk})$ with exception of the challenge key. To model this, we have the adversary output the secret keys of the verification keys they provide in our security games. In the real world, such a key registration can be realized by requiring issuers to prove knowledge of their sk, which in the formal analysis allows a reduction to extract the secret key.

Definition 4 (Unforgeability). *An* AtoSa *signature is unforgeable if for all PPT algorithms $\mathcal{A}$ having access to the oracle $\mathcal{O}^{\mathsf{Sign}()}$, there exists a negligible function ϵ such that:* $\Pr[\mathsf{ExpUnf}_{\mathsf{AtoSa},\mathcal{A}}(\lambda) = 1] \leq \epsilon(\lambda)$ *where the experiment* $\mathsf{ExpUnf}_{\mathsf{AtoSa},\mathcal{A}}(\lambda)$ *is defined in Fig. 1 and Q is the set of queries that $\mathcal{A}$ has issued to the* $\mathcal{O}^{\mathsf{Sign}}$.

$\underline{\mathsf{ExpUnf}_{\mathsf{AtoSa},\mathcal{A}}(\lambda)}$:

- $Q := \emptyset; \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$;
- $(\mathsf{vk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}(\mathsf{pp})$;
- $(j', \mathsf{avk} = \left(\mathsf{vk}_j\right)_{j\in[\ell]}, a\mathsf{sk} = (\mathsf{sk}_j)_{j\in[\ell]\setminus j'}, \mathbb{M}^* = (m_j^*)_{j\in[\ell]}, (\tau^*, \mathbf{T}^*), \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp}, \mathsf{vk}')$
- $(\mathsf{vk}_j^*) := \left(\mathsf{VKeyGen}\left(\mathsf{sk}_j\right)\right)_{j\in[\ell]\setminus j'}$,

return:

$$\begin{pmatrix} \mathsf{VerifyAggr}\left(\mathsf{avk}, \mathbf{T}^*, \sigma^*, \mathbb{M}^*\right) = 1 \ \wedge \forall j \in [\ell], j \neq j' : \\ [\mathsf{vk}_j^*]_{\mathcal{R}_{\mathsf{vk}}} = [\mathsf{vk}_j]_{\mathcal{R}_{\mathsf{vk}}} \wedge \ [\mathsf{vk}']_{\mathcal{R}_{\mathsf{vk}}} = [\mathsf{vk}_{j'}]_{\mathcal{R}_{\mathsf{vk}}} \\ \wedge \forall (m, \mathbf{T}) \in Q : m \neq m_j^* \vee [\mathbf{T}]_{\mathcal{R}_\tau} \neq [\mathbf{T}^*]_{\mathcal{R}_\tau} \end{pmatrix}$$

$\underline{\mathcal{O}^{\mathsf{Sign}}(m, \mathsf{aux}, (\tau, \mathbf{T}))}$:

- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}', \tau, \mathsf{aux}, m)$
- $Q = Q \cup \{m, \mathbf{T}\}$,

return σ

Fig. 1: Experiment $\mathsf{ExpUnf}_{\mathsf{AtoSa},\mathcal{A}}(\lambda)$

Privacy guarantees. Similar to mercurial signatures [32], we define the following privacy notion for randomized keys vk and tags:

Definition 5 (Public key class-hiding). *For all PPT adversaries $\mathcal{A}$, and* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ *there exists a negligible ϵ such that:*

$$\Pr\begin{bmatrix} (\mathsf{vk}_1, \mathsf{sk}_1) \leftarrow \mathsf{KeyGen}(\mathsf{pp}); (\mathsf{vk}_2^0, \mathsf{sk}_2^0) \leftarrow \mathsf{KeyGen}(\mathsf{pp}); \\ r \xleftarrow{\$} \mathbb{Z}_p; \mathsf{vk}_2^1 = \mathsf{ConvertVK}(\mathsf{vk}_1, r); \mathsf{sk}_2^1 = \mathsf{ConvertSK}(\mathsf{sk}_1, r); \\ b \leftarrow \{0,1\}; b' \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}_1,\cdot),\mathsf{Sign}(\mathsf{sk}_2^b,\cdot)}(\mathsf{vk}_1, \mathsf{vk}_2^b) : b' = b \end{bmatrix} \leq \frac{1}{2} + \epsilon(\lambda)$$

Definition 6 (Tag class-hiding). *For all PPT adversaries $\mathcal{A}$ there is a negligible function $\epsilon(\cdot)$ such that*

$$\Pr\begin{bmatrix} b \leftarrow \{0,1\}, \mathsf{BG} \leftarrow \mathsf{BGGen}(1^\lambda), \mathbf{T} \leftarrow \mathcal{T}, \mathbf{T}^{(0)} \leftarrow \mathcal{T}, \\ \mathbf{T}^{(1)} \leftarrow [\mathbf{T}]_{\mathcal{R}}, b^* \leftarrow \mathcal{A}(\mathsf{BG}, \mathbf{T}, \mathbf{T}^{(b)}) : b^* = b \end{bmatrix} - \frac{1}{2} \leq \epsilon(\lambda)$$

The tag class-hiding property for $\mathcal{R}_\tau$ is implied by the DDH assumption.

The following definition guarantees that a signature with tag $\mathbf{T}$ on a message m under vk output by ConvertSig and fed into RndSigTag produces a uniformly random signature under a uniformly random tag (from the respective tag class) and uniformly random key (from the respective key class).

Definition 7 (Origin-hiding of ConvertSig). *For all λ, and $\mathsf{pp} \in \mathsf{Setup}(1^\lambda)$, for all $(\mathsf{vk}, m, \sigma, \mathbf{T})$, if $\mathsf{Verify}(\mathsf{vk}, \mathbf{T}, m, \sigma) = 1$, and $(\omega, \mu) \in \mathbb{Z}_p^*$, then $(\sigma', \mathbf{T}') \leftarrow \mathsf{RndSigTag}(\mathsf{vk}, \mathbf{T}, m, \mathsf{ConvertSig}(\mathsf{vk}, m, \mathbf{T}, \sigma, \omega), \mu)$ outputs uniformly random elements in signature space and $[\mathbf{T}]_{\mathcal{R}_\tau}$) such that $\mathsf{Verify}(\mathsf{vk}', \mathbf{T}', m, \sigma') = 1$, and $\mathsf{vk}' \xleftarrow{\$} \mathsf{ConvertVK}(\mathsf{vk}, \omega)$ is a uniformly random element of $[\mathsf{vk}]_{\mathcal{R}_{\mathsf{vk}}}$.*

We also require a similar definition for ConvertTag and the tag randomization:

Definition 8 (Origin-hiding of ConvertTag). *For all λ, for all $\mathsf{pp} \in \mathsf{Setup}(1^\lambda)$, for all $(\mathsf{vk}, m, \sigma, \mathbf{T})$, if $\mathsf{Verify}(\mathsf{vk}, \mathbf{T}, m, \sigma) = 1$, and $\mu \in \mathbb{Z}_p^*$, then $(\sigma', \mathbf{T}') \leftarrow \mathsf{RndSigTag}(\mathsf{vk}, \mathsf{ConvertTag}(\mathbf{T}, \mu), m, \sigma, \mu)$ outputs uniformly random elements in the signature space and $[\mathbf{T}]_{\mathcal{R}_\tau}$ such that $\mathsf{Verify}(\mathsf{vk}, \mathbf{T}', m, \sigma') = 1$.*

3.3 Construction

We construct the AtoSa scheme based on the PS signature [57]. We can observe that to make PS signatures (h_i, s_i) aggregateable, we need the h_i components to be identical for all signatures to be aggregated. While in the original PS construction h is a random element independently chosen during signing, this can be emulated in AtoSa by generating h for all signatures via a hash function based on some common information embedded in aux. For example, aux, could be a concatenation of all the messages and the tag. This technique was implicitly used in Coconut [63] and Camenisch et al. [18], and has recently been formalized by Crites et al. in [31].

We note that we should be careful when computing h, i.e., in choosing aux, as in PS signatures one can forge signatures when obtaining two signatures on two different messages with respect to the same element h. To prevent forgeries when aiming to aggregate signatures, a unique base h for a set of messages signed under the same tag is required. Therefore, we compute h as a hash of a concatenation of the messages to be signed and corresponding verification keys, denoted as aux. This approach ensures that every signer computes signatures on the same base h. We also introduce a new definition and function:

Aux binding. To ensure this property of h while making our construction modular, we define a straightforward property of $\mathsf{GenAuxTag}(S)$, i.e., no adversary can "open" an aux to two messages for the same signer. This definition is paired with the function VerifyAux which is called by Sign.

Definition 9 (Aux binding). *We split aux into a preimage and an opening: (c, o). For all PPT $\mathcal{A}$, and $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{VKeyGen}(1^\lambda)$ there exists a negligible ϵ such that:*

$$\Pr\left[\begin{array}{l}(h, \mathsf{aux} = (c, o), \mathsf{aux} = (c', o'), \tau, m, \tau', m') \leftarrow \mathcal{A}(\mathsf{vk}); \\ \mathsf{VerifyAux}(\mathsf{sk}, (c, o), \tau, m) = 1 \\ \wedge\ \mathsf{VerifyAux}(\mathsf{sk}, (c', o'), \tau', m') = 1; \\ c = c' \wedge ([\tau]_{\mathcal{R}_\tau} \neq [\tau']_{\mathcal{R}_\tau} \vee m \neq m')\end{array}\right] \leq \epsilon(\lambda)$$

We will then hash the preimage, c in our construction to reduce to the GPS assumption (Generalized Pointcheval-Sanders assumption shown in Appendix C.2) effectively. The o value in this definition may seem unnecessary, but it will become useful when we introduce our IhMA construction in Section 5. We've left aux binding out of our definition and rather defined it in our construction in order to make our definition more generic as aux binding is simply a property we use in the proof to ensure that our construction satisfies the definition of AtoSa.

Synchronicity assumption. We note that when we do not want to fix messages and verification keys in aux beforehand, then we can make assumption as in synchronized aggregate signatures [2, 46] and require each signer to only issue *a single signature per tag*. In this case aux only contains the tag and in the construction below we set $c = P^{\rho_1} || P^{\rho_2}$ and Definition 9 is trivially satisfied.

We involve the tag in signatures by exponentiating the component h with the secret part of the tag h^ρ and compute the component s using this value, which clearly can be checked via a pairing with the tag's public part and verified like a standard PS signature. Moreover, AtoSa allows the

randomization of tag, vk and signatures via a change of representatives tag and vk and a matching signature update.

Our construction. The construction is as follows:

- $\mathsf{Setup}(1^\lambda)$: Run $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, \hat{P}, e) \leftarrow \mathsf{BGGen}(1^\lambda)$ with a prime number order p, where P is a generator of $\mathbb{G}_1$, $\hat{P}$ a generator of $\mathbb{G}_2$. Pick H as a hash function: $H : \{0,1\}^* \rightarrow \mathbb{G}_1$. Output public parameters $\mathsf{pp} = \{\mathsf{BG}, H\}$.
- $\mathsf{KeyGen}\,(\mathsf{pp})$: Choose $(x, y_1, y_2) \xleftarrow{\$} \mathbb{Z}_p$ and set the secret key $\mathsf{sk} = (x, y_1, y_2)$ and verification key $\mathsf{vk} = (\hat{Y}_1 = \hat{P}^{y_1}, \hat{Y}_2 = \hat{P}^{y_2}, \hat{X} = \hat{P}^x)$.
- $\mathsf{VKeyGen}\,(\mathsf{sk})$: On input a secret key $\mathsf{sk} = (x, y_1, y_2)$, output $\mathsf{vk} = (\hat{Y}_1 = \hat{P}^{y_1}, \hat{Y}_2 = \hat{P}^{y_2}, \hat{X} = \hat{P}^x)$.
- $\mathsf{GenAuxTag}(S)$: Given a set $S = \{(m_j, \mathsf{vk}_j)_{j\in[\ell]}\}$, choose $(\rho_1, \rho_2) \xleftarrow{\$} \mathbb{Z}_p$, set $c = P^{\rho_1}||P^{\rho_2}||(m_j, \mathsf{vk}_j)_{j\in[\ell]}$. Next set all $\mathsf{aux}_j = (c, \perp)$. Compute $h = H(c)$ and output aux and a tag pair $(\tau = (\rho_1, \rho_2), \mathbf{T} = (T_1 = h^{\rho_1}, T_2 = h^{\rho_2}))$.
- $\mathsf{VerifyAux}(\mathsf{sk}, \mathsf{aux}, \tau, m_j)$ Parse aux as (c, o). Check that $\tau \in c$ (i.e., that c has the form $P^{\rho_1}||P^{\rho_2}||...$) and $(m_j, \mathsf{vk}) \in c$ where vk is a verification key related to sk (in the same equivalence class). Also check that no other vk_j in aux has the same equivalence class as sk. This can be done by checking that $\hat{Y}_2 = \hat{Y}_1^{\frac{y_2}{y_1}}$ and that $\hat{X} = \hat{Y}_2^{\frac{x}{y_2}}$. If these checks pass, it means that this is in the same equivalence class as the verifier's key. If the check doesn't pass, it means the vk_j is not in the same equivalence class.
- $\mathsf{Sign}(\mathsf{sk}_j, \tau, \mathsf{aux}_j, m_j)$: Given a $\mathsf{sk}_j = (y_{1j}, y_{2j}, x_j)$, τ, aux_j and a message m_j. If $\mathsf{VerifyAux}(\mathsf{sk}_j, \mathsf{aux}_j, \tau, m_j) \neq 1$ return $\perp$. Else, parse aux as (c, o) and compute $h = H(c)$ and output:

$$\sigma_j = \left(h', s_j\right) = \left(h' = h^{\rho_1}, s_j = (h^{\rho_1})^{x_j + y_{1j}\cdot m_j} \cdot (h^{\rho_2})^{y_{2j}}\right)$$

- $\mathsf{Verify}(\mathsf{vk}_j, \mathbf{T}, m_j, \sigma_j)$: Given a vk_j, tag $\mathbf{T} = (T_1, T_2)$, message m_j and signature σ_j, parse σ_j as $\left(h', s_j\right)$ and return 1 if the following checks hold and 0 otherwise:

$$e(h', \hat{X} \cdot \hat{Y}_1^{m_1}) e(T_2, \hat{Y}_2) = (s_j, \hat{P}) \ \wedge \ T_1 = h' \neq 1_{\mathbb{G}}$$

- $\mathsf{AggrSign}(\mathbf{T}, \{(\mathsf{vk}_j, m_j, \sigma_j)\}_{j=1}^{\ell})$: Given ℓ valid signatures $\sigma_j = (h', s_j)$ for m_j under vk_j and the same tag $\mathbf{T}$, where $j \in [\ell]$, outputs an aggregate signature σ on the messages $\mathbb{M} = (m_j)_{j\in[\ell]}$ under the tag $\mathbf{T}$ and aggregated verification key $\mathsf{avk} = (\mathsf{vk}_j)_{j\in[\ell]}$ as: $\sigma' = \left(h', s' = \prod_{j=1}^{\ell} s_j\right)$.
- $\mathsf{VerifyAggr}(\mathsf{avk}, \mathbf{T}, \mathbb{M}, \sigma)$: Given an avk, tag $\mathbf{T}$, messages $\mathbb{M}$ and aggregate signature $\sigma = (h', s)$, it outputs 1 if the following checks holds and 0 otherwise:

$$e\left(h', \prod_{j\in[\ell]} \hat{X}_j \cdot \hat{Y}_{1j}^{m_j}\right) e\left(h^{\rho_2}, \prod_{j\in[\ell]} \hat{Y}_{2j}\right) = e\left(s, \hat{P}\right) \ \wedge \ T_1 = h' \neq 1_{\mathbb{G}}$$

- $\mathsf{ConvertTag}(\mathbf{T}, \mu) \rightarrow \mathbf{T}'$: On input a tag $\mathbf{T}$ and randomness μ, output a randomized tag $\mathbf{T}' = \mathbf{T}^\mu = (T_1^\mu, T_2^\mu)$.
- $\mathsf{RndSigTag}(\mathsf{vk}, \mathbf{T}, m, \sigma, \mu) \rightarrow (\sigma', \mathbf{T}')$: Given a signature σ on message m under a valid tag $\mathbf{T}$ and vk, and randomness μ. Return a randomized signature σ' and a randomized tag:

$$\sigma' = \left(h'^\mu, s^\mu\right), \ \ \mathbf{T}' \leftarrow \mathsf{ConvertTag}(\mathbf{T}, \mu)$$

  where is a valid signature for a new tag representative $\mathbf{T}' \in [\mathbf{T}]_{\mathcal{R}_\tau}$.
- $\mathsf{ConvertSK}(\mathsf{sk}, \omega)$: On input sk and a key converter $\omega \in \mathbb{Z}_p^*$, output a new secret key sk' as $\mathsf{sk}' = \mathsf{sk} \cdot \omega$.
- $\mathsf{ConvertVK}(\mathsf{vk}, \omega)$: On input vk and a key converter $\omega \in \mathbb{Z}_p^*$, output a new public key as $\mathsf{vk}' = \mathsf{vk}^\omega$.
- $\mathsf{ConvertSig}(\mathsf{vk}, m, \mathbf{T}, \sigma, \omega)$: On input a vk, message m, signature σ, tag $\mathbf{T}$, and key converter $\omega \in \mathbb{Z}_p^*$, return a new signature σ' s.t. $\mathsf{Verify}(\mathsf{vk}', \mathbf{T}, m, \sigma') = 1$, where $\mathsf{vk}' \xleftarrow{\$} \mathsf{ConvertVK}(\mathsf{vk}, \omega)$ as follows: $\sigma' = (h', s' = s^\omega)$.

The correctness of our construction follows from inspection. We formally show the unforgeability and privacy notations.

Theorem 1 (Unforgeability). *Our construction achieves the EUF-CMA security stated in Def 4, under the hardness of GPS assumption, stated in Def. 31 in the random oracle model.*

The proof of this theorem is provided in Appendix D.1.

Theorem 2 (Privacy). *Our construction is origin-hiding of ConvertSig, origin-hiding of* RndSigTag*, tag class hiding and has public key class-hiding based on Def. 7, Def. 8, Def. 6, and Def. 5, respectively.*

The proof of Theorems 2 are provided in Appendix D.2.

4 Aggregate Mercurial Signatures With Randomizable Tags

We now present an aggregate mercurial signature with randomizable tags (ATMS). Similar to AtoSa, (see Def. 3), one can aggregate mercurial signatures of different messages under different keys under the same tag and randomize those signatures, public keys, and tags. ATMS differs from AtoSa by in addition supporting equivalence classes on the message space. This further allows the randomization of messages, leading to a feature known from structure-preserving signature on equivalence classes (SPSEQ) and, more precisely, mercurial signatures.
To achieve the aggregation property, we follow the strategy presented by Crites et al. in context of threshold SPS [31], where the authors define a so called Indexed Diffie-Hellman message space $\mathcal{M}^H_{\mathsf{iDH}}$. But the main problem with this approach, as it is defined over both groups, is that we can not define indistinguishable equivalence classes over $\mathbb{G}_1^k \times \mathbb{G}_2^k$, since spanning both groups makes DDH easy and would yield trivial linkability. Note that given both $((M_1, M_2), (N_1, N_2))$ and $((M_1', M_2'), (N_1', N_2'))$, one can easily link them together by checking $e(M_1, N_2') = e(M_2, N_1')$ and $e(M_1', N_2) = e(M_2', N_1)$ holds. So we adapt $\mathcal{M}^H_{\mathsf{iDH}}$ and define a new message space called a Tag-based DH message space $\mathcal{M}^H_{\mathsf{TDH}}$ and its corresponding EQ relation. We essentially define one equivalence class per group and tie them together via the message, the tag, and an index obtained via some auxiliary information (similar to the aux in the case of AtoSa). Indeed we adapt the Diffie-Hellman message space $\mathcal{M}_{\mathsf{DH}}$ to a Tag-based DH message space $\mathcal{M}^H_{\mathsf{TDH}}$ for a tuple $(\mathsf{aux}, h, \mathbf{T}, M, N)$, which includes a tag $\mathbf{T}$ with auxiliary data aux (instead of the *id*).

This new message space then allows us to aggregate and define an equivalence (EQ) relation which gives an indistinguishable message space.

4.1 Formal Definitions

We begin our definitions by introducing Tag-based DH message space $\mathcal{M}^H_{\mathsf{TDH}}$ and give an instantiation in the random oracle model (ROM). Then we define a new EQ relation regarding this message space $\mathcal{M}^H_{\mathsf{TDH}}$, and finally, we define our new primitive ATMS.

A Tag-based DH **message space**. We adapt the message indexing technique introduced by [31] (cf. Def. 1) to tags:

Definition 10 (A Tag-based DH **message space ($\mathcal{M}^H_{\mathsf{TDH}}$)).** *Let H be a random oracle. For the* aux *and tag $\mathbf{T} = (h^{\rho_i})_{i\in[k]}$, we define $\mathcal{M}^H_{\mathsf{TDH}}$ as a tag based* DH *message space, if the following property hold: For the messages vector $(\mathbf{M}, \mathbf{N}) = (M_1, \ldots, M_k, N_1, \ldots, N_k)$ there exists $m_i \in \mathbb{Z}_p$ s.t. for each tuple $(\mathsf{aux}, T_i = h^{\rho_i}, M_i = T_i^{m_i}, N_i = \hat{P}^{m_i})$, the following holds: $e(M_i, \hat{P}) = e(T_i, N_i)$.*

We provide an instantiation in Fig. 2. Let us assume WLOG a message vector with the length $k = 2$ as $\mathbf{m} = (m_1, m_2)$, this can be generalized to any length $k > 1$.

Equivalence relations (EQ) over $\mathcal{M}^H_{\mathsf{TDH}}$. Let the message space $\mathcal{M}^H_{\mathsf{TDH}}$ be defined as $(\mathbf{M}, \mathbf{N}) = (M_1, \ldots . M_k, N_1, \ldots, N_k) \in (\mathbb{G}_1^*)^k \times (\mathbb{G}_2^*)^k$ such that for $(h, \mathbf{T})$, and $i \in [k]$: $e(M_i, \hat{P}) = e(T_i, N_i)$. Now we can define a family of equivalence relations $\mathbb{R}^\ell$ so that for any ℓ with $1 < k \leq \ell$. We define the following equivalence relation $\mathcal{R}_{\mathsf{TDH}} \in \mathbb{R}^\ell$ and the equivalence class $[(\mathbf{M}, \mathbf{N})]_{\mathcal{R}_{\mathsf{TDH}}}$ of a message vector with size k. More concretely, for a fixed bilinear group BG and (k, ℓ), we define $\mathcal{R}_{\mathsf{TDH}} \in \mathbb{R}^\ell$ as follows:

$\mathcal{M}^H_{\mathsf{TDH}}(\mathbf{T}=(h^{\rho_1},h^{\rho_2}),\mathsf{aux},\mathbf{m})$:	$H(\mathsf{aux})$:
– $h \leftarrow H(\mathsf{aux})$ – for $i \in [2]$: • $M_i \leftarrow h^{m_i\rho_i}$ • $N_i \leftarrow \hat{P}^{m_i}$ – **return** $(\mathbf{M},\mathbf{N})$	– If $Q_H[\mathsf{aux}] = \perp$: – $r \xleftarrow{\$} \mathbb{Z}_p$ – $Q_H[\mathsf{aux}] \leftarrow P^r := h$ – **return** $Q_H[\mathsf{aux}]$

Fig. 2: Tag based Diffie-Hellman message space in ROM

Definition 11 (Equivalence relations of $\mathcal{M}^H_{\mathsf{TDH}}$ message spaces). *If vectors of a pair $(\mathbf{M},\mathbf{N}) \in (\mathbb{G}_1^*)^k \times (\mathbb{G}_2^*)^k$ is a message vector from $\mathcal{M}^H_{\mathsf{TDH}}$, then the equivalence relations $[(\mathbf{M},\mathbf{N})]_{\mathcal{R}_{\mathsf{TDH}}}$ defined as*

$$\mathcal{R}_{\mathsf{TDH}} = \left\{ \begin{matrix} (\mathbf{M},\mathbf{N}),(\mathbf{M}',\mathbf{N}') \in (\mathbb{G}_1^* \times \mathbb{G}_2^*)^k \times (\mathbb{G}_1^* \times \mathbb{G}_2^*)^k \Leftrightarrow \exists (\mu,\upsilon) \in \mathbb{Z}_p^* : \\ \mathbf{M}' = \mathbf{M}^{\mu\upsilon}, \mathbf{N}' = \mathbf{N}^{\upsilon} \end{matrix} \right\}$$

Note that the EQ relation for an aggregate signature on a set of vectors $\mathbb{M} = ((\mathbf{M}_j,\mathbf{N}_j))_{j\in[\ell]}$ is the family (set) of relation as above, while all vectors use the same randomness $\mathbb{M} = ((\mathbf{M}_j^{\mu\upsilon},\mathbf{N}_j^{\upsilon}))_{j\in[\ell]}$. For instance, the j'th message vector $(\mathbf{M}_j,\mathbf{N}_j) \in [(\mathbf{M},\mathbf{N})]_{\mathcal{R}^j_{\mathsf{TDH}}}$ is in the class $\mathcal{R}^j_{\mathsf{TDH}} \in \mathbb{R}^\ell$ and if one more signature-message pair is added to the set, we have $\mathcal{R}^{j+1}_{\mathsf{TDH}} \in \mathbb{R}^\ell$, where $j+1 < \ell$. Moreover, we consider the EQ relation for verification keys vk and Tag similar to AtoSa and indicate as $\mathcal{R}_{\mathsf{vk}}$ (see Def. 2.1) and $\mathcal{R}_\tau$ as stated in Def. 3.1. We again denote by $\mathcal{T}$ the space of all tags. We present our ATMS scheme in Definition 12.

Definition 12 (Aggregate Mercurial Signatures with Randomizable Tag (ATMS)). *An ATMS scheme, associated with the parameterized equivalence relations $\mathbb{R}^\ell$, $\mathcal{R}_{\mathsf{TDH}}$, $\mathcal{R}_\tau$ and $\mathcal{R}_{\mathsf{vk}}$, and also message space $\mathcal{M}^H_{\mathsf{TDH}}$ consists of the algorithms:*

$\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: On input the security parameter λ, output the public parameters pp.
$\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{sk},\mathsf{vk})$: On input the public parameters pp, output a key pair $(\mathsf{sk},\mathsf{vk})$.
$\mathsf{VKeyGen}(\mathsf{sk})$: On input a secret key sk, output a verification key vk.
$\mathsf{GenAuxTag}(S) \to (\mathsf{aux}_j,(\tau,\mathbf{T}))$: Given a set $S = \left((\mathbf{M}_j,\mathbf{N}_j),\mathsf{vk}_j\right)_{j\in[n]}$ of messages and keys, output auxiliary data aux_j and a tag pair $(\tau,\mathbf{T})$ where τ is the secret part and $\mathbf{T}$ is the public part of tag and all vk_j should be distinct.
$\mathsf{Sign}(\mathsf{sk}_j,\tau,\mathsf{aux}_j,(\mathbf{M}_j,\mathbf{N}_j)) \to \sigma_j$: On input a secret key sk_j, tag's secret τ, auxiliary data aux_j and message vector $(\mathbf{M}_j,\mathbf{N}_j) \in \mathcal{M}^H_{\mathsf{TDH}}$, output a signature σ_j under the τ, vk_j and $(\mathbf{M}_j,\mathbf{N}_j)$.
$\mathsf{Verify}(\mathsf{vk}_j,\mathbf{T},(\mathbf{M}_j,\mathbf{N}_j),\sigma_j) \to \{0,1\}$: Given a verification key vk_j, tag's public $\mathbf{T}$, message vector $(\mathbf{M}_j,\mathbf{N}_j)$ and signature σ_j, output 1 if σ_j is valid relative to vk_j, $(\mathbf{M}_j,\mathbf{N}_j)$ and $\mathbf{T}$, and 0 otherwise.
$\mathsf{VerifyTag}(\mathbf{T},\tau,\sigma) \to \{0,1\}$: Given a tag's public $\mathbf{T}$, tag's secret signature σ, output 1 if $\mathbf{T}$ is valid relative to σ, and τ, and 0 otherwise.
$\mathsf{AggrSign}(\mathbf{T},(\mathsf{vk}_j,(\mathbf{M}_j,\mathbf{N}_j),\sigma_j)_{j=1}^\ell) \to \sigma'$ Given ℓ signed messages $(\mathbf{M}_j,\mathbf{N}_j)$ in σ_j under vk_j for $j \in [\ell]$ and the same tag $\mathbf{T}$, output a signature σ on the messages $\mathbb{M} = ((\mathbf{M}_j,\mathbf{N}_j))_{j\in[\ell]}$ under the tag $\mathbf{T}$ and verification key $\mathsf{avk} = (\mathsf{vk}_j)_{j\in[\ell]}$.
$\mathsf{VerifyAggr}(\mathsf{avk},\mathbf{T},\mathbb{M},\sigma) \to \{0,1\}$: Given a verification key avk, tag $\mathbf{T}$, messages $\mathbb{M}$ and signature σ, output 1 if σ is valid relative to avk, $\mathbb{M}$ and $\mathbf{T}$, and 0 otherwise.
$\mathsf{ConvertTag}(\mathbf{T},\mu) \to \mathbf{T}'$: On input a tag $\mathbf{T}$ and randomness μ, output a randomized tag $\mathbf{T}' \in [\mathbf{T}]_{\mathcal{R}_\tau}$ (i.e., a new representative of tag).
$\mathsf{ChangRep}((\mathbf{M},\mathbf{N}),\sigma,\mathbf{T},(\mu,\upsilon)) \to (\sigma',\mathbf{T}')$: On input a representative $(\mathbf{M},\mathbf{N}) \in [(\mathbf{M},\mathbf{N})]_{\mathcal{R}_{\mathsf{TDH}}}$, $\mathbf{T} \in [\mathbf{T}]_{\mathcal{R}_\tau}$, signature σ and randomness (μ,υ), return a new signature $((\mathbf{M}',\mathbf{N}'),\mathbf{T}',\sigma')$, where $\mathbf{M}' = \mathbf{M}^{\mu\upsilon} \wedge \mathbf{N}' = \mathbf{N}^{\upsilon} \in [(\mathbf{M},\mathbf{N})]_{\mathcal{R}_{\mathsf{TDH}}}$ and $\mathbf{T}' \leftarrow \mathsf{ConvertTag}(\mathbf{T},\mu)$ are the new representatives and σ' is valid for $(\mathbf{M}',\mathbf{N}')$ and $[\mathbf{T}]_{\mathcal{R}_\tau}$.
This will also apply for a set representative $\mathbb{M}$ such that one can get a new set representative $\mathbb{M}'$ by scaling all message with the same (μ,υ).
$\mathsf{ConvertSK}(\mathsf{sk},\omega) \to \mathsf{sk}'$: On input a sk and key converter ω, output a new secret key sk'.
$\mathsf{ConvertVK}(\mathsf{vk},\omega) \to \mathsf{vk}'$: On input a vk and key converter ω, output a new public key vk'.

$\mathsf{ConvertSig}(\mathsf{vk}, \mathbf{T}, (\mathbf{M}, \mathbf{N}), \sigma, \omega) \rightarrow \sigma'$: On input a vk, message vector $(\mathbf{M}, \mathbf{N})$, signature with tag $(\sigma, \mathbf{T})$, and key converter ω, return a new signature σ' such that $\mathsf{Verify}(\mathsf{vk}', \mathbf{T}, (\mathbf{M}, \mathbf{N}), \sigma') = 1$, where $\mathsf{vk}' \leftarrow \mathsf{ConvertVK}(\mathsf{vk}, \omega)$.

The VerifyTag and VKeyGen are only used for the security game.

4.2 Security Definitions

Correctness. As usual we require that honest signatures verify as expected, but need to consider all the randomizations as well as the aggregation. We formalize this in Appendix B.1.

Unforgeability. The unforgeability game follows the unforgeability definition of AtoSa (see Def. 4). It is slightly modified to fit with our additional EQ relation (Def. 11), i.e., unforgeability is defined with respect to message classes and in addition need to check VerifyTag.

Definition 13 (Unforgeability). *An* ATMS *is unforgeable if for all PPT $\mathcal{A}$ having access to the oracle $\mathcal{O}^{\mathsf{Sign}()}$ there exists a negligible function ϵ s.t:* $\Pr[\mathsf{ExpUnf}_{\mathsf{ATMS},\mathcal{A}}(\lambda) = 1] \leq \epsilon(\lambda)$ *where the experiment* $\mathsf{ExpUnf}_{\mathsf{ATMS},\mathcal{A}}(\lambda)$ *is defined in Fig. 3 and Q is the set of queries that $\mathcal{A}$ has issued to $\mathcal{O}^{\mathsf{Sign}()}$.*

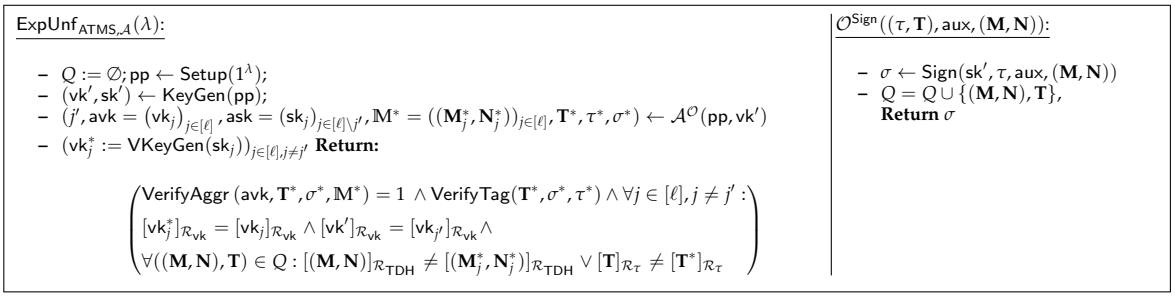


Fig. 3: Experiment $\mathsf{ExpUnf}_{\mathsf{ATMS},\mathcal{A}}(\lambda)$

Privacy guarantees. Similar as in Section 3, we consider the privacy notations *Origin-hiding of ConvertSig*, and *Public key class-hiding* (it is the same as Def. 5). We note that all definitions can be updated due to $\mathcal{M}_{\mathsf{TDH}}^H$ message space (receptively EQ relations of $\mathcal{M}_{\mathsf{TDH}}^H$) instead of the vector $\mathbf{M}$. Origin-hiding of ConvertSig definition can be updated straightforwardly as follows:

Definition 14 (Origin-hiding of ConvertSig **for** ATMS**).** *For all λ, and $\mathsf{pp} \in \mathsf{Setup}(1^\lambda)$, for all $(\mathsf{vk}, (\mathbf{M}, \mathbf{N}), \sigma, \mathbf{T})$, if $\mathsf{Verify}(\mathsf{vk}, \mathbf{T}, (\mathbf{M}, \mathbf{N}), \sigma) = 1$, and $(\omega, v, \mu) \in \mathbb{Z}_p^*$, then $\sigma' \leftarrow \mathsf{ChangRep}((\mathbf{M}, \mathbf{N}), \mathsf{ConvertSig}(\mathsf{vk}, \mathbf{T}, (\mathbf{M}, \mathbf{N}), \sigma, \omega), \mathbf{T}, (v, \mu))$ outputs a uniformly random in the respective spaces s.t. $\mathsf{Verify}(\mathsf{vk}', \mathbf{T}', (\mathbf{M}', \mathbf{N}'), \sigma') = 1$, where $\mathsf{vk}' \xleftarrow{\$} \mathsf{ConvertVK}(\mathsf{vk}, \omega)$ outputs a uniformly random element of $[\mathsf{vk}]_{\mathcal{R}_{\mathsf{vk}}}$.*

However, since this is a variant of SPSEQ we consider the *adaption property* similar to [39] below, an additional property which guarantees that signatures from ChangRep and Sign are identically distributed. This definition also covers Origin-hiding of ConvertTag.

Definition 15 (Perfect Adaption of Signatures). *An* ATMS *scheme perfectly adapts signatures if for all $(\mathsf{vk}, \mathbf{T}, (\mathbf{M}, \mathbf{N}), \sigma, (\mu, v))$ with $(\mathbf{M}, \mathbf{N}) \in \mathcal{M}_{\mathsf{TDH}}^H \wedge \mathsf{Verify}(\mathsf{vk}, \mathbf{T}, (\mathbf{M}, \mathbf{N}), \sigma) = 1 \wedge (\mu, v) \in \mathbb{Z}_p^*$ we have that the output of $(\sigma', \mathbf{T}') \leftarrow \mathsf{ChangRep}(\sigma, (\mathbf{M}, \mathbf{N}), \mathbf{T}, (\mu, v))$ is a uniformly random element in the respective space, conditioned on $\mathsf{Verify}(\mathsf{vk}, \mathbf{T}^\mu, (\mathbf{M}^{\mu v}, \mathbf{N}^v), \sigma') = 1$.*

4.3 Construction

Our construction is inspired by the message-indexed SPS by Crites et al. [31] (see Def. 2.1), which is a variant of Ghadafi's SPS [41] (see Def. 2.1). We use the tag-based message definition $\mathcal{M}_{\mathsf{TDH}}^H$ (Def. 10) instead of the message-indexed (Def. 1). For simplicity, we assume a message vector with the length $k = 2$ as $(\mathbf{M}, \mathbf{N}) = ((M_1, M_2), (N_1, N_2))$, but this can be straightforwardly generalized to any length $k > 1$. Similar to the construction in Section 3.3, we again need aux binding to make this particular construction work.

Definition 16 (Aux binding for ATMs). *We split* aux *into a preimage and an opening:* (c, o). *For all PPT* $\mathcal{A}$, *and* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ *and* $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{VKeyGen}(1^\lambda)$ *there exists a negligible* ϵ *such that:*

$$\Pr\left[\begin{array}{l}(\mathsf{aux} = (c,o), \mathsf{aux} = (c',o'), \tau, (\mathbf{M},\mathbf{N}), \tau', (\mathbf{M}',\mathbf{N}')) \leftarrow \mathcal{A}(\mathsf{vk});\\ \mathsf{VerifyAux}(\mathsf{sk}, (c,o), \tau, (\mathbf{M},\mathbf{N})) = 1\\ \wedge\, \mathsf{VerifyAux}(\mathsf{sk}, (c',o'), \tau', (\mathbf{M}',\mathbf{N}')) = 1 \wedge\\ c = c' \wedge (\tau \neq \tau' \vee (\mathbf{M},\mathbf{N}) \neq (\mathbf{M}',\mathbf{N}'))\end{array}\right] \leq \epsilon(\lambda)$$

Synchronicity assumption. Same as in Section 3.3, instead of fixing messages and verification keys in aux, we can make same assumption as in synchronized aggregate signatures and simply set $c = P^{\rho_1}||P^{\rho_2}$ in the construction below and Definition 9 is trivially satisfied.

Our construction. The construction is as follows:

$\mathsf{Setup}(1^\lambda)$: Run $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, \hat{P}, e) \leftarrow \mathsf{BGGen}(1^\lambda)$ with a prime number order p, where P a generator of $\mathbb{G}_1$, $\hat{P}$ a generator of $\mathbb{G}_2$ and H a hash function: $H : \{0,1\}^* \rightarrow \mathbb{G}_1$, output $\mathsf{pp} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, \hat{P}, H)$.

$\mathsf{KeyGen}(\mathsf{pp})$: Given pp, sample $\mathsf{sk} = (x, y_1, y_2, z_1, z_2) \xleftarrow{\$} (\mathbb{Z}_p^*)^5$, and $\mathsf{vk} = (\hat{X} = \hat{P}^x, \hat{Y}_1 = \hat{P}^{y_1}, \hat{Y}_2 = \hat{P}^{y_2}, \hat{Z}_1 = \hat{P}^{z_1}, \hat{Z}_2 = \hat{P}^{z_2})$.

$\mathsf{VKeyGen}\,(\mathsf{sk})$: Given $\mathsf{sk} = (x, y_1, y_2, z_1, z_2)$, return $\mathsf{vk} = (\hat{X} = \hat{P}^x, \hat{Y}_1 = \hat{P}^{y_1}, \hat{Y}_2 = \hat{P}^{y_2}, \hat{Z}_1 = \hat{P}^{z_1}, \hat{Z}_2 = \hat{P}^{z_2})$.

$\mathsf{GenAuxTag}(S)$: Given a set $S = \{(\mathbf{M}_j, \mathbf{N}_j, \mathsf{vk}_j)_{j\in[n]}\}$, choose $(\rho_1, \rho_2) \xleftarrow{\$} \mathbb{Z}_p$, set $\tau = (\rho_1, \rho_2), \mathbf{T} = (T_1 = h^{\rho_1}, T_2 = h^{\rho_2})$, and $c = \left(P^{\rho_1}||P^{\rho_2}||(\mathbf{N}_j, \mathsf{vk}_j)_{j\in[n]}\right)$, where $h = H(c)$ and $\mathsf{aux}_j = (c, o = \perp)$.

$\mathsf{VerifyAux}(\mathsf{sk}, \mathsf{aux}, (\tau_1, \tau_2), ((M_1, M_2), (N_1, N_2)))$: Extract (T_1, T_2), parse aux as (c, o). Check that $((\mathbf{M},\mathbf{N}), [\mathsf{VKeyGen}(\mathsf{sk})]) \in \mathsf{aux}$ (i.e., $c = ...||((\mathbf{M},\mathbf{N}), [\mathsf{VKeyGen}(\mathsf{sk})]||...)$ s.t no other vk in aux related to sk and check that $(T_1, T_2) = (h^{\tau_1}, h^{\tau_2})$. Compute $h := H(c)$. Output $\bigwedge_{i=1}^{2} e(M_i, \hat{P}) = e(h^{\tau_i}, N_i)$.

$\mathsf{Sign}\left(\mathsf{sk}_j, \tau, \mathsf{aux}_j, (\mathbf{M},\mathbf{N})\right)$: Given a sk_j, $\tau, \mathsf{aux}_j = (c, \perp)$, and message $(\mathbf{M},\mathbf{N}) = ((M_1, M_2), (N_1, N_2)) \in \mathcal{M}_{\mathsf{TDH}}^H$. Parse τ as (ρ_1, ρ_2). Run $\mathsf{VerifyAux}(\mathsf{sk}, \mathsf{aux}, \tau, (\mathbf{M},\mathbf{N}))$ and verify that this outputs 1. If so compute $h = H(c)$ and output a signature as:

$$\sigma = (h, b = \prod_{j\in[2]} h^{\rho_j \cdot z_j}, s = (h^x \cdot \prod_{j\in[2]} M_j^{y_j})).$$

$\mathsf{Verify}(\mathsf{vk}, \mathbf{T}, (\mathbf{M},\mathbf{N}), \sigma)$: Given a vk, tag $\mathbf{T} = (T_1 = h^{\rho_1}, T_2 = h^{\rho_2})$, message $(\mathbf{M},\mathbf{N})$ and signature $\sigma = (h, b, s)$ return 1 if the following holds and 0 otherwise:

$$e(h, \hat{X}) \prod_{j\in[2]} e(M_j, \hat{Y}_j) = e(s, \hat{P}) \;\wedge\; e(b, \hat{P}) = \prod_{j\in[2]} e(T_j, \hat{Z}_j)$$
$$\bigwedge_{j=1}^{2} e(T_j, N_j) = e(M_j, \hat{P})$$

$\mathsf{VerifyTag}(\mathbf{T}, \tau, \sigma)$: Given $\tau = (\tau_1, \tau_2)$, $\sigma = (h, b, s)$, output 1 if $T_i = h^{\tau_i}$ for all $i \in \{1, 2\}$, and 0 otherwise.

$\mathsf{AggrSign}(\mathbf{T}, (\mathsf{vk}_i, (\mathbf{M}_i, \mathbf{N}_i), \sigma_i)_{i=1}^{\ell})$: Given ℓ valid signatures $\sigma_i = (h, b_i, s_i)$ for $(\mathbf{M}_i, \mathbf{N}_i)$ under vk_i and the same tag $\mathbf{T}$ for $i \in [\ell]$, return $\perp$ if all h are not the same, else output a signature σ on the messages $\mathbb{M} = ((\mathbf{M}_i, \mathbf{N}_i))_{i\in[\ell]}$ under the tag $\mathbf{T}$ and aggregated verification key $\mathsf{avk} = (\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell)$ as follows: $\sigma = \left(h, b' = \prod_{i=1}^{\ell} b_i, s' = \prod_{i=1}^{\ell} s_i\right)$.

$\mathsf{VerifyAggr}(\mathsf{avk}, \mathbf{T}, \mathbb{M}, \sigma)$: Given $\mathsf{avk} = (\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell)$, tag $\mathbf{T} = (T_1 = h^{\rho_1}, T_2 = h^{\rho_2})$, messages $\mathbb{M}$ and signature $\sigma = (h, b, s)$, check if the following checks holds and 0 otherwise:

$$\prod_{i\in[\ell]} e(h, \hat{X}_i) \prod_{j\in[2]} e(M_{ij}, \hat{Y}_{ij}) = e(s, \hat{P}) \;\wedge\; e(b, \hat{P}) = \prod_{i\in[\ell]} \prod_{j\in[2]} e(T_j, \hat{Z}_{ij})$$
$$\bigwedge_{j\in[2]\wedge i\in[\ell]} e(T_j, N_{ij}) = e(M_{ij}, \hat{P})$$

$\mathsf{ConvertTag}(\mathbf{T}, \mu) \to \mathbf{T}'$: On input a tag $\mathbf{T}$ and randomness μ, output a randomized tag $\mathbf{T}' = (h^{\rho_1\mu}, h^{\rho_2\mu})$.

$\mathsf{ChangRep}(\sigma, (\mathbf{M}, \mathbf{N}), \mathbf{T}, (\mu, v))$: On input a representative $(\mathbf{M}, \mathbf{N}) \in [(\mathbf{M}, \mathbf{N})]_{\mathcal{R}_{\mathsf{TDH}}}$, $\mathbf{T} \in [\mathbf{T}]_{\mathcal{R}_\tau}$, signature $\sigma = (h, b, s)$, and $(\mu, v) \in (\mathbb{Z}_p^*)^2$, output:

$$\sigma' = \left(h' \leftarrow h^{\mu v}, b' \leftarrow b^{\mu}, s' \leftarrow s^{\mu v}, \mathbf{T}' \leftarrow \mathsf{ConvertTag}(\mathbf{T}, \mu)\right),$$

which is a valid signature for new representatives $(\mathbf{M}^{\mu v} = \mathbf{M}', \mathbf{N}^{v} = \mathbf{N}') \in [(\mathbf{M}, \mathbf{N})]_{\mathcal{R}_{\mathsf{TDH}}}$ and $\mathbf{T}' = (h^{\rho_1\mu}, h^{\rho_2\mu}) \in [\mathbf{T}]_{\mathcal{R}_\tau}$.

$\mathsf{ConvertSK}(\mathsf{sk}, \omega) \to \mathsf{sk}'$: On input a sk and key converter $\omega \in \mathbb{Z}_p^*$, output a new secret key as $\mathsf{sk}' = \mathsf{sk} \cdot \omega$.

$\mathsf{ConvertVK}(\mathsf{vk}, \omega) \to \mathsf{vk}'$: On input a vk and key converter $\omega \in \mathbb{Z}_p^*$, output $\mathsf{vk}' = \mathsf{vk}^{\omega} = (\hat{X}^{\omega}, \hat{Y}_1^{\omega}, \hat{Y}_2^{\omega}, \hat{Z}_1^{\omega}, \hat{Z}_2^{\omega})$.

$\mathsf{ConvertSig}(\mathsf{vk}, (\mathbf{M}, \mathbf{N}), \sigma, \mathbf{T}, \omega) \to \sigma'$: On input a vk, message $(\mathbf{M}, \mathbf{N})$, signature σ with tag $\mathbf{T}$, and key converter $\omega \in \mathbb{Z}_p^*$, returns a new signature σ' as: $\sigma' = (h, b^{\omega}, s^{\omega})$.

Note that one can reduce the number of paring operations in VerifyAggr by using batching verification techniques (cf. [36]).

Theorem 3 (Privacy). *Our construction is origin-hiding of ConvertSig (Def. 7), public key class-hiding (Def. 5), and provides perfect adaption of signatures (Def. 15).*

The proof of Theorem 3 is provided in Appendix D.3.

Theorem 4 (Unforgeability). *Our construction is EUF-CMA secure regarding the definition 13 in the generic group model for Type-III bilinear groups.*

The proof of Theorem 4 is provided in Appendix D.6.

5 Application to AC

As our core application we present Issuer-Hiding Multi-Authority Anonymous Credentials (IhMA). In a multi-authority setting [45], credentials come from ℓ-different credential issuers. Naively, the showing of credentials requires ℓ-independent credentials to be shown. This can be overcome [45] by leveraging aggregate signatures, obtaining a compact AC system with compact-size credentials, and showing costs. However, verifying a user's credentials needs knowledge of all issuers' verification keys, which might violate user privacy. Thus, in the vein of [8] we introduce the issuer-hiding property for multi-authority credentials. We recall that here the verifier can define a set of acceptable issuers in an ad-hoc manner. Then a user can prove that the subset of credentials shown were issued by acceptable issuers without revealing which credential corresponds to which issuer. This is an important feature, especially in multi-authority settings where disclosing issuer keys can reveal too much information compared to a single issuer setting and already lead to identification of the user.

5.1 Formal Definition

Our definition supports multiple users $(u_j)_{j\in[\ell]}$ and multiple credential issuers $(\mathsf{CI}_j)_{j\in[\ell]}$. An issuer can generate a key pair of secret and verification keys $(i\mathsf{sk}, i\mathsf{vk})$ via $\mathsf{IKeyGen}()$. Similarly, users runs $\mathsf{UKeyGen}()$ to generate a user key pair $(u\mathsf{sk}, u\mathsf{vk})$. Each issuer can then issue a credential (cred) on an attribute (a) or attribute-set (A) to a user who can verify the received credential locally. Indeed, when we use AtoSa, we consider an attribute a (i.e., the attribute set includes only one attribute); when we use ATMS, we consider an attribute set, A. We use the notation A, to define security and formal definitions for consistency of definitions.

Users can then use the $\mathsf{CredAggr}$ algorithm to aggregate all credentials and create a single credential valid for all attributes and verification keys. To define the set of accepted issuers, a verifier generates a key-policy *pol* using $\mathsf{GenPolicies}$ (it is known as Presentation policies in [8]), which can be checked for well-formedness by everyone. Finally, with an aggregate credential (disclosing a subset attributes D) and some key-policy *pol* from the verifier, a user uses Show to derive a proof, which a verifier can verify.

Definition 17 (Issuer-Hiding Multi-Authority Credentials (IhMA)). *An* IhMA *is defined by the following algorithms/protocols:*

- Setup: On input a security parameter λ, output public parameters pp (implicit input to all algorithms) .
- IKeyGen: Generate a key pair $(i\mathsf{sk}, i\mathsf{vk})$ for an issuer i.
- UKeyGen: Take a message-key set S, generate a user key pair $(u\mathsf{sk}, u\mathsf{vk})$ which acts as user's identity and auxiliary data aux.
- Issuance: In this protocol, an issuer i associated to $(i\mathsf{sk}, i\mathsf{vk})$ creates a credential cred on an attributes-set A to a user u associated to $(u\mathsf{sk}, u\mathsf{vk})$ as follows:

$$[\mathsf{CredObtain}(u\mathsf{sk}, i\mathsf{vk}, \mathsf{A}) \leftrightarrow \mathsf{CredIssue}(i\mathsf{sk}, u\mathsf{vk}, \mathsf{A})] \rightarrow \mathsf{cred}$$

- CredAggr: Take as input a $u$sk of user and a list of credentials $(i\mathsf{vk}, \mathsf{A}_i, \mathsf{cred}_i)$ for $i \in [\ell]$ and output an aggregated credential cred of attributes-set $\{\mathsf{A}_i\}_{i\in[\ell]}$:

$$\mathsf{CredAggr}\left(u\mathsf{sk}, \{(i\mathsf{vk}, \mathsf{A}_i, \mathsf{cred}_i)\}_{i\in[\ell]}\right) \rightarrow \mathsf{cred}$$

- GenPolicies: A verifier with the secret key $v$sk can define policies defining sets of issuers $\{\mathsf{ivk}\}_{i\in[n]}$ they are willing to accept for certain Show sessions, we have:

$$\mathsf{GenPolicy}(v\mathsf{sk}, \{\mathsf{ivk}\}_{i\in[n]}) \rightarrow pol, \text{ where } n \leq \ell$$

  Note that *pol* defines the sets of accepted issuers by a verifier, but not which attributes a verifier needs to disclose. Thus, *pol* can be reused for multiple contexts, reducing the number of policies.
- Show: In this protocol, a user u with $(u\mathsf{sk}, u\mathsf{vk})$ runs CredShow and interacts with a verifier running CredVerify to prove that she owns a valid credential cred on disclosed attribute sets $D \subseteq \{\mathsf{A}_i\}_{i\in[\ell]}$ issued respectively by one or some credential issuers in *pol*:

$$\begin{bmatrix}\mathsf{CredShow}(u\mathsf{sk}, pol, \{(i\mathsf{vk}, \mathsf{A}_i)\}_{i\in[\ell]}, \mathsf{cred}, D) \leftrightarrow \\ \mathsf{CredVerify}(pol, (i\mathsf{vk}_i)_{i\in[\ell]}, D)\end{bmatrix} \rightarrow (0, 1)$$

5.2 Security Definitions

We define our security model based on the game-based framework in [39, 45], with some modifications to harmonize their definition with the one on IhMA and consider the use of multi-authority and issuer-hiding properties. The adversary $\mathcal{A}$ has access to the following oracles that describe the possible ways to interact with the IhMA. Moreover, we define some global lists that are shared among oracles as $\mathcal{HU}$ a list of honest users and $\mathcal{CU}$ a list of corrupted users, similarly we define $\mathcal{HCI}$ and $\mathcal{CCI}$ for the honest/corrupted credential issuers. Also, $\mathcal{L}_{uk}$ stands for a list of user's keys and $\mathcal{L}_{\mathsf{cred}}$ which is a list of user-credential pairs which includes issued credentials and corresponding attributes and to which user they were issued. A credential in $\mathcal{L}_{\mathsf{cred}}$ can be empty ($\perp$) if the user has not received a credential on this attribute yet. For simplicity we assume a tag τ includes aux as well.

- $\mathcal{O}^{\mathsf{HCI}}(i)$: Create an honest credential issuer with identity i. If i already exists (i.e. $i \in \mathcal{HCI} \cup \mathcal{CCI}$), output $\perp$. Otherwise, run $(i\mathsf{sk}, i\mathsf{vk}) \leftarrow \mathsf{IKeyGen}(i)$, add $(i, i\mathsf{sk}, i\mathsf{vk}) \in \mathcal{HCI}$, and return $i$vk.
- $\mathcal{O}^{\mathsf{CorruptCI}}(i)$: Corrupt a credential issuer i. If i does not exist yet (i.e. $i \notin \mathcal{HCI} \cup \mathcal{CCI}$), create a new corrupted credential issuer by adding i to $\mathcal{CCI}$. Otherwise, if $i \in \mathcal{HCI}$, remove i from $\mathcal{HCI}$, add it to $\mathcal{CCI}$ and output $i$sk. Note that $\mathcal{A}$ does not allow to corrupt the challenge key vk′.
- $\mathcal{O}^{\mathsf{User}}(u, S)$: Take as input a user identity u and issuer/attributes pairs $\{(a_i, \mathsf{vk}_i)\} \in S$. If $u \in \mathcal{HU}$ or $u \in \mathcal{CU}$, return $\perp$. Else, create a fresh entry u by running $(u\mathsf{sk}, u\mathsf{vk}, \mathsf{aux}) \leftarrow \mathsf{UKeyGen}$ and adding u and $(u\mathsf{sk}, u\mathsf{vk}, \mathsf{aux})$ to the list $\mathcal{HU}$ and $\mathcal{L}_{uk}$, receptively. Then, for each $(a_i, \mathsf{vk}_i) \in S$, add $\mathcal{L}_S[i]$. Return $u$vk.
- $\mathcal{O}^{\mathsf{CorruptU}}(u)$: Take as input a user identity u and (optionally) a user public key $u$vk. If $u \notin \mathcal{HU}$, register a new corrupt user with public key $u$vk and add $u \in \mathcal{CU}$. Else, move the entry corresponding to u from $\mathcal{HU}$ and add it to $\mathcal{CU}$, output $u$sk and all the associated credentials items $(u, \mathsf{A}_i, \mathsf{cred}_i)$ of $\mathcal{L}_{\mathsf{cred}}[u]$.

– $\mathcal{O}^{\mathsf{ObtIss}}(u, i, \mathsf{A}_i)$: (Perform an honest issuing/obtaining) Take as input a user identity u, issuer identity i, and attribute(s) A_i. If $u \notin \mathcal{HU} \vee i \notin \mathcal{HCI}$, return $\perp$. Else, find entries $(u\mathsf{sk} \in \mathcal{L}_{uk}, i\mathsf{sk} \in \mathcal{HCI})$, and run the issuing protocols:

$$[\mathsf{CredObtain}(u\mathsf{sk}, i\mathsf{vk}, \mathsf{A}_i) \leftrightarrow \mathsf{CredIssue}(i\mathsf{sk}, u\mathsf{vk}, \mathsf{A}_i)] \rightarrow \mathsf{cred}_i$$

and add the entry $(u, \mathsf{A}_i, \mathsf{cred}_i)$ to $\mathcal{L}_{\mathsf{cred}}$, where cred_i includes aux_i.

– $\mathcal{O}^{\mathsf{Obtain}}(u, i, \mathsf{A}_i)$: (Perform an honest obtaining of a credential with a malicious issuer) On input a user identity $u \in \mathcal{HU}$, issuer identity $i \in \mathcal{CCI}$ and attributes A_i. If $u \notin \mathcal{HU} \vee i \notin \mathcal{CCI}$, return $\perp$. Else, find $u\mathsf{sk} \in \mathcal{L}_{uk}$, and run the Obtain protocol with $\mathcal{A}$:

$$[\mathsf{CredObtain}(u\mathsf{sk}, i\mathsf{vk}, \mathsf{A}_i) \leftrightarrow \mathcal{A}] \rightarrow \mathsf{cred}_i$$

If $\mathsf{cred}_i = \perp$, return $\perp$. Else, append the resulting output $(u, \mathsf{A}_i, \mathsf{cred}_i)$ to $\mathcal{L}_{\mathsf{cred}}$. This oracle is used by $\mathcal{A}$, whom it knows by $i\mathsf{vk}$ impersonating an issuer to issue a credential to an honest user u .

– $\mathcal{O}^{\mathsf{Issue}}(u, i, \mathsf{A}_i)$: (Perform a malicious obtaining of a credential with an honest issuer) On input a user identity $u \in \mathcal{CU}$, issuer $i \in \mathcal{HCI}$, and attributes A_i. If $u \notin \mathcal{CU} \vee i \notin \mathcal{HCI}$, return $\perp$. Else, find entries $i\mathsf{sk} \in \mathcal{HCI}$, and run Issuing with $\mathcal{A}$:

$$[\mathcal{A} \leftrightarrow \mathsf{CredIssue}(i\mathsf{sk}, u\mathsf{vk}, \mathsf{A}_i)] \rightarrow \mathsf{cred}_i$$

Append elements $(u, \mathsf{A}_i, \mathsf{cred}_i)$ to $\mathcal{L}_{\mathsf{cred}}$. This oracle is used by a corrupted user u to get a credential from a honest issuer.

– $\mathcal{O}^{\mathsf{CredShow}}(j, pol, D)$: On input an index of an issuance j, key-policy pol and attributes-subset D. First parses $\mathcal{L}_{\mathsf{cred}}[j]$ as $(u, \mathsf{A}_i, \mathsf{cred}_i)$, where cred_i is the credential issued by an issuer $i\mathsf{vk}$ on A_i for a user u during the i-th query to $\mathcal{O}^{\mathsf{ObtIss}}$ or $\mathcal{O}^{\mathsf{Obtain}}$. If $i \notin \mathcal{HU}$ return $\perp$. Else, run: $\mathsf{CredShow}(u\mathsf{sk}, pol, \{(i\mathsf{vk}, \mathsf{A}_i)\}_{i\in[\ell]}, \mathsf{cred}, D) \leftrightarrow \mathcal{A}$, with the adversary playing the role of verifier.

$(\mathcal{O}^{\mathsf{Obtain}})$ and $(\mathcal{O}^{\mathsf{Issue}})$ are defined specifically for the anonymity ExpAno and the unforgeability ExpUnf, respectively. The other oracles are common between ExpAno and ExpUnf.

Correctness We require that honestly issued credentials shown to honest verifiers always verify with a caveat. If a user does not specify a particular issuer and attribute when User is called, then if that issuer is called to issue that attribute to the user, it is allowed to fail. I.e.: the user must include pairs for every attribute they wish to receive. Further, if the user specifies two attributes for the same issuer, we allow the scheme to return $\perp$ during issuing. This limitation can be overcome practically by having each issuer use a different key for each attribute.

Unforgeability. Unforgeability requires that no adversary can convince a verifier to accept a credential for a set of attributes for which he does not possess all the individual credentials from the accepted issuers set $I = \{i\mathsf{vk}\}_{i\in[\ell]}$. $\mathcal{A}$ can obtain $i\mathsf{vk} \in I$ using $\mathcal{O}^{\mathsf{HCI}}(i)$ and $\mathcal{O}^{\mathsf{CorruptCI}}(i)$. Intuitively, an adversary wins the unforgeability experiment if he is able to convince an honest verifier that he satisfies a certain policy while does not have an appropriate credential. To make the game non-trivial, we impose restrictions that for all corrupted users the disclosed attributes subset D should not pass verification (satisfy attributes credentials).

Definition 18 (Unforgeability). *An* IhMA *is unforgeable if, for all* $\lambda \in \mathbb{N}$ *and for any* PPT *adversary* $\mathcal{A}$*, there exists* $\epsilon(\lambda)$ *s.t* $\Pr[\mathsf{ExpUnf}_{\mathsf{IhMA},\mathcal{A}}(\lambda) = 1] \leq \mathsf{ffl}(\lambda)$*, experiments are defined in Fig 4, where* A_{ui} *means the (set) attribute issued by the issuer* $i\mathsf{vk}$ *to* u*.*

Anonymity. Anonymity requires that a malicious verifier cannot distinguish between two users. Thus we allow the adversary to output two sets of credentials, attributes, as well as a key-policy pol, attribute subset D, and issuers' public keys (can be corrupted). The adversary has adaptive access to an oracle that on the input of two distinct user indexes j_0 and j_1, acts as one of the two credential owners (depending on bit b) in the verification. To make the game non-trivial, we impose restrictions that the subset D is either satisfied or not by both credentials, i.e., $D(\mathsf{A}) = 1 \Rightarrow D \subseteq \cup_{i\in[\ell]}\mathsf{A}_i$ if attributes in A satisfy D and $D(\mathsf{A}) = 0 \Rightarrow D \not\subseteq \cup_{i\in[\ell]}\mathsf{A}_i$ otherwise. The essence of the game is captured by the oracles $\mathcal{O}_b^{\mathsf{Anon}}$ in Fig 5.

Definition 19 (Anonymity). *An* IhMA *is anonymous, if for* $\lambda \in \mathbb{N}$*, any* PPT *adversary* $\mathcal{A}$ *there exists a negligible function* $\epsilon(\lambda)$ *so that* $|\Pr[\mathsf{ExpAno}^0_{\mathsf{IhMA},\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{ExpAno}^1_{\mathsf{IhMA},\mathcal{A}}(\lambda) = 1]| \leq \frac{1}{2} + \epsilon(\lambda)$*, experiments are defined in Fig 5, respectively.*

$\underline{\mathsf{ExpUnf}_{\mathsf{IhMA},\mathcal{A}}(\lambda)}$:

- $(\mathsf{pp}) \leftarrow \mathsf{Setup}(1^\lambda)$; $I \leftarrow \mathcal{A}^{<\mathcal{O}^{\mathsf{CorruptCI}},\mathcal{O}^{\mathsf{HCI}}>}(\mathsf{pp})$;
- $(\mathsf{sk}', \mathsf{vk}') \leftarrow \mathsf{IKeyGen}(\mathsf{pp})$; $I' = I \cup \mathsf{vk}'$;
- $\mathsf{pol} \leftarrow \mathsf{GenPolicies}(I')$;
- $(D, \mathsf{avk} = (i\mathsf{vk})_{i\in[\ell]}) \leftarrow \mathcal{A}^{\langle\mathcal{O}\rangle}(pol, \mathsf{vk}')$;
- $b \leftarrow (\mathcal{A} \leftrightarrow \mathsf{CredVerify}(pol, (i\mathsf{vk})_{i\in[\ell]}, D))$
- **if** $b = 1 \wedge \exists i \in [\ell] : [i\mathsf{vk}] = [\mathsf{vk}'] \ \wedge \ (i\mathsf{vk})_{i\in[\ell]} \subset I' \ \wedge D \not\subset \bigcup_{i\in[\ell]} A_{ui}$, $\forall u \in \mathcal{CU}$
  return 1
- else **return** 0

Fig. 4: Experiment $\mathsf{ExpUnf}_{\mathsf{IhMA},\mathcal{A}}(\lambda)$

$\underline{\mathsf{ExpAno}^b_{\mathsf{IhMA},\mathcal{A}}(\lambda)}$:	$\underline{\mathcal{O}^{\mathsf{Anon}}_b(j_0, j_1, D, pol)}$:
– $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ – $(j_0, j_1, pol, (i\mathsf{vk})_{i\in[\ell]}) \leftarrow \mathcal{A}^{\langle\mathcal{O}\rangle}(\mathsf{pp})$ – $b' \leftarrow \mathcal{A}^{\langle\mathcal{O}^{\mathsf{Anon}}_b,\mathcal{O}\rangle}(\mathsf{st})$ – **return** $(b = b')$	– If j_0 or $j_1 > \lvert\mathcal{L}_{\mathsf{cred}}\rvert$, return $\perp$. – Else, parse $\mathcal{L}_{\mathsf{cred}}[j_0]$ as $(u_0, \mathsf{A}_{0i}, \mathsf{cred}_{0i})_{i\in[\ell]}$ and $\mathcal{L}_{\mathsf{cred}}[j_1]$ as $(u_1, \mathsf{A}_{1i}, \mathsf{cred}_{1i})_{i\in[\ell]}$, such that $\forall i$, $\mathsf{cred}_{bi} \leftarrow [\mathsf{CredObtain}(u\mathsf{sk}_b, i\mathsf{vk}, \mathsf{A}_{bi}) \leftrightarrow \mathsf{CredIssue}(i\mathsf{sk}, u\mathsf{vk}_b, \mathsf{A}_{bi})]$ – If $D(\mathsf{A}_0) \neq D(\mathsf{A}_1) \vee (u_0, u_1) \notin \mathcal{HU}$, return $\perp$. – Otherwise run: $\mathsf{CredShow}(u\mathsf{sk}_b, pol, \{(i\mathsf{vk}, \mathsf{A}_{bi})\}_{i\in[\ell]}, \mathsf{cred}_b, D) \leftrightarrow \mathcal{A}$, where $\mathsf{cred}_b \leftarrow \mathsf{CredAggr}\left(u\mathsf{sk}_b, \{(i\mathsf{vk}, \mathsf{A}_{bi}, \mathsf{cred}_{bi})\}_{i\in[\ell]}\right)$

Fig. 5: Experiment $\mathsf{ExpAno}_{\mathsf{IhMA},\mathcal{A}}(\lambda)$

Issuer-hiding. Issuer-hiding indicates that a malicious verifier cannot distinguish if verification key(s) belongs to the credential's issuer. We allow the adversary to output a set of issuers, as well as a key-policy *pol*. We consider key policies of the form $\{\sigma_i, i\mathsf{vk}\}_{i\in[n]}$, where σ_i is a signature on a given issuer's public key $i\mathsf{vk}$ produced by the verifier. As a result, users can prove the correspondence between a verification key (defined in the key-policy) and the credential verification under that verification key. Note that we assume honest issuers in this definition.

Definition 20 (Issuer-Hiding). *An* IhMA *supports issuer-hiding, if for all* $\lambda \in \mathbb{N}, \ell > 1$, *for all* $D \not\subseteq \cup_{i\in[\ell]}\mathsf{A}_i$ *and* $(\mathsf{pp}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$ *for any* PPT *adversary* $\mathcal{A}$, *there exists a negligible function* $\epsilon(\lambda)$ *s.t:*

$$\Pr\left[\begin{array}{l}(i\mathsf{sk}, i\mathsf{vk})_{i\in[\ell]} \xleftarrow{\$} \mathsf{IKeyGen}(\mathsf{pp});\\ (I_0, I_1, pol, D) \xleftarrow{\$} \mathcal{A}^{\langle\mathcal{O}\rangle}(\mathsf{pp}, i\mathsf{vk}_{i\in[\ell]});\\ (u\mathsf{sk}, u\mathsf{vk}) \xleftarrow{\$} \mathsf{UKeyGen}(\mathsf{pp}); b \xleftarrow{\$} \{0,1\};\\ \forall i\mathsf{vk} \in I_b : (\mathsf{cred}_{b,i}, st) \xleftarrow{\$} \mathsf{CredObtain}(u\mathsf{sk}, i\mathsf{vk}, \mathsf{A}_i) \leftrightarrow : \quad b^* = b\\ \mathsf{CredIssue}(i\mathsf{sk}, u\mathsf{vk}, \mathsf{A}_i);\\ \mathsf{cred}_b \leftarrow \mathsf{CredAggr}\left(u\mathsf{sk}, \{(i\mathsf{vk}, \mathsf{A}_i, \mathsf{cred}_{b,i})\}_{i\in[\ell]}\right)\\ b^* \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{CredShow}}}(pol, I_b, D, \mathsf{A}_{i\in[\ell]})\end{array}\right] \leq \epsilon(\lambda)$$

where both $|I_0| = |I_1|$ *and* $I_0, I_1 \subseteq i\mathsf{vk}_{i\in[\ell]}$ *are one or a set of selected issuer(s).*

5.3 Constructions

Now we are ready to describes our two constructions of IhMA, the first being based on AtoSa (Def. 3) and SPSEQ [39] and the second based on ATMS (Def. 12), a set commitment scheme SC (Def. 2.2), and SPSEQ. To enhance users' privacy and prevent issuers from learning attributes issued by other issuers, we change how aux for the signatures is computed. In particular, we commit to the attributes (messages) instead of including them in plaintext. For example, this can be achieved using a hash-based commitment scheme, where a commitment value c is generated by computing $c := H'(a, r)$ with H' being a hash function modeled as a random oracle, a being the attributing being committed

to, and r a sufficiently large random value. When issuing a credential, users can reveal the relevant message (attribute) a, the opening o, and the commitment value c. The signer then verifies if the c is correct for a and o before issuing the corresponding credential. We modify $\mathsf{GenAuxTag(S)}$ and VerifyAux in AtoSa and ATMS as follows:

- $\mathsf{GenAuxTag(S)}$: Given $S = \{(m_j, \mathsf{vk}_j)_{j\in[\ell]}\}$, choose $(\rho_1, \rho_2) \xleftarrow{\$} \mathbb{Z}_p$, set $c = P^{\rho_1}||P^{\rho_2}||(c_{m_j}||\mathsf{vk}_j)_{j\in[\ell]}$, where c_{m_j} is a hash commitment to j'th message and all vk are distinct. Output $\mathsf{aux} = (c, o_j)$ and tag $\tau = ((\rho_1, \rho_2), (T_1 = h^{\rho_1}, T_2 = h^{\rho_2}))$ with $h = H(c)$.
- $\mathsf{VerifyAux}(\mathsf{sk}, \mathsf{aux}, \tau, m_j)$ Parse aux as (c, o). Check that $\tau \in t$ (i.e., that c has the form: $P^{\rho_1}||P^{\rho_2}||$ ...) check that c_j exists such that $(c_j, \mathsf{vk}) \in t$ and $\mathsf{Open}(c_j, o, m_j) = 1$ where vk is a verification key related to sk (in the same equivalence class). Also check that no other vk in aux has the same equivalence class as sk.

In our IhMA schemes, tags are user identities and are used to verify the user before issuing attributes.

AtoSa based IhMA Construction in Fig. 6. Here, every issuer creates a credential (signature) σ_{1i} on an attribute a_i for the user u with tag τ (and the respective aux) verified with $i$vk by the AtoSa scheme. We cannot reveal the secret part of the tag to signers (issuers) as this would violate the security of the user. To obtain a credential through the Issuing protocols, a user is required to disclose the public parts of tag as identity to the issuer and then authenticate their identity via a ZKPOK.

Interactive signing. We can adapt the signing in a way that signers (issuers) don't learn (ρ_1, ρ_2) as follows:

- u sends $(\mathsf{aux}, (h, \mathbf{T}), \pi)$, where $\mathsf{aux} = P^{\rho_1}||P^{\rho_2}||(c_{m_i}, \mathsf{vk}_i)_{i\in[n]}$ and $\pi = \mathsf{ZKPOK}\{(\rho_1, \rho_2) : T_1 = h^{\rho_1} \wedge T_2 = h^{\rho_2} \wedge u_1 = P^{\rho_1} \wedge u_2 = P^{\rho_2}\}$.
- Signer (issuer) checks if proof π is valid and if so outputs $(h' = h^{\rho_1}, s = (h^{\rho_1})^{x_j + y_{1j}\cdot m_j} \cdot (h^{\rho_2})^{y_{2j}})$

We note that this interactive signing outputs signatures that are identical to that output by Sign and this is used in Issuance.

For the Show protocol, we assume that verifier(s) have signed all accepted issuer keys using an SPSEQ scheme [39]. A user u can take pol and the set of disclosed credentials D, aggregates the respective credentials (signatures) and randomizes the aggregated signature and tag. We note that alternatively, a user could already after Issuance aggregate all credentials to a constant-size (single) credential and then in Show protocol can provide a ZKPOK of the signature and selectively disclose the required attributes (as originally done for PS signatures in [57]). This also yields constant size credentials as noted in Table 1. We stick with the former approach here as it is more efficient for showing credentials, but one can easily switch to the other option. Moreover, In $\mathsf{IhMA_{AtoSa}}$, only one attribute per vk can be issued. However, if an issuer needs to issue multiple attributes, they can easily generate multiple vks.

To hide the issuer's keys, u randomizes them using a random ω and adapts the signature for these randomized keys using ConvertSig. So far, we have created a compact randomized credential (proof) for attributes in D where issuer verification keys of this signature are hidden. The next step is to show that these random verification keys correspond to those keys signed by the verifier (using SPSEQ signatures) in pol. In this direction, u first collects signatures in pol according to issuer keys that are needed in the proof. Then u runs ChangRep of SPSEQ to randomize messages (which are issuer public keys) and signatures with the same randomness ω used in convert, i.e., randomized keys. Randomized issuer keys in a credential match with the messages signed by verifier in pol. Finally, u uses the randomized tag as a pseudonym for communication and provides a ZKPOK of secret part of tag (secret keys and randomness) used in the credentials.

ATMS based IhMA Construction in Fig. 7. We use the framework in [39] in which one can combine mercurial or SPSEQ with a set commitment such that a credential is a signature on set commitment SC. One can then open a subset of messages from this commitment while randomizing both set commitment and signature together. This provides unlinkability and selective disclosure at the same time (see [39]). Unlike the previous construction, we can aggregate credentials immediately after receiving them and have a constant-size credential but still avoid zero-knowledge proof of a

- $\mathsf{Setup}(1^\lambda)$: Run $\mathsf{pp}_{\mathsf{AtoSa}} \leftarrow \Sigma_1.\mathsf{Setup}(1^\lambda) \wedge \mathsf{pp}_{\mathsf{SPSEQ}} \leftarrow \Sigma_2.\mathsf{Setup}(1^\lambda)$, output $\mathsf{pp} = (\mathsf{pp}_{\mathsf{AtoSa}}, \mathsf{pp}_{\mathsf{SPSEQ}})$. The attribute space is $\mathbb{Z}_p$.
- $\mathsf{UKeyGen}(\mathsf{pp}, \mathsf{S})$: Run $(\{\mathsf{aux}_j\}, (\tau, \mathbf{T})) \leftarrow \mathsf{GenAuxTag}(\mathsf{pp}, S)$, and return $(u\mathsf{sk} = \tau, u\mathsf{vk} = \mathbf{T}, \{\mathsf{aux}_j\})$ to u.
- $\mathsf{IKeyGen}(\mathsf{pp})$: Generate $(\mathsf{sk}, \mathsf{vk}) \xleftarrow{\$} \Sigma_1.\mathsf{KeyGen}(\mathsf{pp})$, return $(i\mathsf{sk} = \mathsf{sk}, i\mathsf{vk} = \mathsf{vk})$ to an issuer i.
- Issuance: On input $(\mathbf{T}, \mathsf{aux}_i, a_i)$, u and each issuer i act as follows for an attribute a_i:
  - u sends $(\mathbf{T}, \mathsf{aux}_i, \pi)$, to an issuer i, where π is a zero knowledge proof that the user knows the secret part of the given tag.
  - Issuer checks π is valid and runs $\sigma_i \leftarrow \Sigma_1.\mathsf{Sign}(i\mathsf{sk}, \mathbf{T}, \mathsf{aux}_i, a_i)$ and outputs (σ_i, a_i) to u or aborts if Sign outputs $\perp$.
  - u takes $(i\mathsf{vk}, \mathsf{cred}_i = (a_i, \sigma_i))_{i\in[\ell]}$, checks $\Sigma_1.\mathsf{Verify}(i\mathsf{vk}, a_i, \mathsf{cred}_i)_{i\in[\ell]}$, and saves $\mathsf{cred} = \{\mathsf{cred}_i = (\sigma_i, \tau), \mathsf{A}\}_{i\in[\ell]}$, where $\mathsf{A} = (a_i)_{i\in[\ell]}$.
- Gen-Policies: Generate a key pair $(v\mathsf{sk}, v\mathsf{pk}) \leftarrow \Sigma_2.\mathsf{KeyGen}(\mathsf{pp})$, run $\sigma_{2i} \leftarrow \Sigma_2.\mathsf{Sign}(v\mathsf{sk}, i\mathsf{vk})$ for $i \in I$ where $i\mathsf{vk}$ is a message vector for SPSEQ, return $pol = (v\mathsf{vk}, (i\mathsf{vk}, \sigma_{2i})_{i\in[I]})$.
- Show: On input $\mathsf{cred} = \{(\sigma_i, \tau, \mathsf{A})_{i\in[\ell]}\}$, $pol = (v\mathsf{vk}, (i\mathsf{vk}, \sigma_{2i})_{i\in[I]})$, an D (a set of attributes) from $n \subseteq I$ issuers ($|D| = n$), u prepares a proof for D as:
  1. Run $\sigma \leftarrow \Sigma_1.\mathsf{AggrSign}(\mathbf{T}, (i\mathsf{vk}, a_i, \sigma_i))_{i\in[D]}$ with $\mathsf{avk} = \{i\mathsf{vk}\}_{i\in[D]}$. For $\omega \in \mathbb{Z}_p^*$, run $\mathsf{avk}' \leftarrow \Sigma_1.\mathsf{ConvertVK}(\mathsf{avk}, \omega)$, $\sigma' \leftarrow \Sigma_1.\mathsf{ConvertSig}(\mathsf{avk}, D, \mathbf{T}, \sigma, \omega)$, and randomize $(\sigma'', \mathbf{T}') \leftarrow \Sigma_1.\mathsf{RandSign}(\mathsf{vk}, \mathbf{T}, m, \sigma', v)$ for $v \in \mathbb{Z}_p^*$.
  2. Run $(\sigma'_{2i}, \mathsf{avk}') \xleftarrow{\$} \Sigma_2.\mathsf{ChangRep}(\mathbf{M}_i = \mathsf{vk}_i, \sigma_{2i}, \omega)_{i\in[n]}$ where avk' is the same as $\mathsf{avk}' \leftarrow \Sigma_1.\mathsf{ConvertVK}$.
  3. Prove in zero knowledge that the user knows the secret key for the tag $\mathbf{T}'$, yielding π, send $(\sigma'', \mathsf{Nym} = \mathbf{T}', \sigma'_{2i}, \pi)_{i\in[n]}$ to a verifier V.
- CredVerify: Output 1, if $\pi \wedge \Sigma_1.\mathsf{VerifyAggr}(\mathsf{avk}', \mathbf{T}', D, \sigma') \wedge \Sigma_2.\mathsf{Verify}(v\mathsf{vk}, \mathbf{M}, \sigma'_2) = 1$, where $\mathbf{M} = \mathsf{avk}'$ and $\mathbf{T}' = \mathsf{Nym}$. Output 0 if this check fails.

Fig. 6: Our IhMA scheme (Σ_1 and Σ_2 denote AtoSa and SPSEQ [39], respectively)

signature in showing protocol (because of compatibility of EQ message relation of ATMS and SC randomization).

In the Show protocol, similar to the previous construction, u first collects the signatures required to prove the attributes D from pol. Then, for issuer-hiding similar to AtoSa it randomizes these SPSEQ signatures using ChangRep of SPSEQ with ω. For preparing a proof for D, a user (u) randomizes issuer verification keys in credentials using ConvertVK and converts the ATMS signature using ConvertSig with ω. Subsequently, u randomizes the signature with a tag using ChangRep. Finally, u opens a subset of attributes D from the set commitments. Now a verifier can check if these attributes are in the set commitments signed by some issuers in pol. Same as in the first construction, since all issuer keys are randomized due to the SPSEQ signature the issuers are hidden. We run a ZKPOK to prove that u knows all secret values related to the randomized tag like before. The only point left is the signing of set commitments, which is defined in one source group in [39], but we need both groups. Subsequently, we show how one can combine set commitments with a tag-based DH message space.

Set commitments for $\mathcal{M}^H_{\mathsf{TDH}}$. The main point here is that we need to convert the set commitments space to $\mathcal{M}^H_{\mathsf{TDH}}$, which can be smoothly done as follows: In addition to credentials issuers, we also define a Trusted Authority TA who holds the trapdoor α of the set commitment scheme and can create commitments for the attributes of users who want to register in the system. WLOG, let us for simplicity assume only one attribute set $\mathsf{A} = (\mathsf{A}, \eta)$, where we have a fixed constant η which is never opened in practice and it is the same for all (it is just required for anonymity). It works as follows:

- The user sends a tag $\mathbf{T}$ and aux to TA.
- TA computes a set commitment in both groups $(\mathbf{C} = (C_1, C_2), \hat{\mathbf{C}} = (\hat{C}_1, \hat{C}_2))$ (i.e., $(\mathbf{M}, \mathbf{N})$) with tag, where $(C_2, \hat{C}_2)$ are dummy commitments for a fixed constant η and the other one for the (real) attribute set A. More precisely: TA computes the commitment in $\mathbb{G}_1$ to base h^{ρ_i} and the one in $\mathbb{G}_2$ in base $\hat{P}$: $C_1 = (h^{f_{\mathsf{A}}(\alpha)})^{\rho_1}$, $\hat{C}_1 = \hat{P}^{f_{\mathsf{A}}(\alpha)}$, $C_2 = (h^{\eta})^{\rho_2}$ and $\hat{C}_2 = \hat{P}^{\eta}$ such that such that we have $\bigwedge_{i\in[2]} e(T_i, \hat{C}_i) = e(C_i, \hat{P})$, where $h = H(c)$, $\mathsf{aux} = (c, o)$, $c = P^{\rho_1}||P^{\rho_2}||(c_{\mathsf{A}_i}||\mathsf{vk}_j)_{j\in[2]}$, returns $(\mathbf{C}, \hat{\mathbf{C}})$. Note that $c_{\mathsf{A}} := H'(\mathsf{A}, r)$.

Note that α is a trapdoor kept by TA, but TA does not need to know (ρ_1, ρ_2) (e.g., C_i be computed as $(T_1)^{f_{\mathsf{A}}(\alpha)}$). A multiparty computation protocol can also be used to hide other user details from

TA. A user can first randomize set commitment exactly like our tag-based message with (μ, v) as $(\mathbf{C}^{\mu v}, \hat{\mathbf{C}}^{v})$ and use v as opening information to open any subset values from $\hat{C}_1$ and still verify as follows: verifying the OpenSubset works $e(P, \hat{C}_1) = e(P^{f_D(\alpha)}, W)$. Consequently, we do not need any fundamental change on SC construction, and it works as stated in 2.2. In our construction, we make it explicit as:

– $\mathsf{SC.Commit}_3(\mathsf{A}, \alpha, \mathbf{T}, h) \rightarrow ((\mathbf{C}, \hat{\mathbf{C}}), O)$: On input a set $\mathsf{A} = (\mathsf{A}, \eta)$, $\mathbf{T}$ and h, compute a commitment: $C_1 = (T_1^{f_\mathsf{A}(\alpha)})$, $\hat{C}_1 = \hat{P}^{f_\mathsf{A}(\alpha)}$, $C_2 = (T_2^{\eta})$ and $\hat{C}_2 = \hat{P}^{\eta}$, output $((\mathbf{C}, \hat{\mathbf{C}}), O)$ with $O \leftarrow \perp$.

Now, we can use the same technique as AtoSa to not reveal (ρ_1, ρ_2) to issuers when signing the above commitments $(\mathbf{C}, \hat{\mathbf{C}})$ as follows:

Interactive signing. We can adapt the signing in a way that signers (issuers) don't learn (ρ_1, ρ_2) as follows:

– u sends $(\mathsf{aux}, \mathbf{T}, (\mathbf{C}, \hat{\mathbf{C}}), \pi)$, where
$\pi = \mathsf{ZKPOK}\{(\rho_1, \rho_2) : T_1 = h^{\rho_1} \wedge T_2 = h^{\rho_2} \wedge u_1 = P^{\rho_1} \wedge u_2 = P^{\rho_2}\}$, where P^{ρ_1} and P^{ρ_2} are in aux.
– Signer (issuer) checks if proof π is valid and if so outputs
$(h = H(c), b = \prod T_i^{z_i}, s = (h^x \cdot \prod_{i \in [2]} (C_i)^{y_i}))$.

Again we note that this interactive signing outputs signatures that are identical to that output by Sign and this is used in Issuance.

Achieving constant-size credentials. This can be achieved by following these steps: 1) Users can obtain the (h^{α_i}) values from the TA instead of the commitments. 2) During the issuing phase, users can aggregate all the credentials received from issuers. 3) The commitments can then be recomputed using randomness and the obtained information, eliminating the need to store them. Note that in this case the size of the $|\mathbf{Show}|$ operation will become linear with respect to N instead of K.

Theorem 5. *The above* IhMA *constructions in Fig. 7 and in Fig. 6 are correct, unforgeable, anonymous, and issuer-hiding.*

The proof is presented in Appendix D.4.
To prove the anonymity of ATMS, we need to define a variant of the uber assumption in the next section.

5.4 Uber Assumption

In the anonymity proof of ATMS, we need to claim that two credentials are not distinguishable. Because we manipulate polynomials, a natural approach would be to use the uber-decisional assumption defined by Boyen in [13]. This definition uses the Real-Or-Random paradigm, so we would like to use this theorem to say that a first credential is indistinguishable from random elements and then that these random elements are indistinguishable from a second credential. However, both credentials have an internal structure contrary to the random elements. Indeed these credentials should be publicly checkable. This shows the limit of the Real-Or-Random paradigm and the fact that it seems more accurate to apply the Left-Or-Right paradigm.

Thus, we present a more general definition based on this more powerful paradigm. We give a first attempt in the Figure 8. Similarly to the Real-Or-Random's definition (and also the computational definitions of the uber assumption), the uber problems will often be trivially solvable. So we need to characterize the parameters for which it is not a hard problem. Thus, we need the following definitions.

Definition 21. *Let $\mathbb{F}$ be a field. Let $\mathbf{R} \in \mathbb{F}(X_1, \ldots, X_m)^r$ and $W \in \mathbb{F}(X_1, \ldots, X_m)$. We say that W is* linearly dependent *on $\mathbf{R}$ if there exist coefficients $\{a_i\}_{i=1}^r \in \mathbb{F}^m$ such that*

$$W = \sum_{i=1}^{r} a_i R_i.$$

We say that W is (linearly) independent from $\mathbf{R}$ if it is not linearly dependent on $\mathbf{R}$.

Here, we define a generalization of the definition of [13].

- $\mathsf{Setup}(1^\lambda)$: Run $\mathsf{pp}_{\mathsf{ATMS}} \leftarrow \Sigma_1.\mathsf{Setup}(1^\lambda) \wedge \mathsf{pp}_{\mathsf{SPSEQ}} \leftarrow \Sigma_2.\mathsf{Setup}(1^\lambda) \wedge \mathsf{pp}_{\mathsf{SC}} \leftarrow \mathsf{SC.Setup}$, output $\mathsf{pp} = (\mathsf{pp}_{\mathsf{ATMS}}, \mathsf{pp}_{\mathsf{SPSEQ}}, \mathsf{pp}_{\mathsf{SC}})$.
- $\mathsf{IKeyGen}(\mathsf{pp})$: Generate $(\mathsf{sk}, \mathsf{vk}) \xleftarrow{\$} \Sigma_1.\mathsf{KeyGen}(\mathsf{pp})$, return $(i\mathsf{sk} = \mathsf{sk}, i\mathsf{vk} = \mathsf{vk})$ to an issuer i.
- $\mathsf{UKeyGen}(\mathsf{pp}, S)$: Run $((\tau, \mathbf{T}), \mathsf{aux}) \leftarrow \mathsf{GenAuxTag}(S)$, and return $(u\mathsf{sk} = \tau, u\mathsf{vk} = \mathbf{T})$ to u.
  Then, TA and u interact to computes $((\hat{\mathbf{C}}_i, \mathbf{C}_i)_{i\in[\ell]}) \leftarrow \mathsf{SC.Commit}_3(\mathsf{A}_i, \alpha, \mathbf{T})$, for all attribute sets.
- Issuance: The interaction between an issuer i and a user u for one attribute-set $\mathsf{A} \in \mathbb{Z}_\mathsf{p}$ and $(\mathbf{C}, \hat{\mathbf{C}})$ acts as follows:
  - u hands over $(\mathbf{T}, (\mathbf{C}, \hat{\mathbf{C}}), \mathsf{aux}_i, \pi)$ to an issuer i, where π is zero knowledge proof the secret parts of the tag.
  - An issuer i checks that the proof is correct, then runs $\sigma \leftarrow \Sigma_1.\mathsf{Sign}(i\mathsf{sk}, \mathbf{T}, \mathsf{aux}_i, (\mathbf{C}, \hat{\mathbf{C}}))$, and outputs $(\mathsf{A}, \mathbf{T}, \sigma) = \mathsf{cred}_i$.
  - u takes $(i\mathsf{vk}, \mathsf{cred}_i)$ for $i \in [\ell]$, checks $\Sigma_1.\mathsf{Verify}(i\mathsf{vk}, \mathbf{T}, (\mathbf{C}_i, \hat{\mathbf{C}}_i), \sigma_i)_{i\in[\ell]} = 1$, and outputs $\{\mathsf{cred} = (\sigma_i, \tau), (\mathsf{A}_i, \mathbf{C}_i, \hat{\mathbf{C}}_i)_{i\in[\ell]}\}$.
- Gen-Policies: Generate a key pair $(v\mathsf{sk}, v\mathsf{pk}) \leftarrow \Sigma_2.\mathsf{KeyGen}(\mathsf{pp})$, run $\sigma_{2i} \leftarrow \Sigma_2.\mathsf{Sign}(v\mathsf{sk}, i\mathsf{vk})$ for $i \in I$, return $pol = (v\mathsf{vk}, (i\mathsf{vk}, \sigma_{2i})_{i\in[I]})$.
- Show: On input $\mathsf{cred} = \{(\sigma_i, u\mathsf{sk}, \mathsf{A}_i)_{i\in[\ell]}\}$, $pol = (v\mathsf{vk}, (i\mathsf{vk}, \sigma_{2i})_{i\in[I]})$, and $D \subseteq \mathsf{A}$ from $n \subseteq I$ issuers, u prepares a proof for D as:
  1. Run $(\sigma'_{2i}, \mathsf{avk}') \leftarrow \Sigma_2.\mathsf{ChangRep}(\mathbf{M}_i = \mathsf{vk}_i, \sigma_{2i}, \omega)_{i\in[n]}$ for $\omega \in \mathbb{Z}_p^*$.
  2. Run $\sigma \leftarrow \Sigma_1.\mathsf{AggrSign}(\mathbf{T}, (i\mathsf{vk}, (\mathbf{C}_i, \hat{\mathbf{C}}_i), \sigma_i))_{i\in[n]}$. Convert credentials and issuer keys $\mathsf{avk}' \leftarrow \Sigma_1.\mathsf{ConvertVK}(\mathsf{avk}, \omega)$ and $\sigma' \leftarrow \Sigma_1.\mathsf{ConvertSig}(\mathsf{avk}, (\mathbf{C}, \hat{\mathbf{C}}), \sigma, \mathbf{T}, \omega)$.
  3. Run $(\sigma', \mathbf{T}') \xleftarrow{\$} \Sigma_1.\mathsf{ChangRep}(\sigma, (\mathbf{M}_i, \mathbf{N}_i)_{i\in[n]}, \mathbf{T}, (\mu, \upsilon))$ for (μ, υ), where $(\mathbf{M}_i, \mathbf{N}_i) = (\mathbf{C}_i, \hat{\mathbf{C}}_i)$, and σ' is valid for $(\mathbf{C}'_i = \mathbf{C}_i^{\mu\upsilon}, \hat{\mathbf{C}}'_i = \hat{\mathbf{C}}_i^{\upsilon})_{i\in[n]}$. Create witnesses for attributes $W_j \leftarrow \mathsf{SC.OpenSubset}(\hat{C}_{1j}, A_j, O_j, d_j)$ for $j \wedge d_j \in D$. Aggregate witness $W \leftarrow \mathsf{SC.AggregateAcross}(\{\hat{C}_{1j}, d_j, W_j\}_{j\in[\ell]})$, randomize $W' \leftarrow W^{\mu\upsilon}$.
  4. Prove in zero knowledge that the user knows the secret key for the tag $\mathbf{T}'$, yielding π, send $(\sigma', W', \mathbf{T}', \sigma'_{2i}, \pi, \mathbb{M} = \{(\mathbf{C}'_i, \hat{\mathbf{C}}'_i)\})_{i\in[n]}$ to V.
- CredVerify: Output 1, if $\pi \wedge \Sigma_1.\mathsf{VerifyAggr}(\mathsf{avk}', \mathbf{T}', \mathbb{M}, \sigma') \wedge \Sigma_2.\mathsf{Verify}(v\mathsf{vk}, \mathbf{M}, \sigma'_2) \wedge \mathsf{SC.VerifySubset}(\mathbf{C}', D, W') = 1$, where $\mathbf{M} = \mathsf{avk}'$ is verified by $v\mathsf{vk}$.

Fig. 7: Our IhMA scheme (Σ_1 and Σ_2 denote ATMS and SPSEQ [39], respectively)

Definition 22. *Let $\mathbb{F}$ be a field. Let $\mathbf{R}$, $\mathbf{S}$, $\mathbf{F}$ be vectors of rational functions from $\mathbb{F}(X_1, \ldots, X_m)$ of length r, s, f respectively. We call a* type-1 ("bilinearly") dependant-vector *on $(\mathbf{R}, \mathbf{S}, \mathbf{F})$ a vector $\left((a_{i,k})_{1\le i\le r, 1\le k\le s}, (b_{i,j})_{1\le i\le j\le r} (c_{k,l})_{1\le k\le l\le s}, (d_k)_{1\le k\le f}\right)$ in $\mathbb{F}^{rs+\frac{r(r+1)+s(s+1)}{2}}$ such that*

$$0 = \sum_{i=1}^{r}\sum_{j=1}^{s} a_{i,j}R_iS_j + \sum_{i=1}^{r}\sum_{j=i}^{r} b_{i,j}R_iR_j + \sum_{i=1}^{s}\sum_{j=i}^{s} c_{i,j}S_iS_j + \sum_{k=1}^{f} d_kF_k.$$

We say it is also a type 2 dependent-vector if all the $b_{i,j} = 0$ for all i, j and a type 3 dependent vector if $b_{i,j} = c_{i,j} = 0$ for all i, j. For all $\tau \in \{1, 2, 3\}$, we define the τ-dependant set as the set of all τ-dependant vectors.

Definition 23. *Let* Game *be a decisional game parametrized by a group groupdes. And $\mathcal{A}$ be a PPT adversary. We define the advantage of $\mathcal{A}$ in this game of such an adversary by:*

$$\mathcal{A}\left(\mathsf{Game}_{\mathcal{G}}^{\mathcal{A}}\right) = \left|\Pr\left(\mathsf{Game}_{\mathcal{G}}^{\mathcal{A}} = 1\right) - \frac{1}{2}\right|. \tag{1}$$

Definition 24. *Let $\mathbb{F}$ be a field. Let $(\mathbf{R}, \mathbf{R}')$, $(\mathbf{S}, \mathbf{S}')$, $(\mathbf{F}, \mathbf{F}')$ be pair of vectors of rational functions from $\mathbb{F}(X_1, \ldots, X_m)$ of length r, s, f respectively. Let $\tau \in \{1, 2, 3\}$, we say $(\mathbf{R}, \mathbf{S}, \mathbf{F})$, $(\mathbf{R}', \mathbf{S}', \mathbf{F}')$ are τ-trivially distinguishable, if their τ-dependent sets are distinct.*

In our context, a static assumption is in fact not enough. Then we need to define an adaptative version of the uber problems with the Figure 9. We prove this stronger version with the theorem 6 in the section D.5 for type 3. We claim, our result is easily generalizable for other types (but we only need type 3 in our case).

$(\mathbf{R}^0, \mathbf{S}^0, \mathbf{F}^0, \mathbf{R}^1, \mathbf{S}^1, \mathbf{F}^1)$-**uber**$^{\mathcal{A}}_{\mathcal{G}}$:

- $(g_1, g_2) \leftarrow \mathsf{BG}\,;\, g_T \leftarrow e(g_1, g_2)$
- $\mathbf{x} := (x_1, \ldots, x_m) \xleftarrow{\$} \mathbb{Z}_p^m$
- $b \xleftarrow{\$} \{0,1\}$
- $\mathbf{U} := \left(g_1^{R_1^b(\mathbf{x})}, \ldots, g_1^{R_r^b(\mathbf{x})}\right)$
- $\mathbf{V} := \left(g_2^{S_1^b(\mathbf{x})}, \ldots, g_2^{S_s^b(\mathbf{x})}\right)$
- $\mathbf{W} := \left(g_T^{F_1^b(\mathbf{x})}, \ldots, g_T^{F_f^b(\mathbf{x})}\right)$
- $b' \xleftarrow{\$} \mathcal{A}(\mathbf{U}, \mathbf{V}, \mathbf{W})$
- $\mathit{Return}\big(b = b'\big)$

Fig. 8: Game for the uber assumption relatively to bilinear group $\mathcal{G}$ and adversary $\mathcal{A}$, parametrized by (vectors of) or polynomials $\mathbf{R}^0, \mathbf{S}^0, \mathbf{F}^0, \mathbf{R}^1, \mathbf{S}^1$ and $\mathbf{F}^1$

Uberinteractive$^{\mathcal{A}}_{\mathcal{G}}$	$\mathcal{O}(t \in \{1,2,T\}, R^0, R^1)$:
– $(g_1, g_2) \leftarrow \mathsf{BG}\,;\, g_T \leftarrow e(g_1, g_2)$ – $b \xleftarrow{\$} \{0,1\}$ – $\mathbf{x} = (x_1, \ldots, x_m) \xleftarrow{\$} \mathbb{Z}_p^m$ – $(\mathbf{R}_1^0, \mathbf{R}_2^0, \mathbf{R}_T^0, \mathbf{R}_1^1, \mathbf{R}_2^1, \mathbf{R}_T^1) := ([1], [1], [], [1], [1], [])$ – $b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(g_1, g_2)$ – If $(\mathbf{R}_1^0, \mathbf{R}_2^0, \mathbf{R}_T^0, \mathbf{R}_1^1, \mathbf{R}_2^1, \mathbf{R}_T^1)$ is τ-trivial: – Return a random bit – Else $\mathit{Return}\big(b = b'\big)$	– $\forall b' \in \{0,1\} : \mathbf{R}_t^{b'} := \mathbf{R}_t^{b'} :: R^{b'}$ – $\mathit{Return}\big(g_t^{R^b(x_1,\ldots,x_n)}\big)$

Fig. 9: Adaptative game for the uber assumption relatively to the bilinear group $\mathcal{G}$ and adversary $\mathcal{A}$.

Theorem 6. *Let $\mathcal{A}$ be a adversary that solves decisional uber in a bilinear generic group $\mathcal{G}$ of prime order p of type 3, making at most (m_1, m_2, m_T, m_p) generic queries and (q_1, q_2, q_T) and (d_1, d_2, d_T) the cardinality, and the upper bound on the degrees of (R_1, R_2, R_T). Then*

$$\mathcal{A}\left(\mathit{uberdecisional}^{\mathcal{A}}_{\mathcal{G}}\right) \leq 2\left(\frac{d_1(q_1+m_1)^2 + d_2(q_2+m_2)^2 + \max(d_1+d_2, d_T)(q_T+m_T+m_p)^2}{p}\right).$$

The proof is presented in Appendix D.5.

5.5 Additional Properties

We now discuss how additional features can be obtained via slight modifications of the so far presented approach.

Blind issuing credential for AtoSa**.** We note that our schemes can provide a blind issuing protocol in which a user can receive credentials on blind attributes using the two-party computation technique provided in PS and Coconut [63]. It works as follows:

- A user generates an El-Gamal key-pair $(d_1, D_1 = P_1^d)$; pick a random k and compute an El-Gamal encryption of m as below: $c = \mathsf{Enc}(m) = (a = P^k, b = P^{k \cdot d} \cdot (h')^m)$. Output $(\mathsf{aux}, h, D, c, \pi)$, where π and $h' = h^{\rho_1}$ is defined by: $\pi = \mathsf{NIZK}\{(d, m, k) : D = P^d \wedge c = (P^k, D^k \cdot (h')^m)\}$
- A signer checks the π is correct, and generates blind signatures as follows: $\delta = (a_1' = a^{y_1} = P^{y_1 \cdot k}, b' = (h^{\rho_2})^{y_2} \cdot (h')^x \cdot b^{y_1} = P^{y_1 \cdot k \cdot d} \cdot (h')^{y_1 \cdot m + x} \cdot h^{y_2 \cdot \rho_2})$
- The user decrypts/unblinds signature $\delta = (a', b')$ and gets $h'^{y_1 \cdot m + x} \cdot (h^{\rho_2})^{y_2}$ as follows: $(a')^d = P^{y_1 \cdot k \cdot d}$ and $(h')^{m \cdot y_1 + x} \cdot (h^{\rho_2})^{y_2} = \frac{b'}{(a')^d}$.

As we showed in the IhMA, one can also hide the tag.

Multi-Message Signatures for AtoSa. One can extend this scheme to sign a message vector $\mathbf{m}$ rather than a single m by extending a verification key $\mathsf{vk}_i = (\hat{X}_i, \hat{Y}_{i1}, \ldots, \hat{Y}_{in})$ regarding the number of messages. A signer i can sign a vector $\mathbf{m} = (m_1, \ldots, m_n)$ as $h^{x_i + \sum y_i m_i}$ for $m_i \in \mathbf{m}$ (see [57] for more details).

Non-transferable Credentials. Often it is desirable to prevent users from easily sharing their credentials with others. One common approach to deter such transfers is to leverage a valuable item, such as a secret key, which would also need to be shared if a credential were to be shared [20]. We note that schemes involve users proving their knowledge of tag's secret that represents their identity. We can now use the canonical representative (ρ_1/ρ_2) of the respective tag class as the valuable secret. Then note that when sharing a credential even with a re-randomized tag $(v\rho_1, v\rho_2)$, one can extract the canonical representative and thus also shares the valuable secret. While this feature is important for ACs, however, it is not always easily achievable in all AC systems. In fact, achieving this feature can be quite challenging in some cases, especially in self-binding approaches such as SPSEQ or [29].

Proving Knowledge of AtoSa Signature. One can achieve the proving knowledge of a signature exactly similar to PS. Assume AtoSa signature is $\sigma = (h, s)$, we select random $r, t \leftarrow \mathbb{Z}_p$ and compute $\sigma' \leftarrow (h^r, (s \cdot h^t)^r)$. We send it to the verifier and carries out a zero-knowledge proof of knowledge π (a Schnorr proof) of (m, ρ_1, ρ_2) and t for the signature on a single message: $\mathsf{ZKPOK}\{(m, \rho_1, \rho_2, t) : e(h', \hat{X}) \cdot e(h', Y_1)^m \cdot e(h', Y_1)^{\rho_1} \cdot e(h', Y_2)^{\rho_2} \cdot e(h', \hat{P})^t = e(s', \hat{P})\}$. It can be extended straightforwardly for multi messages (see [57] for more details).

6 Implementation and Evaluation

In the following we present our evaluation based on a Python library in which we implement our primitives ATMS and AtoSa as well as our IhMA protocols (Fig. 7 and Fig. 6). Our implementation is based upon the `bplib` library[16] and petlib [17] with `OpenSSL` bindings[18]. We use the popular pairing friendly curve BN256 which provides efficient type 3 bilinear groups at a security level of around 100 bits. Our measurements have been performed on an Intel Core i5-6200U CPU at 2.30 GHz, 16 GB RAM running Ubuntu 20.04.3.

Benchmark of Primitives. Table 2 shows the mean of the execution time of each algorithm over 500 runs such that AggrSign and VerifyAggr are computed assuming two signers ($n = 2$); the other algorithms are independent of n. ChR/Rnd stands for ChangRep and signature randomization (RandSign) for the ATMS and AtoSa, respectively. PC stands for Pre-Computation, and in ATMS it includes converting messages to the $\mathcal{M}^H_{\mathsf{TDH}}$ message space and generating tags. While in AtoSa, PC includes generating tags and aux using Pedersen commitment, but note that one could also use a hash based commitment instead. We can observe that signing is faster than verifying the signature – due to the pairing operation in the latter. Moreover, verification of ATMS is slower than AtoSa because of additional pairing operations that are needed to check if messages are in $\mathcal{M}^H_{\mathsf{TDH}}$. We increase the number

Table 2: Running times for ATMS and AtoSa (ms)

	PC	Sign	Verify	Convert	ChR/Rnd	AggrSign	VerifyAggr
AtoSa	6	2,5	8,4	4	2,7	0.005	9
ATMS	8.6	3	33	5,4	7,4	0.01	72

(n) of signers from 2 to 10 and show the running time in Fig. 10. Since aggregation is almost free (for $n = 10$ is 0.05 ms), we omit it. We should also note that the result are stated without considering VerifyAux algorithm.

[16] https://github.com/gdanezis/bplib
[17] https://github.com/gdanezis/petlib
[18] https://github.com/dfaranha/OpenPairing

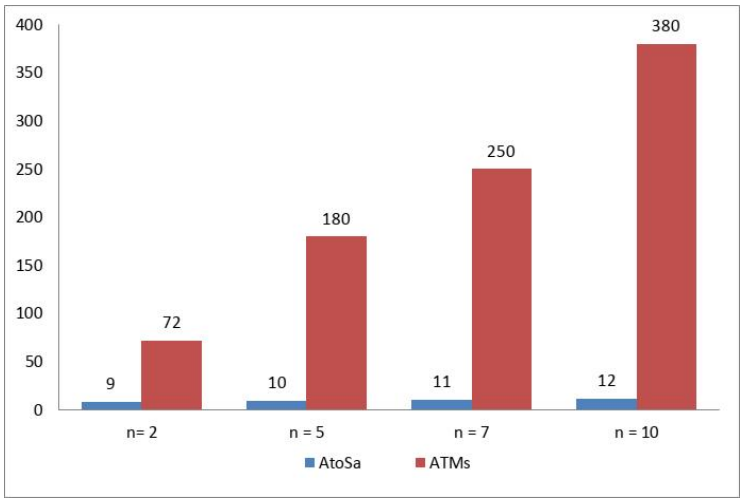
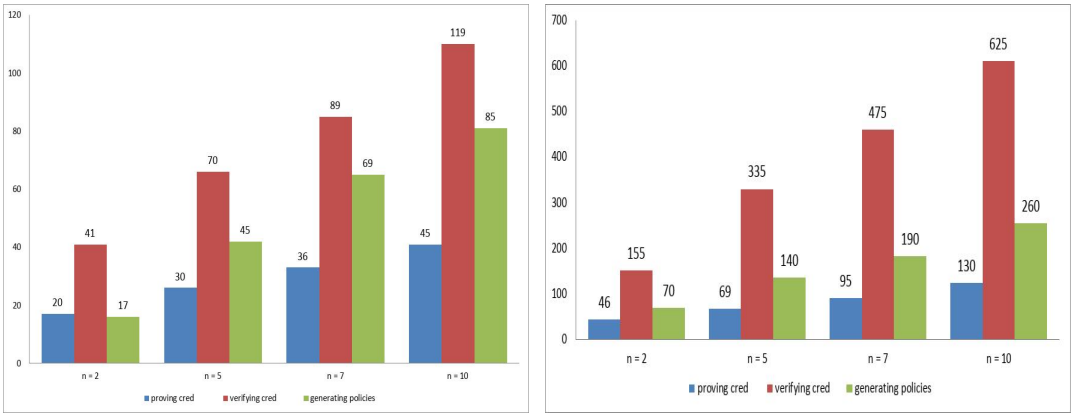


Fig. 10: Running times of VerifyAggr in ATMS & AtoSa (ms)

IhMA **Benchmarks**. IhMA is based upon Schnorr-style discrete logarithm ZKPOK. Our library supports Damgård's technique [30] for obtaining malicious-verifier interactive zero-knowledge proofs of knowledge during the showing and also NIZK obtained via the Fiat-Shamir heuristic. We interpret signers as issuers here and also show n as a number of issuers involved in Showing. For example, $n = 2$ means showing two credentials from 2 different issuers.

Issuing. This protocol does not depend on n, and results are as follows: 1) For IhMA based on AtoSa, including generation of signature, tag, user keys, and aux, it takes 8 ms. 2) For IhMA based on AtoSa, including generation of tag and encoding messages to $\mathcal{M}^{H}_{\mathsf{TDH}}$, with two attributes in each credential it takes 10 ms.

Showing. Fig. 11a shows the runtime of showing for IhMA based on AtoSa. In this experiment, we increase the number of issuers n from 2 to 10 and assume that all attributes are disclosed during verification (the worst-case scenario). Each issuer issues only one attribute, giving a total of n attributes. Fig. 11b shows the time for showing a credentials of IhMA based on ATMS. Here, we have a different setting; we can encode a set of attributes in a credential as we use set commitments. For our evaluation, we have the following parameters: n represents the number of the issuer, t the number of attributes in each set (each credential issued), $d < t$ is the number of disclosed attributes from each attribute set A in the respective commitment C. Here we increase n from 2 to 10, set $t = 2$, and $d = 1$. The total disclosed attributes length $|D| = d \cdot n$ and the total attribute $|A| = n \cdot t$ range from 2 to 10 and 4 to 20, respectively.



(a) Running times of $\mathsf{IhMA}_{\mathsf{AtoSa}}$

(b) Running times of $\mathsf{IhMA}_{\mathsf{ATMS}}$

Fig. 11: Running times of IhMA (ms)

7 Conclusion and Open Questions

This paper introduces the Issuer-Hiding Multi-Authority Anonymous Credentials (IhMA). **MA** means proving possession of attributes from multiple independent credential issuers requires the presentation of independent credentials. Meanwhile, **Ih** means verifying a user's credential does not require disclosing multiple issuers' public keys.

Our proposed solution involves the development of two new signature primitives with versatile randomization features which are independent of interest: 1) Aggregate Signatures with Randomizable Tags and Public Keys (AtoSa) and 2) Aggregate Mercurial Signatures (ATMS), which extend the functionality of AtoSa to support the randomization of messages additionally.

We formalize all notations and provide rigorous security definitions for our proposed primitives. We present provably secure and efficient instantiations of the two primitives and corresponding IhMA systems. Finally, we provide benchmarks based on implementation to demonstrate the practical efficiency of our constructions.

Open Questions and Future Work. Finally, we still have several open questions that merit further investigation. 1) An interesting open question is whether it is possible to present constructions in a fully dynamic setting, i.e., there are no assumptions about prior knowledge of messages and verification keys, nor requirement for a stateful issuer to keep track of the signed information aux. 2) Revocation is another intriguing avenue for future work. While issuer revocation in our scheme is straightforward, as revoked issuers can be excluded from the key policy, user revocation poses greater challenges. Finding effective methods for user revocation within our framework, and for issuer-hiding anonymous credentials in general, are an interesting future research direction.

Acknowledgements. We are very grateful to the anonymous reviewers for their many helpful comments and suggestions. This work has in part been carried out within the scope of Digidow, the Christian Doppler Laboratory for Private Digital Authentication in the Physical World and has partially been supported by the LIT Secure and Correct Systems Lab. Omid Mir gratefully acknowledge financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development, the Christian Doppler Research Association, 3 Banken IT GmbH, ekey biometric systems GmbH, Kepler Universitätsklinikum GmbH, NXP Semiconductors Austria GmbH and Co KG, Österreichische 24 Staatsdruckerei GmbH, and the State of Upper Austria. Daniel Slamanig was supported by the European Commission through ECSEL Joint Undertaking (JU) under grant agreement n°826610 (COMP4DRONES), the European Union through the Horizon Europe research programme under grant agreement n°101073821 (SUNRISE) and by the Austrian Science Fund (FWF) and netidee SCIENCE under grant agreement P31621-N38 (PROFET). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Executive Agency. Neither the European Union nor the granting authority can be held responsible for them. Anna Lysyanskaya and Scott Griffy are supported by NSF Awards 2247305, 2154941 and 2154170, as well as funding from the Peter G. Peterson Foundation and Meta.

References

1. Abe, M., Fuchsbauer, G., Groth, J., Haralambiev, K., Ohkubo, M.: Structure-preserving signatures and commitments to group elements. Journal of Cryptology 29(2), 363–421 (Apr 2016)
2. Ahn, J.H., Green, M., Hohenberger, S.: Synchronized aggregate signatures: new definitions, constructions and applications. In: Al-Shaer, E., Keromytis, A.D., Shmatikov, V. (eds.) ACM CCS 2010. pp. 473–484. ACM Press (Oct 2010)
3. Au, M.H., Susilo, W., Mu, Y.: Constant-size dynamic k-TAA. In: Prisco, R.D., Yung, M. (eds.) SCN 06. LNCS, vol. 4116, pp. 111–125. Springer, Heidelberg (Sep 2006)
4. Backes, M., Hanzlik, L., Kluczniak, K., Schneider, J.: Signatures with flexible public key: Introducing equivalence classes for public keys. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part II. LNCS, vol. 11273, pp. 405–434. Springer, Heidelberg (Dec 2018)
5. Baldimtsi, F., Lysyanskaya, A.: Anonymous credentials light. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013. pp. 1087–1098. ACM Press (Nov 2013)
6. Bellare, M., Namprempre, C., Neven, G.: Unrestricted aggregate signatures. In: Arge, L., Cachin, C., Jurdzinski, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 411–422. Springer, Heidelberg (Jul 2007)

7. Blömer, J., Bobolz, J.: Delegatable attribute-based anonymous credentials from dynamically malleable signatures. In: Preneel, B., Vercauteren, F. (eds.) ACNS 18. LNCS, vol. 10892, pp. 221–239. Springer, Heidelberg (Jul 2018)
8. Bobolz, J., Eidens, F., Krenn, S., Ramacher, S., Samelin, K.: Issuer-hiding attribute-based credentials. In: International Conference on Cryptology and Network Security. pp. 158–178. Springer (2021)
9. Bobolz, J., Eidens, F., Krenn, S., Ramacher, S., Samelin, K.: Issuer-hiding attribute-based credentials. Cryptology ePrint Archive, Report 2022/213 (2022), https://eprint.iacr.org/2022/213
10. Boldyreva, A., Gentry, C., O'Neill, A., Yum, D.H.: Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. Cryptology ePrint Archive, Report 2007/438 (2007), https://eprint.iacr.org/2007/438
11. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 416–432. Springer, Heidelberg (May 2003)
12. Bosk, D., Frey, D., Gestin, M., Piolle, G.: Hidden issuer anonymous credential. Proc. Priv. Enhancing Technol. 2022(4), 571–607 (2022), https://doi.org/10.56553/popets-2022-0123
13. Boyen, X.: The uber-assumption family (invited talk). In: Galbraith, S.D., Paterson, K.G. (eds.) PAIRING 2008. LNCS, vol. 5209, pp. 39–56. Springer, Heidelberg (Sep 2008)
14. Brands, S.: Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy. MIT Press, Cambridge-London (August 2000), http://www.credentica.com/the_mit_pressbook.html
15. Brickell, E.F., Camenisch, J., Chen, L.: Direct anonymous attestation. In: Atluri, V., Pfitzmann, B., McDaniel, P. (eds.) ACM CCS 2004. pp. 132–145. ACM Press (Oct 2004)
16. Brickell, E., Li, J.: Enhanced privacy ID: A direct anonymous attestation scheme with enhanced revocation capabilities. IEEE Trans. Dependable Secur. Comput. 9(3), 345–360 (2012), https://doi.org/10.1109/TDSC.2011.63
17. Camenisch, J., Chen, L., Drijvers, M., Lehmann, A., Novick, D., Urian, R.: One TPM to bind them all: Fixing TPM 2.0 for provably secure anonymous attestation. In: 2017 IEEE Symposium on Security and Privacy. pp. 901–920. IEEE Computer Society Press (May 2017)
18. Camenisch, J., Drijvers, M., Lehmann, A., Neven, G., Towa, P.: Short threshold dynamic group signatures. In: Galdi, C., Kolesnikov, V. (eds.) SCN 20. LNCS, vol. 12238, pp. 401–423. Springer, Heidelberg (Sep 2020)
19. Camenisch, J., Dubovitskaya, M., Haralambiev, K., Kohlweiss, M.: Composable and modular anonymous credentials: Definitions and practical constructions. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015, Part II. LNCS, vol. 9453, pp. 262–288. Springer, Heidelberg (Nov / Dec 2015)
20. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 93–118. Springer, Heidelberg (May 2001)
21. Camenisch, J., Lysyanskaya, A.: A signature scheme with efficient protocols. In: Cimato, S., Galdi, C., Persiano, G. (eds.) SCN 02. LNCS, vol. 2576, pp. 268–289. Springer, Heidelberg (Sep 2003)
22. Camenisch, J., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 56–72. Springer, Heidelberg (Aug 2004)
23. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups (extended abstract). In: Kaliski Jr., B.S. (ed.) CRYPTO'97. LNCS, vol. 1294, pp. 410–424. Springer, Heidelberg (Aug 1997)
24. Camenisch, J., Van Herreweghen, E.: Design and implementation of the idemix anonymous credential system. In: Atluri, V. (ed.) ACM CCS 2002. pp. 21–30. ACM Press (Nov 2002)
25. Chase, M., Perrin, T., Zaverucha, G.: The signal private group system and anonymous credentials supporting efficient verifiable encryption. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020. pp. 1445–1459. ACM Press (Nov 2020)
26. Chaum, D.: Security without identification: Transaction systems to make big brother obsolete. Commun. ACM 28(10), 1030–1044 (1985), https://doi.org/10.1145/4372.4373
27. Chaum, D.: Showing credentials without identification: Signatures transferred between unconditionally unlinkable pseudonyms. In: Pichler, F. (ed.) EUROCRYPT'85. LNCS, vol. 219, pp. 241–244. Springer, Heidelberg (Apr 1986)
28. Cini, V., Ramacher, S., Slamanig, D., Striecks, C., Tairi, E.: Updatable signatures and message authentication codes. In: Garay, J. (ed.) PKC 2021, Part I. LNCS, vol. 12710, pp. 691–723. Springer, Heidelberg (May 2021)
29. Connolly, A., Lafourcade, P., Perez Kempner, O.: Improved constructions of anonymous credentials from structure-preserving signatures on equivalence classes. In: IACR International Conference on Public-Key Cryptography. pp. 409–438. Springer (2022)
30. Cramer, R., Damgård, I., MacKenzie, P.D.: Efficient zero-knowledge proofs of knowledge without intractability assumptions. In: Imai, H., Zheng, Y. (eds.) PKC 2000. LNCS, vol. 1751, pp. 354–372. Springer, Heidelberg (Jan 2000)
31. Crites, E., Kohlweiss, M., Preneel, B., Sedaghat, M., Slamanig, D.: Threshold structure-preserving signatures. Cryptology ePrint Archive, Paper 2022/839 (2022), https://eprint.iacr.org/2022/839, https://eprint.iacr.org/2022/839
32. Crites, E.C., Lysyanskaya, A.: Delegatable anonymous credentials from mercurial signatures. In: Matsui, M. (ed.) CT-RSA 2019. LNCS, vol. 11405, pp. 535–555. Springer, Heidelberg (Mar 2019)

33. Davidson, A., Goldberg, I., Sullivan, N., Tankersley, G., Valsorda, F.: Privacy pass: Bypassing internet challenges anonymously. PoPETs 2018(3), 164–180 (2018), https://doi.org/10.1515/popets-2018-0026
34. Deuber, D., Maffei, M., Malavolta, G., Rabkin, M., Schröder, D., Simkin, M.: Functional credentials. PoPETs 2018(2), 64–84 (Apr 2018)
35. Doerner, J., Kondi, Y., Lee, E., abhi shelat, Tyner, L.: Threshold bbs+ signatures for distributed anonymous credential issuance. Cryptology ePrint Archive, Paper 2023/602 (2023), https://eprint.iacr.org/2023/602, https://eprint.iacr.org/2023/602
36. Ferrara, A.L., Green, M., Hohenberger, S., Pedersen, M.Ø.: Practical short signature batch verification. In: Fischlin, M. (ed.) CT-RSA 2009. LNCS, vol. 5473, pp. 309–324. Springer, Heidelberg (Apr 2009)
37. Fischlin, M.: Communication-efficient non-interactive proofs of knowledge with online extractors. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 152–168. Springer, Heidelberg (Aug 2005)
38. Fuchsbauer, G., Gay, R.: Weakly secure equivalence-class signatures from standard assumptions. In: Abdalla, M., Dahab, R. (eds.) PKC 2018, Part II. LNCS, vol. 10770, pp. 153–183. Springer, Heidelberg (Mar 2018)
39. Fuchsbauer, G., Hanser, C., Slamanig, D.: Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. Journal of Cryptology 32(2), 498–546 (Apr 2019)
40. Garman, C., Green, M., Miers, I.: Decentralized anonymous credentials. In: NDSS 2014. The Internet Society (Feb 2014)
41. Ghadafi, E.: Short structure-preserving signatures. In: Sako, K. (ed.) CT-RSA 2016. LNCS, vol. 9610, pp. 305–321. Springer, Heidelberg (Feb / Mar 2016)
42. Goyal, R., Vaikuntanathan, V.: Locally verifiable signature and key aggregation. Cryptology ePrint Archive, Report 2022/179 (2022), https://eprint.iacr.org/2022/179
43. Hanser, C., Slamanig, D.: Structure-preserving signatures on equivalence classes and their application to anonymous credentials. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part I. LNCS, vol. 8873, pp. 491–511. Springer, Heidelberg (Dec 2014)
44. Hanzlik, L., Slamanig, D.: With a little help from my friends: Constructing practical anonymous credentials. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 2004–2023. ACM Press (Nov 2021)
45. Hébant, C., Pointcheval, D.: Traceable constant-size multi-authority credentials. In: Galdi, C., Jarecki, S. (eds.) Security and Cryptography for Networks - 13th International Conference, SCN 2022, Amalfi, Italy, September 12-14, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13409, pp. 411–434. Springer (2022), https://doi.org/10.1007/978-3-031-14791-3_18
46. Hohenberger, S., Waters, B.: Synchronized aggregate signatures from the RSA assumption. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 197–229. Springer, Heidelberg (Apr / May 2018)
47. Kim, H., Lee, Y., Abdalla, M., Park, J.H.: Practical dynamic group signature with efficient concurrent joins and batch verifications. Cryptology ePrint Archive, Report 2020/921 (2020), https://eprint.iacr.org/2020/921
48. Kim, H., Sanders, O., Abdalla, M., Park, J.H.: Practical dynamic group signatures without knowledge extractors. Cryptology ePrint Archive, Report 2021/351 (2021), https://eprint.iacr.org/2021/351
49. Kreuter, B., Lepoint, T., Orrù, M., Raykova, M.: Anonymous tokens with private metadata bit. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part I. LNCS, vol. 12170, pp. 308–336. Springer, Heidelberg (Aug 2020)
50. Lee, K., Lee, D.H., Yung, M.: Aggregating CL-signatures revisited: Extended functionality and better efficiency. In: Sadeghi, A.R. (ed.) FC 2013. LNCS, vol. 7859, pp. 171–188. Springer, Heidelberg (Apr 2013)
51. Lu, S., Ostrovsky, R., Sahai, A., Shacham, H., Waters, B.: Sequential aggregate signatures and multisignatures without random oracles. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 465–485. Springer, Heidelberg (May / Jun 2006)
52. Lysyanskaya, A.: Security analysis of RSA-BSSA. Cryptology ePrint Archive, Report 2022/895 (2022), https://eprint.iacr.org/2022/895
53. Lysyanskaya, A., Micali, S., Reyzin, L., Shacham, H.: Sequential aggregate signatures from trapdoor permutations. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 74–90. Springer, Heidelberg (May 2004)
54. Mir, O., Slamanig, D., Bauer, B., Mayrhofer, R.: Practical delegatable anonymous credentials from equivalence class signatures. Proc. Priv. Enhancing Technol. 2023(3), 488–513 (2023), https://doi.org/10.56553/popets-2023-0093
55. Neven, G.: Efficient sequential aggregate signed data. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 52–69. Springer, Heidelberg (Apr 2008)
56. Paquin, C., Zaverucha, G.: U-prove cryptographic specification v1.1 (revision 3) (December 2013), https://www.microsoft.com/en-us/research/publication/u-prove-cryptographic-specification-v1-1-revision-3/
57. Pointcheval, D., Sanders, O.: Short randomizable signatures. In: Sako, K. (ed.) CT-RSA 2016. LNCS, vol. 9610, pp. 111–126. Springer, Heidelberg (Feb / Mar 2016)

58. Rosenberg, M., White, J., Garman, C., Miers, I.: zk-creds: Flexible anonymous credentials from zkSNARKs and existing identity infrastructure. Cryptology ePrint Archive, Report 2022/878 (2022), https://eprint.iacr.org/2022/878
59. Sanders, O.: Efficient redactable signature and application to anonymous credentials. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020, Part II. LNCS, vol. 12111, pp. 628–656. Springer, Heidelberg (May 2020)
60. Sanders, O.: Improving revocation for group signature with redactable signature. In: Garay, J. (ed.) PKC 2021, Part I. LNCS, vol. 12710, pp. 301–330. Springer, Heidelberg (May 2021)
61. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT'97. LNCS, vol. 1233, pp. 256–266. Springer, Heidelberg (May 1997)
62. Silde, T., Strand, M.: Anonymous tokens with public metadata and applications to private contact tracing. In: Eyal, I., Garay, J.A. (eds.) Financial Cryptography and Data Security - 26th International Conference, FC 2022, Grenada, May 2-6, 2022, Revised Selected Papers. Lecture Notes in Computer Science, vol. 13411, pp. 179–199. Springer (2022), https://doi.org/10.1007/978-3-031-18283-9_9
63. Sonnino, A., Al-Bassam, M., Bano, S., Meiklejohn, S., Danezis, G.: Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. In: NDSS 2019. The Internet Society (Feb 2019)
64. Tessaro, S., Zhu, C.: Revisiting BBS signatures. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V. Lecture Notes in Computer Science, vol. 14008, pp. 691–721. Springer (2023), https://doi.org/10.1007/978-3-031-30589-4_24

A Bandwidth Analysis of our IhMA Schemes

We present a concrete comparison of schemes in Table 3. We denote the size of zero knowledge proof of the tag as ZKP which as we showed in Section 5, the statements are simple and can be done efficiently with simple Schnorr proofs (the statements are presented in Section 5.3) .

Table 3: Communication complexity of our IhMA schemes (N: total issuers and K: issuers in showing).

	$\mathsf{IhMA}_{\mathsf{AtoSa}}$	$\mathsf{IhMA}_{\mathsf{ATMS}}$
$\lvert$ **cred** $\rvert$	$2N\mathbb{G}_1 + 2\mathbb{Z}_p$	$3N\mathbb{G}_1 + 2\mathbb{Z}_p$
$\lvert$ **show** $\rvert$	$4\mathbb{G}_1 + 4K\mathbb{G}_2 +$ $2K\mathbb{G}_1 + \mathsf{ZKP}$	$6K\mathbb{G}_2 + 6\mathbb{G}_1 +$ $2K\mathbb{G}_1 + \mathsf{ZKP}$

* We present the scheme in a way that supports ad-hoc attribute/issuer aggregation, but for fixed signatures, a constant size credential is achievable.

B Additional Definitions

B.1 Correctness Definitions

Definition 25 (AtoSa **correctness).** *An* AtoSa *is correct if it has the following three properties:*

Basic signature correctness:
For all $\{\mathsf{sk}_i, \mathsf{vk}_i\}_{i\in[\ell]} \leftarrow (\mathsf{KeyGen}(1^k))^n$, $\{m_i\}_{i\in[\ell]} \in \mathbb{Z}_p^{*\ell}$, $(\tau, \{\mathsf{aux}_j\}) = \mathsf{GenTagIdx}(\tau, \{m_i, \mathsf{vk}_i\}_{i\in[\ell]})$ $j \in [\ell]$, we have that $\sigma = \mathsf{Sign}(\mathsf{sk}_j, \tau, \mathsf{aux}_j, m_j)$ and $\mathsf{Verify}(\mathsf{vk}_j, \mathbf{T}, m_j, \sigma) = 1$

Randomizable signature correctness:
For all $(\mathsf{sk}, \mathsf{vk}, m, \mathbf{T}, \tau, \beta, \omega, \mu, \sigma', \mathsf{vk}', \sigma^*, \mathbf{T}^*, \mathbf{T}^\dagger)$ such that $\mathsf{Verify}(\mathsf{vk}, \mathbf{T}, m, \sigma) = 1$, $\beta, \omega \in \mathbb{Z}_p^*$, $\sigma' = \mathsf{ConvertSig}(\mathsf{vk}, \sigma, \omega)$, $\mathsf{vk}' = \mathsf{ConvertVK}(\mathsf{vk}, \omega)$, $\mathsf{sk}' = \mathsf{ConvertSK}(\mathsf{sk}, \omega)$, $(\sigma^*, \mathbf{T}^*) = \mathsf{RndSigTag}(\mathsf{vk}', \mathbf{T}, m, \sigma', \beta)$, $(\mathbf{T}^\dagger) = \mathsf{ConvertTag}(\mathbf{T}, \omega)$,
the following holds:
$\mathsf{ConvertSig}(\mathsf{vk}, \sigma, \omega) = \mathsf{Sign}(\mathsf{sk}', \mathsf{aux}, \tau, m)$ (for a valid aux), $\mathsf{Verify}(\mathsf{vk}', \mathbf{T}^*, m, \sigma^*) = 1$ and
$\mathsf{Verify}(\mathsf{vk}, \mathbf{T}^\dagger, m, \sigma') = 1$.

We've combined the definitions of all randomization functions ($\mathsf{RndSigTag}, \mathsf{ConvertTag}, \ldots$) in this definition, but for ensuring that use of a single randomization function is value, we can set the other randomization factors (β, ω) to 1 indicating no randomization to see that all the randomization functions are correct independent of each other.

Aggregatable signature correctness:
For all $\{\mathsf{sk}_i, \mathsf{vk}_i, m_i, \sigma_i, \mathbf{T}\}_{i\in[\ell]}$ such that $\forall i, \mathsf{Verify}(\mathsf{vk}_i, \mathbf{T}, m_i, \sigma_i) = 1$.

Then, the following holds:
$\sigma' = \mathsf{AggrSign}(\mathbf{T}, \{\mathsf{vk}_i, m_i, \sigma_i\}_{i\in[\ell]})$, $\mathsf{VerifyAggr}\left(a\mathsf{vk}, \mathbf{T}, \{m_i\}_{i\in[\ell]}, \sigma'\right) = 1$, where $a\mathsf{vk} = (\mathsf{vk}_i)_{i\in[\ell]}$.

Randomizable of Aggregatable signature correctness:
For all $a\mathsf{vk} = (\mathsf{vk}_i)_{i\in[\ell]}, \mathbf{T}, \{m_i\}_{i\in[\ell]}, \sigma$ and $\beta, \omega \in \mathbb{Z}_p^*$ such that $\mathsf{VerifyAggr}\left(a\mathsf{vk}, \mathbf{T}, \{m_i\}_{i\in[\ell]}, \sigma\right) = 1$,
$\sigma' = \mathsf{ConvertSig}(\mathsf{vk}, \sigma, \omega)$, $i \in [\ell]$: $\mathsf{vk}_i' = \mathsf{ConvertVK}(\mathsf{vk}_i, \omega)$, $\mathsf{sk}_i' = \mathsf{ConvertSK}(\mathsf{sk}_i, \omega)$ and
$(\sigma^*, \mathbf{T}^*) = \mathsf{RndSigTag}(\mathbf{T}, a\mathsf{vk}, (m_i)_{i\in[n]}, \sigma', \beta)$. $(\mathbf{T}^\dagger) = \mathsf{ConvertTag}(\mathbf{T}, \omega)$,
Then the following holds:
$\mathsf{VerifyAggr}\left(a\mathsf{vk}', \mathbf{T}^*, \{m_i\}_{i\in[\ell]}, \sigma^*\right) = 1$ and $\mathsf{VerifyAggr}\left(a\mathsf{vk}, \mathbf{T}^\dagger, \{m_i\}_{i\in[\ell]}, \sigma'\right) = 1$, where $a\mathsf{vk}' = (\mathsf{vk}_i')_{i\in[\ell]}$.

Definition 26 (ATMs **correctness).** *For all: S,λ,$(\mathbf{M}, \mathbf{N}) \in \mathcal{M}^H_{\mathsf{TDH}}$ such that:*

$$\begin{aligned}
&\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)\\
&\mathsf{aux}, \mathbf{T}, \tau := \mathsf{GenAuxTag}(S)\\
&(\sigma_i)_{i=1}^\ell = (\mathsf{Sign}(\mathsf{sk}_i, \tau, \mathsf{aux}, (\mathbf{M}_i, \mathbf{N}_i)))_{i=1}^\ell,\\
&\sigma' := \mathsf{AggrSign}(\mathbf{T}, (\mathsf{vk}_j, (\mathbf{M}_j, \mathbf{N}_j), \sigma_j)_{j=1}^\ell)
\end{aligned}$$

Then:

$$\begin{aligned}
&\bigwedge_{i=1}^{\ell} \mathsf{Verify}(\mathsf{vk}_i, \mathbf{T}, (\mathbf{M}_i, \mathbf{N}_i), \sigma_i') = 1.\\
&\mathsf{VerifyAggr}(\mathsf{avk}, \mathbf{T}(\mathbf{M}_i, \mathbf{N}_i)_{i=1}^\ell, \sigma')
\end{aligned}$$

Further, for randomization, if $\forall$ τ secret part of the tag, $(m_1^{(1)}, m_2^{(1)})_{i=1}^\ell \in \mathbb{Z}_p^{2\ell}$, keys honestly generated $(\mathsf{sk}_i, \mathsf{vk}_i)_{i=1}^\ell$ and for all randomness (γ, β, ω), for all $\sigma, \{\sigma_i\}_{i\in[\ell]}, \mathbf{T}$, if

$$\begin{aligned}
&\bigwedge_{i=1}^{\ell} \mathsf{Verify}(\mathsf{vk}_i, \mathbf{T}, (\mathbf{M}_i, \mathbf{N}_i), \sigma_i) = 1.\\
&\mathsf{VerifyAggr}(\mathsf{avk}, \mathbf{T}, (\mathbf{M}_i, \mathbf{N}_i)_{i=1}^\ell, \sigma)\\
&(\sigma_i', \mathbf{T}')_{i=1}^\ell = (\mathsf{ChangRep}(\mathsf{vk}_i, \mathbf{T}, (\mathbf{M}_i, \mathbf{N}_i), (\gamma, \beta)))_{i=1}^\ell,\\
&(\mathsf{vk}_i')_{i=1}^\ell := (\mathsf{ConvertVK}(\mathsf{vk}_i, \omega))_{i=1}^\ell\\
&(\sigma_i'')_{i=1}^\ell := (\mathsf{ConvertSig}(\mathsf{vk}_i, (\mathbf{M}_i, \mathbf{N}_i), \mathbf{T}', \sigma_i', \omega))_{i=1}^\ell\\
&(\mathbf{T}_i^\dagger) := \mathsf{ConvertTag}(\mathbf{T}, \omega)\\
&\sigma^* := \mathsf{AggrSign}(\mathbf{T}', (\mathsf{vk}_j, (\mathbf{M}_j, \mathbf{N}_j), \sigma_j'')_{j=1}^\ell)
\end{aligned}$$

then

$$\begin{aligned}
&\mathsf{VerifyAggr}(\mathsf{avk}, \mathbf{T}', (\mathbf{M}_i, \mathbf{N}_i)_{i=1}^\ell, \sigma^*)\\
&\mathsf{VerifyAggr}(\mathsf{avk}, \mathbf{T}^\dagger, (\mathbf{M}_i, \mathbf{N}_i)_{i=1}^\ell, \sigma^*)\\
&\bigwedge_{i=1}^{\ell} \mathsf{Verify}(\mathsf{vk}_i', \mathbf{T}', (\mathbf{M}_i, \mathbf{N}_i), \sigma_i') = 1.\\
&\bigwedge_{i=1}^{\ell} \mathsf{Verify}(\mathsf{vk}_i, \mathbf{T}^\dagger, (\mathbf{M}_i, \mathbf{N}_i), \sigma_i'') = 1.
\end{aligned}$$

Also $\forall(\mathsf{sk},\mathsf{vk})$ *honestly generated* $\mathbf{T}$ *and* $(\mathbf{M},\mathbf{N})$ *then*
$\mathsf{ChangRep}(\mathsf{vk},\mathbf{T},(\mathbf{M},\mathbf{N}),\sigma,\cdot)$ *is a group morphism from* $\mathbb{Z}_p^{*2}$ *to* $\mathcal{SIG}$*, moreover, we have*

$$\omega \mapsto \begin{pmatrix} \mathsf{ConvertSK}(\mathsf{sk},\omega), \mathsf{ConvertVK}(\mathsf{vk},\omega), \\ \mathsf{ConvertSig}(\mathsf{vk},(\mathbf{M},\mathbf{N}),\mathbf{T},\sigma_i',\omega) \end{pmatrix}$$

is a group morphism from $\mathbb{Z}_p^{*2}$ *to* $\mathcal{SK}\times\mathcal{VK}\times\mathcal{SIG}$

C Additional Preliminaries

C.1 Zero-Knowledge Proofs of Knowledge

We define zero-knowledge proofs of knowledge (ZKPOK) and discuss non-interactive versions thereof (NIZK). In our IhMA, we require protocols to prove knowledge of discrete logarithm relations. This can be efficiently realized by relying on Sigma protocols (i.e., three-round public-coin honest-verifier zero-knowledge proofs of knowledge). Sigma protocols are efficient instantiations of ZKPOK which can be converted to (malicious-verifier) zero-knowledge proofs of knowledge, using Damgård's Technique [30] and made non-interactive using different techniques (discussed below).

ZKPoK. Let $L_{\mathbb{R}} = \{x \mid \exists\, w : (x,w) \in \mathbb{R}\} \subseteq \{0,1\}^*$ be a formal language, where $\mathbb{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ is a binary, polynomial-time (witness) relation. For such a relation, the membership of $x \in L_{\mathbb{R}}$ can be decided in polynomial time (in $|x|$) when given a witness w of length polynomial in $|x|$ certifying $(x,w) \in \mathbb{R}$. We assume an interactive protocol $(\mathcal{P},\mathcal{V})$ between a prover $\mathcal{P}$ and a PPT verifier $\mathcal{V}$ and denote the outcome of the protocol as $(\cdot,b) \leftarrow \big(\mathcal{P}(\cdot,\cdot),\mathcal{V}(\cdot)\big)$ where $b=0$ indicates that $\mathcal{V}$ rejects and $b=1$ that it accepts the conversation with $\mathcal{P}$. We require the following properties:

Definition 27 (Completeness). *We call an interactive protocol* $(\mathcal{P},\mathcal{V})$ *for a relation* $\mathbb{R}$ *complete if for all* $x \in L_{\mathbb{R}}$ *and* w *s.t* $(x,w) \in R$ *we have* $(\cdot,1) \leftarrow \big(\mathcal{P}(x,w),\mathcal{V}(x)\big)$ *with probability 1.*

Definition 28 (Zero knowledge (ZK)). *An* $(\mathcal{P},\mathcal{V})$ *for a language* L *is ZK if for any (malicious) verifier* $\mathcal{V}^*$ *, there exists a PPT algorithm* $\mathcal{S}$ *(the simulator) such that:*

$$\{\mathcal{S}(x)\}_{x\in L} \approx \{< (\mathcal{P},\mathcal{V}^\star)(x) >\}_{x\in L},$$

where $(\mathcal{P},\mathcal{V}^*)(x)$ *shows the transcript of communication between* $\mathcal{P}$ *and* $\mathcal{V}^\star$ *on the common input* x*.*

Definition 29 (Knowledge soundness). *We say that* $(\mathcal{P},\mathcal{V})$ *is a proof of knowledge (PoK) relative to an NP relation* $\mathbb{R}$ *if for any malicious prover* $\mathcal{P}^*$ *such that* $(\cdot,1) \leftarrow \big(\mathcal{P}^*(x),\mathcal{V}(x)\big)$ *with probability greater than* ϵ *there exists a PPT knowledge extractor* K *(with rewinding black-box access to* $\mathcal{P}^*$*) such that* $K^{\mathcal{P}^*}(x)$ *returns a value* w *satisfying* $(x,w) \in \mathbb{R}$ *with probability polynomial in* ϵ*.*

If all properties hold, then we denote this interactive protocol as a zero-knowledge proofs of knowledge (ZKPOK).

Non-Interactive Zero-Knowedge Proofs (of Knowledge). One can use the Fiat-Shamir heuristic to transform any Sigma protocol into a non-interactive zero-knowledge proof of knowledge (NIZK). Whenever one requires multiple-extractions in a security proof, a standard measure is to opt for interactive NIZK. We however note that when willing to pay some extra costs, one could instead use straight-line extractable NIZK, e.g., obtained via Fischlin's transformation [37].

C.2 Assumptions

PS assumptions. The PS assumption is an interactive assumption, defined by Pointcheval and Sanders [57] to construct a short randomizable signature known as PS signature.

Definition 30 (PS Assumption [57]). *The PS assumption holds if no PPT adversary* $\mathcal{A}$*, who takes asymmetric pairing* $(p,\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T,P,\hat{P},e)$*, a tuple* $(P^x,\hat{P}^y) \in \mathbb{G}_2^2$ *where* x *and* y *are random scalars in* $\mathbb{Z}_p$*, and unlimited access to PS oracle* $\mathcal{O}^{PS}(m)$ *s.t. on input* $m \in \mathbb{Z}_p^*$ *that chooses a random* $h \in \mathbb{G}_1$ *and outputs the pair* (h,h^{x+my})*, can efficiently generate a tuple* $(h^*,s^*,m^*) \in \mathbb{G}_2^n \times \mathbb{Z}_p$ *such that (1)* $h^* \neq 1_{\mathbb{G}_1}$ *(2)* $s^* = h^{x+ym^*}$*, (3)* $m^* \notin Q$*, where* Q *is the list of queried messages to the* $\mathcal{O}^{PS}(m)$ *oracle.*

The validity of the PS assumption tuple (h^*, s^*, m^*) can be checked as: $e(s^*, \hat{P}) = e(h^*, \hat{P}^x(\hat{P}^y)^{m^*})$.

Generalization of the PS assumption (GPS). GPS was introduced by Kim et al. [47], and splits the PS oracle into two: the first oracle provides basis h sampled uniformly at random and the second oracle takes the message and h as inputs and generates the PS tuple.

Definition 31 (Generalized PS Assumption [47]). *Given a tuple $(P^x, \hat{P}^y) \in \mathbb{G}_2^2$ and two oracles $\mathcal{O}_0^{GPS}()$ and $\mathcal{O}_1^{GPS}(m, h)$ such that: $\mathcal{O}_0^{GPS}() \rightarrow h$, where $h \in \mathbb{G}_1$ is uniformly distributed $\mathcal{O}_1^{GPS}(m, h) \rightarrow s$, where $h \in \mathbb{G}_1$, $m \in \mathbb{Z}_p$, and $s = h^{x+ym} \in \mathbb{G}_1$ as output and if $h \notin Q_0 \vee (h, \star) \in Q_1$ return $\perp$. The GPS assumption holds if no PPT adversary, $\mathcal{A}$, can find a tuple $(h^*, s^*, m^*) \in \mathbb{G}_1^2 \times \mathbb{Z}_p$ such that, (1) $h^* \neq 1_{\mathbb{G}_1}$, (2) $s^* = (h^*)^{x+ym^*}$, (3) $m^* \notin Q$, where $Q_1 = Q_1 \cup (h, m)$ is the list of queried to $\mathcal{O}_1^{GPS}$ oracle by the adversary.*

Authors of [48] expanded the GPS assumption in way that all scalars values are replaces by source group elements, called GPS_2. Additionally, authors of [31] follow up this assumption and define the GPS_3 assumption contains indexed Diffie-Hellman message spaces (cf. Def 1). In particular, they modify the oracle $\mathcal{O}_0^{\mathsf{GPS}_2}$ such that it generates a basis h by taking an index id as input and also acts as answer to RO.

Definition 32 (GPS_3 Assumption [31]). *Given $(P^x, \hat{P}^y) \in \mathbb{G}_2^2$ and two oracles $\mathcal{O}_0^{GPS_3}(id)$ and $\mathcal{O}_1^{GPS_3}(h, M, N)$ such that $\mathcal{O}_0^{GPS_3}(id) \rightarrow h$, where a basis h is generated by taking a tag id as input as follows: if $Q_0[id] = \perp$, else pick $r \xleftarrow{\$} \mathbb{Z}_p$ and compute $Q_0[id] \leftarrow h = P^r$: return $Q_0[id]$, where Q_0 is the list of queried messages to $\mathcal{O}_0^{GPS_3}$. For $\mathcal{O}_1^{GPS_3}(h, M, N) \rightarrow s$, where $h \in \mathbb{G}_1$, $(M, N) \in \mathbb{G}_1 \times \mathbb{G}_2$, if $h \notin Q_0 \vee h \in Q_1$ return $\perp$ else sets $s = h^x M^y \in \mathbb{G}_1$ as output only if h was provided by $\mathcal{O}_0^{GPS_3}(id)$. The GPS_3 assumption holds if no PPT adversary, $\mathcal{A}$, can find a tuple (h^*, M^*, N^*, s^*) such that, (1) $h^* \neq 1_{\mathbb{G}_1}$ and $N^* \neq 1_{\mathbb{G}_2}$, (2) $s^* = (h^*)^x \cdot (M^*)^y$, (3) $\mathsf{dlog}_{h^*}(M^*) = \mathsf{dlog}_{\hat{P}}(N^*)$ (4) $(M^*, N^*) \notin Q$, where Q is the list of queried messages to $\mathcal{O}_1^{GPS_3}$ oracle by the adversary.*

The validity of a solution to GPS_3 assumption can be checking by two pairing product equations: $e(h^*, N^*) = e(M^*, \hat{P}) \wedge e(h^*, \hat{P}^x)e(M^*, \hat{P}^y) = e(s^*, \hat{P})$. Here, similar to [31], we consider the oracle $\mathcal{O}_0^{\mathsf{GPS}}$ as a RO such that $\mathcal{O}_0^{\mathsf{GPS}}(\mathsf{aux}) \rightarrow h$ generates a basis h by taking aux as input s.t,: if $Q_0[\mathsf{aux}] = \perp$, else pick $r \xleftarrow{\$} \mathbb{Z}_p$ and compute $Q_0[\mathsf{aux}] \leftarrow h = P^r$: return $Q_0[\mathsf{aux}]$, where Q_0 is the list of queried messages to $\mathcal{O}_0^{\mathsf{GPS}}$. Note that the condition 3 as $\mathsf{dlog}_{h^*}(M^*) = \mathsf{dlog}_{\hat{P}}(N^*)$ will be $\mathsf{dlog}_{h^*}(M^*) = \mathsf{dlog}_{\hat{\tau}}(N^*)$ regarding to tag based DH message space (Def. 10) that can be checked $e(h^*, N^*) = e(M^*, \hat{T})$.

D Proofs

D.1 Proof of Theorem 1

To simplify our proof and make it more readable, we split our proof via two lemmas such that Lemma 1 indicates that the aggregate signature with a randomizable tag is secure in the RO model (without considering randomizable keys). Lemma 2 stands for randomizable verification keys property and shows that if the aggregate signature with a randomizable tag is secure, meaning lemma 1 is correct, we can achieve an aggregate signature with randomizable verification keys. WLOG, we assume the game only outputs one if the forgery is on the honest signer's exact key $\mathsf{vk}' = \mathsf{vk}_{j'}$ for an index j'.

Lemma 1 (Aggregate Signatures with Randomizable Tags). *Let $\mathcal{A}$ be an adversary against the EUF-CMA security of the aggregate signature scheme (Def. 4). If GPS assumption holds, then our construction in Section 3.3 is unforgeable when $\mathcal{A}$ outputs a forgery on an exact honest verification key instead of an equivalent one. This means that after interacting with the EUF-CMA challenger, no PPT adversary can produce $(\mathsf{avk} = (\mathsf{vk}_j), \mathsf{ask} = (\mathsf{sk}_j), \mathbb{M}^* = (m_i^*), \hat{\tau}^*, \sigma^*)_{j \in [\ell]}$ s.t:*

An adversary interacting with the EUF-CMA challenger produces: $(j', \mathsf{avk} = (\mathsf{vk}_j)_{j \in [\ell]}, \mathsf{ask} = (\mathsf{sk}_j)_{j \in [\ell] \setminus j'}$, $\mathbb{M}^* = (m_j^*)_{j \in [\ell]}, (\tau^*, \mathbf{T}^*), \sigma^*)$ That adversary has defeated the EUF-CMA game if their output satis-

fies:

$$\begin{pmatrix} \mathsf{VerifyAggr}\,(\mathsf{avk}, \mathbf{T}^*, \sigma^*, \mathbb{M}^*) = 1 \wedge \\ \forall j \in [\ell], j \neq j' : [\mathsf{vk}_j^*]_{\mathcal{R}_{\mathsf{vk}}} = [\mathsf{vk}_j]_{\mathcal{R}_{\mathsf{vk}}} \wedge \\ [\mathsf{vk}']_{\mathcal{R}_{\mathsf{vk}}} = [\mathsf{vk}_{j'}]_{\mathcal{R}_{\mathsf{vk}}} \wedge \forall (m, \tau) \in Q : m \neq m_j^* \vee [\mathbf{T}^*] \neq [\mathbf{T}] \end{pmatrix}$$

Proof. We construct a reduction $\mathcal{B}$ using $\mathcal{A}$ against the GPS assumption (Def. 31). The challenger of latter game will be denoted by $\mathcal{C}$. We answer to the random oracle $H(c)$ by calling $\mathcal{O}_0^{\mathsf{GPS}}(c)$ to generate base h where c is part of aux from Definition 9. This is a similar call to $\mathcal{O}_0^{\mathsf{GPS}_3}(id)$ in Definition C.2 but replacing id with c.

The reduction will continue by using the given challenge key from the GPS challenger to sign either messages or tags. The insight for why this proof works comes from the fact that our signature is exactly a multi-message signature (from [57]) on m and $\frac{\rho_2}{\rho_1}$ randomized by ρ_1. Our proof of security will be similar to the proof of multi-message security in [57].

Setup: $\mathcal{B}$ receives from $\mathcal{C}$ values $(\hat{X} = \hat{P}^x, \hat{Y} = \hat{P}^y)$ and pp of BG. $\mathcal{B}$ then computes $\alpha_1, \beta_1, \alpha_2, \beta_2$ and values: $\hat{Y}_1 = \hat{Y}^{\alpha_1}\hat{P}^{\beta_1}, \hat{Y}_2 = \hat{Y}^{\alpha_2}\hat{P}^{\beta_2}$. $\mathcal{B}$ then computes the challenge key for AtoSa as $\mathsf{vk}' = (\hat{X}, \hat{Y}_1, \hat{Y}_2)$ and gives this to the adversary.

Queries: When $\mathcal{A}$ asks a signature query on a tag τ, m, $\mathsf{aux} = (c, o)$, s.t $\mathsf{VerifyAux}(\mathsf{sk}, \mathsf{aux}, \tau, m) = 1$, $\mathcal{B}$ computes a signature as follows:

1. $\mathcal{B}$ first requests from $\mathcal{C}$ a base $h = H(c)$, by calling $h \leftarrow \mathcal{O}_0^{\mathsf{GPS}}(c)$ which is also a RO response. $\mathcal{C}$ (or RO) response as follows: if $Q_0[c] = \perp$, pick $r \xleftarrow{\$} \mathbb{Z}_p^*$ and compute $Q_0[c] \leftarrow h = P^r$: return $Q_0[c]$, where Q_0 is the list of queried messages to $\mathcal{O}_0^{\mathsf{GPS}}$ (or RO). Note that if $h \in Q_1$ we return $\perp$.
2. $\mathcal{B}$ requests from $\mathcal{C}$ to compute s for a message $m^\dagger = \alpha_1 m + \alpha_2 \frac{\rho_2}{\rho_1}$ by calling $s \leftarrow \mathcal{O}_1^{\mathsf{GPS}}(m^\dagger, h)$. $\mathcal{C}$ computes this as $s = h^{x + (\alpha_1 m + \alpha_2 \frac{\rho_1}{\rho_2})y}$. $\mathcal{B}$ then computes $\sigma = \left(h' = h^{\rho_1}, s' = \left(s * h^{\beta_1 m + \beta_2 \frac{\rho_2}{\rho_1}}\right)^{\rho_1}\right)$ and returns this to the adversary. We can see that this verifies with the vk' we gave the adversary earlier.

$$\sigma = \left(h' = h^{\rho_1}, s' = \left(h^{\rho_1 (x + (\alpha_1 y m + \alpha_2 \frac{\rho_2}{\rho_1})y + \beta_1 m + \beta_2 \frac{\rho_2}{\rho_1})}\right)\right)$$

Using the equations from Sign (removing the degeneracy check):

$$e(h', \hat{X} * \hat{Y}_1^m) e(T_2, \hat{Y}_2) = e(s', \hat{P})$$

$$= e(h^{\rho_1}, \hat{P}^x * \hat{P}^{(\alpha_1 y + \beta_1)m}) e(h^{\rho_2}, \hat{P}^{\alpha_2 y + \beta_2}) = e(s', \hat{P})$$

$$= e(h, \hat{P})^{\rho_1 * (x + (\alpha_1 y + \beta_1)m} e(h, \hat{P})^{\rho_2 * (\alpha_2 y + \beta_2)} = e(s', \hat{P})$$

$$= e(h, \hat{P})^{\rho_1 * (x + (\alpha_1 y + \beta_1)m) + \rho_2 * (\alpha_2 y + \beta_2)} = e(s', \hat{P})$$

$$= e(h, \hat{P})^{\rho_1 * (x + m\alpha_1 y + m\beta_1 m + \frac{\rho_2}{\rho_1}\alpha_2 y + \frac{\rho_2}{\rho_1}\beta_2)} = e(s', \hat{P})$$

If we rearrange s' we can see this is the same so the signature verifies correctly:

$$e(s', \hat{P}) = e(h, \hat{P})^{\rho_1 (x + \alpha_1 y m + \beta_1 m + \alpha_2 \frac{\rho_2}{\rho_1} y + \beta_2 \frac{\rho_2}{\rho_1})}$$

3. $\mathcal{A}$ then repeats a polynomial number of signing queries adaptively.

Output: Eventually, $\mathcal{A}$ outputs a forgery as $(j', h'^*, s^*, \tau^* = (\rho_1^*, \rho_2^*, h^{\rho_1^*}, h^{\rho_2^*}))$ on messages $\mathbb{M}^* = (m_1^*, \ldots, m_n^*)$ under the keys $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ and $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$. From the definition, we know that for an index j', a tuple $(m_{j'}^*, \sigma_{j'}^*)$ should be the new signature-message pair under $\mathsf{vk}_{j'} = \mathsf{vk}'$ that is aggregated in σ^* such that $\mathcal{A}$ has never queried both $m_{j'}^*$ and τ^* together. The adversary has also output all other secret keys except for the challenge key. This allows us to isolate this key:

1. $\mathcal{B}$ cancels the tag out from the aggregate signature which is a new tuple as:

$$\sigma^*_{j'} = \left(h^*, s^*_{j'} = \frac{s^*}{\prod_{j \in [\ell] \setminus j'} (h^*)^{x_j + y_{1j} m_j + y_{2j} \frac{\rho^*_2}{\rho^*_1}}} (h^*)^{-\sum \beta_j m^*_j} \right)$$

This signature should now satisfy: $e(h^*, \hat{X}\hat{Y}^{\alpha_1 m^*_{j'} + \alpha_2 \frac{\rho^*_2}{\rho^*_1}}) = e(s^*_{j'}, \hat{P})$.

Send $(\sigma^*_{j'} = (h^*, s^*_{j'}), \alpha_1 m^*_{j'} + \alpha_2 \frac{\rho^*_2}{\rho^*_1})$ under $\mathsf{vk} = (\hat{X}, \hat{Y})$ as a valid GPS response to $\mathcal{C}$.

Because this verifies on the challenge key for a message $\alpha_1 m^*_{j'} + \alpha_2 \frac{\rho^*_2}{\rho^*_1}$, it will be a valid forgery if this message were never queried previously. We can see that, after fixing a challenge key $\hat{Y}$, then $\forall \alpha_1, \alpha_2, \hat{Y}_1, \hat{Y}_2 \in \mathbb{G}_2$, $\exists \beta_1, \beta_2$ s.t. $\hat{Y}_1 = \hat{Y}^{\alpha_1} \hat{P}^{\beta_1}, \hat{Y}_2 = \hat{Y}^{\alpha_2} \hat{P}^{\beta_2}$. This can be seen by setting $\hat{P}^{\beta_1}$ to be $\hat{Y}^{-1}\hat{Y}_1$ (and similar for $\hat{P}^{\beta_2}$). This value for β_1 isn't possible to compute in polynomial time, but it still exists and each choice of β_1 is just as likely to be chosen via random coins as any other element. Further, the distribution of vk values resulting from the choice of β_1, β_2 is uniform. Thus, because of β_1 and β_2, the adversary's view is independent of α_1 and α_2. In the space of message/tag pairs, we have only p^3 sets of pairs $((m, \frac{\rho_2}{\rho_1}), (m', \frac{\rho'_2}{\rho'_1}))$ that satisfy this equation:

$$\alpha_1 m + \alpha_2 \frac{\rho_2}{\rho_1} = \alpha_1 m' + \alpha_2 \frac{\rho'_2}{\rho'_1} \tag{2}$$

(where p is the order of the group). This is because for each combination of $m, \frac{\rho_2}{\rho_1}, m'$, there is a specific value for $\frac{\rho'_2}{\rho'_1}$ that completes the set. Thus there are only p^3 distinct sets that meet Equation 2. Notice that there are p^4 of these sets without the restriction in Equation 2. The adversary samples these sets at random when issuing queries since their view is independent of the chosen α_1, α_2. In the end, the reduction will only fail if a we find a set that satisfies Equation 2 in the adversary's signature queries. Note that the adversary cannot entirely benefit from his or her polynomial number of queries since the pair must contain the adversary's outputted forgery and the adversary's view is independent of α_1, α_2 so their choice of which message to output must be random. Thus, the adversary outputs a forgery $m^*_{j'}, \tau^*$ which forms a pair with each q query issued previously (where q is the polynomial number of signing queries). Thus, the chance that our adversary outputs a forgery that meets Equation 2 with a previous query (which would mean our reduction does not constitute a forgery) is $\frac{p^3 * q}{p^4}$ which is negligible since p is exponential and q is polynomial in the security parameter.

Note that we never ask $\mathcal{O}_1^{\mathsf{GPS}}$ for a second signature on any given h. This is because of Aux binding (Definition 9). The value we pass to $\mathcal{O}_0^{\mathsf{GPS}}$ is based on the messages and tags we sign. Thus, if the adversary asks for a second signature on a particular message/tag pair, the resulting h will either be the same (meaning we can simply return the previous signature) or be a fresh result from $\mathcal{O}_0^{\mathsf{GPS}}$, meaning that this h has not been seen before.

Lemma 2 (Aggregate Tag based Signatures with randomizable Keys). *An adversary cannot produce a valid forgery in Definition 18 without querying the corresponding randomization of the challenge verification key, vk', thus allowing a reduction to extract this randomization and de-randomize the signature to verify with this key.*

To prove the randomization (flexible) public keys property, we follow proof of convert Mercurial signature [32]. Assume there exists a generic group, PPT algorithm $\mathcal{A}$ that can break the unforgeability of aggregate signature scheme randomizable tag and public keys; that is, when given an honest verifier key, vk', $\mathcal{A}$ is able to produce a forgery $(\mathsf{avk} = \{\mathsf{vk}^*_i\}, \mathsf{ask}, \tau^*, \mathbb{M}^*, \sigma^*)$ that satisfies the following conditions with non-negligible probability:

$$[\mathsf{vk}']_{\mathcal{R}_{\mathsf{vk}}} = [\mathsf{vk}^*_1]_{\mathcal{R}_{\mathsf{vk}}} \wedge \forall m \in Q, m^*_1 \neq m \wedge$$
$$\mathsf{VerifyAggr}(\mathsf{avk}, \mathbf{T}^*, \mathbb{M}^*, \sigma^*) = 1$$

Where WLOG, vk_1^* is in the same equivalence class as vk' (with this construction, the adversary's forgery can always be rearranged to produce a forgery like this). The fact that vk_1^* belongs to the same equivalence class as vk' implies that there exists some $\alpha \in \mathbb{Z}_p^*$ such that $\mathsf{vk}' = \mathsf{vk}_1^{\alpha}$. We can construct a PPT reduction, $\mathcal{B}$ that creates a forgery for an aggregate tag based signature scheme using $\mathcal{A}$, then use Lemma 1 to prove our construction secure. The challenger $\mathcal{C}$ in the tag based signature unforgeability game for $\mathcal{B}$ chooses values $(x, y_1, y_2) \xleftarrow{\$} \mathbb{Z}_p^*$, sets $\mathsf{vk}' = (\hat{X}, \hat{Y}_1, \hat{Y}_2) = (\hat{P}^x, \hat{P}^{y_1}, \hat{P}^{y_2})$, and forwards vk' to $\mathcal{B}$.

On input vk', $\mathcal{B}$ operates as follows:

- $\mathcal{B}$ forwards vk' to $\mathcal{A}$ and runs $\mathcal{A}(\mathsf{vk}')$. $\mathcal{B}$ forwards $\mathcal{A}$'s signature queries to the AtoSa (with inflexible public keys) challenger and forwards the results to $\mathcal{A}$. $\mathcal{B}$ also services and records $\mathcal{A}$'s GGM queries.
- $\mathcal{B}$ obtains $\mathcal{A}$'s forgery $(\mathsf{avk} = \{\mathsf{vk}_i^*\}, \mathbb{M}^* = \{m_i^*\}, \sigma^*, \tau^*)$.
- If, via this process, it is possible for $\mathcal{B}$ to obtain α, $\mathcal{B}$ can remove α and outputs $(\mathsf{avk}' = \{(\mathsf{vk}_i^*)^{\alpha}\}, \mathbb{M}^*, \sigma' = (h, (s^*)^{\alpha}), \tau^*)$ as his forgery; else, $\mathcal{B}$ outputs $\perp$.

Now, let us analyze this reduction. One of these vk_i^* is in the same equivalence class as vk'. WLOG, we'll say this is vk_1^*.

Claim 61 *If $[\mathsf{vk}']_{\mathcal{R}_{\mathsf{vk}}} = [\mathsf{vk}_1^*]_{\mathcal{R}_{\mathsf{vk}}}$, then the generic group model reduction $\mathcal{B}$ can obtain $\alpha \in \mathbb{Z}_p^*$ such that $\mathsf{vk}' = (\mathsf{vk}_1^*)^{\alpha}$.*

Proof. Initially, before any queries are made, the elements of $\mathbb{G}_2$ that $\mathcal{A}$ has seen are $\hat{P}$ and $\mathsf{vk}' = (\hat{X}, \hat{Y}_1, \hat{Y}_2)$. Any output in $\mathbb{G}_2$ by the adversary must be from a GGM query of the form:

$$P^{\alpha_1} * \hat{X}^{\alpha_x} * \hat{Y}_1^{\alpha_{y_1}} * \hat{Y}_2^{\alpha_{y_2}}$$

Where $\alpha_1, \alpha_x, \alpha_y \in \mathbb{Z}_p$. We can rewrite this as:

$$P^{\alpha_1 + x\alpha_x + y_1\alpha_{y_1} + y_2\alpha_{y_2}}$$

This must be the form of the adversary's output, $\mathsf{vk}_1^* = \hat{X}_1^*, \hat{Y}_1^*$.

$$\hat{X}_1^* = P^{\alpha_1^{(\hat{X}^*)} + x\alpha_x^{(\hat{X}^*)} + y_1\alpha_{y_1}{}^{(\hat{X}^*)} + y_2\alpha_{y_2}{}^{(\hat{X}^*)}}$$

$$\hat{Y}_{1,1}^* = P^{\alpha_1^{(\hat{Y_1}^*)} + x\alpha_x^{(\hat{Y_1}^*)} + y_1\alpha_{y_1}{}^{(\hat{Y_1}^*)} + y_2\alpha_{y_2}{}^{(\hat{Y_1}^*)}}$$

$$\hat{Y}_{2,1}^* = P^{\alpha_1^{(\hat{Y_2}^*)} + x\alpha_x^{(\hat{Y_2}^*)} + y_1\alpha_{y_1}{}^{(\hat{Y_2}^*)} + y_2\alpha_{y_2}{}^{(\hat{Y_2}^*)}}$$

Where, for example, $\alpha_{y_1}^{(\hat{X}^*)}$ is the adversary's coefficient for the secret value, y_1, when computing their forgery verification key, $\hat{X}^*$. We want to prove that $\alpha_1^{(\hat{X}^*)}$, $\alpha_1^{(\hat{Y_1}^*)}$, $\alpha_1^{(\hat{Y_2}^*)}$, $\alpha_x^{(\hat{Y_1}^*)}$, $\alpha_x^{(\hat{Y_2}^*)}$, $\alpha_{y_1}^{(\hat{Y_2}^*)}$, $\alpha_{y_1}^{(\hat{X}^*)}$, $\alpha_{y_2}^{(\hat{X}^*)}$, and $\alpha_{y_2}^{(\hat{Y}_1^*)}$ are zero and $\alpha_x^{(\hat{X}^*)}$ is equal to $\alpha_{y_1}^{(\hat{Y_1}^*)}$ and $\alpha_{y_2}^{(\hat{Y_2}^*)}$. If so, we will know that $\alpha_x^{(\hat{X}^*)} = \alpha_{y_1}^{(\hat{Y_1}^*)} = \alpha_{y_2}^{(\hat{Y_2}^*)} = \alpha$ and we can compute the forgery for the AtoSa game. We can think of the exponents as polynomials:

$$p_{\hat{X}}^*(x, y) = \alpha_1^{(\hat{X}^*)} + x\alpha_x^{(\hat{X}^*)} + y_1\alpha_{y_1}^{(\hat{X}^*)} + y_2\alpha_{y_2}^{(\hat{X}^*)},$$
$$p_{\hat{Y}_1}^*(x, y) = \alpha_1^{(\hat{Y_1}^*)} + x\alpha_x^{(\hat{Y_1}^*)} + y_1\alpha_{y_1}^{(\hat{Y_1}^*)} + y_2\alpha_{y_2}^{(\hat{Y_1}^*)},$$
$$p_{\hat{Y}_2}^*(x, y) = \alpha_1^{(\hat{Y_2}^*)} + x\alpha_x^{(\hat{Y_2}^*)} + y_1\alpha_{y_1}^{(\hat{Y_2}^*)} + y_2\alpha_{y_2}^{(\hat{Y_2}^*)},$$

Signatures are exclusively in $\mathbb{G}_1$, so the adversary does not learn any more elements in $\mathbb{G}_2$. If the adversary outputs a vk_1^* where $\alpha_1^{(\hat{X}^*)} \neq 0$, $\alpha_{y_1}^{(\hat{X}^*)} \neq 0$, or $\alpha_{y_2}^{(\hat{X}^*)} \neq 0$, then $\hat{X}_1^*$ and $\hat{X}'$ (in vk_1^* and vk') are the result of queries to the GGM of distinct polynomials in $\mathbb{Z}_p[x, y_1, y_2]$ where p is the size of the groups of the bilinear pairing. There is a similar arguement for $\hat{Y}_{1,1}^*$ and $\hat{Y}_{2,1}^*$. Thus, according to the Schwartz-Zippel lemma, the chance that these two polynomials evaluate to the same value when x, y are chosen randomly, is negligible. Thus, if we have $\mathcal{B}$ output random encodings independent of x, y_1, y_2 and later define x, y_1, y_2, there is a negligble chance that $\mathcal{A}$ can compute a non-zero value

for $\alpha_1^{(\hat{X}^*)}$, $\alpha_1^{(\hat{Y_1}^*)}$, $\alpha_1^{(\hat{Y_2}^*)}$, $\alpha_{y_1}^{(\hat{X}^*)}$, $\alpha_{y_2}^{(\hat{X}^*)}$, $\alpha_{y_1}^{(\hat{X}^*)}$, $\alpha_{y_1}^{(\hat{Y_2}^*)}$, $\alpha_{y_2}^{(\hat{X}^*)}$, or $\alpha_{y_2}^{(\hat{Y_1}^*)}$ where the resulting vk_1^* is still in the equivalence class of vk'. This proves claim 1.

After proving that only $\alpha_x^{(\hat{X}^*)}$, $\alpha_{y_1}^{(\hat{Y_1}^*)}$ and $\alpha_{y_2}^{(\hat{Y_2}^*)}$ are non-zero, it holds that $\alpha_x^{(\hat{X}^*)} = \alpha_{y_1}^{(\hat{Y_1}^*)} = \alpha_{y_2}^{(\hat{Y_2}^*)} = \alpha$ as, otherwise, $[\mathsf{vk}_1^*]_{\mathcal{R}} \neq [\mathsf{vk}']_{\mathcal{R}}$.

We can see from the Verify algorithm that, if the reduction can recover α such that $\mathsf{vk}_1^* = (\mathsf{vk}')^{\alpha}$, and $(\mathsf{avk}, m^*, \sigma^*, \tau^*)$ is a valid forgery for our AtoSa scheme with randomizable public keys, then $(\mathsf{avk}', m^*, \sigma', \tau^*)$ is a valid forgery for our AtoSa scheme without randomizable public keys:

$\mathsf{avk} = \{\hat{X}_j, \hat{Y}_{1,j}, \hat{Y}_{2,j}\}$,

$\mathsf{avk}' = \{\hat{X}_j^{\frac{1}{\alpha}}, \hat{Y}_{1,1}^{\frac{1}{\alpha}}, \hat{Y}_{2,1}^{\frac{1}{\alpha}}\}$,

$e(h, \prod_{i\in[\ell]} \hat{X}_j * \hat{Y}_{1,j}^{m}) e(h^{\rho_2}, \prod_{j\in[n]} \hat{Y}_{2,j}) = e(s, \hat{\tau}^*)$,

$e(h, \prod_{i\in[\ell]} \hat{X}_j^{\frac{1}{\alpha}} * \hat{Y}_{1,j}^{\frac{1}{\alpha}m}) e(h^{\rho_2}, \prod_{j\in[n]} \hat{Y}_{2,j}^{\frac{1}{\alpha}}) = e(s^{\frac{1}{\alpha}}, \hat{\tau}^*)$,

We know that $(\mathsf{vk}_1^*)^{\frac{1}{\alpha}} = \mathsf{vk}'$ and so we can use Lemma 1 with $\mathsf{avk}^{\dagger} = \mathsf{avk}^{\frac{1}{\alpha}}, \sigma^{\dagger} = (h, s^{\frac{1}{\alpha}})$ to reduce this to breaking the GPS.

D.2 Proof of Theorem 2

The proof of tag class-hiding follows exactly from the message class hiding of the FHS scheme in [39].

Public key class-hiding proof for AtoSa. To simplify things, whenever the adversary makes a query to either of their oracles, we'll assume the adversary makes a query to both simultaneously. After one query, the ggm adversary in the public-key class-hiding game has elements: $\hat{X}_0$, $\hat{Y}_{1,0}$, $\hat{Y}_{2,0}$, $\hat{X}_{1+b}$, $\hat{Y}_{1,1+b}$, $\hat{Y}_{2,1+b}$, $h_{1,0} = P^{r_1}$, $s_{1,0} = P^{r_1(x_0+y_{1,0}m_1)}$, $h_{1,1} = P^{r_1}$, $s_{1,1} = P^{r_1(x_1+y_{1,1+b}m_1)}$. Here we've compacted the notation from the public-key class-hiding game by making vk_2^0 equal to $(\hat{X}_1, \hat{Y}_{1,1}, \hat{Y}_{2,1})$ and vk_2^1 equal to $(\hat{X}_2, \hat{Y}_{1,2}, \hat{Y}_{2,2})$. Rewriting this with discrete log, the adversary sees encodings of the polynomials: $\hat{x}_0$, $\hat{y}_{1,0}$, $\hat{y}_{2,0}$, $\hat{x}_{1+b}$, $\hat{y}_{1,1+b}$, $\hat{y}_{2,1+b}$, $\mathsf{dlog}_P(h_{1,0}) = r_1$, $\mathsf{dlog}_P(s_{1,0}) = r_1(x_0 + y_{1,0}m_1)$, $\mathsf{dlog}_P(h_{1,1}) = r_1$, $\mathsf{dlog}_P(s_{1,1}) = r_1(x_1 + y_{1,1+b}m_1)$. Where x_*, y_*, and r_* are variables. Because the message space of this signature is in $\mathbb{Z}_p^*$, the adversary cannot base future signatures on previous group elements. Thus, each future query looks as such: $\hat{x}_0$, $\hat{y}_{1,0}$, $\hat{y}_{2,0}$, $\hat{x}_{1+b}$, $\hat{y}_{1,1+b}$, $\hat{y}_{2,1+b}$, $\mathsf{dlog}_P(h_{i,0}) = r_i$, $\mathsf{dlog}_P(s_{i,0}) = r_i(x_0 + y_{1,0}m_i)$, $\mathsf{dlog}_P(h_{i,1}) = r_i$, $\mathsf{dlog}_P(s_{i,1}) = r_i(x_1 + y_{1,1+b}m_i)$.

When $b = 1$, we know that these are related by some randomization factor α as : $\hat{x}_0$, $\hat{y}_{1,0}$, $\hat{y}_{2,0}$, $\alpha\hat{x}_0$, $\alpha\hat{y}_{1,0}$, $\alpha\hat{y}_{2,0}$, $\mathsf{dlog}_P(h_{i,0}) = r_i$, $\mathsf{dlog}_P(s_{i,0}) = r_i(x_0 + y_{1,0}m_i)$, $\mathsf{dlog}_P(h_{i,1}) = r_i$, $\mathsf{dlog}_P(s_{i,1}) = r_i(\alpha x_0 + \alpha y_{1,0}m_i)$.

When $b = 0$, they are not: $\hat{x}_0$, $\hat{y}_{1,0}$, $\hat{y}_{2,0}$, $\hat{x}_2$, $\hat{y}_{1,2}$, $\hat{y}_{2,2}$, $\mathsf{dlog}_P(h_{i,0}) = r_i$, $\mathsf{dlog}_P(s_{i,0}) = r_i(x_0 + y_{1,0}m_i)$, $\mathsf{dlog}_P(h_{i,1}) = r_i$, $\mathsf{dlog}_P(s_{i,1}) = r_i(x_2 + y_{1,2}m_i)$.

We can see that these are made up of distinct polynomials (as long as each $m_i \neq 0$) and thus the encodings will never collide when making queries and thus all encodings will look random regardless of the choice of b. No r_i is reused for a different m_i due to auxiliary binding in Definition 9. This means that any signatures on the same message will be identical and thus each distinct signature will have a new r_i and are thus independent.

Origin-hiding proof for AtoSa. Origin-hiding of ConvertSig is straightforward and clear that outputs of ConvertSig looks like a fresh signature σ' in $\mathbb{G}_1^* \times \mathbb{G}_1^*$ as: For $\omega \in \mathbb{Z}_p^*$, $\mathsf{ConvertSig}(\mathsf{vk}, m, \mathbf{T}, \sigma, \omega)$ outputs (h', s^{ω}), which is a valid signature for $\mathsf{vk}' \xleftarrow{\$} \mathsf{ConvertVK}(\mathsf{vk}, \omega)$ (a uniformly random element of $[\mathsf{vk}]_{\mathcal{R}_{\mathsf{vk}}}$). Next, for $v \in \mathbb{Z}_p^*$, $\mathsf{RndSigTag}(\mathsf{vk}, \mathbf{T}, m, (h', s^{\omega}), v)$, output the randomized signature and tag: $\sigma' = (h'^{v}, s^{v})$, and $\mathbf{T}' = \mathbf{T}^{\mu}$, which are uniformly random elements in the respective space satisfying $\mathsf{Verify}(\mathsf{ConvertVK}(\mathsf{vk}, \omega), \mathbf{T}', m, \sigma') = 1$. So the output is distributed the same as the output of Sign (h', s) for tag $\mathbf{T}$ and vk.

Origin-hiding of RndSigTag is similar to the above, one can simply do the randomization of tag/signature for $v \in \mathbb{Z}_p^*$ as $\mathsf{RndSigTag}(\mathsf{vk}, \mathbf{T}^{\mu}, m, (h'^{v}, s^{v}), v)$, output the randomized signature/tag: $\sigma' = (h'^{v}, s^{v})$, and $\mathbf{T}' = \mathbf{T}^{\mu} = (T_1^{\mu}, T_2^{\mu})$, which are uniformly random elements in the respective space satisfying $\mathsf{Verify}(\mathsf{vk}, \mathbf{T}', m, \sigma') = 1$.

D.3 Proof of Theorem 3

Public key class-hiding proof for ATMs. First, we need to prove that no message signed in either of the oracles can include group elements from a previous signature. We can see that this condition is enforced by looking at the tag-based DH message space in Definition 10. If an adversary receives a signature from the signing oracle, then, to include it in a new $\mathcal{M}^{H}_{\mathsf{TDH}}$ message, they must ensure that:

$$e(M_i, \hat{P}) = e(T_i, N_i) \tag{3}$$

The adversary must also know the discrete log between h and T_i, which can be checked using VerifyTag, which verifies that $T_i = h^{\tau_i}$ for all $i \in \{1, 2\}$. Thus we have that:

$$e(M_i, \hat{P}) = e(h^{\tau_i}, N_i) \tag{4}$$

This means that M_i must be computed on h. But h must not have been used in a distinct signature query before because h is always computed on $(\mathbf{M}, \mathbf{N})$ and thus, changing $(\mathbf{M}, \mathbf{N})$ even by randomizing it to another equivalence class ensures that h changes. There is a similar argument with $\mathbf{T}$ that ensures two queries on the same h means they must have the exact same message and tag.

This means that a generic (GGM) adversary must know the discrete log between h and M_i. Now, because of Equation 3 and the fact that T_i is computed on h, we know that $\mathsf{dlog}_h(M_i) = \mathsf{dlog}_h(T_i)\mathsf{dlog}_{\mathbf{P}}(N_i)$ and these values are in $\mathbb{Z}_p^*$, we can simply compute: $\frac{\mathsf{dlog}_h(M_i)}{\mathsf{dlog}_h(T_i)} = \mathsf{dlog}_{\mathbf{P}}(N_i)$. Thus, the adversary must know the discrete log of N_i.

We can now analyse the adversary's queries in the GGM knowing that the adversary must know the discrete logs of the messages they send to the signing oracle, similar to EUF-CoMA definition in [38]. Similar to our proof of public-key class-hiding for AtoSa, we'll simplify things by giving the adversary signatures from both oracles whenever the adversary makes a query to either of their oracles.

So, when the adversary begins the public-key class-hiding game, they will receive two public keys. The first will be random elements: $\hat{X}_0$, $\hat{Y}_{1,0}$, $\hat{Y}_{2,0}$, $\hat{Z}_{1,0}$, $\hat{Z}_{2,0}$ and the second we'll label: $\hat{X}_{1+b}$, $\hat{Y}_{1,1+b}$, $\hat{Y}_{2,1+b}$, $\hat{Z}_{1,1+b}$, $\hat{Z}_{2,1+b}$ Where: $\hat{X}_1 = \alpha\hat{X}_0$, $\hat{Y}_{1,1} = \alpha\hat{Y}_{1,0}$, $\hat{Y}_{2,1} = \alpha\hat{Y}_{2,0}$, $\hat{Z}_{1,1} = \alpha\hat{Z}_{1,0}$, $\hat{Z}_{2,1} = \alpha\hat{Z}_{2,0}$ and: $\hat{X}_2$, $\hat{Y}_{1,2}$, $\hat{Y}_{2,2}$, $\hat{Z}_{1,2}$, $\hat{Z}_{2,2}$ is another random key. The adversary can now make a number of signing oracle queries, receiving signatures from both keys simultaneously. Because we've established that these cannot depend on previous signature queries, we can predetermine the set of messages the adversary wants to sign. Below we show the structure of the signatures received from each oracle. $\sigma_{i,0}$ is the signature received from the first oracle in query i both when the secret bit $b = 0$ and $b = 1$. $\sigma_{i,1}$ is the signature retrieved from the second oracle when $b = 0$. $\sigma_{i,2}$ is the signature retrieved from the second oracle when $b = 1$. $h_{i,*}, b_{i,*}, s_{i,*}$ are notated similarly where the second subscript denotes which oracle and secret challenger bit this is retrieved from. Thus, on each query, the adversary receives:

$$\sigma_{i,0} = \begin{pmatrix} h_{i,0} = P^{r_{i,0}}, b_{i,0} = P^{r_{i,0}\rho_i z_{1,0} + r_{i,0}\rho_i z_{2,0}}, \\ s_{i,0} = P^{r_{i,0}x_0 + m_{i,0}y_{1,0} + m_{i,1}y_{2,0}} \end{pmatrix}$$

and either:

$$\sigma_{i,1} = \begin{pmatrix} h_{i,1} = P^{r_{i,1}}, b_{i,1} = P^{r_{i,1}\rho_i \alpha z_{1,0} + r_{i,0}\rho_i \alpha z_{2,0}}, \\ s_{i,1} = P^{r_{i,1}\alpha x_0 + m_{i,0}\alpha y_{1,0} + m_{i,1}\alpha y_{2,0}} \end{pmatrix}$$

or:

$$\sigma_{i,2} = \begin{pmatrix} h_{i,2} = P^{r_{i,2}}, b_{i,2} = P^{r_{i,2}\rho_i z_{1,2} + r_{i,2}\rho_i \alpha z_{2,2}}, \\ s_{i,2} = P^{r_{i,2}x_2 + m_{i,0}y_{1,2} + m_{i,1}y_{2,2}} \end{pmatrix}$$

We can look at the discrete logs of these:

$\mathsf{dlog}_P(\sigma_{i,0}) = \left(r_{i,0}, r_{i,0}\rho_i z_{1,0} + r_{i,0}\rho_i z_{2,0}, r_{i,0}x_0 + m_{i,0}y_{1,0} + m_{i,1}y_{2,0}\right)$
and either:
$\mathsf{dlog}_P(\sigma_{i,1}) = \left(r_{i,1}, r_{i,1}\rho_i \alpha z_{1,0} + r_{i,0}\rho_i \alpha z_{2,0}, r_{i,1}\alpha x_0 + m_{i,0}\alpha y_{1,0} + m_{i,1}\alpha y_{2,0}\right)$
or:
$\mathsf{dlog}_P(\sigma_{i,2}) = \left(r_{i,2}, r_{i,2}\rho_i z_{1,2} + r_{i,2}\rho_i \alpha z_{2,2}, r_{i,2}x_2 + m_{i,0}y_{1,2} + m_{i,1}y_{2,2}\right)$
Depending on the challenger secret bit, b.

We can see that these are formally distinct polynomials as long as the user provided values are non-zero which can be checked during signing. Thus, their encoding in the GGM are distinct and the adversary will never be able to match values. In $\mathbb{G}_2$, the advesary only ever receives the verification keys, which they cannot use to distinguish without guessing the randomization of the equivalence class, α, which is from an exponential space, $\mathbb{Z}_p^*$. The adversary can attempt to use the pairing operation to distinguish the games. This allows the adversary to multiply these polynomials with the secrets x, y_1, y_2, z_1, z_2 of the issuer keys. Leading to the following values in $\mathbb{G}_T$:

So, in the case of $b = 0$, the adversary can compute elements in $\mathbb{G}_T$ with the discrete log: Here, we're zero indexing all arrays (notably, elements in public keys like $\hat{Y}_1$, $\hat{Y}_2$ which are now $\hat{Y}_0$ and $\hat{Y}_1$) to make things simpler). We have

$\mathsf{dlog}_{e(P,\hat{P})}(e(h_i^{\eta_\mathsf{L}} b_{i,0}^{\beta_\mathsf{L}} s_{i,0}^{\delta_\mathsf{L}}, (\hat{X}_0)^{\chi_\mathsf{L}} (\hat{Y}_{0,0})^{\gamma_{0,\mathsf{L}}} (\hat{Y}_{1,0})^{\gamma_{1,\mathsf{L}}} (\hat{Z}_{0,0})^{\zeta_{0,\mathsf{L}}} (\hat{Z}_{1,0})^{\zeta_{1,\mathsf{L}}})$.

$$\sigma = (h, b = \prod_{j\in[2]} h^{\rho_j z_j}, s_{i,0} = (h^x \prod_{j\in[2]} M_j^{y_j})).$$

$$\sigma = (h_i = P^{r_i}, b_{i,0} = P^{r_i\rho_0 z_{0,0} + r_i\rho_1 z_{1,0}}, s_{i,0} = P^{r_i x_0 + r_i\rho_0 m_0 y_{0,0} + r_i\rho_1 m_1 y_{0,0}}).$$

$= (r_i\eta_\mathsf{L} + \beta_\mathsf{L} r_i\rho_0 z_{0,0} + \beta_\mathsf{L} r_i\rho_1 z_{1,0} + \delta_\mathsf{L} r_i x_{0,0} + \delta_\mathsf{L} r_i\rho_0 m_0 y_{0,0} + \delta_\mathsf{L} r_i\rho_1 m_1 y_{1,0}) * (x_{0,0}\chi + y_{0,0}\gamma_{0,\mathsf{L}} + y_{1,0}\gamma_{1,\mathsf{L}} + z_{0,0}\zeta_{0,\mathsf{L}} + z_{1,0}\zeta_{1,\mathsf{L}})$

We'll do the same for queries in the right oracle when $b = 0$: $= (r_i\eta_\mathsf{R} + \beta_\mathsf{R} r_i\rho_0\alpha z_{0,0} + \beta_\mathsf{R} r_i\rho_1\alpha z_{1,0} + \delta_\mathsf{R} r_i\alpha x_{0,0} + \delta_\mathsf{R} r_i\rho_0 m_0\alpha y_{0,0} + \delta_\mathsf{R} r_i\rho_1 m_1\alpha y_{1,0}) * (\alpha x_{0,0}\chi + \alpha y_{0,0}\gamma_{0,\mathsf{R}} + \alpha y_{1,0}\gamma_{1,\mathsf{R}} + \alpha z_{0,0}\zeta_{0,\mathsf{R}} + \alpha z_{1,0}\zeta_{1,\mathsf{R}})$

And right oracle queries when $b = 1$: $= (r_i\eta_\mathsf{R} + \beta_\mathsf{R} r_i\rho_0 z_{0,2} + \beta_\mathsf{R} r_i\rho_1 z_{1,2} + \delta_\mathsf{R} r_i x_{0,2} + \delta_\mathsf{R} r_i\rho_0 m_0 y_{0,2} + \delta_\mathsf{R} r_i\rho_1 m_1 y_{1,2}) * (x_{0,2}\chi + y_{0,2}\gamma_{0,\mathsf{R}} + y_{1,2}\gamma_{1,\mathsf{R}} + z_{0,2}\zeta_{0,\mathsf{R}} + z_{1,2}\zeta_{1,\mathsf{R}})$

Now, we add the elements to determine exactly what the adversary can compute. In $b = 0$:
$(r_i\eta_\mathsf{L} + \beta_\mathsf{L} r_i\rho_0 z_{0,0} + \beta_\mathsf{L} r_i\rho_1 z_{1,0} + \delta_\mathsf{L} r_i x_{0,0} + \delta_\mathsf{L} r_i\rho_0 m_0 y_{0,0} + \delta_\mathsf{L} r_i\rho_1 m_1 y_{1,0} + r_i\eta_\mathsf{R} + \beta_\mathsf{R} r_i\rho_0\alpha z_{0,0} + \beta_\mathsf{R} r_i\rho_1\alpha z_{1,0} + \delta_\mathsf{R} r_i\alpha x_{0,0} + \delta_\mathsf{R} r_i\rho_0 m_0\alpha y_{0,0} + \delta_\mathsf{R} r_i\rho_1 m_1\alpha y_{1,0}) * (x_{0,0}\chi + y_{0,0}\gamma_{0,\mathsf{L}} + y_{1,0}\gamma_{1,\mathsf{L}} + z_{0,0}\zeta_{0,\mathsf{L}} + z_{1,0}\zeta_{1,\mathsf{L}} + \alpha x_{0,0}\chi + \alpha y_{0,0}\gamma_{0,\mathsf{R}} + \alpha y_{1,0}\gamma_{1,\mathsf{R}} + \alpha z_{0,0}\zeta_{0,\mathsf{R}} + \alpha z_{1,0}\zeta_{1,\mathsf{R}})$ (5)

And in $b = 1$:
$(r_i\eta_\mathsf{L} + \beta_\mathsf{L} r_i\rho_0 z_{0,0} + \beta_\mathsf{L} r_i\rho_1 z_{1,0} + \delta_\mathsf{L} r_i x_{0,0} + \delta_\mathsf{L} r_i\rho_0 m_0 y_{0,0} + \delta_\mathsf{L} r_i\rho_1 m_1 y_{1,0} + r_i\eta_\mathsf{R} + \beta_\mathsf{R} r_i\rho_0 z_{0,2} + \beta_\mathsf{R} r_i\rho_1 z_{1,2} + \delta_\mathsf{R} r_i x_{0,2} + \delta_\mathsf{R} r_i\rho_0 m_0 y_{0,2} + \delta_\mathsf{R} r_i\rho_1 m_1 y_{1,2}) * (x_{0,0}\chi + y_{0,0}\gamma_{0,\mathsf{L}} + y_{1,0}\gamma_{1,\mathsf{L}} + z_{0,0}\zeta_{0,\mathsf{L}} + z_{1,0}\zeta_{1,\mathsf{L}} + x_{0,2}\chi + y_{0,2}\gamma_{0,\mathsf{R}} + y_{1,2}\gamma_{1,\mathsf{R}} + z_{0,2}\zeta_{0,\mathsf{R}} + z_{1,2}\zeta_{1,\mathsf{R}})$ (6)

These are not idential polynomials in both cases, so we haven't proven this to be indistinguishable yet. We'll label the case where $b = 0$ as game 1 and the case where $b = 1$ as game 2. The adversary can only distinguish the two games if they can find $\rho_*, m_*, \eta_*, \beta_*, \delta_*, \zeta_*, \eta'_*, \beta'_*, \delta'_*, \zeta'_*$ such that the two equations are exact in one game and distinct in the other game. We can think of comparing two equations as subtracting one from the other and seeing if the resulting polynomial is the zero polynomial. This can only be done by canceling out terms. We will look at the terms in which the computed polynomials in $\mathbb{G}_2$ diverge. Any polynomial an adversary could create with two sets of chosen values, they can compute with one, thus, we can ignore the $\eta'_*, \beta'_*, \delta'_*, \zeta'_*$ values. Thus, we investigate ways that the adversary can create zero polynomials in both games and they to determine if there is a zero polynomial in one game that is not identically zero in the other.

After expanding out Equations 5 and 6, we can use automated proving to find that if: $\beta_\mathsf{L} = 0, \beta_\mathsf{R} = 0, \delta_\mathsf{L} = 0, \chi = 0, \delta_\mathsf{R} = 0$ then these games always result in the same polynomial. Thus, we do not need to consider other values. We will iterate through the cases where each of these is non-zero to prove that these polynomials are always distinct with the same chosen values in both games. We will use an automated proving script to prove claims about zeroing out polynomials. We'll call the list of these variables, V.

We can ensure that ρ_* and m_* are non zero (and not equal to each other) during signing by checking if their corresponding elements are equal to $1 \in \mathbb{G}_1$.

We will now step through individual terms in the expansion of Equations 5 and 6 and analyze them to tell when an adversary can obtain zero polynomials. The full polynomial can be computed using a script and is omitted from this appendix.

If χ is not equal to zero, we can look at the following terms to conclude that all the other values must be zero:
In game 1: $((\beta_{\mathsf{L}}\chi\rho_0)r_i x_{0,0}z_{0,0})$.
In game 1: $((\beta_{\mathsf{L}}\chi\rho_1)r_i x_{0,0}z_{1,0})$.
In game 1: $((\chi\delta_{\mathsf{L}})r_i x_{0,0}x_{0,0})$.
In game 1: $((\chi\delta_{\mathsf{L}}\rho_0 m_0)r_i x_{0,0}y_{0,0})$.
In game 1: $((\chi\delta_{\mathsf{L}}\rho_1 m_1)r_i x_{0,0}y_{1,0})$.
In game 1: $((\beta_{\mathsf{R}}\chi\rho_0)\alpha r_i x_{0,0}z_{0,0})$.
In game 1: $((\beta_{\mathsf{R}}\chi\rho_1)\alpha r_i x_{0,0}z_{1,0})$.
In game 1: $((\chi\delta_{\mathsf{R}})\alpha r_i x_{0,0}x_{0,0})$.
In game 1: $((\chi\delta_{\mathsf{R}}\rho_0 m_0)\alpha r_i x_{0,0}y_{0,0})$.
In game 1: $((\chi\delta_{\mathsf{R}}\rho_1 m_1)\alpha r_i x_{0,0}y_{1,0})$.
In game 2: $((\beta_{\mathsf{L}}\chi\rho_0)r_i x_{0,0}z_{0,0})$.
In game 2: $((\beta_{\mathsf{L}}\chi\rho_0)r_i x_{0,2}z_{0,0})$.
In game 2: $((\beta_{\mathsf{L}}\chi\rho_1)r_i x_{0,0}z_{1,0})$.
In game 2: $((\beta_{\mathsf{L}}\chi\rho_1)r_i x_{0,2}z_{1,0})$.
In game 2: $((\chi\delta_{\mathsf{L}})r_i x_{0,0}x_{0,0})$.
In game 2: $((\chi\delta_{\mathsf{L}}\rho_0 m_0)r_i x_{0,0}y_{0,0})$.
In game 2: $((\chi\delta_{\mathsf{L}}\rho_0 m_0)r_i x_{0,2}y_{0,0})$.
In game 2: $((\chi\delta_{\mathsf{L}}\rho_1 m_1)r_i x_{0,0}y_{1,0})$.
In game 2: $((\chi\delta_{\mathsf{L}}\rho_1 m_1)r_i x_{0,2}y_{1,0})$.
In game 2: $((\beta_{\mathsf{R}}\chi\rho_0)r_i x_{0,0}z_{0,2})$.
In game 2: $((\beta_{\mathsf{R}}\chi\rho_0)r_i x_{0,2}z_{0,2})$.
In game 2: $((\beta_{\mathsf{R}}\chi\rho_1)r_i x_{0,0}z_{1,2})$.
In game 2: $((\beta_{\mathsf{R}}\chi\rho_1)r_i x_{0,2}z_{1,2})$.
In game 2: $((\chi\delta_{\mathsf{R}})r_i x_{0,2}x_{0,2})$.
In game 2: $((\chi\delta_{\mathsf{R}}\rho_0 m_0)r_i x_{0,0}y_{0,2})$.
In game 2: $((\chi\delta_{\mathsf{R}}\rho_0 m_0)r_i x_{0,2}y_{0,2})$.
In game 2: $((\chi\delta_{\mathsf{R}}\rho_1 m_1)r_i x_{0,0}y_{1,2})$.
In game 2: $((\chi\delta_{\mathsf{R}}\rho_1 m_1)r_i x_{0,2}y_{1,2})$.
We can see that all the values in V are must be zero if χ is non-zero, since if any are non-zero, we do not obtain the zero polynomial.

We can now look at the distinct combinations of adversarially chosen values including χ in the games:

In game 1 but not game 2: $((\beta_{\mathsf{L}}\chi\rho_0 + \beta_{\mathsf{R}}\chi\rho_0)\alpha r_i x_{0,0}z_{0,0})$.
In game 1 but not game 2: $((\beta_{\mathsf{L}}\chi\rho_1 + \beta_{\mathsf{R}}\chi\rho_1)\alpha r_i x_{0,0}z_{1,0})$.
In game 1 but not game 2: $((\chi\delta_{\mathsf{L}}\rho_0 m_0 + \chi\delta_{\mathsf{R}}\rho_0 m_0)\alpha r_i x_{0,0}y_{0,0})$.
In game 1 but not game 2: $((\chi\delta_{\mathsf{L}}\rho_1 m_1 + \chi\delta_{\mathsf{R}}\rho_1 m_1)\alpha r_i x_{0,0}y_{1,0})$.
Because $\rho_0 \neq 0, \rho_1 \neq 0$ and $\rho_0 \neq \rho_1$, this ensures that these polynomials are non-zero when all other values besides χ in V are 0. Thus, the adversary cannot zero out a term in any of these cases. Thus, in the adversary's assumed query which allows them to distinguish, if χ is non-zero, another variable in V besides χ must be non-zero. But, we previously proved that if χ is non-zero, all the other values must be zero, thus we have a contradiction if $\chi \neq 0$, and so χ must be zero in this distinguishing polynomial.

If δ_{R} is non-zero, we can see that γ_* must be zero because of the following terms:
In game 1: $((\delta_{\mathsf{R}}\gamma_{0,\mathsf{L}}\rho_0 m_0)\alpha r_i y_{0,0}y_{0,0})$.
In game 1: $((\delta_{\mathsf{R}}\gamma_{0,\mathsf{R}}\rho_0 m_0)\alpha\alpha r_i y_{0,0}y_{0,0})$.
In game 1: $((\delta_{\mathsf{R}}\gamma_{1,\mathsf{L}}\rho_1 m_1)\alpha r_i y_{1,0}y_{1,0})$.
In game 1: $((\delta_{\mathsf{R}}\gamma_{1,\mathsf{R}}\rho_1 m_1)\alpha\alpha r_i y_{1,0}y_{1,0})$.
In game 2: $((\delta_{\mathsf{R}}\gamma_{0,\mathsf{L}})r_i x_{0,2}y_{0,0})$.
In game 2: $((\delta_{\mathsf{R}}\gamma_{1,\mathsf{L}})r_i x_{0,2}y_{1,0})$.
In game 2: $((\delta_{\mathsf{R}}\gamma_{0,\mathsf{R}}\rho_0 m_0)r_i y_{0,2}y_{0,2})$.
In game 2: $((\delta_{\mathsf{R}}\gamma_{1,\mathsf{R}}\rho_1 m_1)r_i y_{1,2}y_{1,2})$.

Through automated proofs, we can see that when γ_* are all zero, we have that all terms with δ_R in them are identical. This implies that if δ_R is non-zero, another value in V must be non-zero.

We have a symmetric case for δ_L, implying that if δ_L is non-zero, another value in V must be non-zero:
In game 1: $((\delta_\mathsf{L}\gamma_{0,\mathsf{L}}\rho_0 m_0)r_i y_{0,0}y_{0,0})$.
In game 1: $((\delta_\mathsf{L}\gamma_{0,\mathsf{R}}\rho_0 m_0)\alpha r_i y_{0,0}y_{0,0})$.
In game 1: $((\delta_\mathsf{L}\gamma_{1,\mathsf{L}}\rho_1 m_1)r_i y_{1,0}y_{1,0})$.
In game 1: $((\delta_\mathsf{L}\gamma_{1,\mathsf{R}}\rho_1 m_1)\alpha r_i y_{1,0}y_{1,0})$.
In game 2: $((\delta_\mathsf{L}\gamma_{0,\mathsf{R}})r_i x_{0,0}y_{0,2})$.
In game 2: $((\delta_\mathsf{L}\gamma_{1,\mathsf{R}})r_i x_{0,0}y_{1,2})$.
In game 2: $((\delta_\mathsf{L}\gamma_{0,\mathsf{L}}\rho_0 m_0)r_i y_{0,0}y_{0,0})$.
In game 2: $((\delta_\mathsf{L}\gamma_{1,\mathsf{L}}\rho_1 m_1)r_i y_{1,0}y_{1,0})$.

If β_R is non-zero, we again get that the γ values are non-zero:
In game 1: $((\beta_\mathsf{R}\gamma_{0,\mathsf{L}}\rho_0)\alpha r_i y_{0,0}z_{0,0})$.
In game 1: $((\beta_\mathsf{R}\gamma_{1,\mathsf{L}}\rho_0)\alpha r_i y_{1,0}z_{0,0})$.
In game 1: $((\beta_\mathsf{R}\gamma_{0,\mathsf{R}}\rho_0)\alpha\alpha r_i y_{0,0}z_{0,0})$.
In game 1: $((\beta_\mathsf{R}\gamma_{1,\mathsf{R}}\rho_0)\alpha\alpha r_i y_{1,0}z_{0,0})$.
In game 2: $((\beta_\mathsf{R}\gamma_{0,\mathsf{L}}\rho_0)r_i y_{0,0}z_{0,2})$.
In game 2: $((\beta_\mathsf{R}\gamma_{1,\mathsf{L}}\rho_0)r_i y_{1,0}z_{0,2})$.
In game 2: $((\beta_\mathsf{R}\gamma_{0,\mathsf{R}}\rho_0)r_i y_{0,2}z_{0,2})$.
In game 2: $((\beta_\mathsf{R}\gamma_{1,\mathsf{R}}\rho_0)r_i y_{1,2}z_{0,2})$.

But, different from the δ_* values, if the γ values are non-zero, we still could have a distinct polynomial if only β_R is non-zero:
In game 1 but not game 2: $((\beta_\mathsf{R}\rho_0\zeta_{1,\mathsf{L}} + \beta_\mathsf{R}\rho_1\zeta_{0,\mathsf{L}})\alpha r_i z_{0,0}z_{1,0})$.
In game 2 but not game 1: $((\beta_\mathsf{R}\rho_0\zeta_{1,\mathsf{L}})r_i z_{0,2}z_{1,0})$.
In game 2 but not game 1: $((\beta_\mathsf{R}\rho_1\zeta_{0,\mathsf{L}})r_i z_{0,0}z_{1,2})$.

The adversary could get a zero polynomial if $\rho_0\zeta_{1,\mathsf{L}} + \rho_1\zeta_{0,\mathsf{L}} = 0$, which implies that either $\zeta_{1,\mathsf{L}}$ is non-zero or $\zeta_{0,\mathsf{L}}$ is non-zero. In both cases, we have that β_R is zero, since we have the following terms:
In game 1: $((\beta_\mathsf{R}\rho_0\zeta_{1,\mathsf{L}})\alpha r_i z_{0,0}z_{1,0})$.
In game 2: $((\beta_\mathsf{R}\rho_0\zeta_{1,\mathsf{L}})r_i z_{0,2}z_{1,0})$.
Or:
In game 1: $((\beta_\mathsf{R}\rho_0\zeta_{0,\mathsf{L}})\alpha r_i z_{0,0}z_{0,0})$.
In game 2: $((\beta_\mathsf{R}\rho_0\zeta_{0,\mathsf{L}})r_i z_{0,0}z_{0,2})$.

Thus, this leads to a contradiction, so we can't have either of these be non-zero.

If they're both zero ($\zeta_{0,\mathsf{L}}$ and $\zeta_{1,\mathsf{L}}$), we find that no term with β_R is distinct in the polynomial of game 1 or game 2, implying that another variable in V must be non-zero.

If β_L is non-zero, we again find that the γ values must be zero to cancel the following terms out:
In game 1: $((\beta_\mathsf{L}\gamma_{0,\mathsf{L}}\rho_0)r_i y_{0,0}z_{0,0})$.
In game 1: $((\beta_\mathsf{L}\gamma_{1,\mathsf{L}}\rho_0)r_i y_{1,0}z_{0,0})$.
In game 1: $((\beta_\mathsf{L}\gamma_{0,\mathsf{R}}\rho_0)\alpha r_i y_{0,0}z_{0,0})$.
In game 1: $((\beta_\mathsf{L}\gamma_{1,\mathsf{R}}\rho_0)\alpha r_i y_{1,0}z_{0,0})$.
In game 2: $((\beta_\mathsf{L}\gamma_{0,\mathsf{L}}\rho_0)r_i y_{0,0}z_{0,0})$.
In game 2: $((\beta_\mathsf{L}\gamma_{1,\mathsf{L}}\rho_0)r_i y_{1,0}z_{0,0})$.
In game 2: $((\beta_\mathsf{L}\gamma_{0,\mathsf{R}}\rho_0)r_i y_{0,2}z_{0,0})$.
In game 2: $((\beta_\mathsf{L}\gamma_{1,\mathsf{R}}\rho_0)r_i y_{1,2}z_{0,0})$.

Thus, we get the following distinct terms:
In game 1 but not game 2: $((\beta_\mathsf{L}\rho_0\zeta_{1,\mathsf{R}} + \beta_\mathsf{L}\rho_1\zeta_{0,\mathsf{R}})\alpha r_i z_{0,0}z_{1,0})$.
In game 2 but not game 1: $((\beta_\mathsf{L}\rho_0\zeta_{1,\mathsf{R}})r_i z_{0,0}z_{1,2})$.
In game 2 but not game 1: $((\beta_\mathsf{L}\rho_1\zeta_{0,\mathsf{R}})r_i z_{0,2}z_{1,0})$.

Like in the case with β_{L}, the adversary could get a zero polynomial if $\rho_0\zeta_{1,\mathsf{R}} + \rho_1\zeta_{0,\mathsf{R}} = 0$, which implies that either $\zeta_{1,\mathsf{R}}$ is non-zero or $\zeta_{0,\mathsf{R}}$ is non-zero. In both cases, we have that β_{L} is zero, since we have the following terms:
In game 1: $((\beta_{\mathsf{L}}\rho_0\zeta_{0,\mathsf{R}})\alpha r_i z_{0,0} z_{0,0})$.
In game 2: $((\beta_{\mathsf{R}}\rho_0\zeta_{0,\mathsf{R}}) r_i z_{0,2} z_{0,2})$.
In game 1: $((\beta_{\mathsf{R}}\rho_0\zeta_{1,\mathsf{R}})\alpha\alpha r_i z_{0,0} z_{1,0})$.
In game 2: $((\beta_{\mathsf{R}}\rho_0\zeta_{1,\mathsf{R}}) r_i z_{0,2} z_{1,2})$.

If they're both zero ($\zeta_{0,\mathsf{R}}$ and $\zeta_{1,\mathsf{R}}$), we find that no term with β_{L} is distinct in the polynomial of game 1 or game 2, implying that another variable in V must be non-zero.

Thus, we must have that some pair in $\beta_{\mathsf{L}}, \beta_{\mathsf{R}}, \delta_{\mathsf{L}}, \delta_{\mathsf{R}}$ must be non-zero.

When $\beta_{\mathsf{L}}, \beta_{\mathsf{R}}$ are both non-zero, we get that all γ_* ζ_* are zero (from combinations of previous results). When we zero all these out, (including χ) we find that the polynomial are identical.

When $\beta_{\mathsf{L}}, \delta_{\mathsf{R}}$ are both non-zero, we get that all γ_* values are zeto and all $\zeta_{*,\mathsf{R}}$ are zero. We know from previous results that this means all terms with β_{L} and δ_{R} are identical in game 1 and 2, so in this case, we need another value. There is a symmetric argument for $\beta_{\mathsf{L}}, \delta_{\mathsf{R}}$.

When $\delta_{\mathsf{L}}, \delta\mathsf{R}$ are both non-zet, we get that all γ_* are zero, implying that all terms with δ_* are identical. This, we need another value in V to be non-zero.

As we keep adding more non-zero values in V, we can only ensure that more single sum terms must be zero. Thus, we always zero out the possibilities for both polynomials.

Because the adversary cannot create a identically zero polynomial in one game when in the other game, the polynomial is no identically zero, they cannot find two distinct polynomials that match in one game but not the other. Thus, they cannot distinguish the two games.

Origin-hiding proof. We provide proof of perfect adaption of signatures as: Let $(\mathbf{M}, \mathbf{N}) \in (\mathbb{G}_2^*)^2 \times (\mathbb{G}_1^*)^2$, $\mathsf{vk} \in (\mathbb{G}_2^*)^4$ and $\mathsf{sk} = (y_i, x)_{i\in[3]} \in \mathbb{Z}_p^*$. A signature and tag $(h, b, s, \mathbf{T}) \in \mathbb{G}_1^* \times \mathbb{G}_1^* \times \mathbb{G}_1^* \times (\mathbb{G}_2^*)^2$ satisfying $\mathsf{Verify}(\mathsf{vk}, (\mathbf{M}, \mathbf{N}), \mathbf{T}, (h, b, s)) = 1$ is of the form $(h = P^r, h^{\sum \rho_i \cdot z_i}, h^x \cdot \prod M_i^{y_i}, (h^{\rho_1}, h^{\rho_2}))$ for $r \in \mathbb{Z}_p^*$ regarding to RO. $\mathsf{ChangRep}((\mathbf{M}, \mathbf{N}), \sigma, \mathbf{T}, (\mu, \upsilon))$ for $(\mu, \upsilon) \in \mathbb{Z}_p^*$, outputs

$$\left(h' \leftarrow h^{\mu\upsilon}, b' \leftarrow b^{\mu}, \hat{s}' \leftarrow \hat{s}^{\mu\upsilon}, \mathbf{T}' \leftarrow \mathbf{T}^{\mu}\right)$$

, which are uniformly random elements conditioned $\mathsf{Verify}(\mathsf{vk}, \mathbf{T}^{\mu}, (\mathbf{M}^{\mu\upsilon}, \mathbf{N}^{\upsilon}), \sigma') = 1$.

Note that as clear from the definition, ChangRep not only generates a new representative for the message but also produces a new representative for the tag that ensures the adapted signature is valid for both the new tag representative and the new message representative.

D.4 Proof of Theorem 5

We provide proof for the first and second constructions.

Lemma 3 (Unforgeability construction). *Let* ZKPOK *be a simulation-sound extractable ZKPoK, and* SPSEQ *be unforgeable signature, if* AtoSa *is unforgeable, then the* IhMA *construction Fig 6 is unforgeable.*

Proof. Intuitively, $\mathcal{A}$ has a potential ways of breaking unforgeability: if he can forge a AtoSa signature on the challenge public key (that is, $\mathcal{A}$ does not possession proper attributes, but it can perform verification by forging credentials). We show that if an adversary $\mathcal{A}$ can win the unforgeability game (Def. 18) with non-negligible probability. We then construct an adversary (reduction) $\mathcal{B}$ that breaks the unforgeability of AtoSa (Def. 4). Note that we can extract witness from ZKPOK and assume this will only fail with negligible probability. Lets us assume that $\mathsf{A}_i = a_i$ is an attribute for simplicity, now we show this reduction as follows:

Reduction. The reduction is straightforward. $\mathcal{B}$ interacts with a challenger $\mathcal{C}$ in the unforgeability game of AtoSa and $\mathcal{B}$ simulates the IhMA-unforgeability game for $\mathcal{A}$. $\mathcal{B}$ receives from $\mathcal{C}$ values $(\hat{X} = P^x, \hat{Y}_1 = \hat{P}^{y_1}, \hat{Y}_2 = \hat{P}^{y_2})$ and public parameters pp of BG. Next, $\mathcal{B}$ sets $\mathsf{vk}' = (\hat{X}, \hat{Y}_1, \hat{Y}_2)$ as the challenge key and sends $(\mathsf{pp}, \mathsf{vk}')$ to $\mathcal{A}$. All oracles are executed as in the real game, except for following ones which use the signing oracle in AtoSa instead of using the challenge signing key sk':

$\mathcal{O}^{\mathsf{User}}(u)$: $\mathcal{B}$ takes as input a user identity u. If $u \in \mathcal{HU}$ or $u \in \mathcal{CU}$, return $\perp$. Else, create a fresh entry u by running $(u\mathsf{sk}, u\mathsf{vk}) \leftarrow$ UKeyGen but create aux using commitments, adding u and $(u\mathsf{sk}, u\mathsf{vk}, \mathsf{aux})$ to the list $\mathcal{HU}$ and $\mathcal{L}_{uk}$, receptively. Return $u\mathsf{vk}$.

$\mathcal{O}^{\mathsf{ObtIss}}(i, u, \mathsf{A}_i)$: If $u \notin \mathcal{HU} \vee i \notin \mathcal{HCI} \cup \mathsf{vk}'$, return $\perp$. Else if $i\mathsf{vk} \neq \mathsf{vk}'$, find entries $((u\mathsf{sk} = \tau, \mathsf{aux}) \in \mathcal{L}_{uk}, i\mathsf{sk} \in \mathcal{HCI})$, and compute $\sigma_i \leftarrow \mathsf{Sign}(i\mathsf{sk}, \tau, \mathsf{aux}, \mathsf{A}_i)$ (note that with knowledge of $i\mathsf{sk}$, $\mathcal{B}$ can compute a signature on it's own). Else $i\mathsf{vk} = \mathsf{vk}'$, asks the query $\sigma_i \leftarrow \mathcal{O}^{\mathsf{Sign}}(\mathsf{A}_i, \mathsf{aux}, \tau)$ of AtoSa, which the oracle runs $\mathsf{Sign}(\mathsf{sk}', \tau, \mathsf{aux}, \mathsf{A}_i)$, adds the entry $(u, \mathsf{A}_i, \mathsf{cred}_i)$ to $\mathcal{L}_{\mathsf{cred}}$, where $\mathsf{cred}_i = (\sigma_i, \tau)$.

$\mathcal{O}^{\mathsf{Issue}}(i, u, \mathsf{A}_i)$: If $u \notin \mathcal{CU} \vee i \notin \mathcal{HCI} \cup \mathsf{vk}'$, it returns $\perp$. Else, if $i\mathsf{vk} \neq \mathsf{vk}'$, compute $\sigma_i \leftarrow \mathsf{Sign}(i\mathsf{sk}, \tau, \mathsf{aux}, \mathsf{A}_i)$. Else ask the query $\sigma_i \leftarrow \mathcal{O}^{\mathsf{Sign}}(\mathsf{A}_i, \mathsf{aux}, \tau)$ of AtoSa, add the entry $(u, \mathsf{A}_i, \mathsf{cred}_i)$ to $\mathcal{L}_{\mathsf{cred}}$, where $\mathsf{cred}_i = (\sigma, \tau)$.

Obviously, $\mathcal{B}$ handles any oracle query and never aborts (assuming a simulated ZKPOK for issuing and showing protocols). Thus, at the end of the game, $\mathcal{B}$ simulates all oracles perfectly for $\mathcal{A}$ who is able, with some probability, to prove possession of a credential on attributes D. To do this, $\mathcal{B}$ interacts with $\mathcal{A}$ as verifier in a showing protocol. If $\mathcal{A}$ outputs a valid showing proof as $(\mathsf{avk}, \sigma^* = (h^*, s^*), D, \mathsf{Nym}^* = \mathbf{T}^*)$ and conducting $\pi = \mathsf{ZKPOK}(\mathsf{Nym}^*)$ then $\mathcal{B}$ extracts from the proof of knowledge contained in the Show, lets called the value (ρ_1^*, ρ_2^*) related to the Nym^* (the tag tuple $(\mathbf{T}^*, (\rho_1^*, \rho_2^*))$) and stores all elements. Moreover, no credentials owned by corrupt users can be valid on this set of messages D (as $\mathcal{A}$ can win the unforgeability game). This means that, for all credentials cred_{ui} on A_{ui} and $(u\mathsf{sk})$ with $u \in \mathcal{CCU}$, we have $D \not\subseteq \cup_{i \in [\ell]} \mathsf{A}_{ui}$. From the definition we have that at least one of the key in $\mathsf{vk}_j \in \mathsf{avk}$ should be the challenge key $[\mathsf{vk}_j]_{\mathcal{R}} = [\mathsf{vk}']_{\mathcal{R}}$ such that the related attribute is in the set $\mathsf{A}_j \in D$. Finally we can find all $i\mathsf{sk} \in \mathcal{HCI} \cup \mathcal{CCI}$ corresponding to $[i\mathsf{vk}]_{\mathcal{R}} \in I'$ for all $i \in [\ell]$, and output $\mathsf{ask} = (i\mathsf{sk})_{i \in [\ell]}$. Note that even if adversarial keys are randomized, we can output initial secret keys registered in the list and reduction still works for the AtoSa scheme because of keys class $[\mathsf{vk}]_{\mathcal{R}}$. In all cases, this means that $(\mathsf{avk}, (\tau^*, \mathbf{T}^*), D, \mathsf{ask}, \sigma^*)$ is a valid forgery against our signature scheme, $\mathcal{B}$ breaks thus unforgeability of AtoSa which concludes our proof, assuming SPSEQ is unforgeable.

Lemma 4. *Let* ZKPOK *be a simulation-sound extractable ZKPoK, if* AtoSa *is origin-hiding of ConvertSig and public key class-hiding,* SPSEQ *is perfect adaption, then the* IhMA *construction in Fig 6 is anonymous and issuer hiding.*

Since these properties follow almost immediately from the zero-knowledge property and the privacy notions of the underlying signature AtoSa (Def. 2).

Anonymity. From the randomization (unlinkability) of the AtoSa signature, a tuple $(\sigma = (h', s), \mathbf{T} = (T_1, T_2))$ can be hidden by randomizing them with secret randoms $\mu \in \mathbb{Z}_p^*$ as $(\sigma = (h'^{\mu}, s^{\mu}), \mathbf{T}^{\mu})$, where does not leak any information about the initial tag/signature (tag acts as pseudonym in the interaction). In addition, in the Show protocol, the proof of knowledge of the witness (tag's secret part) is zero-knowledge. This also does not leak any information about secrets either. Consequently, a credential does not leak any information about $u\mathsf{sk}$ or $u\mathsf{vk}$. Note that the Anonymity experiment guarantees that the witnesses used when computing π are valid for both $b = 0$ and $b = 1$ and also the signature/tag is randomized correctly. This means that the Anonymity experiment is computationally indistinguishable from one where the ZKPOK simulator creates π. In the end, the view of $\mathcal{A}$ is independent of b. More formally we provide proof as follows:

Proof. The proof follows a sequence of games until a game where answers for the query to $\mathcal{O}_b^{\mathsf{Anon}}$ is independent of the bit b. Let S be the event, and for $i = 1, \ldots, n$, the construction defines an event S_i in Game_i. In Game_1 we simulate all ZKPoKs. In Game_2 we replace all ConvertTag and ConvertSig calls with freshly generated signatures and In Game_3 we replace the $\mathbf{T}$ with a representative element of the same class.

Game_0**:** The original game as given in Definition 19.

Game_1**:** As Game_0, except that the experiment runs $\mathcal{O}_b^{\mathsf{Anon}}$ as follows: All proofs $\mathsf{NIZK}(\mathsf{Nym}_p)$ in CredShow and Obtain respectively, are simulated.
$\mathsf{Game}_0 \rightarrow \mathsf{Game}_1$: By perfect zero-knowledge of NIZK, we have that

$$\Pr[S_1] = \Pr[S_0]$$

Game_2**:** As Game_1, except for the following changes. Let qu be (an upper bound on) the number of queries made to $\mathcal{O}^{\mathsf{User}}$. At the beginning Game, picks $w \leftarrow [qu]$ (it guesses that the user that owns the j_b-th credential is registered at the w-th call to $\mathcal{O}^{\mathsf{User}}$). Runs $\mathcal{O}^{\mathsf{User}}$, $\mathcal{O}^{\mathsf{CorruptU}}$ and $\mathcal{O}_b^{\mathsf{Anon}}(j_0, j_1, D)$ as follows:

- $\mathcal{O}^{\mathsf{User}}(u, S)$: As in Game_1, but if this is the w-th call then, setting $u* \leftarrow u$.
- $\mathcal{O}^{\mathsf{CorruptU}}(u)$: If $u \in \mathcal{CU}$ or $u \in \mathcal{O}_b^{\mathsf{Anon}}$, it returns $\perp$ (as in the previous games). If $u = u^*$ then the experiment stops and outputs a random bit $b' \xleftarrow{\$} \{0,1\}$. Otherwise, if $u \in \mathcal{HU}$, it returns user $u\mathsf{sk}$ and credentials and moves u from $\mathcal{HU}$ to $\mathcal{CU}$.
- $\mathcal{O}_b^{\mathsf{Anon}}(j_0, j_1, D)$: As in Game_2, except that if $u^* \neq \mathcal{L}_{\mathsf{cred}}[j_b]$, the experiment stops outputting $b' \xleftarrow{\$} \{0,1\}$.

$\mathsf{Game}_1 \rightarrow \mathsf{Game}_2$: By assumption, $\mathcal{O}_b^{\mathsf{Anon}}$ is called at least once with some input (j_0, j_1, D) such that $u_0 \leftarrow \mathcal{L}_{\mathsf{cred}}[j_0], u_1 \leftarrow \mathcal{L}_{\mathsf{cred}}[j_1] \in \mathcal{HU}$. If $u^* = u_b$ then $\mathcal{O}_b^{\mathsf{Anon}}$ does not abort and neither does $\mathcal{O}^{\mathsf{CorruptU}}$ (it cannot have been called on u_b before that call to $\mathcal{O}_b^{\mathsf{Anon}}$ (otherwise $u_b \notin \mathcal{HU}$); if called afterwards, it returns $\perp$, where $u^* \in \mathcal{O}_b^{\mathsf{Anon}}$). Since $u^* = [u_b]$ with probability $\frac{1}{q_u}$, the probability that the experiment does not abort is at least $\frac{1}{q_u}$, and thus

$$\Pr[S_2] \geq (1 - \frac{1}{q_u})\frac{1}{2} + \frac{1}{q_u} \cdot \Pr[S_1]$$

Game_3**:** As Game_2, except $\mathcal{O}_b^{\mathsf{Anon}}(j_0, j_1, D)$: pick a random $\mathbf{T} \leftarrow \mathcal{T}$ and performs the showing with $D = (d_i)_{i \in [k]}$. The only difference is the choice of $\mathbf{T}$.

$\mathsf{Game}_2 \rightarrow \mathsf{Game}_3$: The difference between two games is that we use the tag class hiding Def. 6, which indirectly implies DDH. Indeed, the reduction accepts $\mathbf{T}, \mathbf{T}^b$ from the tag class-hiding challenger and uses these for users. The oracles are simulated as in Game_2, except for the subsequent oracles, which are simulated as follows:

- $\mathcal{O}^{\mathsf{User}}(u, S)$: As in Game_1, but if this is the w-th call then, setting $u^* \leftarrow u$ it sets $u\mathsf{sk}[u] \leftarrow \perp$ and $u\mathsf{vk} \leftarrow \mathbf{T}$.
- $\mathcal{O}_b^{\mathsf{Anon}}(j_0, j_1, D)$: As in Game_2, except that for $u^* = \mathcal{L}_{\mathsf{cred}}[j_b]$, the experiment run show for $\mathbf{T}^b$ which is either $\mathbf{T}^{(0)} \xleftarrow{\$} \mathcal{T}$ or $\mathbf{T}^{(1)} \xleftarrow{\$} [\mathbf{T}]_{\mathcal{R}_\tau}$. Picks b and sends $(\mathbf{T}^b, \sigma_b, \pi)$ to $\mathcal{A}$, and receives b' form $\mathcal{A}$. Return b' as answer to the tag class-hiding game. We thus have:

$$|\Pr[S_2] - \Pr[S_3]| \leq \epsilon_{\mathsf{DDH}}(\lambda) + (1 + 2ql)\frac{1}{p}$$

Game_4**:** As Game_3, except that the experiment runs $\mathcal{O}_b^{\mathsf{Anon}}$ as follows: Like in Game_3, but for $\mu, \upsilon \in \mathbb{Z}_p^*$, all executions of RndSigTag and ConvertSig for the credential and tag $(u_b, \mathsf{A}', \sigma_b) \leftarrow \mathcal{L}_{\mathsf{cred}}[j_b]$ are replaced by freshly generated signatures (note that we can randomize signatures and output uniformly random elements in the respective spaces and we know the related secret keys). So, oracles are simulated as in Game_3.

$\mathsf{Game}_2 \rightarrow \mathsf{Game}_3$: By Origin-hiding (ConvertSig) and (ConvertTag), signatures obtained from RndSigTag and ConvertSig are identically distributed for all $(\mathsf{A}, \mathbf{T}, \mathsf{vk}, \sigma)$. We thus have

$$\Pr[S_3] = \Pr[S_4]$$

At the end, $\mathcal{O}_b^{\mathsf{Anon}}$ returns a fresh signature σ on a random tag and a simulated proof; the bit b is thus information-theoretically hidden from $\mathcal{A}$. probability analysis is similar to [39, 54].

Proof of issuer-hiding for $\mathsf{IhMA}_{\mathsf{AtoSa}}$**.** We can reduce issuer hiding (Definition 20) to public-key class hiding (Definition 5) by constructing a number of hybrids and games, 2ℓ hybrids where ℓ is the number of issuers in the issuer-hiding game.

We'll quickly give an overview of the proof, starting with an explanation of hybrid scheme 0 then explaining each subsequent hybrid. We let hybrid 0 be the original issuer-hiding game where we fix $b = 0$. In hybrid 1, we replace the second $i\mathsf{vk}$ with a key of the same equivalence class as the first (still keeping $b = 0$). As we look at higher number hybrids $(2, 3, 4, ...)$ they each replace more random keys with keys in the same equivalence class as the first. We will use public key class hiding (Definition 5) to prove that these are indistinguishable. We keep replacing $i\mathsf{vk}$ keys until hybrid ℓ where all the keys are in the same equivalence class. We can then make hybrid $\ell + 1$ switch the bit, b, to be $b = 1$ (while still keeping all the $i\mathsf{vk}$s in the same equivalence class). When all the issuer keys are in the same

equivalence class, we know that the difference between $b = 0$ and $b = 1$ is indistinguishable because of origin-hiding (Definition 15). We then let hybrid $\ell + 2$ introduce back a new distict verification key (so that all the verification keys are in the same equivalence class except for one). As the hybrids continue increase beyond ℓ we keep replacing our keys in the same equivalence class with distinct ones and reduce to public key class-hiding to prove indistinguishability. We define hybrids above ℓ this way (all with $b = 1$) until we get to hybrid 2ℓ where all the verification keys are distinct again, but the bit has been flipped $b = 1$. We can see that hybrid 0 is the case where $b = 0$ in the original issuer hiding game and hybrid 2ℓ is the case where $b = 1$ in the game, so if we can prove that all of these hybrids are indistinguishable from eachother, we will have proven issuer hiding. We show these hybrids in Definition 33.

Definition 33 (Hybrid j).

- *If $j \leq \ell$, let $k = j$ and if $j > \ell$, let $k = 2\ell - j$. Generate the first issuer key:* $\mathsf{vk}_0 \xleftarrow{\$} \mathsf{IKeyGen}(\mathsf{pp})$. *Let* $(\mathsf{vk}_i)_{i\in[k-1]}$ *be randomizations of* vk_0 *and generate the rest randomly (storing all secret keys for issuing later).*
- $(I_0, I_1, pol, D) \xleftarrow{\$} \mathcal{A}^{\langle\mathcal{O}\rangle}(\mathsf{pp}, \{\mathsf{vk}_i\}_{i\in[\ell]})$;
- $(\mathsf{sk}_u, \mathsf{vk}_u) \xleftarrow{\$} \mathsf{UKeyGen}(\mathsf{pp})$; *If $j \leq \ell$, let $b = 0$ and if $j > \ell$, let $b = 1$*
- *Run the* CredIssue *function using our stored secret keys:*
  $\forall i\mathsf{vk} \in I_b$:
  $(\mathsf{cred}_{b,i}, st) \xleftarrow{\$} \mathsf{CredObtain}(\mathsf{sk}_u, \mathsf{vk}_i, \mathsf{A}_i) \leftrightarrow \mathsf{CredIssue}(\mathsf{sk}_i, \mathsf{vk}_u, \mathsf{A}_i)$;
- $\mathsf{cred}_b \leftarrow \mathsf{CredAggr}\left(u\mathsf{sk}, \{(i\mathsf{vk}, \mathsf{A}_i, \mathsf{cred}_{b,i})\}_{i\in[\ell]}\right)$
- $b^* \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{CredShow}}}(pol, I_b, D, \mathsf{A}_{i\in[\ell]})$

To reduce to class or origin hiding, we define 2ℓ games. Let's first define game j where $j \leq \ell$. In game j (where $j \leq \ell$), we fix $b = 0$, and the reduction uses the first public key given to it by the public-key class-hiding challenger vk_1 (from Definition 5) for the first j keys (using different randomizations in the same equivalence class). Game j then uses $\mathsf{vk}_2^{b_{\mathsf{PK\text{-}chall}}}$ for the $j + 1$ key (where $b_{\mathsf{PK\text{-}chall}}$ is the secret bit of the public-key class-hiding challenger). Game j then uses IKeyGen to generate the rest of the keys after $j + 1$. We can see these games in Definition 34.

We can see that for game j (where $j \leq \ell$), the adversary's view appears as hybrid j in the case that the public-key class-hiding challenger's secret bit is $b_{\mathsf{PK\text{-}chall}} = 0$ and we appear as hybrid $j + 1$ if the secret bit is $b_{\mathsf{PK\text{-}chall}} = 1$. This is because the public-key class-hiding challenger's secret bit determines whether $\mathsf{vk}_2^{b_{\mathsf{PK\text{-}chall}}}$ is in the same equivalence class as vk_1 and the only difference between the hybrids for the game is whether the key $j + 1$ is in the same equivalence class as j. Thus, because the challenger's secret bit determines which hybrid this reduction appears as, if our signature scheme has public-key class-hiding, each step between hybrids are indistinguishable (for hybrids up to ℓ).

Now let's define the rest of the games (after ℓ). For game $\ell + j$, we set $b = 1$ and it uses randomizations of the challenger key, vk_1 for most of the issuer keys that we give to the adversary ($2\ell - j$ of the keys). This means for $j = \ell$ all the keys are in the same equivalence class (which is indistinguishable because of origin-hiding, which we mentioned while describing the hybrids). Game $\ell + j$ then uses $\mathsf{vk}_2^{b_{\mathsf{PK\text{-}chall}}}$ for one of the issuer keys and generates the rest of the keys on its own.

We can see for these games where $j > \ell$, we appear as hybrid j if the secret bit is $b_{\mathsf{PK\text{-}chall}} = 0$ and as hybrid $j + 1$ if the secret bit is 1 instead. Thus, each step in this hybrid chain is indistinguishable as well.

Definition 34 (Game j).

- *Reduction receives* $\mathsf{vk}_1, \mathsf{vk}_2^{b_{\mathsf{PK\text{-}chall}}}$ *from the challenger.*
- *If $j \leq \ell$, let $k = j$ and if $j > \ell$, let $k = 2\ell - j$. Let* $(i\mathsf{vk})_{i\in[k-1]}$ *be randomizations of* vk_1 *and* $(k\mathsf{vk})$ *be* $\mathsf{vk}_2^{b_{\mathsf{PK\text{-}chall}}}$. *Generate the rest with* IKeyGen.
- $(I_0, I_1, pol, D) \xleftarrow{\$} \mathcal{A}^{\langle\mathcal{O}\rangle}(\mathsf{pp}, i\mathsf{vk}_{i\in[\ell]})$;
- $(u\mathsf{sk}, u\mathsf{vk}) \xleftarrow{\$} \mathsf{UKeyGen}(\mathsf{pp})$; *If $j \leq \ell$, let $b = 0$ and if $j > \ell$, let $b = 1$*

– *Run the* CredIssue *function using the correct signing oracle from the challenger for issuers keys up to k and use our generated secret keys for the rest:*
$\forall i\mathsf{vk} \in I_b$:
$(\mathsf{cred}_{b,i}, st) \xleftarrow{\$} \mathsf{CredObtain}(u\mathsf{sk}, i\mathsf{vk}, \mathsf{A}_i) \leftrightarrow \mathsf{CredIssue}(i\mathsf{sk}, u\mathsf{vk}, \mathsf{A}_i)$;
– $\mathsf{cred}_b \leftarrow \mathsf{CredAggr}\left(u\mathsf{sk}, \{(i\mathsf{vk}, \mathsf{A}_i, \mathsf{cred}_{b,i})\}_{i\in[\ell]}\right)$
– $b^* \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{CredShow}}}(pol, I_b, D, \mathsf{A}_{i\in[\ell]})$
– *This is a hybrid adversary, so they are trying to guess if they're in hybrid j or hybrid $j+1$. The reduction guesses b^* for the public-key class-hiding game.*

Thus, by hybrid argument, we find that the adversary's view in the issuer-hiding game is the same when $b = 0$ and $b = 1$, thus, if our signature scheme has public-key class-hiding, then our IhMA scheme has issuer hiding. The reduction can answer all of the adversary's queries using the signing oracles given to it by the public-key class-hiding oracle which signs messages for each of the public keys.

Lemma 5 (Unforgeability of construction in Fig 7). *Let* ZKPOK, SC, *and* SPSEQ *be a simulation-sound extractable ZKPoK and a binding commitment and unforgeable signature receptively, if* ATMS *is unforgeable, then the* IhMA *construction in Fig 7 is unforgeable.*

Proof. Similar to Lemma 3, we show that if an adversary $\mathcal{A}$ can win the unforgeability game (Def. 18) with non-negligible probability. We then construct an adversary (reduction) $\mathcal{B}$ that breaks the unforgeability of ATMS (Def. 13). Assume that we can extract witness from ZKPOK. It follows the same reductions as AtoSa, so we only show the differences here as follows:

Reduction. $\mathcal{B}$ interacts with a challenger $\mathcal{C}$ in the unforgeability game of ATMS and $\mathcal{B}$ simulates the IhMA-unforgeability for $\mathcal{A}$.

– $\mathcal{B}$ receives from $\mathcal{C}$ values $(\hat{X} = P^x, \hat{Y}_1 = \hat{P}^{y_1}, \hat{Y}_2 = \hat{P}^{y_2})$ and public parameters $\mathsf{pp}_{\mathsf{ATMS}}$. Then, it sets $\mathsf{vk}' = (\hat{X} = P^x, \hat{Y}_1 = \hat{P}^{y_1}, \hat{Y}_2 = \hat{P}^{y_2})$ as the challenge issuer key, $\mathsf{pp}_{\mathsf{SPSEQ}} \leftarrow \Sigma_2.\mathsf{Setup}(1^\lambda)$ and pick α and create set commitment public parameters $\mathsf{pp}_{\mathsf{SC}} \leftarrow \mathsf{SC.Setup}(1^\lambda, \alpha)$ and send vk' and $\mathsf{pp} = (\mathsf{pp}_{\mathsf{ATMS}}, \mathsf{pp}_{\mathsf{SC}}, \mathsf{pp}_{\mathsf{SPSEQ}})$ to $\mathcal{A}$.
– As shown in Lemma 3, $\mathcal{B}$ handles any oracle query and simulates all oracles perfectly for $\mathcal{A}$ similar to Lemma 3, and never aborts, we skip to mention them here.
– If $\mathcal{A}$'s winning condition is not fulfilled, $\mathcal{B}$ aborts.
– Otherwise, $\mathcal{A}$ is able, with some probability, to prove possession of a credential on attributes D^*.

So $\mathcal{B}$ interacts with $\mathcal{A}$ as verifier in a showing protocol. If $\mathcal{A}$ outputs a valid showing proof as $(\mathsf{avk}, (\mathbf{C}^*, \hat{\mathbf{C}}^*), \sigma^*, D^*, W^*, \mathsf{Nym}^* = \hat{\mathbf{T}}^*)$ and conducting $\pi = \mathsf{NIZK}(\mathsf{Nym}^*)$ then $\mathcal{B}$ extracts from the proof of knowledge contained in the Show, called the value $\tau^* = (\rho_1^*, \rho_2^*)$ related to the Nym^* and stores all elements. Also, we know that no credentials owned by corrupt users can be valid on this set of messages D^* (as $\mathcal{A}$ can win the unforgeability game). This means that, for all credentials cred_{ui} on $(u\mathsf{sk})$ with $u \in \mathcal{CCU}$, we have $(D^* \not\subseteq \mathsf{A})$. From the definition we have that at least one of the key in $\mathsf{vk}_j \in \mathsf{avk}$ should be the challenge key $\mathsf{vk}_j = \mathsf{vk}'$ such that the related attribute is in the set $\mathsf{A}_j \in D^*$. Find all $i\mathsf{sk} \in \mathcal{HCI} \cup \mathcal{CCI}$ corresponding to $[i\mathsf{vk}]_{\mathcal{R}} \in [\ell]$. Finally, with all these cases, we conclude that $(\mathsf{avk}, \mathsf{ask}, (\tau^*, \hat{\mathbf{T}}^*), D^*, \sigma^*, (\mathbf{C}^*, \hat{\mathbf{C}}^*))$ is a valid forgery against our signature scheme, where $(\mathbf{C}^*, \hat{\mathbf{C}}^*) = (\mathbf{M}^*, \mathbf{N}^*)$. So $\mathcal{B}$ breaks unforgeability of ATMS which concludes our proof. Note that we also assume SC is a binding and hiding commitment and SPSEQ is unforgeable, so $\mathcal{A}$ can not forge proof by breaking binding of SC or unforgeability of SPSEQ.

Lemma 6. *Let* ZKPOK *be a simulation-sound extractable ZKPoK,* ATMS *is origin-hiding of ConvertSig and ChangRep and public key class-hiding, and* SPSEQ *is perfect adaption, then the* IhMA *construction in Fig 6 is anonymous and issuer hiding.*

Proof. The argument follows the one in Lemma 4 except that we replace privacy notations of AtoSa by ATMS and show that a new uber assumption holds 6 for the randomization set commitments and tag. The proof follows a sequence of games until a game where answers for the query to $\mathcal{O}_b^{\mathsf{Anon}}$ is independent of the bit b. In Game_1 we replace all ChangRep and ConvertSig calls with freshly generated signatures. In Game_2 we simulate all ZKPoKs and In Game_3 we replace the respective commitment vectors $\mathbf{C}$ with a represantive element of the same class.

Game_0**:** The original game as given in Definition 19.

Game_1**:** As Game_0, except that the experiment runs $\mathcal{O}_b^{\mathsf{Anon}}$ as follows: Like in Game_0, but for $\mu, \upsilon \in \mathbb{Z}_p^*$, all executions of ChangRep and ConvertSig for the credential and tag $(i_b, \mathsf{A}', \sigma_b) \leftarrow \mathcal{L}_{\mathsf{cred}}[\mathsf{j}_b]$ are replaced by randomized signatures (note that we can randomize signatures and output uniformly random elements in the respective spaces). So, oracles are simulated as in Game_1.

$\mathsf{Game}_1 \rightarrow \mathsf{Game}_0$: By Origin-hiding (ConvertSig), adapted privacy (ChangRep) signatures obtained from ChangRep and ConvertSig are identically distributed for all $(\mathsf{A}, \mathbf{T}, \mathsf{vk}, (\mathbf{C}, \hat{\mathbf{C}}))$. We thus have

$$\Pr[S_0] = \Pr[S_1]$$

Game_2**:** As Game_1, except that the experiment runs $\mathcal{O}_b^{\mathsf{Anon}}$ as follows: All proofs $\mathsf{NIZK}(\mathsf{Nym}_p)$ in CredShow and ObtIss respectively, are simulated.
$\mathsf{Game}_2 \rightarrow \mathsf{Game}_3$: By perfect zero-knowledge of NIZK, we have that

$$\Pr[S_1] = \Pr[S_2] \Rightarrow \Pr[S_0] = \Pr[S_1] = \Pr[S_2]$$

Game_3**:** As Game_2, except that for $\mathcal{O}_b^{\mathsf{Anon}}(j_0, j_1, D)$: it replaces all $(\mathbf{C}_i, \hat{\mathbf{C}}_i)$ and $\mathbf{T}$ with another vectors in the same equivalence class and performs the showing with $D = (d_i)_{i\in[k]}$ and $W_i \leftarrow f_{d_i}(a)^{-1} \cdot \hat{C}_{1i}$. The only difference is the computation of $(\mathbf{C}_i, \hat{\mathbf{C}}_i)$; while all W_i are distributed as in Game_4, in particular, they are unique elements satisfying VerifySubset.

$\mathsf{Game}_2 \rightarrow \mathsf{Game}_3$: The difference between two games is that we use the uber assumption Def. 6 to create set commitments. Oracles are simulated as in Game_4, except for the following oracles as:

- $\mathcal{O}^{\mathsf{Obtain}}(u, i, \mathsf{A}_i)$: As in Game_2, except for the computation of following values: compute the polynomials $f_1 = F_{\mathsf{A}_i}(\alpha) \cdot \rho_1 \cdot r$, $f_2 = \rho_2 \cdot r \cdot \eta$, $\hat{f}_1 = F_{\mathsf{A}_i}(\alpha)$, and $\hat{f}_2 = \eta$, where $r \xleftarrow{\$} \mathbb{Z}_p$ is for h and come from RO and η is a constant and same for all commitments. Then for $i \in [2]$ it calls $C_i \leftarrow \mathcal{O}^{uber}(1, f_i, f_i)$ and $\hat{C}_i \leftarrow \mathcal{O}^{uber}(2, \hat{f}_i, \hat{f}_i)$ (all C_i and $\hat{C}_i$ are distributed as in the original game.)
- $\mathcal{O}^{\mathsf{CredShow}}(j, pol, D)$: As in Game_2, it computes the witness $W_i \leftarrow f_{d_i}(\alpha)^{-1} \cdot \hat{C}_{1i}$ (W_i is thus distributed as in the original game and $D = (d_i)_{i\in[k]}$.)
- $\mathcal{O}_b^{\mathsf{Anon}}(j_0, j_1, D)$: As in Game_4, with the following difference. For two honest users $u \leftarrow L_{\mathsf{cred}}[j_0]$ and $u' \leftarrow L_{\mathsf{cred}}[j_1]$, first parses $L_{\mathsf{cred}}[j_b]$ as $(u, \mathsf{A}_i, \mathsf{cred}_i)$ and $(u', \mathsf{A'}_i, \mathsf{cred}'_i)$.
  Compute polynomials for each set A_i as $f_{1i} = F_{\mathsf{A}_i}(\alpha) \cdot \rho_1 \cdot r$, $f_{2i} = \rho_2 \cdot r \cdot \eta$, $\hat{f}_{1i} = F_{\mathsf{A}_i}(\alpha)$, and $\hat{f}_{2i} = \eta$.
  Compute polynomials for each set A'_i as $f'_{1i} = F_{\mathsf{A'}_i}(\alpha) \cdot \rho'_1 \cdot r'$, $f'_{2i} = \rho'_2 \cdot r' \cdot \eta$, $\hat{f}'_{1i} = F_{\mathsf{A'}_i}(\alpha)$, and $\hat{f}'_{2i} = \eta$.
  Compute polynomials for the tags $\mathbf{T}$ and $\mathbf{T}'$ as $f'_{t1} = \rho'_1 \cdot r'$, $f'_{t2} = \rho'_2 \cdot r'$, $f_{t1} = \rho_1 \cdot r$, and $f_{t2} = \rho_2 \cdot r$.
  Then it calls $C_{1i} \leftarrow \mathcal{O}^{uber}(1, f_{1i}, f'_{1i})$ and $C_{2i} \leftarrow \mathcal{O}^{uber}(1, f_{2i}, f'_{2i})$, same for $\hat{C}_{1i} \leftarrow \mathcal{O}^{uber}(2, \hat{f}_{1i}, \hat{f}'_{1i})$ and $\hat{C}_{2i} \leftarrow \mathcal{O}^{uber}(2, \hat{f}_{2i}, \hat{f}'_{2i})$. Also, to compute tags, it calls
  $T_1 \leftarrow \mathcal{O}^{uber}(1, f_{t1}, f'_{t1})$ and $T_2 \leftarrow \mathcal{O}^{uber}(1, f_{t2}, f'_{t2})$.
  It sends $(\mathbf{C}_i, \hat{\mathbf{C}}_i)$ and $\mathbf{T}$ to $\mathcal{A}$, and receives b' form $\mathcal{A}$.

$$|\Pr[S_2] - \Pr[S_3]| \leq \epsilon_{uber}(\lambda)$$

  Return b' as answer to the uber assumption. Apart from an error event happening with negligible probability, we have simulated Game_2 if the uber assumption was "real" and Game_3 otherwise.

D.5 Proof of theorem 6

Preliminary notations. We call Δ the statistical distance between two algorithm (interpreted here as distribution of their output). We call $z_{tot} = m_1 + m_2 + m_T + m_p + q_1 + q_2 + q_T$. Because some variables (the j's and the q's are dynamical), we add an entry to precise the temporality of the variable. To be more precise $\forall z \in \{2, z_{tot}\}$ we write $j_1[z]$ the value of j_1 after the z^{th} call to ENC. When we do not precise any entry, it means it is the value of the variable at the end of the game.

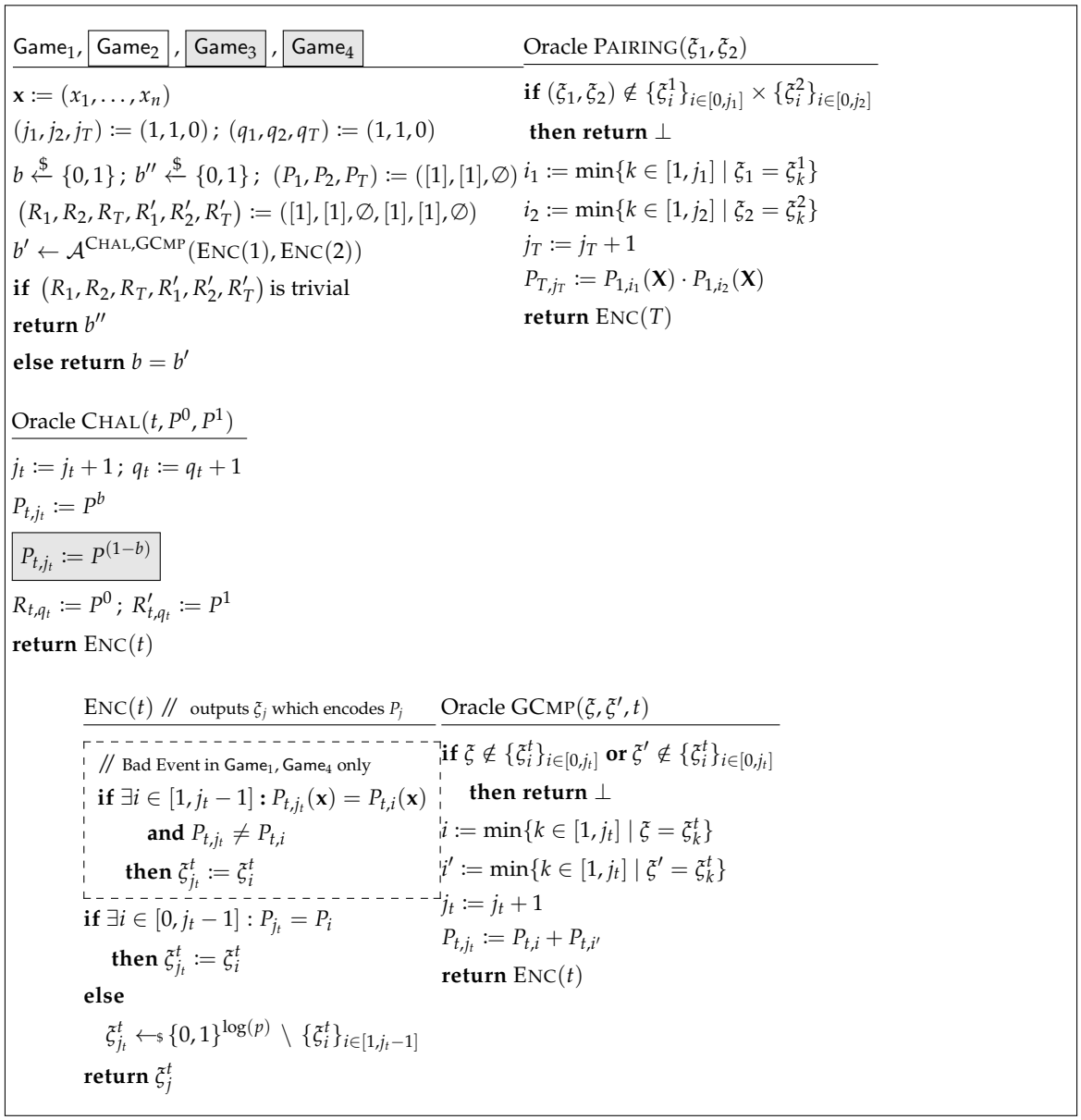


Fig. 12: .

Game$_1$ *to* Game$_2$. We now compare Game$_1$ to Game$_2$. The only difference between the two is when bad event happens in Game$_1$ in the procedure ENC()

$$\exists k \in \{2, z_{tot}\}, \exists t \in \{1,2,T\}, \exists i \in [1, j_t[k] - 1] \textbf{ such that } P_{t,j_t[k]}(\mathbf{x}) = P_{t,i}(\mathbf{x}) \textbf{ and } P_{t,j_t[k]} \neq P_{t,i}.$$

We call this family of disjoint events $\{F_k\}_{k \in \{2, z_{tot}\}}$. let call (d'_1, d'_2, d'_T) an upper bound on the degree of the polynomials in (P_1, P_2, P_T) :

$$\begin{aligned}
\Delta\left(\text{Game}_1\mathcal{A}, \text{Game}_2\mathcal{A}\right) &\leq \Pr[\sqcup_{k \in \{2, z_{tot}\}} F_k] \\
&\leq \sum_{t \in \{1,2,T\}} \Pr[\exists i, j \in [1, j_t]^2 \textbf{ such that } \wedge P_{t,j}(\mathbf{x}) = P_{t,i}(\mathbf{x}) \textbf{ and } P_{t,j_t} \neq P_{t,i} \\
&\leq \frac{j_1^2 d'_1 + d'_2 j_2^2 + d'_T j_T^2}{p}.
\end{aligned}$$

The last equality comes from the Schwartz-Zippel lemma.

We first upperbound for $i \in \{1,2\} : j_i = (q_i + m_i)$, because we can create also new elements in $\mathbb{G}_T$ by using PAIRING. then $j_T \leq (q_T + m_T + m_p)$. There is no way to create elements in $\mathbb{G}_i$ with degree greater than d_i for $i \in \{1,2\}$. Then $d'_i \leq d_i$. But in $\mathbb{G}_T$, we can create element of degree $d_1 + d_2$ by using PAIRING. Then

$$\Delta\left(\mathsf{Game}_1\mathcal{A}, \mathsf{Game}_2\mathcal{A}\right) \leq \frac{d_1(q_1+m_1)^2 + d_2(q_2+m_2)^2 + \max(d_1+d_2, d_T)(q_T+m_T+m_p)^2}{p}. \quad (7)$$

Game_2 *to* Game_3. To simplify the analysis and without any loss of generality, we determinize the adversary and the games itself (when it picks a random string in ENC, we use the same randomness in both games).

The only gap between the two games appears if the condition in the if of ENC is verified in one game and not in the other one.

Let's suppose the first divergence happens in the k^{th} call to ENC. Because the adversary and both games are determinic, we know all the queries are the same until this k^{th} call (included). We suppose an equality is verified in Game_2. Let call $(P^2_{j_t}, P^2_i)$ the corresponding pair of polynomials and $(P^3_{j_t}, P^3_i)$ the analog pair in the Game_3 (in the if condition).

Without any loss of generality we suppose $P^2_{j_t} = P^2_i$ and $P^2_{j_t} \neq P^2_i$).

Case 1, $t = 1$ By looking in details all the queries made by the adversary we can deduce an integer $n := q_1[k]$, and coefficients $(a_i)_{1\leq i\leq n}, (b_i)_{1\leq i\leq n}$ such that $P^2_{j_t} = \sum_{i=1}^n a_i(R_1)_i$, and $P^2_i = \sum_{i=1}^n b_i(R_1)_i$.

Morever, because the behavior of the game was completely analog in both game before this call, we deduce also : $P^3_{j_t} = \sum_{i=1}^n a_i(R_2)'_i$, and $P^3_i = \sum_{i=1}^n b_i(R'_1)_i$.

Then by noticing $1 \in R_2$ and $1 \in R'_2$, and $(P^2_{j_t} - P^2_i) = 0$ and $(P^3_{j_t} - P^3_i) \neq 0$, we deduce that $(R_1, R_2, R_T, R'_1, R'_2, R'_T)$ is trivial, then the output will be a random bit in both games (then it will not appear in Δ).

Case 2, $t = 2$ It is completely analog to the previous case.

Case 3, $t = T$ By looking in details all the queries made by the adversary we can deduce the exitence of coefficients $(a_{u,h})_{1\leq u\leq q_1[k], 1\leq h\leq q_1[k]}, (b_i)_{1\leq i\leq q_t[k]} (c_{u,h})_{1\leq u\leq q_1[k], 1\leq h\leq q_1[k]}, (d_i)_{1\leq i\leq q_t[k]}$ s.t. $P^2_{j_t} = \sum_{u=1}^{q_1[k]}\sum_{h=1}^{q_2[k]} a_{u,h}(R_1)_i \cdot (R_2)_h + \sum_{j=1}^{q_T[k]} b_j(R_T)_j$, and $P^2_i = \sum_{u=1}^{q_1[k]}\sum_{h=1}^{q_2[k]} c_{u,h}(R_1)_u \cdot (R_2)_h + \sum_{j=1}^{q_T[k]} d_j(R_T)_j$,

Morever, because the behavior of the game was completely analog in both game before this call, we deduce also : $P^3_{j_t} = \sum_{u=1}^{q_1[k]}\sum_{h=1}^{q_2[k]} a_{u,h}(R_1)_u \cdot (R_2)_h + \sum_{j=1}^{q_T[k]} b_j(R_T)_j$, and $P^3_i = \sum_{u=1}^{q_1[k]}\sum_{h=1}^{q_2[k]} c_{u,h}(R_1)_u \cdot (R_2)_h + \sum_{j=1}^{q_T[k]} d_j(R_T)_j$, Then by noticing $(P^2_{j_t} - P^2_i) = 0$ and $(P^3_{j_t} - P^3_i) \neq 0$, we deduce that

$(R_1, R_2, R_T, R'_1, R'_2, R'_T)$ is trivial, then the output will be a random bit in both games. We finally deduce:

$$\Delta\left(\mathsf{Game}_3\mathcal{A}, \mathsf{Game}_4\mathcal{A}\right) = 0. \quad (8)$$

Game_3 *to* Game_4. We use exactly the same reasoning as in the comparison between Game_1 and Game_2 to deduce :

$$\Delta\left(\mathsf{Game}_3\mathcal{A}, \mathsf{Game}_4\mathcal{A}\right) \leq \frac{d_1(q_1+m_1)^2 + d_2(q_2+m_2)^2 + \max(d_1+d_2, d_T)(q_T+m_T+m_p)^2}{p}. \quad (9)$$

By combining (7), (8), (9), we can conclude. □

D.6 Proof of Theorem 4

We prove that our scheme satisfies unforgeability (Def. 3) in the generic group model [61] for asymmetric ("Type-3") bilinear groups (for which there are no efficiently computable homomorphisms between $\mathbb{G}$ and $\hat{\mathbb{G}}$). In this model, the adversary is only given *handles* of group elements, which are just uniform random strings. To perform group operations, it uses an oracle to which it can submit handles and is given back the handle of the sum, inversion, etc of the group elements for which it submitted handles. In this proof we use additive notations, and without any loss of generality we suppose that the j^* in the game is equal to 1 in the game 3.

Theorem 7. *If H' is collision resistant, and H is considered as a random oracle, a generic* Adv *that computes at most $(q_{\mathbb{G}}, q_{\mathbb{G}_2}, q_{\mathbb{G}_T})$ group operations and makes up to k queries to signature oracle, and k' queries to H cannot win the game of the figure 3 with probability greater than $\frac{3\left(\left(2k+k'+q_{\mathbb{G}_1}+1\right)\left(5+q_{\mathbb{G}_2}\right)+q_{\mathbb{G}_T}\right)^2}{p-1}$.*

Proof. We consider an adversary that only uses generic group operations on the group elements it receives. After getting the public parameters: $(G, \hat{G})$, a verification key $(\hat{X} = x\hat{G}, \hat{Y}_1 = y_1\hat{G}, Y_2 = y_2\hat{G}, \hat{Z}_1 = z_1\hat{G}, \hat{Z}_2 = z_2\hat{G})$, hashes $(H_{\mathsf{aux}_i})_{i\in[k']}$ computed on queries $(\mathsf{aux}_1, \mathsf{aux}_2, \ldots, \mathsf{aux}_{k'})$ and signatures $\left(H_{\mathsf{aux}_{k_i}}, B_i, S_i\right)_{i=1}^{k}$ computed on queries,

$$\left(\tau_1^{(i)}, \tau_2^{(i)}, \mathsf{aux}_i, \left(\left(M^{(1,i)}, M^{(2,i)}\right), \left(N^{(1,i)}, N^{(2,i)}\right), \omega^{(i)}\right)\right)_{i=1}^{k},$$

(without any loss of generality, we can suppose the message are in $\mathcal{M}_{TDH}^{H}$) the adversary outputs verification keys $\left(\hat{X}^{(i)}, \hat{Y}_1^{(i)}, \hat{Y}_2^{(i)}, \hat{Z}_1^{(i)}, \hat{Z}_2^{(i)}\right)_{1\le i\le \ell}$, messages and public tag

$\left(\left(\left(M^{(1,*,i)}, M^{(2,*,i)}\right), \left(N^{(1,*,i)}, N^{(2,*,i)}\right)_{1\le i\le \ell}, T_1^{(*)}, T_2^{(*)}\right)\right)$, tag's secrets (τ_1^*, τ_2^*), signing keys $\left(\left(\varsigma_x^{(2)}, \varsigma_{y_1}^{(2)}, \varsigma_{y_2}^{(2)}, \varsigma_{z_1}^{(2)}, \varsigma_{z_2}^{(2)}\right), \ldots, \left(\varsigma_x^{(\ell)}, \varsigma_{y_1}^{(\ell)}, \varsigma_{y_2}^{(\ell)}, \varsigma_{z_1}^{(\ell)}, \varsigma_{z_2}^{(\ell)}\right)\right)$, (H^*, B^*, S^*) for them. We call $I := \left\{\mathsf{aux}_j\right\}_{1\le j\le k'}$, and $I' := I \setminus \left\{\mathsf{aux}_{k_j}\right\}_{1\le j\le k}$.

As it must compute any new group element by combining received group elements, it must choose coefficients:

$$\begin{array}{l}
\left(\left(\mu^{(1,i)}, \left(\mu_{h,a}^{(1,i)}\right)_{a\in I}, \left(\mu_{B,j}^{(1,i)}, \mu_{s,j}^{(1,i)}\right)_{1\le j<i}\right),\right.\\
\left(\mu^{(2,i)}, \left(\mu_{h,a}^{(2,i)}\right)_{a\in I}, \left(\mu_{B,j}^{(2,i)}, \mu_{s,j}^{(2,i)}\right)_{1\le j<i}\right),\\
\left.\left(\nu^{(1,i)}, \nu_x^{(1,i)}, \nu_{y,1}^{(1,i)}, \nu_{y,2}^{(1,i)}, \nu_{z,1}^{(1,i)}, \nu_{z,2}^{(1,i)}\right), \left(\nu^{(2,i)}, \nu_x^{(2,i)}, \nu_{y,1}^{(2,i)}, \nu_{y,2}^{(2,i)}, \nu_{z,1}^{(2,i)}, \nu_{z,2}^{(2,i)}\right)\right)_{1\le i\le k},\\
\left(\mu^{(1,*,i)}, \left(\mu_{h,a}^{(1,*,i)}\right)_{a\in I}, \left(\mu_{B,j}^{(1,*,i)}, \mu_{s,j}^{(1,*,i)}\right)_{1\le j\le k}\right),\\
\left(\mu^{(2,*,i)}, \left(\mu_{h,a}^{(2,*,i)}\right)_{a\in I}, \left(\mu_{B,j}^{(2,*,i)}, \mu_{s,j}^{(2,*,i)}\right)_{1\le j\le k}\right),\\
\left(\nu^{(1,*,i)}, \nu_x^{(1,*,i)}, \nu_{y,1}^{(1,*,i)}, \nu_{y,2}^{(1,*,i)}, \nu_{z,1}^{(1,*,i)}, \nu_{z,2}^{(1,*,i)}\right), \left(\nu^{(2,*,i)}, \nu_x^{(2,*,i)}, \nu_{y,1}^{(2,*,i)}, \nu_{y,2}^{(2,*,i)}, \nu_{z,1}^{(2,*,i)}, \nu_{z,2}^{(2,*,i)}\right),\\
\left(\zeta^{(1,*,i)}, \zeta_x^{(1,*,i)}, \zeta_{y,1}^{(1,*,i)}, \zeta_{y,2}^{(1,*,i)}, \zeta_{z,1}^{(1,*,i)}, \zeta_{z,2}^{(1,*,i)}\right), \left(\zeta^{(2,*,i)}, \zeta_x^{(2,*,i)}, \zeta_{y,1}^{(2,*,i)}, \zeta_{y,2}^{(2,*,i)}, \zeta_{z,1}^{(2,*,i)}, \zeta_{z,2}^{(2,*,i)}\right),
\end{array}$$

$$\begin{array}{l}
\left(\chi, \chi_x, \chi_{y,1}, \chi_{y,2}, \chi_{z,1}, \chi_{z,2}\right), \left(\chi^{(i)}, \chi_x^{(i)}, \chi_{y,1}^{(i)}, \chi_{y,2}^{(i)}, \chi_{z,1}^{(i)}, \chi_{z,2}^{(i)}\right)_{2\le i\le \ell},\\
\left(\iota^{(1)}, \iota_x^{(1)}, \iota_{y,1}^{(1)}, \iota_{y,2}^{(1)}, \iota_{z,1}^{(1)}, \iota_{z,2}^{(1)}\right), \left(\iota^{(2)}, \iota_x^{(2)}, \iota_{y,1}^{(2)}, \iota_{y,2}^{(2)}, \iota_{z,1}^{(2)}, \iota_{z,2}^{(2)}\right)\Big),\\
\left(\tau^{(1)}, \left(\tau_{h,u}^{(1)}\right)_{u\in I}, \left(\tau_{B,j}^{(1)}, \tau_{s,j}^{(1)}\right)_{1\le j\le k}\right),\\
\left(\tau^{(2)}, \left(\tau_{h,u}^{(2)}\right)_{u\in I}, \left(\tau_{B,j}^{(2)}, \tau_{s,j}^{(2)}\right)_{2\le j\le k}\right),\\
\left(\eta, (\eta_{h,u})_{u\in I}, (\eta_{B,j}, \eta_{s,j})_{1\le j\le k}\right),\\
\left(\beta, (\beta_{h,u})_{u\in I}, (\beta_{B,j}, \beta_{s,j})_{1\le j\le k}\right), \left(\sigma, \left(\sigma_{h,j}\right)_{j\in I}, (\sigma_{B,j}, \sigma_{s,j})_{1\le j\le k}\right)
\end{array}$$

which define

$$M^{(t,i)} = \mu^{(t,i)}G + \sum_{a\in I}\left(\mu_{h,a}^{(t,i)}H_a\right) + \sum_{j=1}^{(i-1)}\left(\mu_{B,j}^{(t,i)}B_j + \mu_{s,j}^{(t,i)}S_j\right), \forall t \in \{1,2\}, \forall i \in \{1,\dots,k\}$$

$$\hat{N}^{(t,i)} = \nu^{(t,i)}\hat{G} + \nu_x^{(t,i)}\hat{X} + \nu_{y,1}^{(t,i)}\hat{Y}_1 + \nu_{y,2}^{(t,i)}\hat{Y}_2 + \nu_{z,1}^{(t,i)}\hat{Z}_1 + \nu_{z,2}^{(t,i)}\hat{Z}_2, \forall t \in \{1,2\}, \forall i \in \{1,\dots,k\}$$

$$M^{(t,*,i)} = \mu^{(t,*,i)}G + \sum_{a\in I}\left(\mu_{h,a}^{(t,*,i)}H_a\right) + \sum_{j=1}^{k}\left(\mu_{B,j}^{(t,*,i)}B_j + \mu_{s,j}^{(t,*,i)}S_j\right), \forall (t,i) \in \{1,2\}\times\{1,\dots,\ell\}$$

$$\hat{N}^{(t,*,i)} = \nu^{(t,*,i)}\hat{G} + \nu_x^{(t,*,i)}\hat{X} + \nu_{y,1}^{(t,*,i)}\hat{Y}_1 + \nu_{y,2}^{(t,*,i)}\hat{Y}_2 + \nu_{z,1}^{(t,*,i)}\hat{Z}_1 + \nu_{z,2}^{(t,*,i)}\hat{Z}_2, \forall (t,i) \in \{1,2\}\times\{1,\dots,\ell\}$$

$$\hat{X}^{(1)} = \chi\hat{G} + \chi_x\hat{X} + \chi_{y,1}\hat{Y}_1 + \chi_{y,2}\hat{Y}_2 + \chi_{z,1}\hat{Z}_1 + \chi_{z,2}\hat{Z}_2,$$

$$\hat{Y}_t^{(1)} = \iota^{(t)}\hat{G} + \iota_x^{(t)}\hat{X} + \iota_{y,1}^{(t)}\hat{Y}_1 + \iota_{y,2}^{(t)}\hat{Y}_2 + \iota_{z,1}^{(t)}\hat{Z}_1 + \iota_{z,2}^{(t)}\hat{Z}_2, \forall t \in \{1,2\}$$

$$\hat{X}^{(i)} = \chi^{(i)}\hat{G} + \chi_x^{(i)}\hat{X} + \chi_{y,1}^{(i)}\hat{Y}_1 + \chi_{y,2}^{(i)}\hat{Y}_2 + \chi_{z,1}^{(i)}\hat{Z}_1 + \chi_{z,2}^{(i)}\hat{Z}_2, \forall i \in \{2,\dots,\ell\}$$

$$\hat{Y}_t^{(i)} = \iota^{(i,t)}\hat{G} + \iota_x^{(i,t)}\hat{X} + \iota_{y,1}^{(i,t)}\hat{Y}_1 + \iota_{y,2}^{(i,t)}\hat{Y}_2 + \iota_{z,1}^{(i,t)}\hat{Z}_1 + \iota_{z,2}^{(i,t)}\hat{Z}_2, \forall t \in \{1,2\}, \forall i \in \{2,\dots,\ell\}$$

$$\hat{Z}_t^{(1)} = \zeta^{(t)}\hat{G} + \zeta_x^{(t)}\hat{X} + \zeta_{y,1}^{(t)}\hat{Y}_1 + \zeta_{y,2}^{(t)}\hat{Y}_2 + \zeta_{z,1}^{(t)}\hat{Z}_1 + \zeta_{z,2}^{(t)}\hat{Z}_2, \forall t \in \{1,2\}$$

$$\hat{Z}_t^{(i)} = \zeta^{(i,t)}\hat{G} + \zeta_x^{(i,t)}\hat{X} + \zeta_{y,1}^{(i,t)}\hat{Y}_1 + \zeta_{y,2}^{(i,t)}\hat{Y}_2 + \zeta_{z,1}^{(i,t)}\hat{Z}_1 + \zeta_{z,2}^{(i,t)}\hat{Z}_2,$$

$$\forall (t,i) \in \{0,1\}\times\{2,\dots,\ell\}$$

$$T_1^* = \tau^{(1)}G + \sum_{a\in I}\left(\tau_{h,a}^{(1)}H_a\right) + \sum_{j=1}^{k}\left(\tau_{B,j}^{(1)}B_j + \tau_{s,j}^{(1)}S_j\right)$$

$$T_2^* = \tau^{(2)}G + \sum_{a\in I}\left(\tau_{h,a}^{(2)}H_a\right) + \sum_{j=2}^{k}\left(\tau_{B,j}^{(2)}B_j + \tau_{s,j}^{(2)}S_j\right)$$

$$H^* = \eta G + \sum_{a\in I}\left(\eta_{h,a}H_a\right) + \sum_{j=1}^{k}\left(\eta_{B,j}B_j + \eta_{s,j}S_j\right)$$

$$B^* = \beta G + \sum_{a\in I}\left(\beta_{h,a}H_a\right) + \sum_{j=1}^{k}\left(\beta_{B,j}B_j + \beta_{s,j}S_j\right)$$

$$S^* = \sigma G + \sum_{a\in I}\left(\sigma_{h,a}H_a\right) + \sum_{j=1}^{k}\left(\sigma_{B,j}B_j + \sigma_{s,j}S_j\right)$$

Notice that without any loss of generality, the sum is over all the H's[19].

Using this, we can write, for all $1 \leq i \leq k$, the discrete logarithms h_i, b_i, s_i in basis G of the elements $H_i, B_i = \sum_{j=1}^{2} z_j\tau_j^{(i)}H_{\mathsf{aux}_{k_i}}$, $S_i = \left(x\cdot H_{\mathsf{aux}_{k_i}} + y_1M_1^{(i)} + y_2M_2^{(i)}\right)$ from the oracle answers. Then:

$$b_i = \sum_{j=1}^{2} z_j\tau_j^{(i)}\cdot h_{\mathsf{aux}_{k_i}} \tag{10}$$

$$s_i = \left(xh_{\mathsf{aux}_{k_i}} + \sum_{b=1}^{2} y_b\left(\mu^{(b,i)} + \sum_{a\in I}\left(\mu_{h,a}^{(b,i)}h_a\right) + \sum_{j=1}^{(i-1)}\left(\mu_{B,j}^{(b,i)}b_j + \mu_{s,j}^{(b,i)}s_j\right)\right)\right) \tag{11}$$

We interpret these values as multivariate polynomials in variables $x, y_1, y_2, z_1, z_2, (h_u)_{u\in I}$.

First we have to notice

Lemma 7. *In the (GGM + ROM), if the i signing query is valid and if $H_{\mathsf{aux}_{k_i}} = H_{\mathsf{aux}_{k_{i'}}}$, then the i′ is also valid and $S_{\mathsf{aux}_{k_i}} = S_{\mathsf{aux}_{k_i'}}$.*

[19] We know that in some case, there is constraint in the order which force some coefficients to be zero, but we will use that later

First, notice that because $H_{\mathsf{aux}_{k_i}} = H_{\mathsf{aux}_{k_{i'}}}$, it implies

$$\mathsf{aux}_{k_i} = \mathsf{aux}_{k_{i'}}. \tag{12}$$

Then because $\left(G^{\tau_1^{(i)}}, G^{\tau_2^{(i)}}\right)$ is the first coordinate of aux_{k_i}, and because $\left(G^{\tau_1^{(i')}}, G^{\tau_2^{(i')}}\right)$ is the first coordinate of $\mathsf{aux}_{k_{i'}}$, we deduce:

$$\left(G^{\tau_1^{(i)}}, G^{\tau_2^{(i)}}\right) = \left(G^{\tau_1^{(i')}}, G^{\tau_2^{(i')}}\right), \tag{13}$$

then:

$$\left(\tau_1^{(i)}, \tau_2^{(i)}\right) = \left(\tau_1^{(i')}, \tau_2^{(i')}\right), \tag{14}$$

and with the same reasoning we that the vector of keys is the same for both aux.

First we notice $\mathsf{Index}(\mathsf{aux}_{k_i})$ is well defined because by assumption of the lemma the query number i is valid.

We deduce also from $\mathsf{aux}_{k_i} = \mathsf{aux}_{k_{i'}}$ that :

$$\mathsf{vk}'_{\mathsf{Index}(\mathsf{aux}_{k_i})} = \mathsf{vk}_{\mathsf{Index}(\mathsf{aux}_{k_i})}. \tag{15}$$

Then by definition of $\mathsf{Index}(\mathsf{aux}_{k_i})$, we deduce: $\mathsf{vk}'_{\mathsf{Index}(\mathsf{aux}_{k_i})} = \mathsf{vk}$.

Because by definition of $\mathsf{Index}\left(\mathsf{aux}_{k_i}\right)$, we know that $\forall j \neq \mathsf{Index}(\mathsf{aux}_{k_i}) : \mathsf{vk}_j \neq \mathsf{vk}$, we deduce from (15) that: $\forall j \neq \mathsf{Index}\left(\mathsf{aux}_{k_i}\right) : \mathsf{vk}'_j \neq \mathsf{vk}$.

Then it implies $\mathsf{Index}\left(\mathsf{aux}_{k_{i'}}\right)$ is well defined and equals to $\mathsf{Index}\left(\mathsf{aux}_{k_i}\right)$.

Then with the same reasoning we deduce about H:

$$\left(N^{(1,i)}, N^{(2,i)}\right) = \left(N^{(1,i')}, N^{(2,i')}\right). \tag{16}$$

We deduce from (14), (16), and also the fact that $H_{\mathsf{aux}_{k_i}} = H_{\mathsf{aux}_{k_{i'}}}$, and the $e\left(T_j^{(i)}, \cdot\right)$ are injectives:

$$\left(M^{(1,i)}, M^{(2,i)}\right) = \left(M^{(1,i')}, M^{(2,i')}\right). \tag{17}$$

We finally deduce from (12), (14), (16), (17): $S_{k_i} = S_{k'_i}$.

□

Then without any loss of generality we can suppose all the (k_i)'s are distincts, and then

$$\text{All the } h_{\mathsf{Index}\left(\mathsf{aux}_{k_i}\right)} \text{ are distinct.} \tag{18}$$

A successful forgery (H^*, B^*, S^*) on

$$\Bigg(\left(\left(\hat{X}^{(1)}, \hat{Y}_1^{(1)}, \hat{Y}_2^{(1)}, \hat{Z}_1^{(1)}, \hat{Z}_2^{(1)}\right), \ldots, \left(\hat{X}^{(\ell)}, \hat{Y}_1^{(\ell)}, \hat{Y}_2^{(\ell)}, \hat{Z}_1^{(\ell)}, \hat{Z}_2^{(\ell)}\right)\right),$$
$$\left(\left(\varsigma_x^{(2)}, \varsigma_{y,1}^{(2)}, \varsigma_{y,2}^{(2)}, \varsigma_{z,1}^{(2)}, \varsigma_{z,2}^{(2)}\right), \ldots, \left(\varsigma_x^{(\ell)}, \varsigma_{y,1}^{(\ell)}, \varsigma_{y,2}^{(\ell)}, \varsigma_{z,1}^{(\ell)}, \varsigma_{z,2}^{(\ell)}\right)\right),$$
$$\left(\left(\left(M^{(1,*,i)}, M^{(2,*,i)}\right), \left(N^{(1,*,i)}, N^{(2,*,i)}\right)\right)_{1 \leq i \leq \ell}, \left(T_1^{(*)}, T_2^{(*)}\right)\right)\Bigg)$$

satisfies the verification equations

$$\prod_{j \in [\ell]} e(H^*, \hat{X}^{(j)}) \prod_{i \in [2]} e(M^{(i,*,j)}, \hat{Y}_i^{(j)}) = e(S^*, \hat{G}) \ \wedge \ e(B^*, \hat{G}) = \prod_{j \in [\ell], i \in [2]} e(T_i^*, \hat{Z}_i^{(j)})$$
$$\bigwedge_{i \in [2] \wedge j \in [\ell]} e(T_j^*, N^{(i,*,j)}) = e(M^{(i,*,j)}, \hat{G})$$

Using the coefficients defined above and considering the logarithms in basis $e(P, \hat{P})$, we obtain:

$$\left(\eta + \sum_{a\in I}(\eta_{h,a}h_a) + \sum_{j=1}^{k}(\eta_{B,j}b_j + \eta_{s,j}s_j)\right)\left(\chi + \chi_x x + \chi_{y,1}y_1 + \chi_{y,2}y_2 + \chi_{z,1}z_1 + \chi_{z,2}z_2\right) +$$
$$\sum_{t\in[2]}\left(\mu^{(t,*,1)} + \sum_{a\in I}\left(\mu_{h,a}^{(t,*,1)}h_a\right) + \sum_{j=1}^{k}\left(\mu_{B,j}^{(t,*,1)}b_j + \mu_{s,j}^{(t,*,1)}s_j\right)\right)\left(\iota^{(1,t)} + \iota_x^{(1,t)}x + \iota_{y,1}^{(1,t)}y_1 + \iota_{y,2}^{(1,t)}y_2 + \iota_{z,1}^{(1,t)}z_1 + \iota_{z,2}^{(1,t)}z_2\right)$$
$$+ \sum_{2\le i\le\ell}\left(\left(\eta + \sum_{a\in I}(\eta_{h,a}h_a) + \sum_{j=1}^{k}(\eta_{B,j}b_j + \eta_{s,j}s_j)\right)\left(\chi^{(i)} + \chi_x^{(i)}x + \chi_{y,1}^{(i)}y_1 + \chi_{y,2}^{(i)}y_2 + \chi_{z,1}^{(i)}z_1 + \chi_{z,2}^{(i)}z_2\right) +\right.$$
$$\left.\sum_{t\in[2]}\left(\mu^{(t,*,i)} + \sum_{a\in I}\left(\mu_{h,j}^{(t,*,i)}h_a\right) + \sum_{j=1}^{k}\left(\mu_{B,j}^{(t,*,i)}b_j + \mu_{s,j}^{(t,*,i)}s_j\right)\right)\left(\iota^{(i,t)} + \iota_x^{(i,t)}x + \iota_{y,1}^{(i,t)}y_1 + \iota_{y,2}^{(i,t)}y_2\iota_x^{(i,t)}x + \iota_{z,1}^{(i,t)}z_1 + \iota_{z,2}^{(i,t)}z_2\right)\right) \tag{19}$$

$$= \left(\sigma + \sum_{a\in I}(\sigma_{h,a}h_a) + \sum_{j=1}^{k}(\sigma_{B,j}b_j + \sigma_{s,j}s_j)\right)$$
$$\left(\beta + \sum_{a\in I}(\beta_{h,a}h_a) + \sum_{j=1}^{k}(\beta_{B,j}b_j + \beta_{s,j}s_j)\right)$$
$$= \sum_{t\in\{0,1\}}\left(\tau^{(t)} + \sum_{a\in I}\left(\tau_{h,a}^{(t)}h_a\right) + \sum_{j=1}^{k}\left(\tau_{B,j}^{(t)}b_j + \tau_{s,j}^{(t)}s_j\right)\right)\left(\zeta^{(1,t)} + \zeta_x^{(1,t)}x + \zeta_{y,1}^{(1,t)}y_1 + \zeta_{y,2}^{(1,t)}y_2 + \zeta_{z,1}^{(1,t)}z_1 + \zeta_{z,2}^{(1,t)}z_2\right) \tag{20}$$
$$+ \sum_{i\in[2;\ell],t\in\{0,1\}}\left(\tau^{(t)} + \sum_{a\in I}\left(\tau_{h,a}^{(t)}h_a\right) + \sum_{j=1}^{k}\left(\tau_{B,j}^{(t)}b_j + \tau_{s,j}^{(t)}s_j\right)\right)\left(\zeta^{(i,t)} + \zeta_x^{(i,t)}x + \zeta_{y,1}^{(i,t)}y_1 + \zeta_{y,2}^{(i,t)}y_2 + \zeta_{z,1}^{(i,t)}z_1 + \zeta_{z,2}^{(i,t)}z_2\right)$$
$$\forall i \in \{1,\dots,\ell\}, \forall t \in \{1,2\}: \tag{21}$$
$$\left(\tau^{(t)} + \sum_{a\in I}\left(\tau_{h,a}^{(t)}h_a\right) + \sum_{j=1}^{k}\left(\tau_{B,j}^{(t)}b_j + \tau_{s,j}^{(t)}s_j\right)\right)\left(\nu^{(t,*,i)} + \nu_x^{(b,*,i)}x + \nu_{y,1}^{(t,*,i)}y_1 + \nu_{y,2}^{(t,*,i)}y_2 + \nu_{z,1}^{(t,*,i)}z_1 + \nu_{z,2}^{(t,*,i)}z_2\right) \tag{22}$$
$$= \left(\mu^{(b,*,i)} + \sum_{a\in I}\left(\mu_{h,a}^{(b,*,i)}h_a\right) + \sum_{j=1}^{k}\left(\mu_{B,j}^{(b,*,i)}b_j + \mu_{s,j}^{(b,*,i)}s_j\right)\right)$$

We can use also the second winning condition (i.e. $[\mathsf{vk}_i']_{\mathcal{R}_{\mathsf{vk}}} = [\mathsf{vk}_i]_{\mathcal{R}_{\mathsf{vk}}}$) is verified.

It implies $\exists u \in \mathbb{Z}_p^*$ such that:

$$\chi\hat{G} + \chi_x\hat{X} + \chi_{y,1}\hat{Y}_1 + \chi_{y,2}\hat{Y}_2 = u\hat{X} \tag{23}$$
$$\forall t \in \{1,2\}: \iota^{(t)}\hat{G} + \iota_x^{(t)}\hat{X} + \iota_{y,1}^{(t)}\hat{Y}_1 + \iota_{y,2}^{(t)}\hat{Y}_2 + \iota_{z,1}^{(t)}\hat{Z}_1 + \iota_{z,2}^{(t)}\hat{Z}_2 = u\hat{Y}_t \tag{24}$$
$$\forall t \in \{1,2\}: \zeta^{(t)}\hat{G} + \zeta_x^{(t)}\hat{X} + \zeta_{y,1}^{(t)}\hat{Y}_1 + \zeta_{y,2}^{(t)}\hat{Y}_2 + \zeta_{z,1}^{(t)}\hat{Z}_1 + \zeta_{z,2}^{(t)}\hat{Z}_2 = u\hat{Z}_t \tag{25}$$

and:

$\forall i \in [2;\ell]: \exists u^{(i)} \in \mathbb{Z}_p^*$ such that:

$$\chi^{(i)}\hat{G} + \chi_x^{(i)}\hat{X} + \chi_{y,1}^{(i)}\hat{Y}_1 + \chi_{y,2}^{(i)}\hat{Y}_2 = u^{(i)}\varsigma_x^{(i)}\hat{G} \tag{26}$$
$$\forall t \in \{1,2\}: \iota^{(i,t)}\hat{G} + \iota_x^{(i,t)}\hat{X} + \iota_{y,1}^{(i,t)}\hat{Y}_1 + \iota_{y,2}^{(i,t)}\hat{Y}_2 + \iota_{z,1}^{(i,t)}\hat{Z}_1 + \iota_{z,2}^{(i,t)}\hat{Z}_2 = u^{(i)}\varsigma_{y,t}^{(i)}\hat{G} \tag{27}$$
$$\forall t \in \{1,2\}: \zeta^{(i,t)}\hat{G} + \zeta_x^{(i,t)}\hat{X} + \zeta_{y,1}^{(i,t)}\hat{Y}_1 + \zeta_{y,2}^{(i,t)}\hat{Y}_2 + \zeta_{z,1}^{(i,t)}\hat{Z}_1 + \zeta_{z,2}^{(i,t)}\hat{Z}_2 = u^{(i)}\varsigma_{z,t}^{(i)}\hat{G} \tag{28}$$

Using the coefficients defined above and considering the logarithms in base $\hat{G}$, we obtain:

$$\forall t \in \{1,2\} : y_t \left(\chi + \chi_x x + \chi_{y,1} y_1 + \chi_{y,2} y_2 + \chi_{z,1} z_1 + \chi_{z,2} z_2\right) = \tag{29}$$
$$x \left(\iota^{(t)} + \iota_x^{(t)} x + \iota_{y,1}^{(t)} y_1 + \iota_{y,2}^{(t)} y_2 + \iota_{z,1}^{(t)} z_1 + \iota_{z,2}^{(t)} z_2\right)$$
$$\forall t \in \{1,2\} : z_t \left(\chi + \chi_x x + \chi_{y,1} y_1 + \chi_{y,2} y_2 + \chi_{z,1} z_1 + \chi_{z,2} z_2\right) = \tag{30}$$
$$x \left(\zeta^{(t)} + \zeta_x^{(t)} x + \zeta_{y,1}^{(t)} y_1 + \zeta_{y,2}^{(t)} y_2 + \zeta_{z,1}^{(t)} z_1 + \zeta_{z,2}^{(t)} z_2\right)$$
$$\forall i \in [2;\ell] : \chi^{(i)} + \chi_x^{(i)} x + \chi_{y,1}^{(i)} y_1 + \chi_{y,2}^{(i)} y_2 + \chi_{z,1}^{(i)} y_1 + \chi_{z,2}^{(i)} y_2 = u^{(i)} \varsigma_z^{(i)} \tag{31}$$
$$\forall i \in [2;\ell], \forall t \in \{1,2\} : \iota^{(i,t)} + \iota_x^{(i,t)} x + \iota_{y,1}^{(i,t)} y_1 + \iota_{y,2}^{(i,t)} y_2 + \iota_{z,1}^{(i,t)} z_1 + \iota_{z,2}^{(i,t)} z_2 = u^{(i)} \varsigma_{y,t}^{(i)} \tag{32}$$
$$\forall i \in [2;\ell], \forall t \in \{1,2\} : \zeta^{(i,t)} + \zeta_x^{(i,t)} x + \zeta_{y,1}^{(i,t)} y_1 + \zeta_{y,2}^{(i,t)} y_2 + \zeta_{z,1}^{(i,t)} z_1 + \zeta_{z,2}^{(i,t)} z_2 = u^{(i)} \varsigma_{z,t}^{(i)}. \tag{33}$$

We follow the standard proof technique for results in the generic group model and now consider an "ideal" game in which the challenger treats all the (handles of) group elements as elements of $\mathbb{Z}_p[x, y_1, y_2, z_1, z_2, h_{\mathsf{aux}_1}, \ldots, h_{\mathsf{aux}_k}]$, that is, polynomials whose indeterminates represent the secret values chosen by the challenger.

We first show that in the ideal game if the adversary's output satisfies the verification equations, then the third winning condition, $[(\mathbf{M}, \mathbf{N})]_{\mathcal{R}} \neq [(\mathbf{M}_j^*, \mathbf{N}_j^*)]_{\mathcal{R}}, \forall (\mathbf{M}, \mathbf{N}) \in Q$, is not satisfied, which demonstrates that the ideal game cannot be won. We then compute the statistical distance from the adversary's point of view between the real and the ideal game at the end of the proof.

In the ideal game we thus interpret the two equalities (19), (20), (22) as polynomial equalities over the ring $\mathbb{Z}_p[x, y_1, y_2, z_1, z_2, h_{\mathsf{aux}_1}, \ldots, h_{\mathsf{aux}_k}]$. By identifying coefficients of (29), and (30)

$$\forall t \in \{1,2\} : \chi = \iota^{(t)} = \iota_x^{(t)} = \iota_{y,t}^{(1-t)} = \zeta_{z,t}^{(1-t)} = \chi_{y,t} = \chi_{z,t} = 0, \tag{34}$$
$$\chi_x = \iota_{y,t}^{(t)} = \zeta_{y,t}^{(t)}. \tag{35}$$

Because $\hat{X}^* \neq 0$, we deduce

$$\chi_x \neq 0 \tag{36}$$

From (31), (32), (33), we deduce:

$$\forall i \in [2;\ell], \forall t \in \{1,2\} : \iota^{(i,t)} = u^{(i)} \varsigma_{y,t}^{(i)}, \zeta^{(i,t)} = u^{(i)} \varsigma_{z,t}^{(i)}, \chi^{(i)} = u^{(i)} \varsigma_x^{(i)}. \tag{37}$$
$$\forall i \in [2;\ell], \forall t \in \{1,2\} : \chi_{y,1}^{(i)} = \chi_{y,2}^{(i)} = \chi_x^{(i)} = \iota_x^{(i,t)} = \iota_{y,1}^{(i,t)} = \iota_{z,1}^{(i,t)} \tag{38}$$
$$= \iota_{y,2}^{(i,t)} = \iota_{z,2}^{(i,t)} = \zeta_x^{(i,t)} = \zeta_{y,1}^{(i,t)} = \zeta_{z,1}^{(i,t)} = \zeta_{y,2}^{(i,t)} = \zeta_{z,2}^{(i,t)} = 0.$$

Then if we apply (34), (35),(37) (38) the equation (19) becomes:

$$\begin{aligned}
&\left(\eta + \sum_{a \in I} (\eta_{h,a} h_a) + \sum_{j=1}^{k} \left(\eta_{B,j} b_j + \eta_{s,j} s_j\right)\right) \chi_x x + \\
&\sum_{t \in [2]} \left(\mu^{(t,*,1)} + \sum_{u \in I} \left(\mu_{h,u}^{(t,*,1)} h_u\right) + \sum_{j=1}^{k} \left(\mu_{B,j}^{(t,*,1)} b_j + \mu_{s,j}^{(t,*,1)} s_j\right)\right) \chi_x y_t + \\
&\sum_{2 \leq i \leq \ell} \left(\left(\eta + \sum_{a \in I} (\eta_{h,a} h_a) + \sum_{j=1}^{k} \left(\eta_{B,j} b_j + \eta_{s,j} s_j\right)\right) u^{(i)} \varsigma_x^{(i)} + \right. \\
&\left. \sum_{t \in [2]} \left(\mu^{(t,*,i)} + \sum_{a \in I} \left(\mu_{h,a}^{(t,*,i)} h_a\right) + \sum_{j=1}^{k} \left(\mu_{B,j}^{(t,*,i)} b_j + \mu_{s,j}^{(t,*,i)} s_j\right)\right) u^{(i)} \varsigma_{y_t}^{(i)}\right) \\
&= \left(\sigma + \sum_{a \in I} (\sigma_{h,a} h_a) + \sum_{j=1}^{k} \left(\sigma_{B,j} b_j + \sigma_{s,j} s_j\right)\right)
\end{aligned} \tag{39}$$

Now we look this equation modulo (y_1, y_2, z_1, z_2) (remark that $(s_j, b_j) = \left(x h_{\mathsf{aux}_{k_j}}, 0\right)$ in this quotient ring):

$$\begin{aligned}
&\left(\eta + \sum_{a\in I'}(\eta_{h,a}h_a) + \sum_{j=1}^{k}\left(\eta_{s,\mathsf{aux}_{k_j}}h_{\mathsf{aux}_{k_j}} + \eta_{s,j}x h_{\mathsf{aux}_{k_j}}\right)\right)\chi_x x + \\
&\sum_{2\le i\le \ell}\left(\left(\eta + \sum_{a\in I}(\eta_{h,a}h_a) + \sum_{j=1}^{k}\left(\eta_{s,j}x h_{\mathsf{aux}_j}\right)\right)\chi_x^{(i)} + \right. \\
&\left.\sum_{t\in[2]}\left(\mu^{(t,*,i)} + \sum_{a\in I}\left(\mu_{h,a}^{(t,*,i)}h_a\right) + \sum_{j=1}^{k}\left(\mu_{s,j}^{(t,*,i)}x h_{\mathsf{aux}_{k_j}}\right)\right)\iota_{y,t}^{(i,t)}\right) \\
&= \left(\sigma + \sum_{a\in I}(\sigma_{h,a}h_a) + \sum_{j=1}^{k}\left(\sigma_{s,j}x h_{\mathsf{aux}_j}\right)\right)
\end{aligned} \tag{40}$$

If we look the coefficients of $x^2 h_{\mathsf{aux}_{k_j}}$, we deduce:

$$\forall j \le k : \eta_{s,j}\chi_x = 0 \tag{41}$$

And from equation (36)and property (18) we deduce:

$$\forall j \le k : \eta_{s,j} = 0 \tag{42}$$

Then if we use (42), the equation (40) becomes:

$$\begin{aligned}
&\left(\eta + \sum_{u\in I}\eta_{h,u}h_u\right)\chi_x x + \sum_{2\le i\le \ell}\left(\left(\eta + \sum_{u\in I}\eta_{h,u}h_u\right)\chi_x^{(i)} + \right. \\
&\left.\sum_{t\in[2]}\left(\mu^{(t,*,i)} + \sum_{a\in I}\mu_{h,a}^{(t,*,i)}h_a + \sum_{j=1}^{k}\left(\mu_{s,j}^{(t,*,i)}x h_{\mathsf{aux}_j}\right)\right)\iota_{y,b}^{(i,b)}\right) \\
&= \left(\sigma + \sum_{u\in I}(\sigma_{h,u}h_u) + \sum_{j=1}^{k}\left(\sigma_{s,j}x h_{\mathsf{aux}_j}\right)\right)
\end{aligned} \tag{43}$$

then, we look the coefficients of the monomial x, in (43) we deduce:

$$\chi_x \eta = 0 \tag{44}$$

And from equation (36) we deduce:

$$\eta = 0 \tag{45}$$

We apply (34), (35),(37) (38) in (20). And it becomes:

$$\begin{aligned}
&\left(\beta + \sum_{a\in I}(\beta_{h,a}h_a) + \sum_{j=1}^{k}\left(\beta_{B,j}b_j + \beta_{s,j}s_j\right)\right) \\
&= \sum_{t\in\{1,2\}}\left(\tau^{(t)} + \sum_{a\in I}\left(\tau_{h,a}^{(t)}h_a\right) + \sum_{j=1}^{k}\left(\tau_{B,j}^{(t)}b_j + \tau_{s,j}^{(t)}s_j\right)\right)\chi_x z_t \\
&+ \sum_{i\in[2;\ell],t\in\{1,2\}}\left(\tau^{(t)} + \sum_{a\in I}\left(\tau_{h,a}^{(t)}h_a\right) + \sum_{j=1}^{k}\left(\tau_{B,j}^{(t)}b_j + \tau_{s,j}^{(t)}s_j\right)\right)\zeta^{(i,t)}
\end{aligned} \tag{46}$$

Now we look this equation modulo $(y_1, y_2, h_1, \ldots, h_{k'})$ (remark that $(s_j, b_j) = (0, 0)$ in this quotient ring):

$$\beta = \sum_{t\in\{1,2\}} \tau^{(t)}\chi_x z_t + \sum_{i\in[2;\ell],t\in\{1,2\}} \tau^{(t)}\zeta^{(i,t)} \tag{47}$$

It implies by looking z monomials

$$\forall t \in \{1,2\} : \tau^{(t)}\chi_x = 0$$

And from equation (36) we deduce:

$$\forall t \in \{1,2\} : \tau^{(t)} = 0 \tag{48}$$

Then (47) becomes

$$\beta = 0 \tag{49}$$

Now let consider (51) by using (49) and (48):

$$\begin{aligned}
&\left(\sum_{a\in I}(\beta_{h,a}h_a) + \sum_{j=1}^{k}\left(\beta_{B,j}b_j + \beta_{s,j}s_j\right)\right) \\
&= \sum_{t\in\{1,2\}}\left(\sum_{a\in I}\left(\tau^{(t)}_{h,a}h_a\right) + \sum_{j=1}^{k}\left(\tau^{(t)}_{B,j}b_j + \tau^{(t)}_{s,j}s_j\right)\right)\chi_x z_t \\
&\quad + \sum_{i\in[2;\ell],t\in\{1,2\}}\left(\tau^{(t)} + \sum_{a\in I}\left(\tau^{(t)}_{h,a}h_a\right) + \sum_{j=1}^{k}\left(\tau^{(t)}_{B,j}b_j + \tau^{(t)}_{s,j}s_j\right)\right)\zeta^{(i,t)}
\end{aligned} \tag{50}$$

Then, we look this equation modulo (y_1, y_2) (remark that $s_j = xh_{\mathsf{aux}_j}$ in this quotient ring):

$$\begin{aligned}
&\left(\sum_{a\in I}(\beta_{h,a}h_a) + \sum_{j=1}^{k}\left(\beta_{B,j}b_j + \beta_{s,j}xh_{\mathsf{aux}_j}\right)\right) \\
&= \sum_{t\in\{1,2\}}\left(\sum_{a\in I}\left(\tau^{(t)}_{h,a}h_a\right) + \sum_{j=1}^{k}\left(\tau^{(t)}_{B,j}b_j + \tau^{(t)}_{s,j}xh_{\mathsf{aux}_j}\right)\right)\chi_x z_t \\
&\quad + \sum_{i\in[2;\ell],t\in\{1,2\}}\left(\sum_{a\in I}\left(\tau^{(t)}_{h,a}h_a\right) + \sum_{j=1}^{k}\left(\tau^{(t)}_{B,j}b_j + \tau^{(t)}_{s,j}xh_{\mathsf{aux}_j}\right)\right)\zeta^{(i,t)}
\end{aligned} \tag{51}$$

We look monomials of degree 2 in z_1, z_2, and we deduce:

$$\sum_{t\in\{1,2\}}\sum_{j=1}^{k}\tau^{(t)}_{B,j}b_j z_t = 0$$

Then, because all the b_j's are independant (because the h's involved are all distinct, we deduce:

$$\forall t \in \{1,2\}\forall j \in \{1,\ldots,k\} : \tau^{(t)}_{B,j} = 0 \tag{52}$$

Now we look in the same equation monomials in $z_t xh_{t'}$:

$$0 = \sum_{t\in\{1,2\}}\sum_{j=1}^{k}\left(\tau^{(t)}_{s,j}xh_{\mathsf{aux}_j}\right)\chi_x z_t$$

With the same reasoning of previously (independancy of the $z_t xh_{t'}$, we deduce:

$$\forall t \in \{1,2\}\forall j \in \{1,\ldots,k\}\tau^{(t)}_{s,j}\chi_x = 0 \tag{53}$$

Because (36), we deduce:

$$\forall t \in \{1,2\} \forall j \in \{1,\ldots,k\} : \tau_{s,j}^{(t)} = 0 \tag{54}$$

Now we apply (48), (52) and (54) in (22):

$$\forall i \in \{1,\ldots,\ell\}, \forall t \in \{1,2\} : \tag{55}$$

$$\left(\sum_{a\in I}\left(\tau_{h,a}^{(t)} h_a\right)\right)\left(\nu^{(t,*,i)} + \nu_x^{(t,*,i)} x + \nu_{y,1}^{(t,*,i)} y_1 + \nu_{y,2}^{(t,*,i)} y_2 + \nu_{z,1}^{(t,*,i)} z_1 + \nu_{z,2}^{(t,*,i)} z_2\right) \tag{56}$$

$$= \left(\mu^{(t,*,i)} + \sum_{a\in I}\left(\mu_{h,a}^{(t,*,i)} h_a\right) + \sum_{j=1}^{k}\left(\mu_{B,j}^{(t,*,i)} b_j + \mu_{s,j}^{(t,*,i)} s_j\right)\right)$$

By looking constant coefficients, we deduce:

$$\forall i \in \{1,\ldots,\ell\}, \forall t \in \{1,2\} : \mu^{(t,*,i)} = 0 \tag{57}$$

Now we look equation (43) with the knowledge of (45), (57):

$$\begin{aligned}
&\left(\sum_{u\in I}\eta_{h,u} h_u\right)\chi_x x + \sum_{2\leq i\leq \ell}\left(\left(\sum_{u\in I}\eta_{h,u} h_u\right)\chi_x^{(i)} + \right. \\
&\left.\sum_{t\in[2]}\left(\sum_{a\in I}\mu_{h,a}^{(t,*,i)} h_a + \sum_{j=1}^{k}\left(\mu_{s,j}^{(t,*,i)} x h_{\mathsf{aux}_j}\right)\right)\iota_{y,b}^{(i,b)}\right) \\
&= \left(\sigma + \sum_{u\in I}\left(\sigma_{h,u} h_u\right) + \sum_{j=1}^{k}\left(\sigma_{s,j} x h_{\mathsf{aux}_j}\right)\right)
\end{aligned} \tag{58}$$

By looking constant coefficients, we deduce:

$$\sigma = 0 \tag{59}$$

Now let's look (56), by using the knowledge (57).

$$\forall i \in \{1,\ldots,\ell\}, \forall t \in \{1,2\} :$$

$$\left(\sum_{a\in I}\left(\tau_{h,a}^{(t)} h_a\right)\right)\left(\nu^{(t,*,i)} + \nu_x^{(t,*,i)} x + \nu_{y,1}^{(t,*,i)} y_1 + \nu_{y,2}^{(t,*,i)} y_2 + \nu_{z,1}^{(t,*,i)} z_1 + \nu_{z,2}^{(t,*,i)} z_2\right) \tag{60}$$

$$= \left(\sum_{a\in I}\left(\mu_{h,a}^{(t,*,i)} h_a\right) + \sum_{j=1}^{k}\left(\mu_{B,j}^{(t,*,i)} b_j + \mu_{s,j}^{(t,*,i)} s_j\right)\right)$$

We look (39) by using (42), (45), (59), (67), (59),

$$\begin{aligned}
&\left(\sum_{a\in I}\left(\eta_{h,a} h_a\right) + \sum_{j=1}^{k}\left(\eta_{B,j} b_j\right)\right)\chi_x x + \\
&\sum_{t\in[2]}\left(\sum_{u\in I}\left(\mu_{h,u}^{(t,*,1)} h_u\right) + \sum_{j=1}^{k}\left(\mu_{B,j}^{(t,*,1)} b_j + \mu_{s,j}^{(t,*,1)} s_j\right)\right)\chi_x y_t + \\
&\sum_{2\leq i\leq \ell}\left(\left(\eta + \sum_{a\in I}\left(\eta_{h,a} h_a\right) + \sum_{j=1}^{k}\left(\eta_{B,j} b_j\right)\right)u^{(i)}\varsigma_x^{(i)} + \right. \\
&\left.\sum_{t\in[2]}\left(\sum_{a\in I}\left(\mu_{h,a}^{(t,*,i)} h_a\right) + \sum_{j=1}^{k}\left(\mu_{B,j}^{(t,*,i)} b_j + \mu_{s,j}^{(t,*,i)} s_j\right)\right)u^{(i)}\varsigma_{y_t}^{(i)}\right) \\
&= \left(\sum_{a\in I}\left(\sigma_{h,a} h_a\right) + \sum_{j=1}^{k}\left(\sigma_{B,j} b_j + \sigma_{s,j} s_j\right)\right)
\end{aligned} \tag{61}$$

We look this equation modulo (y_1, y_2) :

$$\begin{aligned}
&\left(\sum_{a\in I}\left(\eta_{h,a}h_a\right)+\sum_{j=1}^{k}\left(\eta_{B,j}b_j\right)\right)\chi_x x+\\
&\sum_{2\le i\le \ell}\left(\left(\eta+\sum_{a\in I}\left(\eta_{h,a}h_a\right)+\sum_{j=1}^{k}\left(\eta_{B,j}b_j\right)\right)u^{(i)}\varsigma_x^{(i)}+\right.\\
&\left.\sum_{t\in[2]}\left(\sum_{a\in I}\left(\mu_{h,a}^{(t,*,i)}h_a\right)+\sum_{j=1}^{k}\left(\mu_{B,j}^{(t,*,i)}b_j+\mu_{s,j}^{(t,*,i)}xh_{\mathsf{aux}_j}\right)\right)u^{(i)}\varsigma_{y_t}^{(i)}\right)\\
&=\left(\sum_{a\in I}\left(\sigma_{h,a}h_a\right)+\sum_{j=1}^{k}\left(\sigma_{B,j}b_j+\sigma_{s,j}xh_{\mathsf{aux}_j}\right)\right)
\end{aligned}\tag{62}$$

We look all monomials of degree 3, and the equation become:

$$\left(\sum_{j=1}^{k}\left(\eta_{B,j}b_j\right)\right)\chi_x x=0$$

We use still the fact all the h_{aux_j} are distinct and (36) to deduce:

$$\forall j\in\{1,\ldots,k\}:\eta_{B,j}=0\tag{63}$$

Now, we can use the fact all queries are well formed, it means:

$$\forall i\in\{1,\ldots,k\},\forall t\in\{1,2\}:e\left(\tau_t^{(i)}H_{\mathsf{aux}_i},N^{(t,i)}\right)=e\left(M^{(t,i)},\hat{G}\right)\tag{64}$$

Then in terms of polynomials:

$$\begin{aligned}
&\forall i\in\{1,\ldots,k\},\forall t\in\{1,2\}:\\
&\left(\mu^{(t,i)}+\sum_{u\in I}\left(\mu_{h,u}^{(t,i)}h_u\right)+\sum_{j=1}^{(i-1)}\left(\mu_{B,j}^{(t,i)}b_j+\mu_{s,j}^{(t,i)}s_j\right)\right)\\
&=\tau_t^{(i)}h_{\mathsf{aux}_{k_i}}\left(\nu^{(t,i)}+\nu_x^{(t,i)}x+\nu_{y,1}^{(t,i)}y_1+\nu_{y,2}^{(t,i)}y_2+\nu_{z,1}^{(t,i)}z_1+\nu_{z,2}^{(t,i)}z_2\right)
\end{aligned}\tag{65}$$

We look this equation modulo (y_1, y_2), it becomes:

$$\begin{aligned}
&\forall i\in\{1,\ldots,k\},\forall t\in\{1,2\}:\\
&\left(\mu^{(t,i)}+\sum_{u\in I}^{k}\left(\mu_{h,u}^{(t,i)}h_u\right)+\sum_{j=1}^{i-1}\left(\mu_{B,j}^{(t,i)}b_j+\mu_{s,j}^{(t,i)}xh_{\mathsf{aux}_j}\right)\right)\\
&=\tau_t^{(i)}h_{\mathsf{aux}_{k_i}}\left(\nu^{(t,i)}+\nu_x^{(t,i)}x+\nu_{z,1}^{(t,i)}z_1+\nu_{z,2}^{(t,i)}z_2\right)
\end{aligned}\tag{66}$$

By looking the constant coefficient and the coefficients of the monomials $h_{\mathsf{aux}_{k_i}}, xh_{\mathsf{aux}_{k_i}}, h_{\mathsf{aux}_{k_i}}z_t, z_t$ and x:

$$\begin{aligned}
&\forall i\in\{1,\ldots,k\},\forall t\in\{1,2\}:\\
&\mu^{(t,i)}=0 &(67)\\
&\forall u\neq\mathsf{aux}_{k_i}:\mu_{h,u}^{(t,i)}=0 &(68)\\
&\forall j\neq i:\mu_{s,j}^{(t,i)}=0 &(69)\\
&\forall j\neq i:\mu_{B,j}^{(t,i)}=0 &(70)\\
&\nu^{(t,i)}\tau^{(i)}=\mu_{h,\mathsf{aux}_{k_i}}^{(t,i)} &(71)\\
&\tau_t^{(i)}\nu_x^{(t,i)}=\tau_t^{(i)}\nu_{z,1}^{(t,i)}=\tau_t^{(i)}\nu_{z,2}^{(t,i)}=0 &(72)
\end{aligned}$$

Then by applying (67), (68),(69), (70), (71) we deduce:

$$\forall t \in \{1,2\}, \forall i \in \{1,\dots,k'\} : M^{(t,i)} = \nu^{(t,i)} \tau^{(i)} H_{\mathsf{aux}_{k_i}} \tag{73}$$

Because $\tau_t^{(i)} \neq 0$, we deduce from (72):

$$\forall i \in \{1,\dots,k\}, \forall t \in \{1,2\} : \nu_x^{(t,i)} = \nu_{z,1}^{(t,i)} = \nu_{z,2}^{(t,i)} = 0 \tag{74}$$

Then by applying (67), (68),(69),(70), (74) to (65):

$$\begin{array}{l} \forall i \in \{1,\dots,k\}, \forall t \in \{1,2\} : \\ \left(\mu_{h,\mathsf{aux}_{k_i}}^{(t,i)} h_{\mathsf{aux}_{k_i}}\right) = \tau_t^{(i)} h_{\mathsf{aux}_{k_i}} \left(\nu^{(t,i)} + \nu_{y,1}^{(t,i)} y_1 + \nu_{y,2}^{(t,i)} y_2\right) \end{array} \tag{75}$$

We deduce form this equation by looking degree 2 monomials

$$\nu_{y,1}^{(t,i)} = \nu_{y,2}^{(t,i)} = 0 \tag{76}$$

From (74), and (76), we deduce:

$$\forall t \in \{1,2\}, \forall i \in \{1,\dots,k'\} : \hat{N}^{(t,i)} = \nu^{(t,i)} \hat{G} \tag{77}$$

Then, by applying (67),(68), (69), (70) the equation (11) can be rewritten:

$$s_i = \left(x h_{\mathsf{aux}_{k_i}} + \sum_{t=1}^{2} y_b \mu_{h,\mathsf{aux}_{k_i}}^{(t,i)} h_{\mathsf{aux}_{k_i}}\right) \tag{78}$$

Now, let reuse the equation (61) with the knowledge of (63),(**??**).

$$\begin{array}{l} \left(\sum_{a\in I} (\eta_{h,a} h_a)\right) \chi_x x + \\ \sum_{t\in[2]} \left(\sum_{u\in I} \left(\mu_{h,u}^{(t,*,1)} h_u\right) + \sum_{j=1}^{k} \left(\mu_{B,j}^{(t,*,1)} b_j + \mu_{s,j}^{(t,*,1)} s_j\right)\right) \chi_x y_t + \\ \sum_{2\leq i\leq \ell} \left(\left(\sum_{a\in I} (\eta_{h,a} h_a)\right) u^{(i)} \varsigma_x^{(i)} + \right. \\ \left. \sum_{t\in[2]} \left(\sum_{a\in I} \left(\mu_{h,a}^{(t,*,i)} h_a\right) + \sum_{j=1}^{k} \left(\mu_{B,j}^{(t,*,i)} b_j + \mu_{s,j}^{(t,*,i)} s_j\right)\right) u^{(i)} \varsigma_{y_t}^{(i)} \right) \\ = \left(\sum_{a\in I} (\sigma_{h,a} h_a) + \sum_{j=1}^{k} (\sigma_{B,j} b_j + \sigma_{s,j} s_j)\right) \end{array} \tag{79}$$

And if we look degree 3 monomials of this equation by having in mind (10), (78), we deduce:

$$\begin{array}{l} \sum_{t\in[2]} \sum_{j=1}^{k} \left(\mu_{B,j}^{(t,*,1)} b_j + \mu_{s,j}^{(t,*,1)} s_j\right) \chi_x y_t = 0 \\ \sum_{j=1}^{k} h_{\mathsf{aux}_{k_j}} \left(\sum_{t\in[2]} \mu_{B,j}^{(t,*,1)} \tau_t^{(j)} z_t + \chi_x y_t \mu_{s,j}^{(t,*,1)} \left(x + \sum_{t=1}^{2} y_b \mu_{h,\mathsf{aux}_{k_i}}^{(t,j)}\right)\right) = 0 \end{array}$$

Because $s_i \neq 0$, (from (78)) and (36), and also because all the $h_{\mathsf{aux}_{k_j}}$'s are distinct, we deduce:

$$\forall t \in [2] \forall j \in \{1,\dots,k\} : \mu_{s,j}^{(t,*,1)} = 0 \tag{80}$$

And because the $\tau_t^{(j)}$'s are non zero, we deduce:

$$\forall j \in \{1,\dots,k\}, \forall t \in \{1,2\} : \mu_{B,j}^{(t,*,1)} = 0 \tag{81}$$

Then we use (81) and (80) in (60) for $i = 1$

$$\begin{aligned}
&\forall t \in \{1,2\} : \\
&\left(\sum_{a\in I}\left(\tau_{h,a}^{(t)} h_a\right)\right)\left(\nu^{(t,*,1)} + \nu_x^{(t,*,1)} x + \nu_{y,1}^{(t,*,1)} y_1 + \nu_{y,2}^{(t,*,1)} y_2 + \nu_{z,1}^{(t,*,1)} z_1 + \nu_{z,2}^{(t,*,1)} z_2\right) \\
&= \sum_{a\in I}\left(\mu_{h,a}^{(t,*,1)} h_a\right)
\end{aligned} \tag{82}$$

And by looking degree 2 monomials and arguing the T_t are non zero, we deduce:

$$\forall t \in \{1,2\} : \nu_x^{(t,*,1)} = \nu_{y,1}^{(t,*,1)} = \nu_{y,2}^{(t,*,1)} = \nu_{z,1}^{(t,*,1)} = \nu_{z,2}^{(t,*,1)} = 0 \tag{83}$$

It means

$$\forall t \in \{1,2\} : \hat{N}^{(t,*,1)} = \nu^{(t,*,1)} \hat{G} \tag{84}$$

Now we can use the fact, VerifyTag output 1. It implies

$$\forall t \in \{1,2\} : T_t^* = \tau_t^* H^* \tag{85}$$

Then:

$$\forall t \in \{1,2\} : \sum_{a\in I}\left(\tau_{h,a}^{(t)} h_a\right) = \tau_t^* \left(\sum_{a\in I}\left(\eta_{h,a} h_a\right)\right)$$

It implies by using the non equality between the h:

$$\forall t \in \{1,2\} \forall a \in I : \tau_{h,a}^{(t)} = \tau_t^* \eta_{h,a} \tag{86}$$

We apply (80),eq:mubjzero, (86) to (79):

$$\begin{aligned}
&\left(\sum_{a\in I}\left(\frac{\tau_{h,a}^{(t)}}{\tau_t^*} h_a\right)\right)\chi_x x + \sum_{t\in[2]}\left(\sum_{u\in I}\left(\mu_{h,u}^{(t,*,1)} h_u\right)\right)\chi_x y_t + \\
&\sum_{2\le i\le \ell}\left(\left(\sum_{a\in I}\left(\eta_{h,a} h_a\right)\right) u^{(i)} \varsigma_x^{(i)} + \right. \\
&\left.\sum_{t\in[2]}\left(\sum_{a\in I}\left(\mu_{h,a}^{(t,*,i)} h_a\right) + \sum_{j=1}^{k}\left(\mu_{B,j}^{(t,*,i)} b_j + \mu_{s,j}^{(t,*,i)} s_j\right)\right) u^{(i)} \varsigma_{y_t}^{(i)}\right) \\
&= \left(\sum_{a\in I}\left(\sigma_{h,a} h_a\right) + \sum_{j=1}^{k}\left(\sigma_{B,j} b_j + \sigma_{s,j} s_j\right)\right)
\end{aligned} \tag{87}$$

We only look degree two monomials in this equation modulo (z_1, z_2) (to erase the b's) by recalling (78):

$$\begin{aligned}
&\left(\sum_{a\in I}\left(\frac{\tau_{h,a}^{(t)}}{\tau_t^*} h_a\right)\right)\chi_x x + \sum_{t\in[2]}\left(\sum_{u\in I}\left(\mu_{h,u}^{(t,*,1)} h_u\right)\right)\chi_x y_t + \\
&\sum_{2\le i\le \ell}\sum_{t\in[2]}\sum_{j=1}^{k}\mu_{s,j}^{(t,*,i)} s_j u^{(i)} \varsigma_{y_t}^{(i)} = \sum_{j=1}^{k}\left(\sigma_{s,j} s_j\right)
\end{aligned}$$

Now we use (36):

$$\left(\sum_{a\in I}\left(\frac{\tau_{h,a}^{(t)}}{\tau_t^*}h_a\right)\right)x+\sum_{t\in[2]}\left(\sum_{u\in I}\left(\mu_{h,u}^{(t,*,1)}h_u\right)\right)y_t$$
$$=\sum_{j=1}^{k}\frac{1}{\chi_x}\left(\sum_{2\leq i\leq \ell}\sum_{t\in[2]}\sigma_{s,j}-\mu_{s,j}^{(t,*,i)}u^{(i)}\varsigma_{y_t}^{(i)}\right)s_j \tag{88}$$

We define:

$$\forall j\in\{1,\dots,k\}:\alpha_j:=\frac{1}{\chi_x}\left(\sum_{2\leq i\leq \ell}\sum_{t\in[2]}-\mu_{s,j}^{(t,*,i)}u^{(i)}\varsigma_{y_t}^{(i)}+\sigma_{s,j}\right) \tag{89}$$

By using (78), we deduce:

$$\forall u\in I':\frac{\tau_{h,u}^{(t)}}{\tau_t^*}=\mu_{h,u}^{(t,*,1)}=0 \tag{90}$$

Then applying (89), (90) on (88):

$$\sum_{j=1}^{k}h_{\mathsf{aux}_{k_j}}\left(\left(\frac{\tau_{h,\mathsf{aux}_{k_j}}^{(t)}}{\tau_t^*}\right)x+\sum_{t\in[2]}\mu_{h,u}^{(t,*,1)}y_t\right)=\sum_{j=1}^{k}\alpha_j s_j$$

We use now (78) in this equation:

$$\sum_{j=1}^{k}h_{\mathsf{aux}_{k_j}}\left(\left(\frac{\tau_{h,\mathsf{aux}_{k_j}}^{(t)}}{\tau_t^*}\right)x+\sum_{t\in[2]}\mu_{h,\mathsf{aux}_{k_j}}^{(t,*,1)}y_t\right)= \tag{91}$$

$$\sum_{j=1}^{k}\alpha_j\left(xh_{\mathsf{aux}_{k_i}}+\sum_{t=1}^{2}y_b\mu_{h,\mathsf{aux}_{k_j}}^{(t,j)}h_{\mathsf{aux}_{k_j}}\right)$$

We deduce:

$$\forall j\in\{1,\dots,k\}\forall t\in\{1,2\}:\frac{\tau_{h,\mathsf{aux}_{k_j}}^{(t)}}{\tau_t^*}=\alpha_j \tag{92}$$

$$\mu_{h,\mathsf{aux}_{k_j}}^{(t,*,1)}=\alpha_j\mu_{h,\mathsf{aux}_{k_j}}^{(t,j)} \tag{93}$$

Now we apply (83), (90), (92), (93) to (82)

$$\forall t\in\{1,2\}:$$
$$\sum_{j=1}^{k}\alpha_j\nu^{(t,*,1)}\tau_t^*h_{\mathsf{aux}_{k_j}}=\sum_{j=1}^{k}\alpha_j\mu_{h,\mathsf{aux}_{k_j}}^{(t,j)}h_{\mathsf{aux}_{k_j}} \tag{94}$$

Let's define

$$K=\{i\in\{1,\dots,k\}|\alpha_i\neq 0\}. \tag{95}$$

Because $H^*\neq 0$ is non zero we deduce:

$$K\neq\emptyset. \tag{96}$$

Now, by combining (93), (95) we deduce:

$$\forall j \notin K : \mu_{h,\mathsf{aux}_{k_j}}^{(t,*,1)} = 0 \tag{97}$$

And by applying (95) in (94):

$$\begin{aligned}&\forall t \in \{1,2\} : \\ &\sum_{j\in K} \tau_t^* \nu^{(t,*,1)} h_{\mathsf{aux}_{k_j}} = \sum_{j\in K} \mu_{h,\mathsf{aux}_{k_j}}^{(t,j)} h_{\mathsf{aux}_{k_j}}\end{aligned}$$

Then :

$$\forall t \in \{1,2\} \forall j \in K : \nu^{(t,*,1)} \cdot \tau_t^* = \mu_{h,\mathsf{aux}_{k_j}}^{(t,j)} \tag{98}$$

Then, we deduce, by applying (67), (68), (69), (70), (98) :

$$\forall t \in \{1,2\}, \forall j \in K : M^{(t,j)} = \nu^{(t,*,1)} \cdot \tau_t^* H_{\mathsf{aux}_{k_j}} \tag{99}$$

Now we apply (48), (52), (54), (92), (95) in (50):

$$\begin{aligned}&\left(\sum_{a\in I} (\beta_{h,a} h_a) + \sum_{j=1}^{k} \left(\beta_{B,j} b_j + \beta_{s,j} s_j\right)\right) \\ &= \sum_{t\in\{1,2\}} \left(\sum_{j\in K} \tau_t^* \alpha_j h_{\mathsf{aux}_{k_j}}\right) \chi_x z_t \\ &\quad + \sum_{i\in[2;\ell],t\in\{1,2\}} \left(\sum_{j\in K} \tau_t^* \alpha_j h_{\mathsf{aux}_{k_j}}\right) \zeta^{(i,t)}\end{aligned} \tag{100}$$

We look monomials non-constant in z_1 or z_2 and by recalling (10) and (78), we deduce:

$$\begin{aligned}&\sum_{j=1}^{k} \beta_{B,j} \sum_{t\in\{1,2\}} z_t \tau_j^{(t)} \\ &= \sum_{t\in\{1,2\}} \left(\sum_{j\in K} \alpha_j h_{\mathsf{aux}_{k_j}}\right) \chi_x \tau_t^* z_t\end{aligned} \tag{101}$$

We deduce, because the $\tau_j^{(t)}$'s are non zero:

$$\forall j \notin K : \beta_{B,j} = 0 \tag{102}$$

$$\forall j \in K, \forall t \in \{1,2\} : \beta_{B,j} \tau_j^{(t)} = \alpha_j \chi_x \tau_t^* \tag{103}$$

It means, because (36) and the $\tau_j^{(t)}$'s are non zero:

$$\forall j \in K : \frac{\beta_{B,j}}{\alpha_j \chi_x} \left(\tau_j^{(1)}, \tau_j^{(2)}\right) = (\tau_1^*, \tau_2^*) \tag{104}$$

It implies, by non-zeroness of τ_1^* and τ_2^*):

$$\forall j \in K : \beta_{B,j} \neq 0. \tag{105}$$

If we synthetize our knowledge ((84),(85), (99)) we obtain:

$$(\mathbf{N}^*, \mathbf{M}^*, \mathbf{T}^*) \tag{106}$$

$$= \left(\left(\nu^{(1,*,1)} \hat{G}, \nu^{(2,*,1)} \hat{G}\right), \left(\nu^{(1,*,1)} \tau_1^* H^*, \nu^{(2,*,1)} \tau_2^* H^*\right), (\tau_1^* H^*, \tau_2^* H^*)\right) \tag{107}$$

Let $i \in K$ (such an i exists from (96)). By using (73), eq:Nchapeaudevoile, and (104)

$$(\mathbf{N}_i, \mathbf{M}_i, \mathbf{T}_i) \tag{108}$$

$$= \Big(\left(\nu^{(1,*,1)} \hat{P}, \nu^{(2,*,1)} \hat{P} \right), \left(\nu^{(1,*,1)} \tau_1^* H_{\mathsf{aux}_{k_i}}, \nu^{(2,*,1)} \tau_2^* H_{\mathsf{aux}_{k_i}} \right), \tag{109}$$

$$\frac{\beta_{B,i}}{\alpha_i \chi_x} \left(\tau_1^* H_{\mathsf{aux}_{k_i}}, \tau_2^* H_{\mathsf{aux}_{k_i}} \right) \Big) \tag{110}$$

And because (105), we deduce the forgery doesn't check the final winning condition. So the scheme is secure.

Difference between ideal and real game We start with upper-bounding the degree of the polynomials that can be generated by the adversary. We first remark by combining (10), and (78), that the maximum degree of the elements in G_1 is 2 and 1 in G_2. Now, we can count the number of elements used by the adversary, there is $\left(2k + k' + q_{\mathbb{G}_1} + 1\right)$ elements in $\mathbb{G}_1$, and $\left(5 + q_{\mathbb{G}_2}\right)$ in $\mathbb{G}_2$.

So it implies there is at most $\left(2k + k' + q_{\mathbb{G}_1} + 1\right)\left(5 + q_{\mathbb{G}_2}\right) + q_{\mathbb{G}_T}$ elements of degree 3 in $\mathbb{G}_T$.

The "ideal" model and the generic group model differ if and only if two elements are distinct as polynomial but identical as (handle of a) group element. That is, if we evaluate two different polynomials corresponding to the group $\mathbb{G}_T$ at scalar values $x, y_1, y_2, z_2, z_2 h_1, \ldots, h_{k'}$ and obtain the same result after pairings. Then by applying Schwartz-Zippel lemma, we deduce the following upperbound $\frac{3\left(\left(2k+k'+q_{\mathbb{G}_1}+1\right)\left(5+q_{\mathbb{G}_2}\right)+q_{\mathbb{G}_T}\right)^2}{p-1}$.