Constant-Size Issuer Hiding for BBS Credentials via Randomizable Keys

Andrea Flamini, Karla Friedrichs, and Anja Lehmann

Hasso Plattner Institute, University of Potsdam, Germany
{andrea.flamini,karla.friedrichs,anja.lehmann}@hpi.de

Abstract. Anonymous credentials (AC) equip users with credentials on attested attributes, which enable them to prove data-minimizing statements over their attributes. However, in standard ACs, each credential presentation reveals the credential issuer, which could be more information than intended and necessary, e.g., when merely proving age or personhood. *Issuer-Hiding Anonymous Credentials* (IHAC) address this limitation and hide the issuer in the presentation. That is, they only reveal that the user has a credential from an issuer within a certain trust set, referred to as the *policy*. Recent works by Sanders and Traoré, and Katz and Sefranek show how to add issuer hiding to PS- and BBS-based credentials while keeping presentations constant-size, i.e., not scaling in the number of issuers. However, both constructions require the verifier to generate dedicated policy-key pairs, turning verification into a secret-key operation. Managing these verifier-specific keys introduces additional complexity and affects the resulting practical privacy and security guarantees. An orthogonal work by Flamini et al. avoids verifier-specific keys but at the cost of presentations that scale linearly—or logarithmically when using more involved proof systems—in the policy size, which becomes a bottleneck for scenarios with hundreds or thousands of issuers. In this work, we close this gap and propose IHBBS_τ, the first IHAC scheme from standard BBS signatures that achieves *constant-size* presentations and is *publicly verifiable*, i.e., does not rely on any verifier-specific secret keys. At the core of our construction is a new technique to *multiplicatively* randomize BBS public keys and adapt the signatures accordingly, which we believe to be of independent interest.

Keywords: issuer hiding · anonymous credentials · BBS

1 Introduction

Anonymous credentials (AC) are an established tool to enable secure yet privacy-preserving authentication [Cha85,CL01,CL03,CL04]. The setup is as in classical certificates: a trusted issuer signs a list of user attributes, which forms the user's credential. The user can then present their attested attributes in a publicly verifiable way using the credential. The distinguishing feature of ACs is that they allow to make these presentations in a privacy-preserving way: with every

presentation, the user can decide which attributes they want to reveal, and presentations are unlinkable even when the issuer is corrupt. The current desire to build secure and privacy-respecting digital identity systems around the globe has renewed real-world interest in these techniques. In particular, the European effort to build a European Digital Identity (EUDI) Wallet [eud23] is rooted in the eIDAS2.0 regulation that mandates unlinkable authentications with minimal disclosure [eid24], effectively requiring the use of ACs.

Interestingly, the privacy expectations for ACs go beyond what they naturally offer: a typical example is that minimal disclosure of a university degree should merely reveal that the user has a certain degree, but not from which university. While it is trivial to hide any attribute that contains the university's name, this information is still leaked in a normal AC system: the issuer's identity is always revealed in a presentation through their public key. Likewise, in national identity documents, the issuer is typically the country or local state a user is registered in. Thus, even a seemingly anonymous proof of age or personhood would reveal the user's home state. In fact, the practical deployment of large-scale systems might even distribute the load over several smaller issuance authorities, e.g., on federal state or municipality level. Revealing the issuer public with every presentation would then significantly reduce the anonymity set. In the context of the EUDI wallet, *issuer hiding* has therefore been identified as an explicit requirement for a ZKP-based solution (`ZKP01(v)`) [eud26].

Issuer-Hiding Anonymous Credentials. ACs that reduce the leakage of information about the issuer are known as *Issuer-Hiding Anonymous Credentials* (IHAC). Informally, in IHAC the verifier no longer learns that the user holds a credential from a specific issuer with key ipk_i, but only that they own one from an issuer within a well-defined set (or *policy*) $pol := \{ipk_1, \ldots, ipk_n\}$ of trusted issuers. Such a policy could, for instance, contain all EU member states or all universities of a certain country. Issuer hiding has been studied explicitly only recently [BEK+21,CLPK22], and already inspired several further constructions [MBG+23,ST24,KS25,FFK+26,KKLT26]. In essence, there are currently three main approaches how to construct such IHAC:

OR proofs over the policy. This is the naïve and most generic approach to issuer hiding, also described by [BEK+21]: it consists of generating a standard OR proof [CDS94] that proves that the user owns a credential from one of the issuers in *pol*. Using this approach, the proof size results to be linear in the number of issuers, which can be reduced to logarithmic for instance with [GGHAK22]. A recent work by Flamini et al. [FFK+26] applies and optimizes the OR-proof technique to BBS credentials. Their construction leverages a new technique to randomize an issuer's BBS public key and adapt the BBS signature accordingly, so that the OR proof is run only over the issuer keys rather than over full credential verification equations. This reduces the amount a presentation grows per added issuer in the policy, and applying Stacking Sigmas [GGHAK22] or Groth–Kohlweiss proofs [GK15] makes presentations only scale logarithmically in the number of issuers in the policy.

The work by Connolly et al. [CLPK22], based on structure-preserving signatures, follows the same strategy, but the issuer-hiding privacy property therein requires that all issuers are honest[1], an assumption that anonymous credentials usually try to avoid.

Constant proof size using public policy key. A technique introduced by Bobolz et al. [BEK+21] aims to achieve *constant* presentation size with respect to the number of issuers. This relies on an additional entity and cryptographic scheme to authenticate the accepted policy. Each policy is authenticated through a trusted central entity which signs each public key $ipk_i \in pol$ through its key (vsk, vpk) as $\sigma_i \leftarrow \mathsf{Sign}(vsk, ipk_i)$. The main observation is that this reduces the problem to proving that the user holds a credential *cred* signed by an ipk_j for which they also know a signature valid under vpk. The authors propose a generic construction that combines an AC scheme with a compatible signature scheme for Sign. The latter must hence be able to sign public keys of the main AC, typically group elements, and allow for efficient proofs of knowledge of a signature. Bobolz et al. instantiate their blueprint with Groth signatures [Gro15].

Also the work by Mir et al. [MBG+23], based on structure-preserving signatures somewhat falls into that category, but it considers a setting where the same attributes are certified by multiple issuers, and these credentials get aggregated into a compact presentation token. As in [CLPK22], the issuer-hiding privacy property therein requires that all issuers are honest.

Constant proof size using a verifier-specific key. Sanders and Traoré [ST24] recently show that issuer hiding can be achieved without a zero-knowledge proof (beyond what is needed for the selective disclosure part), which yields an even more compact solution. They use PS-based credentials [PS16] and propose an algebraic approach. On a high level, holders modify a credential to make it depend on all issuer keys included in the policy at once. Crucially, they have to use some cryptographic values that the targeted verifier generates for each policy. These values come with a secret component that the verifier needs when verifying the presentation. Its secrecy is also important to ensure unforgeability, and thus must be generated and maintained individually by every verifier. The main benefit is that [ST24] achieves more compact presentations in comparison to [BEK+21], at least for small number of attributes; this compactness gets lost if the number of attributes in a credential is high. The recent work by Katz and Sefranek [KS25] transfers this approach to BBS credentials.

Systematization of IHAC *schemes.* We first take a systematic view on the existing approaches and classify them according to the setup and trust they require towards verifiers. Our classification identifies three types of policy setup:

Type-0: no policy key. In the simplest case, only the policy *pol*, containing trusted issuer keys $ipk_1, \ldots, ipk_n$, is needed to generate and verify presentations. The recent OR-proof-based BBS scheme [FFK+26] is Type-0.

[1] A malicious issuer would be able to find out if the presented credential is one of those it has issued.

Type-1: universal policy key. Policies have an associated public key pk_{pol}, as introduced by Bobolz et al. [BEK+21], which must be generated in a trusted manner. Generation and verification of issuer-hiding presentations require pk_{pol} in addition to pol. In Bobolz et al.'s scheme sketched above, the policy key for a policy pol would be $pk_{pol} \leftarrow (vpk, \{\sigma_i\}_{\forall ipk_i \in pol})$.

Type-2: per-verifier policy key. A policy key has a secret counter part sk_{pol}, and presentations created with some pk_{pol} require the corresponding secret key to verify. Hence, the policy-key pair must be generated by the verifiers themselves. Sanders and Traoré [ST24] as well as Katz and Sefranek [KS25] use this approach.

We observe that these three types of IHAC have different implications for the level of practical privacy users can expect. Intuitively, issuer-hiding credentials should increase the privacy of users compared to standard ACs, as they do not disclose which issuer originally issued their credential. However, because presentations are now created for a specific policy pol, and possibly also a policy key pk_{pol}, privacy may actually decrease if this is not handled carefully.

For instance, a malicious verifier could make a holder use a unique policy (key): if the holder caches this policy (key) for efficiency and re-uses it in every subsequent presentation, the presentations become easily traceable. To avoid this, pol and pk_{pol} must either be ephemeral, i.e., sent from verifier to holder before every presentation, or different holders must be sure to use identical values.

Privacy and security challenges for Type-2. Note that the policies and their keys scale linearly in the number of included issuers. Thus, sending and verifying ephemeral policy keys for every presentation would render the compactness benefits of Type-1 and Type-2 solutions useless. Therefore, we do not believe sending ephemeral values to be plausible in practice. Instead, a trusted repository can make the policies and keys available universally: verifiers send a compact identifier of the desired policy, and users fetch the corresponding artifacts either from the repository or cache. For Type-0 and Type-1 IHAC this yields an efficient and privacy-preserving solution. However, for Type-2, we consider this approach unfavorable: for once, the trusted repository must manage a large number of policy keys pk_{pol}, as every verifier needs their own individual key even when they are for the same policy. The repository now also learns which policy which verifier is accepting, and—depending on the concrete implementation—possibly even for which verifier a user is requesting the policy key for.

Finally, handling secret key material on the verifier side is not without challenges either, as it directly impacts the unforgeability guarantees with respect to that verifier: the verifier must choose between generating a fresh policy-key pair for each session or maintaining a long-term policy-key pair. As argued above, a long-term key pair is certainly better in terms of efficiency. However, the verifier must then ensure proper protection of the secret key, as its exposure would allow an attacker to forge arbitrary presentations, voiding the credential system of all security guarantees. Having ephemeral keys for each session would be better for security, but remove the efficiency benefits of the Type-2 IHAC solutions. Overall,

while one can imagine scenarios suitable to Type-2 IHAC, e.g., when verifiers are also issuers, the settings in which Type-2 IHAC schemes can realistically achieve the promised efficiency seem limited.

Efficiency challenges for Type-0. While Type-0 IHAC avoids questions related to the distribution of public policy keys, it seems to imply presentations that scale (linearly, or logarithmically through optimization) with the policy size. Flamini et al. [FFK+26] show that Type-0 BBS instantiations can offer good performance for policy sets of up to 64 issuers, but there are natural use cases with much larger sets of issuers, where the inherent efficiency constraints of Type-0 become a bottleneck. For instance, all universities of the European Union (well over 1000), or age proofs accepted across the EU when multiple issuers per country are involved: for non-discriminatory deployment it is desirable not to rely solely on a centralized national digital-identity issuer, but to also let, e.g., banks or schools issue age credentials to their existing user bases. At this point a constant-size, Type-1 solution is much better suited.

The need for IHAC *from BBS.* As BBS credentials are undergoing standardization [LKWL26,SB25] and have been identified as the most promising candidate for a ZKP-based EUDI Wallet solution [TS126], it is particularly appealing to extend BBS with issuer-hiding capabilities in the Type-1 flavor. We therefore ask the natural question:

Can we design a Type-1 IHAC *scheme from BBS credentials with constant-size presentations?*

Neither of the two existing constant-size IHAC constructions satisfactorily answers this question. The recent issuer-hiding BBS scheme by Katz and Sefranek [KS25] is of Type-2 and crucially relies on verifier-specific secret keys, with all the privacy and security drawbacks discussed above. The generic framework of Bobolz et al. [BEK+21] is of Type-1, but is not directly compatible with BBS: it requires the underlying credential and the policy-key scheme to jointly support authentication of *group elements*, and Bobolz et al. accordingly instantiate their blueprint with Groth signatures [Gro15] rather than BBS.

Even setting BBS compatibility aside, the Bobolz et al. framework imposes two structural costs that any direct instantiation inherits, and that we would like to avoid for the sake of efficiency. First, the framework relies on a second, full-fledged signature scheme to authenticate the issuer public keys in the policy. This signature scheme must sign group elements and admit efficient proofs of knowledge of a signature, which forces relatively heavy choices such as Groth signatures and brings non-trivial computational overhead, including pairings into $\mathbb{G}_T$. Second, every presentation must combine *two* NIZKPs of knowledge of a signature: one for the credential (which is unavoidable for any AC), and one for the policy signature on the issuer public key. The latter is essentially the price the framework pays for hiding which key in the policy the credential was issued under. We therefore refine our question:

Can we authenticate the issuer public keys in the policy through a lightweight mechanism, avoiding a second signature scheme over group elements?

1.1 Our Contributions

In this work we positively answer the above questions and propose IHBBS_τ, the first Type-1 IHAC scheme built on top of standard BBS credentials. Our construction has constant-size presentations, does not require verifier-specific keys, and is analyzed in the AGM. Beyond the construction itself, we make two further contributions: a new *multiplicative* key randomization technique for BBS signatures, which we believe to be of independent interest; and a unified security framework that captures privacy and unforgeability for all three types of IHAC schemes in a single model, together with a systematic discussion of the privacy impacts of the different types. We give a comparison of our scheme and the related work in Table 1.

IHBBS_τ*: constant-size BBS-*IHAC *without verifier-specific keys.* Our construction IHBBS_τ realizes a Type-1 IHAC on top of standard BBS, with presentation size independent of the policy size and everlasting privacy. In spirit, IHBBS_τ follows the framework of Bobolz et al. [BEK+21], where a trusted entity certifies each issuer public key ipk_i in the policy so that the holder can prove possession of a credential under a certified key. This direct application of Bobolz et al.'s approach inherits the structural costs discussed above: a second, heavy signature scheme and an additional NIZKP of signature knowledge per presentation; and on top requires a security proof in the Generic Group Model (GGM).

IHBBS_τ overcomes all three limitations. It removes all computations in $\mathbb{G}_T$, requires only a minimal extension on top of standard BBS, and is analyzed in the AGM rather than the GGM—in contrast to prior constant-size constructions. In fact, IHBBS_τ even removes any zero-knowledge proof related to the issuer hiding, an efficiency perk that was so far only achieved by Type-2 constructions [ST24,KS25]. The key observation is that one does not need a full-fledged signature scheme to authenticate issuer public keys, since the main trust is already established through the unforgeability of BBS signatures on the attributes. We therefore replace the policy signature by a much simpler primitive that we call a *tag*. Tags are designed to be *controllably malleable*: given a tag τ on a message m, anyone can derive a valid tag τ' on a randomization m^α of m, but tags for unrelated messages remain unforgeable. This is exactly what we need: holders first randomize the issuer public key (together with the BBS signature) and then publish the randomized key in the clear together with its adapted tag, with no further proof of knowledge required. We give a very simple tag construction that provides this controlled malleability while preventing tag forgeries beyond legitimate randomizations.

Multiplicative key randomization for BBS. At the heart of IHBBS_τ lies a new randomization technique for BBS public keys, which we believe to be of independent interest. BBS public keys are of the form $ipk = \tilde{X} = \tilde{G}_2^x$, and we show

that they can be randomized as $\tilde{X}^\star = ipk^r$ for a random $r \leftarrow\!\!\$\ \mathbb{Z}_p^*$, while the corresponding BBS signature $cred = (A, e)$ on attributes $\mathbf{a}$ can be adapted into $cred^\star \leftarrow (A^{1/r}, er)$ that verifies on $\mathbf{a}$ under $\tilde{X}^\star$. Note that this is not a full randomization of the BBS signature, but it is sufficient for our use case, as BBS signatures are never revealed in an AC system anyway: only knowledge of them is proven, and for issuer hiding we ultimately rely only on the randomization of the issuer public key.

An alternative would be to randomize *ipk additively*, e.g., by computing $ipk \cdot \tilde{G}_2^r$ so that the secret counterpart is $x + r$. This is the approach taken in the concurrent Type-0 construction of Flamini et al. [FFK+26]. For our Type-1 setting, however, multiplicative randomization is essential, and additive randomization would not suffice. Concretely, in IHBBS_τ the trusted entity tags each policy public key as $\tilde{T}_i \leftarrow ipk_i^{1/x_v}$ using a secret x_v, and the user randomizes $(ipk_i, \tilde{T}_i)$ into $(ipk_i^r, \tilde{T}_i^r)$. Without knowing x_v or the secret keys of honest issuers, no one can produce a tag on a public key outside the policy. The analogous additive design would tag keys as $\tilde{T}_i \leftarrow ipk_i \cdot \tilde{G}_2^{1/x_v}$, but this is insecure: from any single pair $(ipk_i, \tilde{T}_i)$ everyone can extract $\tilde{G}_2^{1/x_v}$ and forge a valid tag for an arbitrary public key. A small technical price is that multiplicative randomization only covers $\mathbb{G}_2 \setminus \{\mathbf{1}_{\mathbb{G}_2}\}$, so we explicitly exclude the trivial secret key 0 from key generation and sample r from $\mathbb{Z}_p^*$ rather than $\mathbb{Z}_p$.

Unified security framework for Type-0/1/2 IHAC. Along the way, we introduce a unified security framework that captures both privacy and unforgeability for all three flavors of IHAC: Type-0, Type-1, and Type-2 IHAC schemes. Further, we make explicit the qualitative differences between the three types: in particular, the inherent trade-offs of Type-2 constructions, which arise from handling secret policy material on the verifier side. We hope that this unified treatment, together with our systematization of the three types, provides a useful reference point for future work on issuer-hiding credentials.

Table 1. Comparison of some relevant IHAC approaches. **BBS** refers to the compatibility with standard BBS credentials. **Overhead** denotes the extra presentation costs needed for n issuers and $|\mathcal{U}|$ undisclosed attributes. **No** $\mathbb{G}_T$ captures if the presentation requires computations in $\mathbb{G}_T$, and **Public Vf** whether verification requires a secret policy key.

Type	Scheme	BBS	Overhead	Model	No $\mathbb{G}_T$	Public Vf
0	[FFK+26]	✓	$O(n)/O(\log n)$	ROM	✓	✓
1	[BEK+21]	✗	$O(1)$	GGM + ROM	✗	✓
1	$\mathsf{IHBBS}_{\mathtt{Gro}}$	(✓)	$O(1)$	GGM + ROM	✗	✓
1	IHBBS_τ	✓	$O(1)$	AGM + ROM	✓	✓
2	[ST24]	✗	$O(1)$	GGM + ROM	✓	✗
2	[KS25]	✓	$O(1)$	GGM + ROM	✓	✗

Concurrent Work. Concurrently to ours, Kaaniche et al. [KKLT26] also develop a Type-1 BBS-based IHAC scheme. Among other contributions, they propose a Type-1 construction that follows the framework of Bobolz et al. [BEK+21], introducing a BBS-like signature scheme over group elements to certify issuer public keys. Their presentation protocol, however, still involves computing a zero-knowledge proof of knowledge of a signed message—namely, of the issuer's public key under the policy signature—which is precisely what our construction is designed to avoid in order to keep the presentation protocol lightweight and compact. We refer to Section 6 for a detailed comparison of performance of our scheme and their pairing based scheme.

Kaaniche et al. additionally claim that our construction suffers from a critical vulnerability that would render it unsuitable for real-world deployment. The alleged attack proceeds as follows: a malicious issuer in the policy generates a rogue public key by randomizing its own, and computes a valid tag for that rogue key by applying the same randomness; the issuer can then issue credentials under this rogue public key that appear unlinkable to its real identity. However, that is not an attack on the unforgeability of our scheme: it relies on a corrupt issuer that is part of the policy, which is explicitly excluded from the unforgeability experiment—in our model and in theirs alike. We further note that a malicious issuer could in any case go much further, and fully undermine all unforgeability guarantees. This is a general limitation in issuer-hiding schemes, and investigating whether security models—and schemes—can tolerate a mild form of issuer corruption is an interesting open problem.

Paper History. An earlier version of this work contained both a Type-0 and a Type-1 BBS-based IHAC construction. The Type-0 scheme has been split off and merged into a separate paper by Flamini et al. [FFK+26], which now also contains a detailed study of how the linear size overhead of OR-proof-based presentations can be reduced through more advanced proof techniques such as Stacking Sigmas and Groth–Kohlweiss proofs. This paper now focuses exclusively on the constant-size Type-1 construction IHBBS_τ, and additionally develops a common framework and security model that covers all three IHAC types together with a systematic discussion of their respective privacy and efficiency trade-offs.

2 Preliminaries

Notation. Let λ represent the security parameter of our schemes. Throughout the algorithms given in this work, we use $\leftarrow$ for deterministic, and $\leftarrow\!\!\$$ for probabilistic assignment or sampling from a set. We designate by $\emptyset$ empty sets and by () empty dictionaries, where values are accessed by key.

(Bilinear) groups. We consider throughout this work groups $\mathbb{G}$ of prime order p. The presented BBS-based schemes rely on asymmetric bilinear groups. We use the convention of denoting the source groups with $\mathbb{G}_1$ and $\mathbb{G}_2$, and the target group with $\mathbb{G}_T$. For convenience, we write elements $G \in \mathbb{G}_1$ in plain, elements

$\tilde{G} \in \mathbb{G}_2$ with a tilde, and $\hat{G} \in \mathbb{G}_T$ with a hat. All three groups are of order p, and a the respective generators are called G_1, $\tilde{G}_2$, and $\hat{G}_T$. As an additional structure, there is a *pairing*, an efficiently computable map $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ which fulfills *bilinearity* (for any $x, y \in \mathbb{Z}_p$, it holds that $\mathbf{e}(G_1^x, \tilde{G}_2^y) = \mathbf{e}(G_1, \tilde{G}_2)^{xy}$) and *non-degeneracy* (for all generators G_1 and $\tilde{G}_2$, $\mathbf{e}(G_1, \tilde{G}_2)$ is also a generator). For the setup of public parameters, we assume a bilinear group generation algorithm $\mathsf{BGGen}(1^\lambda) \xmapsto{\$} (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbf{e}, G_1, \tilde{G}_2, \hat{G}_T)$.

Digital signatures. Digital signatures are defined by a set of four algorithms: $\mathsf{PGen}(1^\lambda)$ outputs public parameters pp, $\mathsf{KGen}(pp)$ generates a signer-key pair (sk, pk), $\mathsf{Sign}(sk, m)$ lets the signer create a signature σ on a message m, and $\mathsf{Vf}(pk, \sigma, m)$ returns 1 iff σ is a valid signature of m under key pk.

The standard security notions for signatures are *existential* and *strong* unforgeability under chosen message attacks (UF-CMA and SUF-CMA), respectively. These notions are captured by two very similar experiments: the adversary receives from the challenger the public parameters and a public key pk. Then, it can request a polynomial number $q(\lambda)$ of signatures for messages m_i of its choice, receiving the associated signature $\sigma_i \leftarrow \mathsf{Sign}(sk, m_i)$ every time. Finally, the adversary produces a forgery $(m^\dagger, \sigma^\dagger)$. For both UF-CMA and SUF-CMA, it is checked that $\sigma^\dagger$ is valid on $m^\dagger$ and under pk. In the weaker UF-CMA, the adversary wins if it never queried a signature on $m^\dagger$ (i.e., $\forall i \in [q] : m^\dagger \neq m_i$); in the stronger SUF-CMA, the adversary additionally wins if $m^\dagger$ was among the queries, but $\sigma^\dagger$ is fresh (i.e., $\forall i \in [q] : (m^\dagger, \sigma^\dagger) \neq (m_i, \sigma_i)$).

BBS signatures and anonymous credentials. BBS signatures permit efficient ZKPs of knowledge of a signature and therefore lend themselves to anonymous credentials [CL04,CDL16,TZ23]. We assume all algorithms receive the public parameters pp, but omit them from the inputs for brevity.

Definition 1 (BBS Signatures). *BBS signatures are defined by the following algorithms. Here, messages are always a vector* $\mathbf{a}$ *of* ℓ *elements* $(a_1, \ldots, a_\ell)$.

$\mathsf{PGen}(1^\lambda)$: *Generate* $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbf{e}, G_1, \tilde{G}_2, \hat{G}_T) \leftarrow\!\!\$\ \mathsf{BGGen}(1^\lambda)$. *Then sample* $(H_1, \ldots, H_\ell) \leftarrow\!\!\$\ \mathbb{G}_1^\ell$ *and return* $pp \leftarrow (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbf{e}, G_1, \tilde{G}_2, H_1, \ldots, H_\ell)$.

$\mathsf{KGen}(pp)$: *Sample* $x \leftarrow\!\!\$\ \mathbb{Z}_p$. *Compute* $\tilde{X} \leftarrow \tilde{G}_2^x$, *set* $sk \leftarrow x$, *and* $pk \leftarrow \tilde{X}$.

$\mathsf{Sign}(sk, \mathbf{a})$: *Compute* $C(\mathbf{a}) \leftarrow G_1 \prod_{i=1}^{\ell} H_i^{a_i}$. *Sample* $e \leftarrow\!\!\$\ \mathbb{Z}_p$ *and compute* $A \leftarrow C(\mathbf{a})^{\frac{1}{e+x}}$. *If* $A = 1_{\mathbb{G}_1}$, *return* $\perp$. *Else, output* (A, e).

$\mathsf{Vf}(pk, (A, e), \mathbf{a})$: *Check* $A \neq 1_{\mathbb{G}_1}$. *Set* $C(\mathbf{a}) \leftarrow G_1 \prod_{i=1}^{\ell} H_i^{a_i}$ *and return 1 iff* $\mathbf{e}(A, \tilde{X}\tilde{G}_2^e) = \mathbf{e}(C(\mathbf{a}), \tilde{G}_2)$.

Note that an alternative check for Vf is $\mathbf{e}(A, \tilde{X}) = \mathbf{e}(C(\mathbf{a})A^{-e}, \tilde{G}_2)$.

To build AC from BBS, signatures (A, e) on attributes $\mathbf{a}$ serve as credentials, and ZKPs allows the holder to present their credential disclosing only a subset of the attributes. Such efficient ZKPs of knowledge for BBS signatures are given by [TZ23]; we repeat one of them below. We denote the indices of revealed

attributes by $\mathcal{D}$, while corresponding attribute values are called $(a_i)_{i\in\mathcal{D}}$. Analogously, we call the indices of hidden attributes $\mathcal{U}$. Generation and verification of such presentations of BBS credentials is defined as follows:

$\mathsf{Present}_{\mathsf{BBS}}(ipk, cred, \mathbf{a}, \mathcal{D}, nonce)$: *Parse* $\tilde{X} \leftarrow ipk$. *The statement to be proven is* $(\tilde{X}, \mathcal{D}, (a'_i)_{i\in\mathcal{D}})$, *for some disclosed attributes* $(a'_i)_{i\in\mathcal{D}}$. *The witness is the credential* (A, e) *valid under* $\tilde{X}$ *and over attributes* $\mathbf{a}$ *that match the revealed ones, i.e.,* $\{a'_i\}_{i\in\mathcal{D}} = \{a_i\}_{i\in\mathcal{D}}$.
Set $C(\mathbf{a}) \leftarrow G_1 \prod_{i=1}^{\ell} H_i^{a_i}$ *and* $C(\mathbf{a}') \leftarrow G_1 \prod_{i\in\mathcal{D}} H_i^{a_i}$, *sample* $r \leftarrow\!\!\$\ \mathbb{Z}_p^*$ *and compute* $\overline{A} \leftarrow A^r$, *and* $\overline{B} \leftarrow C(\mathbf{a})^r \overline{A}^{-e}$. *Note that this implies* $\overline{B} = C(\mathbf{a}')^r \left(\prod_{j\in\mathcal{U}} H_i^{ra_i}\right) \overline{A}^{-e}$. *Finally, the holder computes* π *as:*

$$\mathsf{NIZKP}\left\{ \left((\alpha, (\beta_j)_{j\in\mathcal{U}}, \gamma), (\overline{B}, C(\mathbf{a}'), (H_j)_{j\in\mathcal{U}}, \overline{A})\right) : \overline{B} = C(\mathbf{a}')^{\alpha} \left(\prod_{j\in\mathcal{U}} H_j^{\beta_j}\right) \overline{A}^{\gamma} \right\}$$

To compute the challenge, all public values, in particular, including the nonce, need to be hashed. Obverse that we know the witness $(r, (ra_j)_{j\in\mathcal{U}}, -e)$. *Finally, we output pres* $\leftarrow (\overline{A}, \overline{B}, \pi) \in \mathbb{G}_1^2 \times \mathbb{Z}_p^{|\mathcal{U}|+3}$.

$\mathsf{VfPres}_{\mathsf{BBS}}(ipk, pres, \mathcal{D}, (a'_i)_{i\in\mathcal{D}}, nonce)$: *Parse* $\tilde{X} = ipk$ *and* $(\overline{A}, \overline{B}, \pi) \leftarrow pres$. *Compute* $C(\mathbf{a}') \leftarrow G_1 \prod_{i\in\mathcal{D}} H_i^{a'_i}$. *Return 1 iff the following three checks hold: (1)* $\overline{A} \neq 1_{\mathbb{G}_1}$, *(2)* $\mathbf{e}(\overline{A}, \tilde{X}) = \mathbf{e}(\overline{B}, \tilde{G}_2)$, *and (3)* π *is valid.*

Idealized models. Proving the security of the constructions proposed in this work will require idealized models, specifically, the Random Oracle Model and the Algebraic Group Model. Below we recall how these models are defined.

The Random Oracle Model (ROM) [BR93] allows to treat cryptographic hash functions H as random oracles RO. It assumes that the output $\mathsf{RO}(m)$ upon any input m is indistinguishable from the output of a truly random function $\mathsf{f}(m)$. Additionally, the ROM allows us to observe all queries the adversary makes to RO, and allows us to program the outputs of RO for specific inputs m.

In the Algebraic Group Model (AGM) [FKL18], adversaries are modeled as *algebraic*. Such an adversary is unable to generate fresh elements from a group, it can only compute them from elements it has seen before. Technically, every time that the adversary outputs a group element $X \in \mathbb{G}$, we force it to also output a representation $\bar{\alpha} = (\alpha_1, \ldots, \alpha_n)$ that constructs X with respect to the group elements it has seen so far $(X_1, \ldots, X_n)$, i.e. $X = \prod_{i\in[n]} X_i^{\alpha_i}$. The AGM is a weaker assumption [JM24] than the Generic Group Model (GGM) [Mau05], which was used in prior work on IHAC schemes [KS25,BEK+21,ST24,MBG+23].

3 Issuer-Hiding Anonymous Credentials (Type-1)

In this section we formally define Issuer-Hiding Anonymous Credentials (IHAC) for the Type-1 setting we focus on in this work. An extended security framework for all three types of policy settings is given in Section 7 along with a discussion of the impact they have on the practical privacy (and efficiency) of users.

3.1 Syntax of IHAC Schemes

We first recall standard Anonymous Credentials (AC), upon which IHAC extend. Regular AC schemes are made up of the following set of algorithms. For setup, PGen generates public parameters and IssKGen generates issuer keys (ipk, isk). Credentials *cred* for some attributes $\mathbf{a} = (a_1, \ldots, a_\ell)$ are issued by Issue[2], using key *isk*, and are verified with VfCred. Holders run Present to generate presentations *pres* of their credential, which in turn are verified by VfPres. To minimize the information disclosed by a presentation, holders may reveal only a predicate over their attributes instead of the full attributes. In this work, we concentrate on the *selective disclosure* predicate, but our approach can easily be extended to other predicates. Present and VfPres also take a *nonce* as input, which in practice can be used to bind presentations to a particular session or context.

Standard AC are issuer-revealing: the VfPres algorithm takes some issuer public key *ipk* as input and accepts only presentations of credentials created by this specific issuer. In contrast, Issuer-Hiding Anonymous Credentials verify presentations with respect to a *policy* $pol = \{ipk_1, \ldots, ipk_n\}$, the set of public keys of trusted issuers: given such a policy, VfPres will accept presentations of credentials created by *any of the issuers in the policy.*

The Type-1 (or *universal policy key*) setting uses for the creation and verification of presentations not only the policy *pol*, i.e. the set of accepted issuers, but also *an associated policy public key* pk_{pol}. This policy key is created through an algorithm SetPolicy for a given policy *pol*[3], and can be verified to be valid for *pol* using VfPolicy. Note that pk_{pol} is entirely public; it needs to be generated only once for a policy, can be delegated to a trusted party (or generated in a distributed fashion, for instance by the issuers in the policy) and be publicly checked for well-formedness. This type is due to Bobolz et al. [BEK+21].

Definition 2 (Type-1 IHAC). *A Type-1* IHAC *scheme is defined by the set of algorithms* (PGen, IssKGen, Issue, VfCred, SetPolicy, VfPolicy, Present, VfPres)*:*

$\mathsf{PGen}(\lambda) \xmapsto{\$} pp$: *generates public parameters pp. For simplicity, we will assume here and in the following that the public parameters of the* IHAC *scheme are shared by all the issuers in the system.*

$\mathsf{IssKGen}(pp) \xmapsto{\$} (isk, ipk)$: *generates an issuer-key pair.*

$\mathsf{Issue}(isk, \mathbf{a} = (a_1, \ldots, a_\ell)) \xmapsto{\$} cred$: *issues a credential for the attributes in* $\mathbf{a}$*.*

$\mathsf{VfCred}(ipk, cred, \mathbf{a}) \to 0/1$: *checks if credential cred is valid for attributes* $\mathbf{a}$ *and under issuer key ipk.*

$\mathsf{SetPolicy}(pol) \xmapsto{\$} pk_{pol}$: *generates a policy public key* pk_{pol} *for pol.*

$\mathsf{VfPolicy}(pol, pk_{pol}) \to 0/1$: *verifies if* pk_{pol} *is valid for the keys in pol.*

$\mathsf{Present}(pol, ipk, pk_{pol}, cred, \mathbf{a}, \mathcal{D}, nonce) \xmapsto{\$} pres$: *presents cred valid under ipk, disclosing only attributes* $(a_i)_{i\in\mathcal{D}}$*, w.r.t. policy* $pol = \{ipk_0, \ldots, ipk_n\}$ *with policy key* pk_{pol} *and for context nonce.*

[2] Issue could be replaced by an interactive protocol to support blind issuance.

[3] Note that we define the policy *pol* strictly as the set of issuer public keys. Some related works [BEK+21,MBG+23,KS25] denote with *pol* a merged object that contains both issuer public keys and the corresponding policy key pk_{pol}.

$\mathsf{VfPres}(pol, pk_{pol}, pres, \mathcal{D}, (a'_i)_{i\in\mathcal{D}}, nonce) \to 0/1$: *validates pres under policy pol and* pk_{pol}*, for revealed attributes* $(a'_i)_{i\in\mathcal{D}}$*, and context nonce.*

3.2 Security Model for IHAC

We now define the security properties that Type-1 IHAC schemes must satisfy. Our model comprises the two main properties expected from AC: *unlinkability* and *unforgeability*. We extend both to the issuer-hiding case, building upon the security models of [BEK+21,ST24,KS25,FFK+26]. Further, it is standard to consider *correctness*, i.e., the guarantee that a scheme fulfills the desired functionality if all parties behave honestly. This property is straightforward to define and prove for each construction, so we defer its treatment to Appendix A.1.

Issuer-Hiding Unlinkability. Classic unlinkability for anonymous credentials guarantees that presentations reveal nothing about the holder beyond the disclosed attributes. In standard unlinkability experiments, an adversary chooses two credentials that verify under the issuer public key and share a set of disclosed attributes. The experiment creates a presentation using one of these credentials and the adversary must guess which one has been used.

The central goal of IHAC is to strengthen this privacy notion: where basic AC produce presentations that reveal the issuer that produced the underlying credential, we want IHAC to hide even this aspect of the credential.Therefore, *issuer-hiding* unlinkability allows the adversary to specify credentials issued under *different* keys. The experiment is given in Game 1.

We model a strong adversary that controls policy generation, issuance, and verification. For each challenge, $\mathcal{A}$ chooses the inputs of two holders. As disclosures, policy (and policy key) and nonce of a presentation are publicly verifiable, these must be identical among the two input sets. Some sanity checks on the inputs are made to exclude trivial attacks where presentation of one of the input sets would simply fail: the attribute sets must overlap at the disclosed indices, the issuer keys must be part of the policy, and the credentials must be valid. Type-1 constructions also require pk_{pol} to be valid for the given policy. Note that any of these checks is easily reproducible by a wary holder.

We consider a scheme to be unlinkable if $\mathcal{A}$ is unable to differentiate presentations created from either input set—this implies that $\mathcal{A}$ learns nothing about the credential, hidden attributes, or underlying issuer of a presentation.

Remark that the game implicitly assumes that holders are able to ensure they receive the same policy and policy key as other holders. We discuss in Section 7.3 how any implementation must carefully provide such a guarantee to holders, as otherwise privacy breaks.

Definition 3 (Unlinkability). *A Type-1* IHAC *scheme* Π *is unlinkable, if* $\forall$ *PPT adversaries the following advantage w.r.t. Game 1 is negligible:*

$$\mathsf{Adv}^{\mathsf{UNLINK}}_{\Pi}(\mathcal{A}, \lambda) := \left| \Pr\left[\mathsf{Exp}^{\mathsf{UNLINK},0}_{\mathcal{A},\Pi}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{UNLINK},1}_{\mathcal{A},\Pi}(\lambda) = 1\right] \right|$$

Game 1: Unlinkability (Type-1)

$\mathsf{Exp}^{\mathsf{UNLINK},b}_{\mathcal{A},\mathsf{IHAC}}(\lambda)$

$pp \leftarrow\!\$\ \mathsf{PGen}(1^\lambda)$
$b' \leftarrow\!\$\ \mathcal{A}^{\mathcal{O}_{\mathsf{Chal}}}(pp)$
return b'

$\mathcal{O}_{\mathsf{Chal}}(pol, pk_{pol}, \mathcal{D}, nonce, ipk^{(0)}, ipk^{(1)}, cred^{(0)}, cred^{(1)}, \mathbf{a}^{(0)}, \mathbf{a}^{(1)})$

if $\mathsf{VfPolicy}(pol, pk_{pol}) = 0$: **return** $\bot$
if $(a_i^{(0)})_{i\in\mathcal{D}} \neq (a_i^{(1)})_{i\in\mathcal{D}}$: **return** $\bot$
for $d \in \{0,1\}$:
  if $ipk^{(d)} \notin pol \ \vee\ \mathsf{VfCred}(ipk^{(d)}, cred^{(d)}, \mathbf{a}^{(d)}) = 0$: **return** $\bot$
return $\mathsf{Present}(pol, pk_{pol}, ipk^{(b)}, cred^{(b)}, \mathbf{a}^{(b)}, \mathcal{D}, nonce)$

Other unlinkability definitions. Our work closely follows the existing models on issuer-hiding unlinkability: [BEK+21], [ST24], [KS25], and [FFK+26] also give left-or-right type notions with the same adversarial abilities, attack goals, and sanity checks, except for syntactical differences.

Issuer-Hiding Unforgeability. Unforgeability ensures that malicious holders are not able to generate presentations with respect to a policy *pol*, unless they have a credential from an issuer in that specific policy. This property is captured by an experiment where the adversary, during the training phase, is given access to the following oracles:

$\mathcal{O}_{\mathsf{KGen}}()$ generates keys (isk, ipk) for honest issuers. It records the pair and returns ipk to the adversary.
$\mathcal{O}_{\mathsf{SetPolicy}}(pol)$ represents a trusted entity that manages policies and creates policy public keys. The adversary must 'register' any policy here that it wants to use in a forgery. Only honest issuers are accepted in policies.
$\mathcal{O}_{\mathsf{Issue}}(ipk, \mathbf{a})$ generates credentials on behalf of honest issuers. The credentials are recorded but not given to the adversary.
$\mathcal{O}_{\mathsf{Corrupt}}(i_{cred})$ corrupts the i_{cred}-th credential, revealing it to the adversary.
$\mathcal{O}_{\mathsf{Present}}(pol, pk_{pol}, i_{cred}, \mathcal{D}, nonce)$ allows the adversary to obtain presentations of non-corrupted credentials.

Eventually, $\mathcal{A}$ outputs its forgery $pres^\dagger$ and wins the unforgeability experiment if it is a valid presentation for (1) a registered policy and (2) attributes for which no matching credential was ever leaked by $\mathcal{O}_{\mathsf{Corrupt}}$, and the presentation is fresh, i.e., not generated by $\mathcal{O}_{\mathsf{Present}}$.

Definition 4 (Unforgeability). *A Type-1* IHAC *scheme Π is* unforgeable*, if $\forall$ PPT adversaries the following advantage w.r.t. Game 2 is negligible:*

$$\mathsf{Adv}^{\mathsf{UNF}}_{\Pi}(\mathcal{A}, \lambda) := \Pr\left[\mathsf{Exp}^{\mathsf{UNF}}_{\mathcal{A},\Pi}(\lambda) = 1\right]$$

The need for $\mathcal{O}_{\mathsf{SetPolicy}}$. It might surprise that we enforce honest policy generation. Intuitively, it could seem sufficient to exclude malicious issuers in *pol* and verify adversarial pk_{pol}'s via $\mathsf{VfPolicy}$. Such a security notion would, however, not be achieved by the construction by Bobolz et al. [BEK+21], as well as our

Game 2: Unforgeability (Type-1)

$\underline{\mathsf{Exp}^{\mathsf{UNF}}_{\mathcal{A},\mathsf{IHAC}}(\lambda)}$
$pp \leftarrow\!\$\ \mathsf{PGen}(1^\lambda);\quad Q_{\mathsf{keys}}, Q_{\mathsf{creds}} \leftarrow ();\quad Q_{\mathsf{pols}}, Q_{\mathsf{pres}}, Q_{\mathsf{cor}} \leftarrow \emptyset$
$(pol^\dagger, pk^\dagger_{pol}, pres^\dagger, \mathcal{D}^\dagger, (a'^\dagger_i)_{i\in\mathcal{D}^\dagger}, nonce^\dagger) \leftarrow\!\$$
$\quad \mathcal{A}^{\mathcal{O}_{\mathsf{SetPolicy}}, \mathcal{O}_{\mathsf{KGen}}, \mathcal{O}_{\mathsf{Issue}}, \mathcal{O}_{\mathsf{Corrupt}}, \mathcal{O}_{\mathsf{Present}}}(pp)$
return 0 **if** :
$\quad$ // Policy not trusted
$\quad (pol^\dagger, pk^\dagger_{pol}) \notin Q_{\mathsf{pols}}$
$\quad$ // Matching corrupt credential exists
$\quad \vee\ \exists (cred, \mathbf{a}, ipk) \in Q_{\mathsf{cor}}$:
$\quad\quad (a'^\dagger_i)_{i\in\mathcal{D}^\dagger} = (a_i)_{i\in\mathcal{D}^\dagger} \wedge ipk \in pol^\dagger$
$\quad$ // Presentation not fresh
$\quad \vee\ (pol^\dagger, pk^\dagger_{pol}, \mathcal{D}^\dagger, (a'^\dagger_i)_{i\in\mathcal{D}^\dagger}, nonce^\dagger) \in Q_{\mathsf{pres}}$
return $\mathsf{VfPres}(pol^\dagger, pk^\dagger_{pol}, pres^\dagger, \mathcal{D}^\dagger, (a'^\dagger_i)_{i\in\mathcal{D}^\dagger}, nonce^\dagger)$

$\underline{\mathcal{O}_{\mathsf{Present}}(pol, pk_{pol}, i_{cred}, \mathcal{D}, nonce)}$
if $\mathsf{VfPolicy}(pol, pk_{pol}) = 0 \vee$ **if** $i_{cred} \notin Q_{\mathsf{creds}}$: **return** $\bot$
$(cred, \mathbf{a}, ipk) \leftarrow Q_{\mathsf{creds}}[i_{cred}]$
if $ipk \notin pol$: **return** $\bot$
$Q_{\mathsf{pres}} \leftarrow Q_{\mathsf{pres}} \cup \{(pol, pk_{pol}, \mathcal{D}, (a_i)_{i\in\mathcal{D}}, nonce)\}$
return $\mathsf{Present}(pol, pk_{pol}, ipk, cred, \mathbf{a}, \mathcal{D}, nonce)$

$\underline{\mathcal{O}_{\mathsf{SetPolicy}}(pol)}$
if $pol \not\subseteq Q_{\mathsf{keys}}$: **return** $\bot$
$pk_{pol} \leftarrow\!\$\ \mathsf{SetPolicy}(pol)$
$Q_{\mathsf{pols}} \leftarrow Q_{\mathsf{pols}} \cup \{(pol, pk_{pol})\}$
return pk_{pol}

$\underline{\mathcal{O}_{\mathsf{KGen}}()}$
$(isk, ipk) \leftarrow\!\$\ \mathsf{KGen}(pp)$
$Q_{\mathsf{keys}}[ipk] \leftarrow isk$
return ipk

$\underline{\mathcal{O}_{\mathsf{Issue}}(ipk, \mathbf{a})}$
if $ipk \notin Q_{\mathsf{keys}}$: **return** $\bot$
$cred \leftarrow\!\$\ \mathsf{Issue}(Q_{\mathsf{keys}}[ipk], \mathbf{a})$
$Q_{\mathsf{creds}}[|Q_{\mathsf{creds}}|] \leftarrow (cred, \mathbf{a}, ipk)$

$\underline{\mathcal{O}_{\mathsf{Corrupt}}(i_{cred})}$
if $i_{cred} \notin Q_{\mathsf{creds}}$: **return** $\bot$
$Q_{\mathsf{cor}} \leftarrow Q_{\mathsf{cor}} \cup Q_{\mathsf{creds}}[i_{cred}]$
return $Q_{\mathsf{creds}}[i_{cred}]$

scheme: policy keys therein are to some extent malleable and allow the adversary to secretly include more issuers to pk_{pol} than those advertised in the policy, while pk_{pol} still verifies for the original pol. This allows for a trivial win, where the adversary employs a credential from one of the added, 'hidden' issuers in the policy. Schemes with such malleable policy keys hence require honest generation of policy keys. This is what $\mathcal{O}_{\mathsf{SetPolicy}}$ models. In practice, policy keys should be generated by a mutually trusted authority (or, to avoid a central authority, in distributed fashion), anyway, so we believe this to be a reasonable assumption.

Interestingly, this is an aspect where Type-2 IHACs can have an advantage, although mostly from a theoretical standpoint: Type-1 schemes in this weaker model essentially require the verifier to trust all issuers *and* the policy generator. As the policy generator in Type-2 is the verifier itself, this additional assumption is not needed. However, as discussed in Section 7.3, practically, verifiers might want to avoid sending pk_{pol} at every presentation in order to reduce the communication overhead.

Comparison to other unforgeability definitions. Our model follows in spirit the existing IHAC models on the respective levels, but is slightly more generous in the adversarial capabilities: the Type-1 model of Bobolz et al. [BEK+21] restricts the adversary to generate a single policy and associated policy key, whereas we allow multiple out of which any can be used in the forgery.

4 Multiplicative Randomization for BBS Public Keys

At the center of our BBS-based IHAC scheme is a new randomization technique for BBS public keys, which we believe to be of independent interest. In [FFK+26], the authors already show that BBS signatures are signatures with additively randomizable keys [CGH+24] with the following randomization procedure: the user samples $r \leftarrow\!\!\$\ \mathbb{Z}_p$, and to randomize the issuer public key ipk computes $ipk^\star \leftarrow ipk\tilde{G}_2^r$. To adapt a signature (A, e) over $\mathbf{a}$ issued by ipk, so that it verifies for the same attributes $\mathbf{a}$ under $ipk^\star$, the user computes $(A^\star, e^\star) \leftarrow (A, e - r)$.

Here, we show that BBS allows also for *multiplicative* randomization, which finally enables the design of our Type-1 construction.

Re-randomizing BBS signatures via Adapt. Let (A, e) be a BBS signature on the set of attributes $\mathbf{a}$ generated by the issuer with public key $ipk = \tilde{X} = \tilde{G}_2^x$. We define an algorithm to randomize BBS public keys and adapt the associated signatures via the following map Adapt:

$\mathsf{Adapt}(\tilde{X}, (A, e), \alpha)$: takes as input a public key $\tilde{X} \neq \mathbf{1}_{\mathbb{G}_2}$, a BBS signature (A, e), and a scalar $\alpha \in \mathbb{Z}_p^*$, and outputs $(\tilde{X}^\alpha, (A^{\frac{1}{\alpha}}, e\alpha))$.

Note that, if (A, e) is a valid signature over a set of attributes $\mathbf{a}$ under the public key $\tilde{X}$, then for any $\alpha \in \mathbb{Z}_p^*$ the resulting $(A^{\frac{1}{\alpha}}, e\alpha)$ is valid on the same attributes $\mathbf{a}$ and under the public key $\tilde{X}^\alpha$, since $\mathbf{e}(A^{\frac{1}{\alpha}}, \tilde{X}^\alpha \tilde{G}_2^{e\alpha}) = \mathbf{e}(A, \tilde{X}\tilde{G}_2^e) = \mathbf{e}(C(\mathbf{a}), \tilde{G}_2)$. Also, if α is sampled uniformly at random, then also $\tilde{X}^\alpha$ is distributed uniformly in $\mathbb{G}_2 \setminus \{\mathbf{1}_{\mathbb{G}_2}\}$.

Further, we define additional notions to help the understanding of our security proofs, namely the *equivalence classes of BBS public-key/signature pairs* induced by Adapt, and the VfEquivalence algorithm that can be used to check if two pairs belong to the same class.

Equivalence class induced by Adapt. Let (A, e) be a BBS signature and $\tilde{X} \neq \mathbf{1}_{\mathbb{G}_2}$. We define the equivalence class of $(\tilde{X}, (A, e))$ induced by Adapt as:

$$[\tilde{X}, (A, e)] \leftarrow \{\mathsf{Adapt}(\tilde{X}, (A, e), \alpha)\}_{\alpha \in \mathbb{Z}_p^*}$$

It is easy to see that these sets are indeed equivalence classes (satisfying reflexivity, symmetry and transitivity) where $(\tilde{X}_2, (A_2, e_2)) \in [(\tilde{X}_1, (A_1, e_1))]$ if and only if $\exists \alpha \in \mathbb{Z}_p^* : e_1\alpha = e_2 \wedge \tilde{X}_1^\alpha = \tilde{X}_2 \wedge A_1^{\frac{1}{\alpha}} = A_2$. Therefore, leveraging the first condition ($e_2 = e_1\alpha$), there is an efficient algorithm to verify whether two BBS signatures for the same message vector live in the same class.

Verifying equivalence of public key-signature pairs via VfEquivalence. The following algorithm outputs 1 if two given signatures belong to the same class.

$\mathsf{VfEquivalence}((\tilde{X}_1, (A_1, e_1)), (\tilde{X}_2, (A_2, e_2)))$: takes two signatures $(\tilde{X}_1, (A_1, e_1))$ and $(\tilde{X}_2, (A_2, e_2))$ where $\tilde{X}_1 \neq \mathbf{1}_{\mathbb{G}_2}$, $\tilde{X}_2 \neq \mathbf{1}_{\mathbb{G}_2}$ and considers three cases:

1. if $e_1 \neq 0 \wedge e_2 \neq 0$: compute $\epsilon \leftarrow \frac{e_2}{e_1}$ and return 1 if $\tilde{X}_1^{\epsilon} = \tilde{X}_2 \wedge A_1^{\frac{1}{\epsilon}} = A_2$
2. if $e_1 = e_2 = 0$: return 1 if $\mathbf{e}(A_1, \tilde{X}_1) = \mathbf{e}(A_2, \tilde{X}_2)$
3. otherwise return 0

Note that $\alpha \neq 0$ by definition, so if between two signatures only one of e_1, e_2 is 0, they do not live in the same class.

5 BBS-based Type-1 IHAC Construction: IHBBS_{τ}

Our main construction for Type-1 IHAC pivots from the framework by Bobolz et al. [BEK+21]. The core idea therein is to entirely remove the need for an OR proof by relying on a second signature scheme that authenticates all issuer keys in $pol = \{ipk_1, \ldots, ipk_n\}$, then augmenting the presentation with a proof that the credential verifies under a hidden but authenticated public key.

As a starting point, we describe in Section 5.1 how the framework could be instantiated with BBS credentials, exploiting their key-randomizability. This intermediate construction $\mathsf{IHBBS}_{\mathtt{Gro}}$ combines BBS for the credential and Groth signatures [Gro15] for the policy signatures. While being the first BBS-based Type-1 construction, $\mathsf{IHBBS}_{\mathtt{Gro}}$ still has shortcomings: it requires expensive computations in $\mathbb{G}_T$ and does not rely on BBS signatures alone. Therefore, we propose a different approach: IHBBS_{τ}. The main observation we exploit here is that a full-fledged signature for authenticating the issuer keys is not necessary. We substitute the Groth signature with a new primitive we call *tag*, introduced in Section 5.2. Tags are malleable, but only for a controlled class of randomizations. This gives us the flexibility to combine the tag-based authentication in a simple way with BBS credentials and their randomizable issuer keys. This yields our IHBBS_{τ} scheme, presented in Section 5.3, where the issuer hiding requires no modification to the standard BBS zero-knowledge proof. We prove the scheme secure in Section 5.4 in the AGM, which improves upon prior compact IHAC schemes that need the GGM. Besides, transforming our IHBBS_{τ} construction into a Type-2 construction yields mild efficiency improvements over [KS25], which we sketch in Appendix E.

5.1 Warmup: $\mathsf{IHBBS}_{\mathtt{Gro}}$ (based on [BEK+21])

Our first contribution is to combine the randomization technique for BBS key-signature pairs and the framework presented in [BEK+21] to obtain the first Type-1 IHAC scheme for BBS signatures, which we call $\mathsf{IHBBS}_{\mathtt{Gro}}$. We sketch the main idea here and refer to Appendix B.3 for details.

For the policy public key pk_{pol}, a fresh Groth [Gro15] key pair $(sk_{\mathsf{G}}, pk_{\mathsf{G}})$ is sampled and every BBS issuer key $\tilde{X}_k \in pol$ is signed as $\sigma_k \leftarrow \mathsf{Sign}_{\mathsf{Groth}}(sk_{\mathsf{G}}, \tilde{X}_k)$. The policy key is then set as $pk_{pol} \leftarrow (pk_{\mathsf{G}}, (\sigma_1, \ldots, \sigma_n))$. For presentations, the holder knows a BBS credential (A, e) by issuer key $\tilde{X}$ and a Groth signature σ on $\tilde{X}$ valid under pk_{G}. The holder first randomizes $\tilde{X}$ as $\tilde{X}^{\star} \leftarrow \tilde{X}^{\alpha}$ and adapts the BBS signature using the map $(\tilde{X}^{\star}, (A^{\star}, e^{\star})) \leftarrow \mathsf{Adapt}(\tilde{X}, (A, e), \alpha)$.

Then they prove knowledge of (1) a signature σ on some $\tilde{X}$ that is hidden in the randomized $\tilde{X}^\star$, and (2) of the adapted BBS credential valid under $\tilde{X}^\star$. The framework of [BEK+21] needs the signature scheme for policy keys to sign group elements (the BBS public key) and to allow for efficient proofs of knowledge, which motivated the choice of Groth signatures.

Shortcomings. Our randomization technique allows to use BBS signatures for the attribute credential instead of Groth (as in [BEK+21]), but we observe multiple limitations of $\mathsf{IHBBS}_{\mathtt{Gro}}$. First, efficiency is far from optimal as both proof generation and verification require computations in $\mathbb{G}_T$. While the proof size is constant, proving knowledge of a Groth signature costs an additional 3 elements in $\mathbb{G}_2$, 1 element in $\mathbb{G}_1$ and 3 scalars. Second, the use of Groth signatures for the policy key requires a second non-standard signature scheme for the overall credential scheme. This adds complexity and delays to any real-world deployment that needs proper standardization, such as the EUDI wallet. Third, the security proof relies on the strong GGM. This is typical for signatures over group elements, e.g., [Gro15,FHS19].

5.2 Avoiding Signatures on Group Elements: Tags

The issues described above seem to mainly stem from the use of Groth signatures, and more generally the need to have a secure signature that signs (1) over group elements and (2) allows for efficient proofs of knowing a signature. We observe that full-fledged signatures with these properties are not needed here. In particular, we do not need fully unforgeable authentication as we only authenticate BBS public keys, and the user must still know a valid BBS signature under that key. Thus, we can offload most of the heavy lifting on the BBS unforgeability, whereas the policy authentication can be malleable to some degree, as long as the adversary extend authentication to some key it controls.

We realize this with a relaxed variant of a signature scheme that we call a *tag*. This is still a public key primitive with secret-key-based authentication and public verification. In terms of security, we ensure that forging a tag for a *specific* message (unrelated to those previously signed) is infeasible, but randomization of the authenticated messages (in our case, BBS public keys) is possible. The latter property would clearly make this relaxed primitive not UF-CMA.

It turns out that the desired *tag* functionality can be realized with a simple algebraic operation. To build some intuition, we introduce it as a standalone construction and describe informally the relaxed unforgeability-like guarantees it provides. After this warm-up, we present our Type-1 IHAC construction.

A simple tag construction. We model Tag schemes to consist of a set of four algorithms $(\mathsf{PGen}, \mathsf{KGen}, \mathsf{TagMsg}, \mathsf{VfTag})$ that are syntactically and semantically analogous to the corresponding signature algorithms. Our construction algTag has scalars $x_v \leftarrow\!\!\$\ \mathbb{Z}_p^*$ as secret keys and message space $\mathcal{M} = \mathbb{G}_2 \setminus \{\mathbf{1}_{\mathbb{G}_2}\}$. For tagging, it simply exponentiates messages with the secret key, and it uses the pairing operation for verification.

Definition 5 (algTag). algTag *is defined as the following set of algorithms:*

$\mathsf{PGen}(1^\lambda)$: *Generate parameters* $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbf{e}, G_1, \tilde{G}_2, \hat{G}_T) \leftarrow\!\!\$\ \mathsf{BGGen}(1^\lambda)$ *and return* $pp \leftarrow (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbf{e})$.

$\mathsf{KGen}(pp)$: *Sample* $x_v \leftarrow\!\!\$\ \mathbb{Z}_p^*$, , *set* $sk \leftarrow x_v$, *then sample* $G_v \leftarrow\!\!\$\ \mathbb{G}_1 \setminus \mathbf{1}_{\mathbb{G}_1}$, *set* $T_v \leftarrow G_v^{1/x_v}$ *and set* $pk \leftarrow (G_v, T_v)$. *Return* (sk, pk).

$\mathsf{TagMsg}(sk, \tilde{M})$: *Parse* $x_v \leftarrow sk$. *Return* $\perp$ *return* $\tilde{T} \leftarrow (\tilde{M}^{1/x_v})$.

$\mathsf{VfTag}(pk, \tilde{T}, \tilde{M})$: *Parse* $(G_v, T_v) \leftarrow pk$. *Return 1 iff* $G_v \neq \mathbf{1}_{\mathbb{G}_1}$, $T_v \neq \mathbf{1}_{\mathbb{G}_1}$, $\tilde{M} \in \mathbb{G}_2 \setminus \{\mathbf{1}_{\mathbb{G}_2}\}$ *and* $\mathbf{e}(T_v, \tilde{M}) = \mathbf{e}(G_v, \tilde{T})$.

Note that starting from a message-tag pair $(\tilde{M}, \tilde{T})$, a user can easily randomize it by choosing $\alpha \leftarrow\!\!\$\ \mathbb{Z}_p^*$ and computing $(\tilde{M}^\alpha, \tilde{T}^\alpha)$. Yet, the resulting message-tag pair will always be in relation to a known message-tag pair. This constitutes the intuition for the security guarantee provided by algTag: without knowing the tag secret key, the adversary is limited to the message-tag pairs granted to it as well as message-tags pairs derived from the granted pairs by some exponent that the adversary knows. Looking ahead, when using algTag to tag issuer keys, the adversary will be unable to learn a tag of an *unrelated* key, in particular, one that it controls.

Remark 1. While tags resemble SPS-EQ on a surface level, this perspective is unhelpful for our construction and security analysis. Tags fulfill the structural properties of SPS in that messages, tags, and public keys are group elements, and that verification checks the equality of products of pairings of group elements. Also, just as SPS-EQ, tags offer a "changing the representation" mechanism: we can shift a tag for one *ipk* to be valid on a different one. But this is also the catch: with *ipk* being a single element, the message space collapses to a single equivalence class, making SPS-EQ unforgeability useless: unforgeability ensures that an adversary cannot sign representatives of a fresh equivalence class–but here, knowing one tag, we have a tag over the only equivalence class, and can obtain tags for any representative.

5.3 IHBBS_τ Scheme

Now we are ready for our main construction: the Type-1 IHAC scheme IHBBS_τ. Again, attribute credentials are plain BBS signatures, so parameters, issuer keys, as well as credential generation and verification are as in the standard BBS scheme. The policy generation then uses our simple algTag scheme from above. It generates a fresh tag key x_v with public key (G_v, T_v) and authenticates all BBS public keys through tags $\tilde{T}_1, \ldots, \tilde{T}_n$. Presentations are very simple and avoid any ZKPs besides the inevitable proof of knowledge of a credential: the holder (1) randomizes their issuer public key and its tag accordingly, (2) adapts the BBS credential, and (3) outputs the randomized public-key/tag pair together with a BBS presentation for the randomized public key.

Definition 6 (IHBBS_τ). *Let* $\mathsf{BBS} = (\mathsf{PGen}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ *be as in Definition 1 and the tag scheme* $\mathsf{algTag} = (\mathsf{PGen}, \mathsf{KGen}, \mathsf{TagMsg}, \mathsf{VfTag})$ *be as in Definition 5. Then,* IHBBS_τ *is defined as the following set of algorithms:*

$\mathsf{PGen}(1^\lambda)$: *Run* $pp_{\mathsf{BBS}} \leftarrow (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbf{e}, G_1, \tilde{G}_2, H_1, ..., H_\ell) \leftarrow\!\$\ \mathsf{PGen}_{\mathsf{BBS}}(1^\lambda)$, *set* $pp_{\mathsf{algTag}} \leftarrow (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbf{e})$ *and return* $pp \leftarrow (pp_{\mathsf{BBS}}, pp_{\mathsf{algTag}})$.

$\mathsf{IssKGen}(pp)$: *Run* $(isk, ipk) \leftarrow\!\$\ \mathsf{KGen}_{\mathsf{BBS}}(pp)$ *and return* (isk, ipk) *if* $isk \neq 0$ *(which happens with overwhelming probability), otherwise return* $\perp$.

$\mathsf{Issue}(isk, \mathbf{a})$: *Return* $cred = (A, e) \leftarrow\!\$\ \mathsf{Sign}_{\mathsf{BBS}}(isk, \mathbf{a})$.

$\mathsf{VfCred}(ipk, cred, \mathbf{a})$: *Return* $\mathsf{Vf}_{\mathsf{BBS}}(ipk, (A, e), \mathbf{a})$.

$\mathsf{SetPolicy}(pol)$: *Generate* $(x_v, (G_v, T_v)) \leftarrow\!\$\ \mathsf{KGen}_{\mathsf{algTag}}(pp_{\mathsf{algTag}})$. *For each issuer public key* $\tilde{X}_k \in pol$, *compute a tag* $\tilde{T}_k \leftarrow \mathsf{TagMsg}(x_v, \tilde{X}_k)$. *Return* $pk_{pol} \leftarrow ((G_v, T_v), \tilde{T}_1, \ldots, \tilde{T}_n)$.

$\mathsf{VfPolicy}(pol, pk_{pol})$: *Parse* $((G_v, T_v), \tilde{T}_1, \ldots, \tilde{T}_n) \leftarrow pk_{pol}$. *Return* 1 *if and only if* $|pol| = n$ *and* $\forall \tilde{X}_k \in pol : \mathsf{VfTag}((G_v, T_v), \tilde{T}_k, \tilde{X}_k) = 1$.

$\mathsf{Present}(pol, pk_{pol}, ipk, cred, \mathbf{a}, \mathcal{D}, nonce)$: *Let* $\tilde{X} \leftarrow ipk$ *and* $(A, e) \leftarrow cred$. *Ensure* $\mathsf{VfPolicy}(pol, pk_{pol})$ *and* $\tilde{X} \in pol$. *Retrieve from* pk_{pol} *the tag* $\tilde{T}$ *for key* $\tilde{X}$. *Sample* $\alpha \leftarrow\!\$\ \mathbb{Z}_p^*$, *randomize* $(\tilde{X}^\star, (A^\star, e^\star)) \leftarrow \mathsf{Adapt}(\tilde{X}, (A, e), \alpha)$[4] *and* $\tilde{T}^\star \leftarrow \tilde{T}_i^\alpha$ *accordingly.*

Prove knowledge of a credential under the randomized key $\tilde{X}^\star$*:*
Sample $r \leftarrow\!\$\ \mathbb{Z}_p^*$, *set* $\overline{A} \leftarrow (A^\star)^r$, *set* $\overline{B} \leftarrow C(\mathbf{a})^r \overline{A}^{-e^\star}$, *and compute* π *as*

$$\mathsf{NIZKP}\Big\{\big((\rho, (\upsilon_j)_{j\in\mathcal{U}}, \epsilon, \delta), (\overline{B}, C(\mathbf{a}'), (H_j)_{j\in\mathcal{U}}, \overline{A})\big) :\ \overline{B} = C(\mathbf{a}')^\rho \prod_{j\in\mathcal{U}} H_j^{\upsilon_j} \overline{A}^\epsilon\Big\}.$$

Finally, return $pres \leftarrow (\tilde{X}^\star, \tilde{T}^\star, \overline{A}, \overline{B}, \pi)$.

$\mathsf{VfPres}(pol, pk_{pol}, pres, \mathcal{D}, (a'_i)_{i\in\mathcal{D}}, nonce)$: *Parse* $((G_v, T_v), \tilde{T}_1, \ldots, \tilde{T}_n) \leftarrow pk_{pol}$ *and* $(\tilde{X}^\star, \tilde{T}^\star, \overline{A}, \overline{B}, \pi) \leftarrow pres$. *Return* 1 *iff the following three checks pass: (1)* $\mathsf{VfPolicy}(pol, pk_{pol})$, *(2)* $\mathsf{VfPres}_{\mathsf{BBS}}(\tilde{X}^\star, (\overline{A}, \overline{B}, \pi), \mathcal{D}, (a'_i)_{i\in\mathcal{D}}, nonce)$, *and (3)* $\mathsf{VfTag}_{\mathsf{algTag}}((G_v, T_v), \tilde{T}^\star, \tilde{X}^\star)$.

Remark 2. Note that x_v is used only in a one-time offline operation (SetPolicy), and can be erased after policy generation, reducing the risk of compromise only to the time frame of generation of pk_{pol}. This is one of the advantage of $Type{-}1$ over $Type - 2$ IHAC, where sk_{pol} is required long-term by every verifier, increasing the attack surface.

5.4 Security Analysis of IHBBS_τ

IHBBS_τ achieves unlinkability and unforgeability. Whereas the former is easily satisfied designing a proof that leverages on the ZK simulator underlying NIZKP of knowledge of BBS credentials, and the randomizability of public key-tag pairs, the latter proof is more involved. Unforgeability stems from BBS unforgeability and the restricted malleability of tags that we outlined. As sketched in Section 5.2, the security of the tag scheme cannot be captured as a stand-alone property, as it is deeply intertwined with the unforgeability of BBS signatures. Thus, here we argue both in a combined security proof, for which we resort to an argument in the AGM.

[4] Recall $\tilde{X}^\star \leftarrow \tilde{X}^\alpha$ and $(A^\star, e^\star) \leftarrow (A^{\frac{1}{\alpha}}, \alpha e)$.

Unlinkability. Presentations created by IHBBS_τ leak nothing about the credential, hidden attributes, or the employed issuer key, since first the key-tag pair is perfectly randomized, such that any key of the policy could be the tagged one, and second we perform a BBS presentation, which is zero-knowledge.

Theorem 1. IHBBS_τ *is* UNLINK *if* H *is a random oracle.*

Proof (sketch). Starting from the unlinkability game with $b = 0$, we replace challenge and response of the presentation by a simulation, relying on the zero-knowledge property of BBS presentations. Next we create the elements $\tilde{X}^\star, \overline{A}, \overline{B}$ and the tag $\tilde{T}^\star$ from the input set (1) instead of (0). This does not change the adversary's view, since $\tilde{X}^\star$ is uniformly random from $\mathbb{G}_2$ in both cases, $\overline{A}$ is uniformly random from $\mathbb{G}_1$, and $\overline{B}$ and the tag $\tilde{T}^\star$ are uniquely determined by the former elements. Finally we revert from simulation to an honest presentation of inputs (1) and end up in the unlinkability game with $b = 1$. We give the full proof in Appendix C.2.

Unforgeability. Unforgeability of IHBBS_τ stems from the SUF-CMA of the underlying BBS credentials and the guarantees that the tags provide.

Since the proof leverages an algebraic property of the tag scheme, we rely on the AGM, but only for operations in $\mathbb{G}_2$. While applying the AGM to both groups might improve tightness, restricting it to $\mathbb{G}_2$ highlights the gap between our proof and a standard model argument, potentially guiding future work toward a standard model reduction or an improved construction.

Theorem 2. IHBBS_τ *is* UNF *under SUF-CMA of BBS signatures in the AGM and ROM.*

Proof (sketch). We define a reduction $\mathcal{B}$ that runs one of two reductions $\mathcal{B}_1$ and $\mathcal{B}_2$ that both reduce to the SUF-CMA of BBS signatures. The two reductions are indistinguishable from the adversary's point of view, as both simulate the unforgeability experiment in Game 2 perfectly. Both reductions rewind the adversary to extract a credential associated to $\mathcal{A}$'s forgery, remaining indistinguishable under $\mathcal{A}$'s view. Then the reductions evaluate a property of the extracted credential and react in different ways but with the shared goal of forging a BBS signature. The evaluated property of the extracted credential is whether it *belongs to the same equivalence class induced by the map* Adapt *of one of the credentials (with associated issuer key) issued to* $\mathcal{A}$ *as a response to queries to* $\mathcal{O}_{\mathsf{Corrupt}}$[5]. We call this event Coll_i for reduction $\mathcal{B}_i$—note that Coll_1 and Coll_2 occur with the same probability since the information provided by the reductions to $\mathcal{A}$ so far have the same distribution.

The reductions $\mathcal{B}_1$ and $\mathcal{B}_2$ cover complementary cases of Coll, with $\mathcal{B}_2$ successfully reducing to SUF-CMA of BBS signatures if Coll_2 occurs and $\mathcal{B}_1$ successfully breaking SUF-CMA of BBS signatures if $\neg\mathsf{Coll}_1$ occurs. Since the interaction of

[5] In this proof sketch we omit an additional signature for the sake of simplicity.

$\mathcal{B}$ with $\mathcal{A}$ hides which reduction is actually running, informally, for any adversary $\mathcal{A}$, $\mathcal{B}$ will run the "right reduction" with probability 1/2.

We sketch both reductions to the SUF-CMA security of BBS signatures here and provide the full proof in Appendix C.3.

Reduction $\mathcal{B}_1$. We split the description of the attacker into setup, training, and forgery phase.

Setup phase. The reduction $\mathcal{B}_1$ to the SUF-CMA of BBS signatures forwards the public parameters pp_{BBS} received from challenger $\mathcal{C}$ and stores the public key $\tilde{X}$ it has received. Then, $\mathcal{B}_1$ computes the "leakage" associated to the public key $(\tilde{G}_2, \tilde{X})$, which is a pair of elements $(U_1, U_2) \in \mathbb{G}_1^2$ s.t. $\mathbf{e}(U_1, \tilde{X}) = \mathbf{e}(U_2, \tilde{G}_2)$. $\mathcal{B}_1$ can compute this leakage from any signature it queries from the BBS strong unforgeability game.[6]

Training phase. We simulate $\mathcal{A}$'s oracles as follows. When $\mathcal{A}$ asks for a new issuer public key, the reduction samples $\alpha_i \leftarrow\!\$\ \mathbb{Z}_p^*$ and returns $\tilde{X}_i \leftarrow \tilde{X}^{\alpha_i}$. When the adversary corrupts a credential from $\tilde{X}_i$ for $\mathbf{a}$, $\mathcal{B}_1$ sends a signing query to $\mathcal{C}$ for the same attributes $\mathbf{a}$: it receives (A, e) and sends to $\mathcal{A}$ the adapted signature $(\tilde{X}_i, (A', e')) \leftarrow \mathsf{Adapt}(\tilde{X}, (A, e), \alpha_i)$. The signature is well distributed as e was sampled randomly by $\mathcal{C}$.

$\mathcal{B}_1$ answers to the i-th $\mathcal{O}_{\mathsf{SetPolicy}}$ query by sampling $\beta_i, \gamma_i \leftarrow\!\$\ \mathbb{Z}_p^*$ and setting $(G_{v,i}, T_{v,i}) \leftarrow (U_2^{\gamma_i \beta_i}, U_1^{\gamma_i})$, a tag public key associated to the secret key $x\beta_i$. $\mathcal{B}_1$ does not know x, but since the issuer keys are generated randomizing $\tilde{X}$, it can compute the associated tags setting, for every $\tilde{X}_j \in pol_i$, $\tilde{T}_j \leftarrow \tilde{G}_2^{\frac{\alpha_i}{\beta_i}}$. This enables $\mathcal{B}_1$ to generate a policy public key which is distributed as an honestly generated one. When the adversary queries for a presentation, $\mathcal{B}_1$ simulates a presentation randomizing a key-tag pair included in the queried pk_{pol}, and simulating a proof of knowledge of a BBS credential under the same randomized key (running the HVZK simulator of BBS presentations and programming the random oracle). This last operation might overwrite the hash table Q_{RO} that $\mathcal{B}_1$ programs to answer random oracle queries, but this happens with negligible probability. Also presentations generated in this way have the right distribution.

Forgery phase. Eventually, the adversary produces a forgery attempt

$$(pol^\dagger, pk_{pol}^\dagger, \mathcal{D}^\dagger, (a_i^\dagger)_{i \in \mathcal{D}^\dagger}, nonce^\dagger, (\tilde{X}^\dagger, \tilde{T}^\dagger, \pi^\dagger))$$

$\mathcal{B}_1$ rewinds $\mathcal{A}$ (as in [FLL25]) to extract a credential associated to the public key $\tilde{X}^\dagger$, and aborts if it does not extract a BBS credential $(A^\dagger, e^\dagger)$ for $\mathbf{a}^\dagger$ valid under $\tilde{X}^\dagger$.[7]

[6] (U_1, U_2) will be useful to generate the tag public key of a special policy. This signature on randomly chosen attributes we obtain from the SUF-CMA game is very unlikely to collide with the forgery that $\mathcal{B}_1$ extracts.

[7] Note that the rewinding step, with the associated tightness loss can be omitted if the proof of knowledge of the BBS credential is generated using online-extractable transforms like [Fis05,Unr15].

We call Coll_1 the event that $\tilde{X}^\dagger, (A^\dagger, e^\dagger)$ live in the same equivalence class of any other signature, together with the associated issuer key, issued to $\mathcal{A}$ during the experiment execution, or in the class of the random signature $\tilde{X}, (A^*, e^*)$ generated by $\mathcal{C}$ during the setup phase. If Coll_1 happens, $\mathcal{B}_1$ aborts. Here we stress that Coll_1 is an event triggered by a property of the credential extracted from $\mathcal{A}$, based on the signatures it has received during the training phase, which are distributed as honestly generated signatures.

Since $\mathcal{A}$ is algebraic, it provides $\mathcal{B}_1$ with a representation of $(\tilde{X}^\dagger, \tilde{T}^\dagger)$ with respect to the elements in $\mathbb{G}_2$ previously seen, and since $\mathcal{B}_1$ knows the representation of the elements in $\mathbb{G}_2$ it sends to $\mathcal{A}$ with respect to $\tilde{G}_2, \tilde{X}$, this boils down to $\mathcal{B}_1$ learning (1) $\tilde{X}^\dagger = \tilde{X}^a \tilde{G}_2^b$ and (2) $\tilde{T}^\dagger = \tilde{X}^c \tilde{G}_2^d$. Let $\beta^\dagger$ be the value s.t. the tag secret key of the forged policy is $\beta^\dagger x$. Then $\mathcal{B}_1$ also knows that for an unknown x it holds $G_v^\dagger = (T_v^\dagger)^{x\beta^\dagger}$. Since the randomized key and associated tag are valid for $pk_{pol}^\dagger$, we have (3) $\tilde{X}^\dagger = (\tilde{T}^\dagger)^{x\beta^\dagger}$. Relations (1), (2), (3) can be combined with (4) $\tilde{X} = \tilde{G}_2^x$, to construct the equation (see the full explanation in Appendix C.3 and D)

$$c\beta^\dagger x^2 + (d\beta^\dagger - a)x - b = 0 \mod p \tag{1}$$

where x is to be determined. This will be the crucial step that enables $\mathcal{B}_1$ to generate a BBS forgery, and that raises the need the AGM on $\mathbb{G}_2$.

In the extended analysis in Appendix C.3, we show that, when the equation is not undetermined, then the reduction can solve Equation (1) finding also a solution to $\log_{\tilde{G}_2} \tilde{X}$, revealing the BBS secret key. When Equation (1) is undetermined (i.e., all the coefficients are 0), the representations $\mathcal{B}_1$ obtains are $\tilde{X}^\dagger = \tilde{X}^a$ and $\tilde{T}^\dagger = \tilde{G}_2^{\frac{a}{\beta^\dagger}}$, so $\mathcal{B}_1$ can apply the update map to the extracted credential: $(\tilde{X}, (A, e)) \leftarrow \mathsf{Adapt}(\tilde{X}^\dagger, (A^\dagger, e^\dagger), \frac{1}{a})$. The latter is a forgery, and $\mathcal{B}_1$ breaks SUF-CMA of BBS signatures.

Now we design another reduction, again to the SUF-CMA of BBS signatures, that succeeds if Coll_2 (which is the analogous of Coll_1 that can happen to $\mathcal{B}_2$) happens, and aborts otherwise.

Reduction $\mathcal{B}_2$. The reduction $\mathcal{B}_2$ is defined as $\mathcal{B}_1$ for the setup phase.
Training phase. After receiving the i-th query to $\mathcal{O}_{\mathsf{KGen}}$, $\mathcal{B}_2$ samples $u_i \leftarrow\!\!\$\ \{0,1\}$, $\mathcal{B}_2$ samples $\alpha_i \leftarrow\!\!\$\ \mathbb{Z}_p^*$ and if $u_i = 0$, returns $\tilde{X}_i \leftarrow \tilde{G}^{\alpha_i}$, otherwise $\tilde{X}_i \leftarrow \tilde{X}^{\alpha_i}$. The public keys have the right distribution and they are now divided in two groups: those for which $\mathcal{B}_2$ knows the corresponding secret key, and the randomizations of $\tilde{X}$. $\mathcal{B}_2$ answers to queries to $\mathcal{O}_{\mathsf{Corrupt}}$ honestly if the public key of the queried issuer $\tilde{X}_i$ is associated to $u_i = 0$, otherwise it answers as $\mathcal{B}_1$ does if $u_i = 1$, querying the SUF-CMA challenger for a signature, then adapting it to the right public key. $\mathcal{B}_2$ answers queries to $\mathcal{O}_{\mathsf{SetPolicy}}$ honestly generating a tag key pair and tagging the public keys in the policy. Presentations are simulated as $\mathcal{B}_1$ does.
Forgery phase. Eventually, the adversary produces a forgery attempt

$$(pol^\dagger, pk_{pol}^\dagger, \mathcal{D}^\dagger, (a_i^\dagger)_{i\in\mathcal{D}^\dagger}, nonce^\dagger, (\tilde{X}^\dagger, \tilde{T}^\dagger, \pi^\dagger)).$$

$\mathcal{B}_2$ aborts if it cannot extract a credential for $\tilde{X}^\dagger$ after rewinding $\mathcal{A}$, as $\mathcal{B}_1$ does.

As for $\mathcal{B}_1$, we define Coll_2 as the event that the extracted credential lives in the same class as one of the credentials issued to $\mathcal{A}$ or in the class of the random signature generated by $\mathcal{C}$ during the setup phase. If $\neg\mathsf{Coll}_2$ happens, $\mathcal{B}_2$ aborts.

Let (A, e) be such a signature issued by $\tilde{X}_j$ for the same extracted attributes $\mathbf{a}^\dagger$ such that $((A^\dagger, e^\dagger), \tilde{X}^\dagger) \in [(A, e), \tilde{X}_j]$. Then we define $\epsilon \leftarrow \frac{e}{e^\dagger}$, and observe that $\tilde{X}_j = (\tilde{X}^\dagger)^\epsilon$. Also, since the adversary is algebraic, it provides $\mathcal{B}_2$ with a representation of $\tilde{X}^\star$ with respect to the elements in $\mathbb{G}_2$ it has previously seen. As for $\mathcal{B}_1$, also $\mathcal{B}_2$ knows the representation of every element in $\mathbb{G}_2$ given to $\mathcal{A}$ w.r.t. $\tilde{X}$ and $\tilde{G}_2$: $\tilde{X}^\dagger = \tilde{X}^a \tilde{G}_2^b$. This means that $\mathcal{B}_2$ can compute a representation of $\tilde{X}_j = \tilde{X}^{\epsilon a} \tilde{G}_2^{\epsilon b}$.

Now we observe that whenever $b = 0$ and also the bit $u_j = 0$ (sampled while $\mathcal{B}_2$ was generating $\tilde{X}_j$) $\mathcal{B}_2$ knows $\tilde{G}_2^{\alpha_j} = \tilde{X}^{\epsilon a}$, i.e. $x = \frac{\alpha_j}{\epsilon a}$, then $\mathcal{B}_2$ can trivially win the SUF-CMA experiment for BBS signature against $\mathcal{C}$ as it has extracted the secret counterpart of $\tilde{X}$.

Similarly, whenever $b \neq 0$ and the bit $u_j = 1$, (sampled while $\mathcal{B}_2$ was generating $\tilde{X}_j$), $\mathcal{B}_2$ knows $\tilde{X}_2^{\alpha_j} = \tilde{X}^{\epsilon a} \tilde{G}_2^{\epsilon b}$, i.e. $(\alpha_j - \epsilon a)x = \epsilon b \neq 0$, i.e., $(\alpha_j - \epsilon a) \neq 0$ and most importantly $x = \frac{\epsilon b}{\alpha_j - \epsilon a}$. Then $\mathcal{B}_2$ can trivially win the SUF-CMA experiment for BBS signature against $\mathcal{C}$ as it has extracted the secret counterpart of $\tilde{X}$. We include in this case of the analysis also the event that the credential that collides is (A^*, e^*) issued by $\tilde{X}$ at the beginning of the experiment to generate the leakage for $\tilde{X}$ (i.e. $\tilde{X}_j = \tilde{X}$, and $\alpha_j = 1$).

The last crucial observation is that the adversary is oblivious of whether $u_j = 0$ or $u_j = 1$, and this allows us to argue that $\mathcal{B}_2$, when Coll_2 occurs, will be able to extract the secret key of the SUF-CMA experiment half of the times.

6 Performance Comparison

In this section we evaluate the efficiency of IHBBS_τ.

We provide a comparison in four parts. First, we examine the compact (even if not constant-size) Type-0 constructions presented in [FFK+26] based on Stacking Sigmas ($\mathsf{S\Sigma}$) [GGHAK22] and Groth–Kohlweiss (GK) [GK15]. Since a major contribution of this work is optimization, it rely on the numbers reported by the authors. Second, we compare IHBBS_τ to the Type-1 construction $\mathsf{IHBBS}_{\mathtt{Gro}}$, and third, to the concurrent work by Kaaniche et al. [KKLT26]. Here, our comparison is on the construction-level: we confront the number of operations performed during a presentation and the number of elements that make up presentations, policy keys, and policies. Finally, as a footnote, we remark that transforming IHBBS_τ into a Type-2 construction yields a minor improvement over the recent BBS-based construction by [KS25].

In all cases, we focus on the computational complexity for the user, since it is the most resource-restricted party. We evaluate the schemes in terms of communication and computational complexity *on top of* performing a plain BBS presentation, which is included in each approach.

Comparing IHBBS_τ *to* $\mathsf{S\Sigma}$ *and* GK *in [FFK+26].* The proof size overhead of the issuer-hiding constructions $\mathsf{S\Sigma}$ and GK is, according to [FFK+26], $96 + 64\log(n)$ and $32 + 288\log(n)$ bytes, respectively, where n is the number of issuers in a policy. This means that, even for large policies with around 2500 issuers (e.g., the universities in the EU), the proof size overhead is under 1KB for $\mathsf{S\Sigma}$ and under 4KB for GK, which are practical values, even if exceeding our 0.19KB in proof size overhead of IHBBS_τ due to the addition of randomized public key and the associated tag.

The more significant burden for the Type-0 constructions $\mathsf{S\Sigma}$ and GK is the computational overhead for the prover, because as the size of policies grows, exceeding hundred or thousands of issuers, the computational overhead grows close to a full second, based on the experimental evaluation in [FFK+26]. In contrast, in our construction the computational overhead is constant in the size of the policy and only requires computing two exponentiations in $\mathbb{G}_2$ and one in $\mathbb{G}_1$.

Comparing IHBBS_τ *to* $\mathsf{IHBBS}_{\mathtt{Gro}}$ *and [KKLT26].* Results for $\mathsf{IHBBS}_{\mathtt{Gro}}$, IHBBS_τ, and the construction by [KKLT26] are given in Table 2. The computational complexity estimation is based on the number of exponentiations ($\mathtt{e}_{\mathbb{G}}$ is an exponentiation in group $\mathbb{G}$) and pairing ($\mathtt{p}$) operations that the algorithms require. The concrete numbers related to the computational and communication overheads (that do not depend on the size of the policy, or the attributes in the credential) have been collected using https://zka.lc/, for curve BLS12-381, implemented in arkworks (https://github.com/arkworks-rs) v0.4.2 using a AWS EC2 m5.2xlarge machine. When implementing BBS over the BLS12-381 curve, scalars in $\mathbb{Z}_p$ are 32 bytes large, group elements in $\mathbb{G}_1$ are 48 bytes large, while elements in $\mathbb{G}_2$ are 96 bytes large.

Here, we save 4 exponentiations in $\mathbb{G}_1$, 2 exponentiations in $\mathbb{G}_2$, and 2 pairings, resulting in an almost 75% performance improvement over the overhead of $\mathsf{IHBBS}_{\mathtt{Gro}}$. The overall proof size overhead is also more than halved, as IHBBS_τ proofs only consist of 2 $\mathbb{G}_2$ elements. We also note that size of the policy key pk_{pol} is only half as large in IHBBS_τ: for each verifier, there is only 1 $\mathbb{G}_2$ element added, as opposed to 1 $\mathbb{G}_1$ and 2 $\mathbb{G}_2$ elements in $\mathsf{IHBBS}_{\mathtt{Gro}}$. Besides, IHBBS_τ is proven secure in the AGM and does not rely on the GGM.

For Kaaniche et al. [KKLT26], we focus on the pairing-based construction, since their pairing-free version assumes a setting where the user must interact with the issuer for every presentation, and is thus less comparable. The authors do not provide an evaluation of their pairing-based scheme, so we made the following set of assumption for the numbers shown in Table 2. To begin with, the work leaves unspecified how to instantiate the NIZKPs used for presentations, so we make a standard choice of a Schnorr-type proof for linear relations and [CS03, Section 6] for the proof of inequality of discrete logarithms. Furthermore, the construction requires for each BBS public key $\tilde{X}_i = \tilde{G}_2^{x_i}$ that the $\mathbb{G}_1$ analogue $G_1^{x_i}$ is also provided, together with a NIZKP of equality of discrete logarithm between $(\tilde{G}_2, \tilde{X}_i)$ and (G_1, X_i); the latter adding two $\mathbb{Z}_p$ elements. Although not explicitly stated, it seems necessary to include these additional artifacts in the

policy, which triggers a blowup in size. Leaving out elements already included in *pol*, the public policy key consists only of one BBS signature per issuer public key and one additional BBS public key ($\in \mathbb{G}_2$) to verify these signatures.

In total, the policy keys in [KKLT26] are slightly smaller than ours, but at the cost of more than doubling the policy size. Overall this makes communication less efficient, since the user has to fetch both policy and policy key in order to validate policy public keys and create a presentation.

Table 2. Performance comparison between Type-1 IHAC schemes built on top of BBS credentials, namely our intermediate construction $\mathsf{IHBBS}_{\mathtt{Gro}}$, our main construction IHBBS_{τ}, and the pairing-based construction described in the concurrent work [KKLT26]. We specify also the idealized model (in addition to the ROM) under which the scheme is proven secure, the policy key size and the policy size. Since policy and policy key are always used together, we omit from pk_{pol} any elements that are already included in *pol*.

Scheme	Comput. OH	Commun. OH	$\lvert pk_{pol}\rvert$	$\lvert pol\rvert$
[KKLT26]	$\mathsf{e}_{\mathbb{G}_2} + 12\mathsf{e}_{\mathbb{G}_1}$	$\mathbb{G}_2 + 4\mathbb{G}_1 + 5\mathbb{Z}_p$	$\mathbb{G}_2 + n(\mathbb{G}_1 + \mathbb{Z}_p)$	$n(\mathbb{G}_2 + \mathbb{G}_1 + 2\mathbb{Z}_p)$
concrete numbers	3.6 ms	448 B	96+80n B	208n B
$\mathsf{IHBBS}_{\mathtt{Gro}}$	$2\mathsf{p} + 4\mathsf{e}_{\mathbb{G}_2} + 5\mathsf{e}_{\mathbb{G}_1}$	$3\mathbb{G}_2 + \mathbb{G}_1 + 3\mathbb{Z}_p$	$n(2\mathbb{G}_2 + \mathbb{G}_1)$	$n\mathbb{G}_2$
concrete numbers	8.7 ms	432 B	240n B	96n B
IHBBS_{τ}	$2\mathsf{e}_{\mathbb{G}_2} + \mathsf{e}_{\mathbb{G}_1}$	$2\mathbb{G}_2$	$n\mathbb{G}_2 + 2\mathbb{G}_1$	$n\mathbb{G}_2$
concrete numbers	2.2 ms	192 B	96($n+1$) B	96n B

Comparing a Type-2 transform of IHBBS_{τ} *to [KS25].* We remark that any Type-1 scheme can be used also as a Type-2 scheme if we let the verifier generate the policy public key. This conjecture can remove the need for a trusted party that generates policies. Therefore, in Appendix E we compare a conjectured scheme $\mathsf{IHBBS}_{\tau 2}$, obtained by turning IHBBS_{τ} into a Type-2 scheme and applying some straightforward optimizations, to the scheme of Katz and Sefranek [KS25]. The prover's computational complexity is essentially the same, as exponentiations in $\mathbb{G}_2$ are roughly 2-3 times slower than in $\mathbb{G}_1$, whereas we slightly improve the communication complexity by one element in $\mathbb{G}_2$ and one scalar.

7 IHAC Framework & Impact of Policy Types

In Section 3 we defined IHAC for the Type-1 setting that is the focus of this work. To put our construction into context, we now broaden the picture and describe how the other two types of IHAC schemes used in the literature fit our framework: Type-0, which uses no policy key at all, and Type-2, which uses a verifier-specific policy-key pair. We give the syntax of both types as light modifications of the Type-1 syntax (Definition 2), sketch the resulting changes to the security model (with the formal games deferred to Appendix A), and close with a discussion of how the choice of policy type affects practical privacy and efficiency.

7.1 Syntax of Type-0 and Type-2 IHAC

Type-0 (no policy key). The simplest constructions take the policy pol, i.e., the set of accepted issuers, as the sole authentication target and use no additional policy key. This is the *Type-0* class: there is no policy-key generation step, no pk_{pol} to verify, and Present and VfPres depend on pol only. Schemes in this class include the recent BBS-based constructions of Flamini et al. [FFK+26], which apply optimized OR-proof techniques to the policy.

Syntactically, a Type-0 IHAC scheme is obtained from Definition 2 by removing the policy-key apparatus altogether: it consists of (PGen, IssKGen, Issue, VfCred, Present, VfPres), where

- SetPolicy and VfPolicy are absent, and
- Present and VfPres take no pk_{pol} for input

All other algorithms are inherited unchanged from the Type-1 syntax.

Type-2 (verifier policy key). Type-2 IHAC extends Type-1 by adding a *secret counterpart* sk_{pol} to the policy public key pk_{pol}, which is required to verify presentations. Since the secrecy of sk_{pol} is essential for unforgeability, it cannot be shared across verifiers: every verifier must generate its own policy-key pair, and credential holders must obtain the policy public key from every verifier individually—even for presentations under the same policy. This type was introduced by Sanders and Traoré [ST24], and recently used by Katz and Sefranek [KS25] in their BBS-based scheme.

Syntactically, a Type-2 IHAC scheme is obtained from Definition 2 by upgrading SetPolicy to output a key pair instead of a single public key, and by passing the secret part to VfPres:

- $\mathsf{SetPolicy}(pol) \xrightarrow{\$} (sk_{pol}, pk_{pol})$: run by the verifier, generates a policy-key pair for pol.
- $\mathsf{VfPres}(pol, pk_{pol}, sk_{pol}, pres, \mathcal{D}, (a'_i)_{i\in\mathcal{D}}, nonce) \to \{0,1\}$: the verifier additionally uses sk_{pol} to check $pres$.

All other algorithms remain as for Type-1. Following [ST24,KS25], VfPolicy can still be defined as a public-coin sanity check on pk_{pol} relative to pol.

Relation between the types. The types form a natural hierarchy. Type-0 schemes can be cast as Type-1 by letting SetPolicy output an empty pk_{pol}, VfPolicy always return 1, and Present and VfPres ignore pk_{pol}. Analogously, Type-1 schemes can be cast as Type-2 by letting SetPolicy output an empty sk_{pol} and VfPres ignore sk_{pol}. All security properties introduced in Section 3.2 are preserved under these transformations.

7.2 Security Model Across Types

The unlinkability and unforgeability games defined in Section 3.2 for Type-1 IHAC adapt to the other two types in a straightforward way. We describe the necessary changes informally here and give the full definitions in Appendix A.

Type-0. In the unlinkability game (Game 1), the challenge oracle $\mathcal{O}_{\mathsf{Chal}}$ no longer takes pk_{pol} as input and the VfPolicy check is dropped; the call to Present likewise drops pk_{pol}. In the unforgeability game (Game 2), the policy registration oracle $\mathcal{O}_{\mathsf{SetPolicy}}$ retains its role of restricting the adversary to policies built from honest issuers—this is needed independently of the existence of a policy key, since the unforgeability winning condition still has to refer to a registered policy—but it no longer runs SetPolicy and the bookkeeping set Q_{pols} stores policies only. All occurrences of pk_{pol} in the oracle inputs, in $\mathcal{O}_{\mathsf{Present}}$, and in the forgery output are removed; the VfPolicy check in $\mathcal{O}_{\mathsf{Present}}$ is dropped. The winning conditions are otherwise unchanged.

Type-2. Two changes are needed. First, SetPolicy now produces a key pair (sk_{pol}, pk_{pol}), of which only pk_{pol} is exposed to the adversary; the experiment retains sk_{pol} to use in verification. Second, because VfPres requires the secret sk_{pol}, the adversary cannot verify presentations on its own (at least when the policy keys were generated by the challenger), so the unforgeability experiment provides a *presentation verification oracle* $\mathcal{O}_{\mathsf{Vf}}(pol, pk_{pol}, pres, \mathcal{D}, (a'_i)_{i\in\mathcal{D}}, nonce)$. This oracle runs VfPres on the adversarially provided inputs and a sk_{pol} recorded in Q_{pols}. The unlinkability game requires no further changes beyond the syntactic upgrade of $\mathcal{O}_{\mathsf{Chal}}$ to receive (pol, pk_{pol}) generated by SetPolicy rather than chosen freely by the adversary—since the adversary now plays the role of the verifier and knows sk_{pol}, the experiment must ensure that pk_{pol} is well-formed. These adaptations align with the unforgeability and unlinkability notions of [ST24,KS25].

7.3 Impact of IHAC Type on Privacy

The three types of IHAC differ not only syntactically, but also in the practical privacy guarantees they offer. In this subsection we argue that Type-0 and Type-1 are strictly preferable to Type-2 from a privacy standpoint, and carve out the exact assumptions that any deployment of IHAC must rely on to achieve meaningful unlinkability.

Unlinkability is the principal privacy property of AC: a presentation should disclose the minimum amount of information necessary for authentication, and, in particular, the holder should stay anonymous among all presentations with the same disclosures. For this to be meaningful, the corresponding *anonymity set* must be large. A malicious verifier might attempt to de-anonymize users by shrinking the anonymity set: for instance, by forcing a holder to present a uniquely identifying attribute value. Issuer-hiding credentials intuitively create *larger* anonymity sets than standard ACs. However, extra care is needed—especially for Type-2 IHAC—to ensure that policies and policy keys cannot themselves be used to reduce the anonymity set.

Anonymity sets. Anonymity sets are determined by the values revealed in a presentation. For standard ACs, this is the nonce, the disclosed attribute indices $\mathcal{D}$, the disclosed attributes $(a_i)_{i\in\mathcal{D}}$, and the issuer public key ipk. Nonces are ephemeral (one per presentation) and thus do not permit linking, so the

remaining three values define the anonymity set: a presentation is anonymous among all holders that have a credential from the same issuer attesting the same disclosed attributes. IHAC presentations replace the issuer key ipk by a policy $pol = \{ipk_1, \ldots, ipk_n\}$ and, for Type-1 and Type-2, additionally by the corresponding policy public key pk_{pol}. The anonymity set therefore depends on how many other holders make presentations under *the same* policy and policy key.

1. Policy must not be unique. The first observation is that the policy must not create a unique trace: it must be shared widely among users. If holders use *distinct* policies, they become trivially traceable across presentations, regardless of the disclosed attributes. This can happen accidentally—e.g., when a policy is assembled on the user side and inadvertently omits or includes one issuer—or maliciously: a verifier or policy provider could "tag" a holder by handing them a slightly tampered policy, e.g., a pol^*_{EU} in which the public key for Liechtenstein is replaced by a unique tracing key. The holder can still produce valid presentations under pol^*_{EU} (unless they actually hold a Liechtenstein credential), but every such presentation is then linkable.

This is an inherent challenge in all three types of IHAC. A simple mitigation is a trusted repository that publishes standardized policies, e.g., pol_{EU} containing the current public keys of all EU member states, or pol_{UniPL} containing all universities in Poland; holders regularly fetch the most recent policies from there. Verifiers refer to such policies via small identifiers, and the disclosed policy then no longer reveals information about the holder beyond standardized, widely shared sets.

2. Policy key *must not be unique.* Whenever a policy key pk_{pol} enters the presentation, the previous observation extends to pk_{pol}: it is not enough for holders to share the same policy, they must also share the same pk_{pol}. For Type-1 IHAC this is easily handled by the same trusted-repository solution: a policy and its single, publicly verifiable pk_{pol} are published together, and all holders use the canonical pair. Verifiers can pre-fetch pk_{pol} once and re-use it.

The picture changes substantially for Type-2 IHAC, where policy keys are by design verifier-specific. A central repository can no longer publish a single pk_{pol} per policy: every verifier must register and have its own pk_{pol} stored. Beyond the sheer scale (the repository must hold a key pair per verifier-pol combination), this also increases the trust placed in the third party, which now sees which verifiers a holder authenticates to. Worse, a verifier could deliberately use a slightly unique pk_{pol} at every interaction to recognize returning holders, and the verifier-specificity of pk_{pol} makes such anomalies hard to detect.

Alternative for Type-2 IHAC? An attempt to limit the privacy risks of Type-2 is to go to the opposite extreme: holders are provided with policy and pk_{pol} from the verifier on the fly and never reuse stored or pre-fetched values. This prevents tracking through repeated use of the same tailored pk_{pol}, but at a high cost. First, freshly delivered policies are easy to tweak—e.g., dropping some issuer to single out a specific population—and the holder has no shared baseline to compare

against. Second, every pk_{pol} must be checked for well-formedness against pol at presentation time, eliminating the efficiency gains of validated, pre-fetched values. Third, communication cost grows: pk_{pol} in Type-2 schemes [ST24,KS25] is linear in the policy size, so transmitting it for every presentation reintroduces the very overhead that constant-size IHAC aims to remove.

Overall, given these privacy and efficiency drawbacks of Type-2 IHAC, the Type-0 and Type-1 classes appear strictly preferable for real-world deployment—which is precisely why we have focused on a Type-1 construction in this work.

Acknowledgements

This work was partially funded by SPRIND, the Federal Agency for Breakthrough Innovation, and Common Codes. The authors are solely responsible for the content of this publication; the positions presented here do not reflect the views of SPRIND and Common Codes.

References

BEK+21. Jan Bobolz, Fabian Eidens, Stephan Krenn, Sebastian Ramacher, and Kai Samelin. Issuer-hiding attribute-based credentials. In Mauro Conti, Marc Stevens, and Stephan Krenn, editors, *CANS 21*, volume 13099 of *LNCS*, pages 158–178. Springer, Cham, December 2021.

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

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

CDS94. Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 174–187. Springer, Berlin, Heidelberg, August 1994.

CGH+24. Sofía Celi, Scott Griffy, Lucjan Hanzlik, Octavio Perez-Kempner, and Daniel Slamanig. SoK: Signatures with randomizable keys. In Jeremy Clark and Elaine Shi, editors, *FC 2024, Part II*, volume 14745 of *LNCS*, pages 160–187. Springer, Cham, March 2024.

Cha85. David Chaum. Security without identification: transaction systems to make big brother obsolete. *Commun. ACM*, 28(10):1030–1044, October 1985.

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

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

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

CLPK22. Aisling Connolly, Pascal Lafourcade, and Octavio Perez-Kempner. Improved constructions of anonymous credentials from structure-preserving signatures on equivalence classes. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022, Part I*, volume 13177 of *LNCS*, pages 409–438. Springer, Cham, March 2022.

CS03. Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 126–144. Springer, Berlin, Heidelberg, August 2003.

eid24. Regulation (eu) 2024/1183 of the european parliament and of the council of 11 april 2024 amending regulation (eu) no 910/2014 as regards establishing the european digital identity framework. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex:32024R1183, 2024.

eud23. The European Digital Identity Wallet Architecture and Reference Framework. https://digital-strategy.ec.europa.eu/en/library/european-digital-identity-wallet-architecture-and-reference-framework, 2023.

eud26. A.2.3.31 Topic 53 Zero-Knowledge Proofs. From: The European Digital Identity Wallet Architecture and Reference Framework. v2.8.0. https://github.com/eu-digital-identity-wallet/eudi-doc-architecture-and-reference-framework/blob/main/docs/annexes/annex-2/annex-2.02-high-level-requirements-by-topic.md#a2331-topic-53-zero-knowledge-proofs, 2026. [Accessed 11-02-2026].

FFK+26. Andrea Flamini, Karla Friedrichs, Jonathan Katz, Watson Ladd, Anja Lehmann, and Marek Sefranek. Issuer-hiding BBS-based anonymous credentials without policy keys. Cryptology ePrint Archive, Paper 2026/870, 2026.

FFL26. Andrea Flamini, Karla Friedrichs, and Anja Lehmann. Constant-size issuer hiding for bbs credentials via randomizable keys. *Cryptology ePrint Archive*, 2026.

FHS19. Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. *Journal of Cryptology*, 32(2):498–546, April 2019.

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

FKL18. Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Cham, August 2018.

FLL25. Andrea Flamini, Eysa Lee, and Anna Lysyanskaya. Multi-holder anonymous credentials from BBS signatures. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VI*, volume 16005 of *LNCS*, pages 325–357. Springer, Cham, August 2025.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Berlin, Heidelberg, August 1987.

GGHAK22. Aarushi Goel, Matthew Green, Mathias Hall-Andersen, and Gabriel Kaptchuk. Stacking sigmas: A framework to compose Σ-protocols for disjunctions. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 458–487. Springer, Cham, May / June 2022.

GK15. Jens Groth and Markulf Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 253–280. Springer, Berlin, Heidelberg, April 2015.

Gro15. Jens Groth. Efficient fully structure-preserving signatures for large messages. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 239–259. Springer, Berlin, Heidelberg, November / December 2015.

JM24. Joseph Jaeger and Deep Inder Mohan. Generic and algebraic computation models: When AGM proofs transfer to the GGM. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part V*, volume 14924 of *LNCS*, pages 14–45. Springer, Cham, August 2024.

KKLT26. Nesrine Kaaniche, Seyni Kane, Maryline Laurent, and Jacques Traoré. Improved issuer hiding for BBS-based anonymous credentials. Cryptology ePrint Archive, Paper 2026/555, 2026.

KS25. Jonathan Katz and Marek Sefranek. Issuer hiding for bbs-based anonymous credentials. *Cryptology ePrint Archive*, 2025.

LKWL26. Tobias Looker, Vasilis Kalos, Andrew Whitehead, and Mike Lodder. The BBS Signature Scheme. Internet-Draft draft-irtf-cfrg-bbs-signatures-10, Internet Engineering Task Force, January 2026. Work in Progress.

Mau05. Ueli M. Maurer. Abstract models of computation in cryptography (invited paper). In Nigel P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *LNCS*, pages 1–12. Springer, Berlin, Heidelberg, December 2005.

MBG+23. Omid Mir, Balthazar Bauer, Scott Griffy, Anna Lysyanskaya, and Daniel Slamanig. Aggregate signatures with versatile randomization and issuer-hiding multi-authority anonymous credentials. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 30–44. ACM Press, November 2023.

PS16. David Pointcheval and Olivier Sanders. Short randomizable signatures. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 111–126. Springer, Cham, February / March 2016.

SB25. Manu Sporny and Greg Bernstein. Data integrity BBS cryptosuites v1.0. Candidate recommendation, W3C, February 2025. https://www.w3.org/TR/2025/CRD-vc-di-bbs-20250218/.

ST24. Olivier Sanders and Jacques Traoré. Compact issuer-hiding authentication, application to anonymous credential. *PoPETs*, 2024(3):645–658, July 2024.

Game 3: Correctness (Type-0, $\boxed{\text{Type-1}}$)

$\underline{\mathsf{Exp}^{\mathsf{Correctness}}_{\mathsf{IHAC}}(\lambda, n, \ell)}$

$\boxed{pk_{pol} \leftarrow\!\$\ \mathsf{SetPolicy}(pol)}$; $\quad cred \leftarrow\!\$\ \mathsf{Issue}(isk^*, \mathbf{a})$

$pres \leftarrow\!\$\ \mathsf{Present}(pol, \boxed{pk_{pol}}, ipk^*, cred, \mathbf{a}, \mathcal{D}, nonce)$

return $\mathsf{VfCred}(ipk^*, cred, \mathbf{a}) \ \wedge\ \mathsf{Vf}(pol, \boxed{pk_{pol}}, pres, \mathcal{D}, (a'_i)_{i\in\mathcal{D}}, nonce)$

$\boxed{\wedge\ \mathsf{VfPolicy}(pol, pk_{pol}) = 1}$

TS126. Specification for the implementation of Zero-Knowledge Proofs based on multi-message signatures in the EUDI Wallet. v0.3. https://github.com/eu-digital-identity-wallet/eudi-doc-standards-and-technical-specifications/blob/main/docs/technical-specifications/ts14-zkps-from-mms.md, 2026. [Accessed 11-02-2026].

TZ23. Stefano Tessaro and Chenzhi Zhu. Revisiting BBS signatures. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 691–721. Springer, Cham, April 2023.

Unr15. Dominique Unruh. Non-interactive zero-knowledge proofs in the quantum random oracle model. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 755–784. Springer, Berlin, Heidelberg, April 2015.

A Full security framework for Type-0, Type-1, and Type-2 IHAC

In the following, we extend the security model presented for Type-1 in Section 3.2 to Type-0 and Type-2 IHAC.

A.1 Correctness

Correctness of IHAC requires intuitively that honestly generated artifacts verify as expected, i.e., honestly generated policy keys are valid for the underlying policy; honestly issued credentials verify for the creating issuer key; and honestly generated presentations verify for the given policy, policy key, nonce, and any subset of the attributes that are vouched for by the credential.

For readability we give a correctness experiment for Type-0 and Type-1 in Game 3 and separately for Type-2 in Game 4, and define the notion jointly in Definition 7. The three experiments are conceptually the same and merely account for the type-specific artifacts. Type-1 and Type-2 incorporate and additional guarantee that honestly generates policy public key verify for the underlying policy.

Definition 7 (Correctness (All Types)). *We define a (Type-0, Type-1, Type-2)* IHAC *scheme Π to be correct, if for all $\lambda, n, \ell \in \mathbb{N}^+$; $pp \leftarrow\!\$\ \mathsf{PGen}(1^\lambda)$; $(isk_1, ipk_1), \ldots, (isk_n, ipk_n) \leftarrow\!\$\ \mathsf{KGen}(pp)$; $pol \subseteq \{ipk_i\}_{i\in[n]}$ with $|pol| > 0$;*

Game 4: Correctness (Type-2)

$\underline{\mathsf{Exp}^{\mathsf{Correctness}}_{\mathsf{IHAC}}(\lambda, n, \ell)}$
$(sk_{pol}, pk_{pol}) \leftarrow\!\$\ \mathsf{SetPolicy}(pol); \qquad cred \leftarrow\!\$\ \mathsf{Issue}(isk^*, \mathbf{a})$
$pres \leftarrow\!\$\ \mathsf{Present}(pol, pk_{pol}, ipk^*, cred, \mathbf{a}, \mathcal{D}, nonce)$
return $\mathsf{VfCred}(ipk^*, cred, \mathbf{a}) \ \wedge\ \mathsf{Vf}(pol, pk_{pol}, sk_{pol}, pres, \mathcal{D}, (a'_i)_{i\in\mathcal{D}}, nonce) \ \wedge$
$\mathsf{VfPolicy}(pol, pk_{pol}) = 1$

Game 5: Unlinkability (Type-0, Type-1, Type-2)

$\underline{\mathsf{Exp}^{\mathsf{UNLINK},b}_{\mathcal{A},\mathsf{IHAC}}(\lambda)}$
$pp \leftarrow\!\$\ \mathsf{PGen}(1^\lambda)$
$b' \leftarrow\!\$\ \mathcal{A}^{\mathcal{O}_{\mathsf{Chal}}}(pp)$
return b'

$\underline{\mathcal{O}_{\mathsf{Chal}}(pol, \boxed{pk_{pol},}\ \mathcal{D}, nonce, ipk^{(0)}, ipk^{(1)}, cred^{(0)}, cred^{(1)}, \mathbf{a}^{(0)}, \mathbf{a}^{(1)})}$
$\boxed{\textbf{if } \mathsf{VfPolicy}(pol, pk_{pol}) = 0 : \textbf{return } \bot}$
if $(a_i^{(0)})_{i\in\mathcal{D}} \neq (a_i^{(1)})_{i\in\mathcal{D}} : \textbf{return } \bot$
for $d \in \{0,1\}$:
  if $ipk^{(d)} \notin pol \ \vee\ \mathsf{VfCred}(ipk^{(d)}, cred^{(d)}, \mathbf{a}^{(d)}) = 0 : \textbf{return } \bot$
return $\mathsf{Present}(pol, \boxed{pk_{pol},}\ ipk^{(b)}, cred^{(b)}, \mathbf{a}^{(b)}, \mathcal{D}, nonce)$

$(isk^*, ipk^*) \in \{(isk_i, ipk_i)\}_{i\in[n]}$ *with* $ipk^* \in pol; \mathbf{a} \in \mathcal{M}^\ell; \mathcal{D} \subseteq [\ell]; nonce \in \{0,1\}^*$; *the probability of the correctness experiment in Game 3 for Type-0 and* $\boxed{\textit{Type-1}}$ *and in Game 4 for Type-2, respectively, outputting 0 is negligible, i.e., there exists a negligible function* $\mathsf{negl}(\lambda)$ *s.t.*

$$\Pr\Big[\mathsf{Exp}^{\mathsf{Correctness}}_{\Pi}(\lambda, n, \ell) = 0\Big] \leq \mathsf{negl}(\lambda).$$

A.2 Issuer-Hiding Unlinkability

The unlinkability property has little variation between the three types. Type-0 simply removes the policy key pk_{pol} and thus also the validity check of pk_{pol} inside $\mathcal{O}_{\mathsf{Chal}}$. For Type-2, we can employ the unlinkability definition of Type-1 without any changes. We state the games in Game 5 and the definition in the following:

Definition 8 (Unlinkability (All Types)). *A (Type-0, Type-1, Type-2)* IHAC *scheme* Π *is unlinkable, if for all PPT adversaries the following advantage w.r.t. the type-specific game in Game 5 is negligible:*

$$\mathsf{Adv}^{\mathsf{UNLINK}}_{\Pi}(\mathcal{A}, \lambda) := \Big|\Pr\Big[\mathsf{Exp}^{\mathsf{UNLINK},0}_{\mathcal{A},\Pi}(\lambda) = 1\Big] - \Pr\Big[\mathsf{Exp}^{\mathsf{UNLINK},1}_{\mathcal{A},\Pi}(\lambda) = 1\Big]\Big|$$

A.3 Issuer-Hiding Unforgeability

For the unforgeability definition, Type-0 again requires some syntactical changes due to the lack of a public policy key. In $\mathcal{O}_{\mathsf{Present}}$ we omit the sanity check on pk_{pol}. $\mathcal{O}_{\mathsf{SetPolicy}}$ no longer needs to generate anything, it merely records policies containing only honest issuer keys for later use in the winning conditions. The

oracle could be omitted and replaced by a simple check for Type-0; we decided to keep it for easy of comparison between the IHAC types. We highlight the difference to the Type-1 definition in Game 6.

Type-2 also modifies $\mathcal{O}_{\mathsf{SetPolicy}}$, which now needs to generate a secret key. If we reveal this sk_{pol} to the adversary, we do not expect the scheme to stay unforgeable w.r.t. that policy key. Therefore, we never reveal the sk_{pol} for the trusted policies output by $\mathcal{O}_{\mathsf{SetPolicy}}$. But the addition of a secret verification key requires a second change if we want to maintain the same adversarial abilities as for Type-0 and Type-1. We remark that, as for the other IHAC types, the adversary still has the ability to request presentations under untrusted policies at the $\mathcal{O}_{\mathsf{Present}}$ oracle, and this extends to policies that the adversary generated themselves and that they know the secret policy key for. However, since in Type-2 IHAC the sk_{pol} is necessary for verification, $\mathcal{A}$ has trouble checking whether some presentation is valid under some trusted policy key. Implicitly, this ability is still encoded in the game, since we verify a presentation when we check the winning conditions. However, for clarity, we decide to model this ability explicitly as well and add a verification oracle $\mathcal{O}_{\mathsf{Vf}}$ that grants the possibility to verify presentations under trusted policies. Other changes to the game are purely syntactical, and the updated game is shown in Game 7.

Let us give the definition of issuer-hiding unforgeability for our full security framework:

Definition 9 (Unforgeability (All Types)). *A (Type-0, Type-1, Type-2)* IHAC *scheme Π is unforgeable, if for all PPT adversaries the following advantage is negligible w.r.t. the experiments in Game 6 for Type-0 and* Type-1 *and the experiment in Game 7 for Type-2:*

$$\mathsf{Adv}^{\mathsf{UNF}}_{\Pi}(\mathcal{A},\lambda) := \Pr\left[\mathsf{Exp}^{\mathsf{UNF}}_{\mathcal{A},\Pi}(\lambda) = 1\right]$$

B IHBBS$_{\mathtt{Gro}}$: BBS-based Type-1 IHAC via [BEK+21]

In this section we sketch how to obtain IHBBS$_{\mathtt{Gro}}$, a Type-1 IHAC Scheme from BBS credentials via the framework of Bobolz et al. [BEK+21]. To do so, we recall the definition of the Groth signature scheme, the protocols used to prove knowledge of a Groth signature and of the signed message via a sigma protocol requiring to perform operations in $\mathbb{G}_T$. Finally, we describe how to apply our map Adapt for the randomization of BBS signatures and public keys to smoothly obtain a Type-1 IHAC scheme.

B.1 Groth Signature Scheme with Efficient Protocols

We recall the Groth digital signature scheme presented in [Gro15]. This signature scheme allows to sign group elements from a bilinear group. It can be defined to sign elements in the group $\mathbb{G}_1$ or $\mathbb{G}_2$; since we will be use this scheme to sign BBS public keys, we will present it in the latter mode.

Game 6: Unforgeability (Type-0, [Type-1])

$\underline{\mathsf{Exp}^{\mathsf{UNF}}_{\mathcal{A},\mathsf{IHAC}}(\lambda)}$
$pp \leftarrow\!\$\ \mathsf{PGen}(1^\lambda);\quad Q_{\mathsf{keys}}, Q_{\mathsf{creds}} \leftarrow ();\quad Q_{\mathsf{pols}}, Q_{\mathsf{pres}}, Q_{\mathsf{cor}} \leftarrow \emptyset$
$(pol^\dagger, \boxed{pk^\dagger_{pol},} pres^\dagger, \mathcal{D}^\dagger, (a_i'^\dagger)_{i\in\mathcal{D}^\dagger}, nonce^\dagger) \leftarrow\!\$$
$\quad \mathcal{A}^{\mathcal{O}_{\mathsf{SetPolicy}}, \mathcal{O}_{\mathsf{KGen}}, \mathcal{O}_{\mathsf{Issue}}, \mathcal{O}_{\mathsf{Corrupt}}, \mathcal{O}_{\mathsf{Present}}}(pp)$
return 0 **if** :
$\quad$ // Policy not trusted
$\quad (pol^\dagger, \boxed{pk^\dagger_{pol}}) \notin Q_{\mathsf{pols}}$
$\quad$ // Matching corrupt credential exists
$\quad \vee\ \exists (cred, \mathbf{a}, ipk) \in Q_{\mathsf{cor}}$:
$\quad\quad (a_i'^\dagger)_{i\in\mathcal{D}^\dagger} = (a_i)_{i\in\mathcal{D}^\dagger} \wedge ipk \in pol^\dagger$
$\quad$ // Presentation not fresh
$\quad \vee\ (pol^\dagger, \boxed{pk^\dagger_{pol}}, \mathcal{D}^\dagger, (a_i'^\dagger)_{i\in\mathcal{D}^\dagger}, nonce^\dagger) \in Q_{\mathsf{pres}}$
return $\mathsf{VfPres}(pol^\dagger, \boxed{pk^\dagger_{pol},} pres^\dagger, \mathcal{D}^\dagger, (a_i'^\dagger)_{i\in\mathcal{D}^\dagger}, nonce^\dagger)$

$\underline{\mathcal{O}_{\mathsf{Present}}(pol, \boxed{pk_{pol},} i_{cred}, \mathcal{D}, nonce)}$
$\boxed{\textbf{if } \mathsf{VfPolicy}(pol, pk_{pol}) = 0 : \textbf{return } \bot}$
if $i_{cred} \notin Q_{\mathsf{creds}}$: **return** $\bot$
$(cred, \mathbf{a}, ipk) \leftarrow Q_{\mathsf{creds}}[i_{cred}]$
if $ipk \notin pol$: **return** $\bot$
$Q_{\mathsf{pres}} \leftarrow Q_{\mathsf{pres}} \cup \{(pol, \boxed{pk_{pol}}, \mathcal{D}, (a_i)_{i\in\mathcal{D}}, nonce)\}$
return $\mathsf{Present}(pol, \boxed{pk_{pol},} ipk, cred, \mathbf{a}, \mathcal{D}, nonce)$

$\underline{\mathcal{O}_{\mathsf{SetPolicy}}(pol)}$
if $pol \not\subseteq Q_{\mathsf{keys}}$: **return** $\bot$
$\boxed{pk_{pol} \leftarrow\!\$\ \mathsf{SetPolicy}(pol)}$
$Q_{\mathsf{pols}} \leftarrow Q_{\mathsf{pols}} \cup \{(pol\boxed{, pk_{pol}})\}$
$\boxed{\textbf{return } pk_{pol}}$

$\underline{\mathcal{O}_{\mathsf{KGen}}()}$
$(isk, ipk) \leftarrow\!\$\ \mathsf{KGen}(pp)$
$Q_{\mathsf{keys}}[ipk] \leftarrow isk$
return ipk

$\underline{\mathcal{O}_{\mathsf{Issue}}(ipk, \mathbf{a})}$
if $ipk \notin Q_{\mathsf{keys}}$: **return** $\bot$
$cred \leftarrow\!\$\ \mathsf{Issue}(Q_{\mathsf{keys}}[ipk], \mathbf{a})$
$Q_{\mathsf{creds}}[|Q_{\mathsf{creds}}|] \leftarrow (cred, \mathbf{a}, ipk)$

$\underline{\mathcal{O}_{\mathsf{Corrupt}}(i_{cred})}$
if $i_{cred} \notin Q_{\mathsf{creds}}$: **return** $\bot$
$Q_{\mathsf{cor}} \leftarrow Q_{\mathsf{cor}} \cup Q_{\mathsf{creds}}[i_{cred}]$
return $Q_{\mathsf{creds}}[i_{cred}]$

Definition 10 (Groth). *The algorithms defining the Groth signature scheme are the following:*

$\mathsf{PGen}(1^\lambda)$: *Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be groups of prime order p, where the former two are generated by G_1 and $\tilde{G}_2$, respectively, and $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be the pairing operation. Sample $\tilde{Y} \leftarrow\!\$\ \mathbb{G}_2$ and set the set of public parameters $pp \leftarrow (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbf{e}, G_1, \tilde{G}_2, \tilde{Y})$.*

$\mathsf{KGen}(pp)$: *Sample $sk = v \leftarrow\!\$\ \mathbb{Z}_p^*$ and compute the associated public key $pk = V \leftarrow G_1^v$. Output (sk, pk).*

$\mathsf{Sign}(sk, \tilde{M})$: *Sample $r \leftarrow\!\$\ \mathbb{Z}_p^*$ and output*

$$\sigma = (R, \tilde{S}, \tilde{T}) \leftarrow (G_1^r, (\tilde{Y}\tilde{G}_2^v)^{\frac{1}{r}}, (\tilde{Y}^v \tilde{M})^{\frac{1}{r}}).$$

$\mathsf{Vf}(pk, (R, \tilde{S}, \tilde{T}), \tilde{M})$: *Verify the signature $(R, \tilde{S}, \tilde{T})$ of $\tilde{M}$ by checking*

$$\mathbf{e}(R, \tilde{S}) = \mathbf{e}(G_1, \tilde{Y})\mathbf{e}(V, \tilde{G}_2) \quad \wedge \quad \mathbf{e}(R, \tilde{T}) = \mathbf{e}(V, \tilde{Y})\mathbf{e}(G_1, \tilde{M}).$$

The signature scheme also defines an algorithm to randomize a digital signature which consists in sampling a random $r' \leftarrow\!\$\ \mathbb{Z}_p$ and compute $(R', \tilde{S}', \tilde{T}') \leftarrow (R^{r'}, \tilde{S}^{\frac{1}{r'}}, \tilde{T}^{\frac{1}{r'}})$.

Note that the first two components of a signature do not depend on the message signed but only on the signer key v and the randomness r sampled while computing the signature. This means that the randomization algorithm allows to perfectly randomize the first two component of a signature of the issuer

Game 7: Unforgeability (Type-2)

$\underline{\mathsf{Exp}^{\mathsf{UNF}}_{\mathcal{A},\mathsf{IHAC}}(\lambda)}$
$pp \leftarrow\!\$\ \mathsf{PGen}(1^\lambda);\quad Q_{\mathsf{keys}}, Q_{\mathsf{creds}}, Q_{\mathsf{pols}} \leftarrow ();\quad Q_{\mathsf{pres}}, Q_{\mathsf{cor}} \leftarrow \emptyset$
$(pol^\dagger, pk^\dagger_{pol}, pres^\dagger, \mathcal{D}^\dagger, (a_i'^\dagger)_{i\in\mathcal{D}^\dagger}, nonce^\dagger) \leftarrow\!\$$
$\quad \mathcal{A}^{\mathcal{O}_{\mathsf{SetPolicy}}, \mathcal{O}_{\mathsf{KGen}}, \mathcal{O}_{\mathsf{Issue}}, \mathcal{O}_{\mathsf{Corrupt}}, \mathcal{O}_{\mathsf{Present}}, \mathcal{O}_{\mathsf{Vf}}}(pp)$
return 0 **if** :
$\quad$ // Policy not trusted
$\quad (pol^\dagger, pk^\dagger_{pol}) \notin Q_{\mathsf{pols}}$
$\quad$ // Matching corrupt credential exists
$\quad \vee\ \exists (cred, \mathbf{a}, ipk) \in Q_{\mathsf{cor}}$:
$\quad\quad (a_i'^\dagger)_{i\in\mathcal{D}^\dagger} = (a_i)_{i\in\mathcal{D}^\dagger} \wedge ipk \in pol^\dagger$
$\quad$ // Presentation not fresh
$\quad \vee\ (pol^\dagger, pk^\dagger_{pol}, \mathcal{D}^\dagger, (a_i'^\dagger)_{i\in\mathcal{D}^\dagger}, nonce^\dagger) \in Q_{\mathsf{pres}}$
$sk^\dagger_{pol} \leftarrow Q_{\mathsf{pols}}[(pol^\dagger, pk^\dagger_{pol})]$
return
$\quad \mathsf{VfPres}(pol^\dagger, pk^\dagger_{pol}, sk^\dagger_{pol}, pres^\dagger, \mathcal{D}^\dagger, (a_i'^\dagger)_{i\in\mathcal{D}^\dagger}, nonce^\dagger)$

$\underline{\mathcal{O}_{\mathsf{Present}}(pol, pk_{pol}, i_{cred}, \mathcal{D}, nonce)}$
if $\mathsf{VfPolicy}(pol, pk_{pol}) = 0 \vee$ **if** $i_{cred} \notin Q_{\mathsf{creds}}$: **return** $\bot$
$(cred, \mathbf{a}, ipk) \leftarrow Q_{\mathsf{creds}}[i_{cred}]$
if $ipk \notin pol$: **return** $\bot$
$Q_{\mathsf{pres}} \leftarrow Q_{\mathsf{pres}} \cup \{(pol, pk_{pol}, \mathcal{D}, (a_i)_{i\in\mathcal{D}}, nonce)\}$
return $\mathsf{Present}(pol, pk_{pol}, ipk, cred, \mathbf{a}, \mathcal{D}, nonce)$

$\underline{\mathcal{O}_{\mathsf{Vf}}(pol, pk_{pol}, pres, \mathcal{D}, (a'_i)_{i\in\mathcal{D}}, nonce)}$
if $(pol, pk_{pol}) \notin Q_{\mathsf{pols}}$: **return** $\bot$
$sk_{pol} \leftarrow Q_{\mathsf{pols}}[(pol, pk_{pol})]$
return $\mathsf{VfPres}(pol, pk_{pol}, sk_{pol}, pres, \mathcal{D}, (a'_i)_{i\in\mathcal{D}}, nonce)$

$\underline{\mathcal{O}_{\mathsf{SetPolicy}}(pol)}$
if $pol \not\subseteq Q_{\mathsf{keys}}$: **return** $\bot$
$(sk_{pol}, pk_{pol}) \leftarrow\!\$\ \mathsf{SetPolicy}(pol)$
$Q_{\mathsf{pols}}[(pol, pk_{pol})] \leftarrow sk_{pol}$
return pk_{pol}

$\underline{\mathcal{O}_{\mathsf{KGen}}()}$
$(isk, ipk) \leftarrow\!\$\ \mathsf{KGen}(pp)$
$Q_{\mathsf{keys}}[ipk] \leftarrow isk$
return ipk

$\underline{\mathcal{O}_{\mathsf{Issue}}(ipk, \mathbf{a})}$
if $ipk \notin Q_{\mathsf{keys}}$: **return** $\bot$
$cred \leftarrow\!\$\ \mathsf{Issue}(Q_{\mathsf{keys}}[ipk], \mathbf{a})$
$Q_{\mathsf{creds}}[|Q_{\mathsf{creds}}|] \leftarrow (cred, \mathbf{a}, ipk)$

$\underline{\mathcal{O}_{\mathsf{Corrupt}}(i_{cred})}$
if $i_{cred} \notin Q_{\mathsf{creds}}$: **return** $\bot$
$Q_{\mathsf{cor}} \leftarrow Q_{\mathsf{cor}} \cup Q_{\mathsf{creds}}[i_{cred}]$
return $Q_{\mathsf{creds}}[i_{cred}]$

with public key V. This is useful if we want to build a NIZKP of knowledge of a signature which verifies under V.

Algorithm 3 (Generation and Verification of Groth Presentations). *The statement we will need to prove is that we know a signature under the public key V, and the witness is the signature $(R, \tilde{S}, \tilde{T})$ that verifies under V, and the message $\tilde{M}$ signed in it.*

$pres \leftarrow\!\$\ \mathsf{Present}(pk, cred, \mathbf{a}, \mathcal{D}, nonce)$*: The prover*

- *runs the randomization algorithm on the signature sampling $\rho \leftarrow\!\$\ \mathbb{Z}_p^*$ and computing $(R', \tilde{S}', \tilde{T}') \leftarrow (R^\rho, \tilde{S}^{\frac{1}{\rho}}, \tilde{T}^{\frac{1}{\rho}})$;*
- *randomizes the message by sampling $\gamma \leftarrow\!\$\ \mathbb{Z}_p^*$ computing $\tilde{M}' \leftarrow \tilde{M}^{\frac{1}{\gamma}}$ and blinds the third component of the signature[8] sampling $\delta \leftarrow\!\$\ \mathbb{Z}_p^*$ and computing $(R'', \tilde{S}'', \tilde{T}'') \leftarrow (R', \tilde{S}', \tilde{T}'^{\frac{1}{\delta}})$;*

[8] The third component is the only one depending on the message.

– *computes* π *as:*

$$\mathsf{NIZKP}\Big\{ \Big((a,b),(R'',\tilde{T}'',V,\tilde{Y},G_1,\tilde{M}')\Big) : \mathbf{e}(R'',\tilde{T}'')^a = \mathbf{e}(V,\tilde{Y})\mathbf{e}(G_1,\tilde{M}')^b \Big\}$$

Note that the prover knows a witness (δ,γ) *such that* $\mathbf{e}(R'',\tilde{T}'')^\delta = \mathbf{e}(V,\tilde{Y})\mathbf{e}(G_1,\tilde{M}')^\gamma$.

The proof is given by $pres \leftarrow \Big((R'',\tilde{S}'',\tilde{T}''),\tilde{M}',\pi\Big)$.

$0/1 \leftarrow \mathsf{VfPres}(pk,\mathcal{D},(a'_i)_{i\in\mathcal{D}},nonce,pres)$*: The verifier returns 1 iff:*

– $\mathbf{e}(R'',\tilde{S}'') = \mathbf{e}(G_1,\tilde{Y})\mathbf{e}(V,\tilde{G}_2)$*;*

– *the proof* π *is valid.*

We remark that every time the prover and the verifier must compute an exponentiation of elements in $\mathbb{G}_T$, they compute the exponentiation over $\mathbb{G}_1$ which is the most efficient group, and then compute a pairing to obtain the desired element of $\mathbb{G}_T$.

B.2 Security of Groth's NIZKP

It is possible to show that the interactive protocol underlying the algorithm described above is a sigma protocol which can be made non-interactive using a general purpose transform like Fiat-Shamir [FS87], or Fischlin [Fis05]. The first message of the sigma protocol includes the generation of $R'',\tilde{S}'',\tilde{T}'',\tilde{M}'$, and the first message of π. The challenge is random in $\mathbb{Z}_p$ and the response is given by the responses of π. It is easy to see that the sigma protocol is correct, now we show that it satisfies HVZK and special soundness.

Zero-knowledge. First of all we must assume that a pair $(R_0,\tilde{S}_0)$ s.t. $\mathbf{e}(R_0,\tilde{S}_0) = \mathbf{e}(G_1,\tilde{Y})\mathbf{e}(V,\tilde{G}_2)$ is publicly available. This can be assumed because every signature generated by the owner of the public key V will reveal such a pair, and similarly any presentation of a credential under the policy public key generated by the owner of V will reveal such a pair.

To simulate a presentation, a simulator samples $r \leftarrow\!\!\$\ \mathbb{Z}_p^*$ and randomizes the pair $(R_0,\tilde{S}_0)$ as $(R'',\tilde{S}'') \leftarrow (R_0^r,\tilde{S}_0^{\frac{1}{r}})$. Then it samples $\tilde{T}'',\tilde{M}' \leftarrow\!\!\$\ \mathbb{G}_2$. Note that these elements are distributed as in a real proof.

With this in place, we invoke the simulator of the Schnorr-like sigma protocol to simulate knowledge of a representation of $\mathbf{e}(V,\tilde{Y}) = \mathbf{e}(R'',\tilde{T}'')^a\mathbf{e}(G,\tilde{M}')^b$: it generates $z_a,z_b,ch \leftarrow\!\!\$\ \mathbb{Z}_p$, computes the first message

$$\hat{U} \leftarrow \mathbf{e}(R'',\tilde{T}'')^{z_a}\mathbf{e}(G,\tilde{M}')^{z_b}\mathbf{e}(V,\tilde{Y})^{-ch},$$

and outputs the transcript

$$(\underbrace{R'',\tilde{S}'',\tilde{T}'',\tilde{M}',\hat{U}}_{\text{first message}},ch,\ \underbrace{z_a,z_b}_{\text{response}}).$$

If we instantiate the NIZKP using Fiat-Shamir, the zero-knowledge simulator must program the random oracle setting $H(pp, V, R'', \tilde{S}'', \tilde{T}'', \tilde{M}', \hat{U}) = ch$, which overwrites the hash table only with negligible probability, since the input $(R'', \tilde{S}'', \tilde{T}'', \tilde{M}', \hat{U})$ to the hash function has high min-entropy. Then, the simulator sets $\pi \leftarrow (\hat{U}, ch, z_a, z_b)$.

Note that the Schnorr-like sigma protocol is run in $\mathbb{G}_T$, which is the least efficient and largest group of the bilinear setting. To mitigate the increase in proof size, we recall that $\hat{U}$ can be omitted from π and be reconstructed from ch, z_a, z_b. To conclude, the resulting simulated proof will be given by $R'', \tilde{S}'', \tilde{T}'', \tilde{M}', \pi$.

Special soundness. Given two valid transcripts for the same first message and distinct challenges

$$((R'', \tilde{S}'', \tilde{T}'', \tilde{M}', \hat{U}), ch, z_a, z_b) \text{ and } ((R'', \tilde{S}'', \tilde{T}'', \tilde{M}', \hat{U}), ch', z'_a, z'_b),$$

it is possible to extract (δ, γ) so that $\mathbf{e}(R'', \tilde{T}'')^{\delta} = \mathbf{e}(V, \tilde{Y})\mathbf{e}(G, \tilde{M}')^{\gamma}$ using the standard extractor of the Schnorr sigma protocol.

Thus, the extractor learns a valid message-signature pair, which is the witness of the proof: it holds that $\mathbf{e}(R'', \tilde{S}'') = \mathbf{e}(G_1, \tilde{Y})e(V, \tilde{G}_2)$ and $\mathbf{e}(R'', \tilde{T}''^{\delta}) = \mathbf{e}(V, \tilde{Y})\mathbf{e}(G, \tilde{M}'^{\gamma})$, so by learning (δ, γ), the extractor may compute $(R'', \tilde{S}'', \tilde{T}''^{\delta})$ as a signature for the message $\tilde{M}'^{\gamma}$ under the public key V.

B.3 $\mathsf{IHBBS}_{\mathtt{Gro}}$ Scheme

Next, we describe how to apply the framework by Bobolz et al. [BEK+21] to design a Type-1 IHAC scheme for a BBS credential. Essentially, we combine our technique to randomize the public key of a BBS credential with the framework in [BEK+21]. We formally describe the scheme $\mathsf{IHBBS}_{\mathtt{Gro}}$ as follows.

$\mathsf{PGen}(1^{\lambda})$: Run $pp_{\mathsf{BBS}} \leftarrow\!\!\$\ \mathsf{BBS.PGen}(1^{\lambda})$ and $pp_{\mathsf{Gro15}} \leftarrow\!\!\$\ \mathsf{Gro15.PGen}(1^{\lambda})$ and return $pp \leftarrow (pp_{\mathsf{BBS}}, pp_{\mathsf{Gro15}})$.

$\mathsf{IssKGen}(pp)$: Return $(isk, ipk) \leftarrow\!\!\$\ \mathsf{BBS.KGen}(pp_{\mathsf{BBS}})$.

$\mathsf{Issue}(isk, \mathbf{a})$: Return $cred = (A, e) \leftarrow\!\!\$\ \mathsf{BBS.Sign}(isk, \mathbf{a})$.

$\mathsf{VfCred}(ipk, cred, \mathbf{a})$: Return $\mathsf{BBS.Vf}(ipk, (A, e), \mathbf{a})$.

$\mathsf{SetPolicy}(pol)$: The trusted third party generates a new Groth key pair $(v, X_v = G_1^v) \leftarrow\!\!\$\ \mathsf{Gro15.KGen}(pp_{\mathsf{Gro15}})$ for the specific policy pol and signs every issuer public key $\tilde{X}_i \in pol$ computing σ_i setting $pk_{pol} = ((\tilde{X}_i, \sigma_i)_{i \in pol}, X_v)$.

$\mathsf{VfPolicy}(pol, pk_{pol})$: Verify that the signatures σ_i of the public keys $\tilde{X}_i \in pol$ verify under the public key X_v.

$\mathsf{Present}(pol, pk_{pol}, \tilde{X}_i, cred, \mathcal{D}, nonce)$: Ensure that that $\mathsf{VfPolicy}(pol, pk_{pol}) = 1$ and that $\tilde{X}_i \in pol$. Retrieve from pk_{pol} the signature σ_i for key $\tilde{X}_i$. Randomize $\tilde{X}_i$ and $cred$ accordingly:

- $r \leftarrow\!\!\$\ \mathbb{Z}_p$
- $(\tilde{X}^{\star}, (A^{\star}, e^{\star})) \leftarrow \mathsf{Adapt}(\tilde{X}_i, (A, e), r)$[9]

[9] $\tilde{X}^{\star} \leftarrow \tilde{X}_i^r$ and $(A^{\star}, e^{\star}) \leftarrow (A^{\frac{1}{r}}, re)$

Compute a NIZKP of knowledge of a Groth signature $\sigma_i = (R, \tilde{S}, \tilde{T})$ of $\tilde{M} = \tilde{X}_i$ where $\tilde{M}' \leftarrow \tilde{M}^r$ which ends up being the same $\tilde{X}^\star = \tilde{M}'$ (setting $\gamma = \frac{1}{r}$ according to Algorithm 3)

- randomize σ sampling $\rho \leftarrow\!\$\ \mathbb{Z}_p^*$ and compute $(R', \tilde{S}', \tilde{T}') \leftarrow (R^\rho, \tilde{S}^{\frac{1}{\rho}}, \tilde{T}^{\frac{1}{\rho}})$
- unlike in Algorithm 3 now we do not randomize the message $\tilde{X}_i$ because we use the same randomization above, namely $\tilde{X}^\star = \tilde{X}_i^r$
- blinds the third component of the signature sampling $\delta \leftarrow\!\$\ \mathbb{Z}_p^*$ and computing $(R'', \tilde{S}'', \tilde{T}'') \leftarrow (R', \tilde{S}', \tilde{T}'^{\frac{1}{\delta}})$.
- compute $\pi_1 \leftarrow\!\$\ \mathsf{NIZKP}\Big\{(a, b) \in \mathbb{Z}_p : \mathbf{e}(R'', \tilde{T}'')^a = \mathbf{e}(V, \tilde{Y})\mathbf{e}(G, \tilde{M}')^b\Big\}$[10]

Compute a BBS presentation π_2 of the attributes $(a_i)_{i\in\mathcal{D}}$ from $\mathbf{a}, (A^\star, e^\star)$ valid under $\tilde{X}^\star$. Return $pres \leftarrow \Big((R'', \tilde{S}'', \tilde{T}''), \tilde{X}^\star, \pi_1, \pi_2\Big)$.

$\mathsf{VfPres}(pol, pk_{pol}, \mathcal{D}, (a'_i)_{i\in\mathcal{D}}, pres, nonce)$: The verifier checks that the presentation is valid verifying the combined BBS and Groth-15 presentations that are linked by the randomized public key $\tilde{X}^\star$.

The security of this construction follows from [BEK+21, Section 4.2].

C IHBBS_τ: Full Proofs

C.1 Correctness of IHBBS_τ

It is easy to see that construction IHBBS_τ fulfills correctness (Game 3) except with negligible probability.

Theorem 4. IHBBS_τ *is correct if the* BBS *signature and presentation schemes are correct, and if the tag scheme* algTag *is correct.*

Proof. Let $\lambda, n, \ell \in \mathbb{N}^+$; $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbf{e}, G_1, \tilde{G}_2, H_1, \ldots H_\ell) \leftarrow\!\$\ \mathsf{PGen}(1^\lambda)$; $(isk_1, ipk_1), \ldots, (isk_n, ipk_n) \leftarrow\!\$\ \mathsf{KGen}_{\mathsf{BBS}}(pp)$; $pol \subseteq \{ipk_i\}_{i\in[n]}$ with $|pol| > 0$. Choose some $(isk^*, ipk^*) \in \{(isk_i, ipk_i)\}_{i\in[n]}$ with $ipk^* \in pol$. Choose attributes $\mathbf{a} \in \mathcal{M}^\ell$ and $\mathcal{D} \subseteq [\ell]$. Choose some $nonce \in \{0,1\}^*$.

In the correctness game, first a policy key pk_{pol} is generated, where a tag key pair $(x_v, (G_v, T_v))$ is generated and used to tag each issuer public key in the policy. Next, a credential is issued on the given attributes $\mathbf{a}$, under issuer key isk^* and finally, a presentation of the credential is performed, presenting $(a'_i)_{i\in\mathcal{D}}$ with respect to $nonce$.

For Type-1 constructions, correctness asks three conditions to be fulfilled:

Valid credential. Firstly, the credential must be valid under ipk^* and for attributes $\mathbf{a}$: this follows from correctness of BBS signatures, since as in IHBBS_0, in IHBBS_τ credentials are BBS signatures over $\mathbf{a}$, and VfCred performs exactly the BBS signature verification.

[10] The prover knows a witness $(\delta, \frac{1}{r})$ such that $\mathbf{e}(R'', \tilde{T}'')^\delta = \mathbf{e}(V, \tilde{Y})\mathbf{e}(G, \tilde{M}')^{\frac{1}{r}}$.

Valid presentation. Secondly, we need that the presentation is valid for the disclosed attributes $(a'_i)_{i\in\mathcal{D}}$ and the nonce, under the fixed policy and pk_{pol}. This has once more three sub-conditions. The key pk_{pol} must be valid for pol, which it is if the tag scheme is correct, as we honestly tagged all issuer keys $\in pol$. The BBS presentation must be valid, which it is except with negligible probability if the ZKP over a BBS credential is correct, since it presented $(A^\star, e^\star)$ which is a valid credential over $\mathbf{a}$ under $\tilde{X}^\star$: with $\tilde{X}^\star$ is computed as $(ipk^*)^\alpha$ and $(A^\star, e^\star)$ computed as $(A^{1/\alpha}, \alpha e)$, we have:

$$\begin{aligned} &\mathbf{e}(A^\star, \tilde{X}^\star) = \mathbf{e}(C(\mathbf{a})(A^\star)^{-e^\star}, \tilde{G}_2) \\ \Leftrightarrow\ &\mathbf{e}(A^{1/\alpha}, (ipk^*)^\alpha) = \mathbf{e}(C(\mathbf{a})(A^{1/\alpha})^{-\alpha e}, \tilde{G}_2) \\ \Leftrightarrow\ &\mathbf{e}(A, ipk^*) = \mathbf{e}(C(\mathbf{a})A^{-e}, \tilde{G}_2) \end{aligned}$$

where the last equation holds since (A, e) is a valid signature of $\mathbf{a}$ under ipk^*. Finally, we also need $\tilde{T}^\star$ to be a valid tag for $\tilde{X}^\star$ under the tag key pair (G_v, T_v). Indeed, $\mathsf{VfTag}_{\mathsf{algTag}}((G_v, T_v), \tilde{T}^\star, \tilde{X}^\star) = 1$ (unless with negligible probability $\tilde{T}^\star = 1_{\mathbb{G}_2}$), since:

$$\begin{aligned} &\mathbf{e}(T_v, \tilde{X}^\star) = \mathbf{e}(G_v, \tilde{T}^\star) \\ \Leftrightarrow\ &\mathbf{e}(T_v, (ipk^*)^\alpha) = \mathbf{e}(G_v, \tilde{T}^\alpha) \\ \Leftrightarrow\ &\mathbf{e}(T_v, ipk^*)^\alpha = \mathbf{e}(G_v, \tilde{T})^\alpha \end{aligned}$$

which holds if the tag scheme algTag is correct.

Valid policy key. Thirdly, we want that the policy key pk_{pol} is valid for policy pol. This was already checked inside VfPres. □

C.2 Unlinkability of IHBBS_τ

We extend the proof given in Section 5.4 with additional detail, *emphasized* in the following.

Proof (Theorem 1).

Game_0. This is exactly the unlinkability game in with $b = 0$, where oracle $\mathcal{O}_{\mathsf{Chal}}$ always returns a presentation of input set 0.

Game_1. We do lazy sampling for the random oracle H. This change is indistinguishable.

Game_2. We continue to create $ipk^\star$ from $ipk^{(0)}$ and to create $\overline{A}$ and $\overline{B}$ from $cred^{(0)}$ and $\mathbf{a}^{(0)}$, but then call the BBS zero-knowledge simulator to simulate the rest of the proof. The simulator is allowed to program the random oracle. *In detail: we start by computing* $ipk^\star, \overline{A}$, *and* $\overline{B}$ *as before, by randomizing the*

inputs $ipk^{(0)}$, $cred^{(0)}$, *and using* $\mathbf{a}^{(0)}$. *The rest of the proof is simulated: we sample random responses* $z_r, (z_j)_{j\in[\ell]\setminus\mathcal{D}}, z_e \leftarrow\!\!\$\ \mathbb{Z}_p^{\ell-|\mathcal{D}|+2}$ *and a random challenge* $ch \leftarrow\!\!\$\ \mathbb{Z}_p$. *Then, we simulate the first message* $U \leftarrow C(\mathbf{a}')^{z_r}(\prod_{j\in[\ell]\setminus\mathcal{D}} H_j^{z_j})\overline{A}^{z_e}\overline{B}^{ch}$. *Finally, we need to program the random oracle to return the right value during verification of the proof, so we set* $\mathsf{H}(pp, pol, \mathcal{D}, ipk^\star, \overline{A}, \overline{B}, U, nonce) \coloneqq ch$. *With overwhelming probability, the random oracle is yet undefined at this point, since* $ipk^\star$ *and* $\overline{A}$ *are truly random elements that the adversary queried before only with negligible probability.* Thus, challenge and response of the proof are independent of the witnesses except for what is leaked by $ipk^\star, \overline{A}$, and $\overline{B}$. This change is indistinguishable if the BBS presentation is zero-knowledge.

Game_3. Next, we create the announcements $ipk^\star, \overline{A}, \overline{B}$ and the tag $\tilde{T}^\star$ from the input set 1 instead of the input set 0. *This does not change the distribution of the adversary's view:* $ipk^\star$ *will be a uniformly random element from* $\mathbb{G}_2$ *(*$(ipk^{(b)})^\alpha$ *for a random* $\alpha \leftarrow\!\!\$\ \mathbb{Z}_p$*) in both cases, and* $\overline{A}$ *will be a uniformly random element from* $\mathbb{G}_1$ *(*$(A^{(b)})^{r/\alpha}$ *for* $(A^{(b)}, e^{(b)}) \leftarrow cred^{(b)}$ *and a uniformly random* $r \leftarrow\!\!\$\ \mathbb{Z}_p$*). Then,* $\overline{B}$ *is uniquely determined by* $\overline{A}$ *and* $ipk^\star$ *as* $\overline{B} = \overline{A}^{dlog(G_2, ipk^\star)}$. *Further, the tag* $\tilde{T}^\star$ *is uniquely determined by* $ipk^\star$ *and the policy key* pk_{pol} *(which is independent of b) as* $\tilde{T}^\star = (ipk^\star)^{dlog(G_v, X_v)}$, *where* G_v *and* X_v *are part of the policy key. We are sure this computation is possible in both worlds: there must be a* $\tilde{T}_i$ *matching both* $ipk^{(0)}$ *and* $ipk^{(1)}$ *due to the checks* $ipk^{(b)} \in pol$ *and* $\mathsf{VfPolicy}(pol, pk_{pol})$*; additionally we are sure that the discrete logarithm between* $\tilde{T}^{(0)}$ *and* $ipk^{(0)}$ *on the one hand and* $\tilde{T}^{(1)}$ *and* $ipk^{(1)}$ *are exactly* $dlog(G_v, X_v)$*, due to the* $\mathsf{VfPolicy}$ *check. Note also that due to the changes made in* Game_2*, neither holder-specific inputs nor the randomization values* α *and* r *are further used in the proof.*

Game_4. We remove the simulator again and perform a regular presentation of $ipk^{(1)}$, $cred^{(1)}$, and $\mathbf{a}^{(1)}$ instead, which is indistinguishable if the BBS presentation is zero-knowledge.

Game_5. Finally, since the random oracle no longer requires programming, we remove the lazy sampling introduced in the first game hop. This leaves us with exactly the unlinkability game with $b = 1$.
Overall, we have shown $\mathsf{Exp}^{\mathsf{UNLINK},0}_{\mathcal{A},\mathsf{IHBBS}_\tau}$ and $\mathsf{Exp}^{\mathsf{UNLINK},1}_{\mathcal{A},\mathsf{IHBBS}_\tau}$ to be distinguishable only with negligible probability as long as the BBS presentation is zero-knowledge, which concludes the proof. □

C.3 Unforgeability of IHBBS_τ

We augment the proof sketch of Theorem 2 with additional detail. Let us first state the extended version of the theorem:

Theorem 5 (IHBBS_τ UNF, extended theorem). *Let* $\mathcal{A}$ *be an algebraic adversary in the ROM against* UNF *of* IHBBS_τ *that requests at most* q_k *different*

issuer keys and at most q_p different policies. Then there exists an adversary $\mathcal{B}$ in the strong unforgeability game against BBS *signatures such that*

$$\Pr[\mathcal{B}\ \textit{wins}] \geq \frac{1}{4} \cdot \left(\frac{\left(\mathsf{Adv}^{\mathsf{UNF}}_{\mathcal{A},\mathsf{IHBBS}_\tau}\right)^2}{q_H} + \mathsf{negl}(\lambda) \right).$$

Proof (Theorem 5). We design two reductions $\mathcal{B}_1, \mathcal{B}_2$ to the strong unforgeability of BBS signatures that will be building blocks for our main reduction $\mathcal{B}$.

Reduction $\mathcal{B}_1$.

Setup phase. The reduction $\mathcal{B}_1$ receives from $\mathcal{C}$ the following information:

$$(p, \mathbf{e}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, G_1, \tilde{G}_2), (H_1, \ldots, H_m), \tilde{X},$$

and forwards to $\mathcal{A}$

$$(p, \mathbf{e}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, G_1, \tilde{G}_2), (H_1, \ldots, H_m).$$

$\mathcal{B}_1$ computes a pair $(U_1, U_2) \in \mathbb{G}_1^2$ s.t. $\mathbf{e}(U_1, \tilde{X}) = \mathbf{e}(U_2, \tilde{G}_2)$ which will be needed during the simulation of the IHAC unforgeability game. To do so, it samples a random vector of attributes $\mathbf{a}^* \leftarrow\!\!\$\ \mathbb{Z}_p^m$ and sends a sign query to $\mathcal{C}$ for $\mathbf{a}^*$ obtaining (A^*, e^*), valid under $\tilde{X}$. From these values it can compute $U_1 \leftarrow A^*$ and $U_2 \leftarrow C(\mathbf{a}^*)(A^*)^{-e^*}$ which satisfy $\mathbf{e}(U_1, \tilde{X}) = \mathbf{e}(U_2, \tilde{G}_2)$ by the verification equation of BBS signatures. Note that (U_1, U_2) are distributed uniformly random in the set of pairs of elements in $\mathbb{G}_1$ satisfying $\mathbf{e}(U_1, \tilde{X}) = \mathbf{e}(U_2, \tilde{G}_2)$.

Training phase. $\mathcal{B}_1$ operates the oracles for $\mathcal{A}$ as follows:

$\mathcal{O}_{\mathsf{RO}}$: Upon reception of a query for m to $\mathcal{O}_{\mathsf{RO}}$, if the input was already added to Q_{RO}, return $Q_{\mathsf{RO}}[m]$, otherwise, sample uniformly at random $d \leftarrow\!\!\$\ \mathbb{Z}_p$, set $Q_{\mathsf{RO}}[m] \leftarrow d$ and return d.

$\mathcal{O}_{\mathsf{KGen}}$: When the adversary asks for a new public key, sample $\alpha_i \leftarrow\!\!\$\ \mathbb{Z}_p^*$ and return $\tilde{X}_i \leftarrow \tilde{X}^{\alpha_i}$. The public key is added to Q_{keys}.

$\mathcal{O}_{\mathsf{Issue}}$: Return nothing but record the *ipk* and $\mathbf{a}$ corresponding to the i-th query to this oracle.

$\mathcal{O}_{\mathsf{Corrupt}}$: When corruption of a credential from $\tilde{X}_i$ for $\mathbf{a}$ is queried, send a sign query to $\mathcal{C}$ for attributes $\mathbf{a}$ and receive (A, e). Update the signature computing $(\tilde{X}_i, (A', e')) \leftarrow \mathsf{Adapt}(\tilde{X}, (A, e), \alpha_i)$ and send (A', e') (a valid signature of $\mathbf{a}$ under $\tilde{X}_i$) to $\mathcal{A}$. The credential is added to Q_{cor}. Note that, (A, e) being generated honestly by $\mathcal{C}$ sampling a random e, the adapted credential has the right distribution.

$\mathcal{O}_{\mathsf{SetPolicy}}$: When the adversary queries for the i-th policy public key for pol_i, sample $\beta_i, \gamma_i \leftarrow\!\!\$\ \mathbb{Z}_p^*$ and set $(G_{v,i}, T_{v,i}) \leftarrow (U_2^{\gamma_i \beta_i}, U_1^{\gamma_i})$—a tag public key associated to the secret key $x\beta_i$, which is unknown (and is a randomization of the secret key from the BBS unforgeability game). Then, for every $\tilde{X}_j \in pol_i$, compute $\tilde{T}_j \leftarrow \tilde{G}_2^{\frac{\alpha_i}{\beta_i}}$ and return

$$pk_{pol_i} \leftarrow ((G_{v,i}, T_{v,i}), (\tilde{X}_j, \tilde{T}_j)_{j \in pol_k}).$$

The generated policy key is added to Q_{pols}.

$\mathcal{O}_{\mathsf{Present}}$: When the adversary requests a presentation under a policy (pol, pk_{pol}) and of the i_{cred}-th credential generated by $\mathcal{O}_{\mathsf{Issue}}$ for the attribute indices in $\mathcal{D}$ and nonce $nonce$, we verify pk_{pol} and assert that the ipk corresponding to the i_{cred}-th call to $\mathcal{O}_{\mathsf{Issue}}$ is part of the policy. In case any of the checks fails, return $\perp$. Otherwise, simulate a presentation as follows:

- Parse $(pol, (G_v, T_v), (\tilde{X}_j, \tilde{T}_j)_{j \in pol}) \leftarrow pk_{pol}$.
- Construct a uniformly random and valid key-tag pair by randomizing the public key and the tag $\tilde{X}_i, \tilde{T}_i$ of the issuer of the credential indexed by i_{cred}, which must be in the policy. Hence, randomize $(\tilde{X}_i, \tilde{T}_i) \in pk_{pol}$ by sampling $r \leftarrow\!\!\$\ \mathbb{Z}_p$ and computing $(\tilde{X}^\star, \tilde{T}^\star) \leftarrow (\tilde{X}_i^r, \tilde{T}_i^r)$.
- Run the simulator of BBS presentations[11] with inputs $(\tilde{X}^\star, \mathcal{D}, (a'_i)_{i \in \mathcal{D}})$ where $(a'_i)_{i \in \mathcal{D}}$ are the attributes with indices in $\mathcal{D}$ of the vector of attributes queried in the i_{cred}-th query to $\mathcal{O}_{\mathsf{Issue}}$[12]. Note that this simulator needs to program the random oracle (including also the $nonce$ in the input), but this programming overwrites Q_{RO} only with probability $\leq \frac{q_H + q_P}{p^2}$ where q_H is the maximum number of RO queries, and q_P the maximum number of presentation queries. The reason is that the sigma protocol for BBS presentations has first messages which include the values $(\overline{A}, \overline{B})$ sampled from a space of size p, and U, sampled as well from a space of size p, and the hash table contains at most $q_H + q_P$ entries. If Q_{RO} needs to be overwritten, $\mathcal{B}_1$ aborts, otherwise let $\pi^\star$ be the simulated BBS proof, and return $(\tilde{X}^\star, \tilde{T}^\star, \pi^\star)$ (which is indistinguishable from a real presentation). The presentation is then added to Q_{pres}.

We call '$\mathcal{B}_1$ $\mathsf{simulates}$' the event that $\mathcal{B}_1$ correctly simulates all the oracle queries. The only queries that can fail are the queries to $\mathcal{O}_{\mathsf{Present}}$. It is possible to show, using a similar argument as in [FLL25, Appendix G.1], that $\mathcal{B}_1$ does not abort with probability $\Pr[\mathcal{B}_1\ \mathsf{simulates}] \geq 1 - \frac{(q_H + q_P)^2}{p^2}$, which is overwhelming.

Forgery phase. Eventually, the adversary produces a forgery attempt

$$(pol^\dagger, pk_{pol^\dagger}, \mathcal{D}^\dagger, (a_i^\dagger)_{i \in \mathcal{D}^\dagger}, nonce^\dagger, (\tilde{X}^\dagger, \tilde{T}^\dagger, \pi^\dagger)),$$

[11] This simulator leverages on the HVZK simulator of BBS presentations presented in [TZ23, Section 5]

[12] The simulator needs in input also the leakage of the randomized public key, but this can be obtained from the leakage of the public key $\tilde{X}_i$, randomizing it accordingly.

which is a winning forgery if (1) $(pol^\dagger, pk_{pol^\dagger}) \in Q_{\mathsf{pols}}$, $pol^\dagger \subseteq Q_{\mathsf{keys}}$, (2) it has not previously corrupted a credential from one issuer in the certified policy $pol^\dagger$, for $(a_i^\dagger)_{i\in\mathcal{D}^\dagger}$ (3) it is not a presentation previously queried to $\mathcal{O}_{\mathsf{Present}}$ and (4) that is a presentation of a credential from a random public key $\tilde{X}^\dagger$ for which the adversary can produce a valid tag $\tilde{T}^\dagger$[13] w.r.t. the tag public key $(G_v^\dagger, T_v^\dagger) \in pk_{pol^\dagger}$. The reduction can extract a valid BBS credential for the presented attributes valid under the public key $\tilde{X}^\dagger$ using the same rewinding strategy as in [FLL25, Lemma 6] and we call '$\mathcal{B}_1$ extracts' the event that $\mathcal{B}_1$ extracts such a credential. Note that $\mathcal{B}_1$ receives a valid forgery from $\mathcal{A}$ with probability

$$\begin{aligned}\epsilon_1 = \Pr\left[\mathcal{A}^{\mathcal{B}_1}\mathsf{wins} \wedge \mathcal{B}_1 \text{ simulates}\right] &= \Pr\left[\mathcal{A}^{\mathcal{B}_1}\mathsf{wins} | \mathcal{B}_1 \text{ simulates}\right] \Pr[\mathcal{B}_1 \text{ simulates}] \\ &\geq \mathsf{Adv}^{\mathsf{UNF}}_{\mathcal{A},\mathsf{IHBBS}_\tau} \cdot \left(1 - \frac{(q_H + q_P)^2}{p^2}\right),\end{aligned}$$

where '$\mathcal{A}^{\mathcal{B}_1}\mathsf{wins}$' denotes that $\mathcal{A}$, interacting with $\mathcal{B}_1$, wins the issuer-hiding unforgeability game. Then, $\mathcal{B}_1$ can extract such a credential with probability

$$\begin{aligned}\Pr[\mathcal{B}_1 \text{ extracts}] \geq \frac{\epsilon_1^2}{q_H} - \frac{\epsilon_1}{p} \geq &\left(\mathsf{Adv}^{\mathsf{UNF}}_{\mathcal{A},\mathsf{IHBBS}_\tau} \cdot \left(1 - \frac{(q_H + q_P)^2}{p^2}\right)\right)^2 / q_H \\ &- \left(\mathsf{Adv}^{\mathsf{UNF}}_{\mathcal{A},\mathsf{IHBBS}_\tau} \cdot \left(1 - \frac{(q_H + q_P)^2}{p^2}\right)\right) / p.\end{aligned}$$

by rewinding the adversary $\mathcal{A}$ [14]. We can rewrite the inequality above as

$$\Pr[\mathcal{B}_1 \text{ extracts}] \geq \frac{\left(\mathsf{Adv}^{\mathsf{UNF}}_{\mathcal{A},\mathsf{IHBBS}_\tau}\right)^2}{q_H} + \mathsf{negl}(\lambda), \tag{2}$$

where the negligible terms are collected in the function $\mathsf{negl}(\lambda)$.
If $\neg\mathcal{B}_1$ extracts happens, $\mathcal{B}_1$ aborts.

At this point, $\mathcal{B}_1$ has extracted a credential $(A^\dagger, e^\dagger)$ for $\mathbf{a}^\dagger$ valid under $\tilde{X}^\dagger$, and the presentation is for pol_k.
We define the event 'Coll_1' as the event that either:

- the extracted credential $((A^\dagger, e^\dagger), \tilde{X}^\dagger) \in [(A, e), \tilde{X}_j]^{\mathbf{a}^\dagger}$ for some credential (A, e) on the same attributes $\mathbf{a}^\dagger$ issued to $\mathcal{A}$ on behalf of $\tilde{X}_j$ over the experiment execution, or
- $\mathbf{a}^* = \mathbf{a}^\dagger$ and $((A^\dagger, e^\dagger), \tilde{X}^\dagger) \in [(A^*, e^*), \tilde{X}]^{\mathbf{a}^\dagger}$, where $\mathbf{a}^*$ are the random attributes queried to the BBS signature strong unforgeability game (receiving the signature (A^*, e^*)) to generate U_1, U_2.

[13] We recall that $\tilde{X}^\dagger \neq 1_{\mathbb{G}_2}, \tilde{T}^\dagger \neq 1_{\mathbb{G}_2}$ is not a valid key-tag pair.
[14] Note that this tightness loss, and the need for rewinding, could be avoided if one used online-extractable compilers such as [Fis05,Unr15] to compile the proof of knowledge of a BBS credential, which is part of the presentation protocol.

Note that $\mathcal{B}_1$ can check for Coll_1 using the efficient algorithm $\mathsf{VfEquivalence}$ with input $(\tilde{X}^\dagger, (A^\dagger, e^\dagger))$, the extracted credential and randomized public key, and $\tilde{X}, (A, e)$ for all the signatures queried to $\mathcal{C}$.
If Coll_1 happens, $\mathcal{B}_1$ aborts.

Next, since the adversary is algebraic, it provides $\mathcal{B}_1$ with a representation of $(\tilde{X}^\dagger, \tilde{T}^\dagger)$ with respect to the elements in $\mathbb{G}_2$ it has previously seen that are:

- the generator of $\mathbb{G}_2$ which is $\tilde{G}_2$
- public keys of the issuers, that are powers of $\tilde{X}$
- the tags included in policies that are powers of $\tilde{G}_2$
- the public keys and tags included in the presentations that are randomizations of group elements in $\mathbb{G}_2$ provided by $\mathcal{A}$, so that are linear combinations of the group elements previously listed

Therefore, omitting the elements of the fourth category, $\mathcal{B}_1$ can reconstruct:

$$\tilde{X}^\dagger = \underbrace{\tilde{G}_2^{\alpha_1} \cdot \left(\prod_{pol_i \in Q_{\mathsf{pols}}} \prod_{r \in pol_i} \tilde{T}_r^{\delta_{1,i,r}} \right)}_{\text{Known representation w.r.t. } \tilde{G}_2} \cdot \underbrace{\prod_{i \in \mathcal{O}_{\mathsf{KGen}}} \tilde{X}_i^{\beta_{1,i}}}_{\text{Known representation w.r.t. } \tilde{X}}$$

$$\tilde{T}^\dagger = \underbrace{\tilde{G}_2^{\alpha_2} \cdot \left(\prod_{pol_i \in Q_{\mathsf{pols}}} \prod_{r \in pol_i} \tilde{T}_r^{\delta_{2,i,r}} \right)}_{\text{Known representation w.r.t. } \tilde{G}_2} \cdot \underbrace{\prod_{i \in \mathcal{O}_{\mathsf{KGen}}} \tilde{X}_i^{\beta_{2,i}}}_{\text{Known representation w.r.t. } \tilde{X}}$$

We make a series of observations which will allow us to prove an auxiliary equation that will ultimately allow us to bound $\mathcal{B}_1's$ winning probability.

First, observe that $\mathcal{B}_1$ can thus reconstruct the representation of every group element in $\mathbb{G}_2$ received from $\mathcal{A}$ w.r.t. $\tilde{G}_2, \tilde{X}$, and in particular it learns the representation of $\tilde{X}^\dagger = \tilde{X}^a \tilde{G}_2^b$ and $\tilde{T}^\dagger = \tilde{X}^c \tilde{G}_2^d$.

Second, let us parse $pk_{pol^\dagger}$ as $(G_v^\dagger, T_v^\dagger), \{(\tilde{X}_{i,\dagger}, \tilde{T}_{i,\dagger}) : \tilde{X}_{i,\dagger} \in pol^\dagger\}$. Since $(pol^\dagger, pk_{pol^\dagger}) \in Q_{\mathsf{pols}}$, $\mathcal{B}_1$ knows the value $\beta^\dagger$ such that $(T_v^\dagger)^{x\beta^\dagger} = G_v^\dagger$ according to how queries to $\mathcal{O}_{\mathsf{SetPolicy}}$ are answered.

Finally, $(\tilde{X}^\dagger, \tilde{T}^\dagger)$ being a valid key-tag pair under the tag public key $(G_v^\dagger, T_v^\dagger)$, and $a, b, c, d, \beta^\dagger \in \mathbb{Z}_p, \beta^\dagger \neq 0$ being known scalars that satisfy, for an undetermined $x \in \mathbb{Z}_p^*$, we know the following relations to hold:

(i) $G_v^\dagger = (T_v^\dagger)^{x\beta^\dagger}$
(ii) $\tilde{X}^\dagger = \tilde{X}^a \tilde{G}_2^b$
(iii) $\tilde{T}^\dagger = \tilde{X}^c \tilde{G}_2^d$
(iv) $\tilde{X} = \tilde{G}_2^x$

Now, we show the following auxiliary equation:

$$c\beta^\dagger x^2 + (d\beta^\dagger - a)x - b = 0 \mod p. \tag{3}$$

In fact, as $(\tilde{X}^\dagger, \tilde{T}^\dagger) \neq (\mathbf{1}_{\mathbb{G}_2}, \mathbf{1}_{\mathbb{G}_2})$ is a valid key-tag pair under $(G_v^\dagger, T_v^\dagger)$,[15] it satisfies $\mathbf{e}(T_v^\dagger, \tilde{X}^\dagger) = \mathbf{e}(G_v^\dagger, \tilde{T}^\dagger)$ then, according to (i), and by the bilinearity and non-degeneracy of $\mathbf{e}$,

$$\mathbf{e}(T_v^\dagger, \tilde{X}^\dagger) = \mathbf{e}((T_v^\dagger)^{x\beta^\dagger}, \tilde{T}^\dagger) \implies \mathbf{e}(T_v^\dagger, \tilde{X}^\dagger) = \mathbf{e}(T_v^\dagger, (\tilde{T}^\dagger)^{x\beta^\dagger}) \implies$$
$$\mathbf{e}(T_v^\dagger, \tilde{X}^\dagger(\tilde{T}^\dagger)^{-x\beta^\dagger}) = \mathbf{1}_{\mathbb{G}_T} \implies \tilde{X}^\dagger = (\tilde{T}^\dagger)^{x\beta^\dagger}$$

By applying the constraints in (ii) and (iii) we can write

$$\tilde{X}^\dagger = (\tilde{T}^\dagger)^{x\beta^\dagger} \implies \tilde{X}^a \tilde{G}_2^b = (\tilde{X}^c \tilde{G}_2^d)^{x\beta^\dagger}$$

and by applying the constraint in (iv) we can write

$$\tilde{X}^a \tilde{G}_2^b = (\tilde{X}^c \tilde{G}_2^d)^{x\beta^\dagger} \implies \tilde{G}_2^{ax} \tilde{G}_2^b = (\tilde{G}_2^{cx} \tilde{G}_2^d)^{x\beta^\dagger} \implies \tilde{G}_2^{ax+b} = \tilde{G}_2^{c\beta^\dagger x^2 + d\beta^\dagger x}.$$

Finally, since $\tilde{G}_2$ is a generator of $\mathbb{G}_2$, we can write

$$\tilde{G}_2^{c\beta^\dagger x^2 + d\beta^\dagger x} = \tilde{G}_2^{ax+b} \implies c\beta^\dagger x^2 + (d\beta^\dagger - a)x - b = 0 \mod p.$$

Let us rename $d' \leftarrow d\beta^\dagger$ and $c' \leftarrow c\beta^\dagger$, hence, Equation (3) can be rewritten as

$$c'x^2 + (d' - a)x - b = 0 \mod p.$$

We know that a value x satisfying Equation (3) exists[16], namely $\log_{\tilde{G}_2} \tilde{X}$ (by the way $G_v^\dagger, T_v^\dagger$ have been computed). However, it being a quadratic equation modulo a prime p, it might have either 1, 2 or p solutions (the latter if the equation is undetermined with $(c', d' - a, b) = (0, 0, 0)$).

Let us consider the following two cases:

- The first, corresponds to the case where the equation is undetermined as $d' = a$ and $b = c' = 0$ (which implies $b = c = 0$ as $\beta^\dagger \neq 0$), so $\tilde{X}^\dagger = \tilde{X}^a$ and that $\tilde{T}^\dagger = \tilde{G}_2^d = \tilde{G}_2^{\frac{a}{\beta^\dagger}}$. In this case, $\mathcal{B}_1$ has computed $\tilde{X}^\dagger = \tilde{X}^a$ and $\tilde{T}^\dagger = \tilde{G}_2^d = \tilde{G}_2^{\frac{a}{\beta^\dagger}}$, as $d' = d\beta^\dagger = a$ and $b = c' = 0$. Then, using the extracted BBS signature $((A^\dagger, e^\dagger), \mathbf{a}^\dagger)$, $\mathcal{B}_1$ can compute $(A', e') \leftarrow \mathsf{Adapt}(\tilde{X}^\dagger, (A^\dagger, e^\dagger), \frac{1}{a})$ to obtain a signature of $\mathbf{a}^\dagger$ valid under the public key $\tilde{X}$.
  This is a forgery unless the signature lives in the same class of another signature queried to the BBS unforgeability game, but this is not the case, as $\mathcal{B}_1$ did not abort as a consequence of Coll, so $\mathcal{B}_1$ wins the strong unforgeability experiment for BBS signature.

[15] $\tilde{X}^\dagger, \tilde{T}^\dagger$ are both different from $\mathbf{1}_{\mathbb{G}_2}$, because the verification of presentations relies on VfTag, which outputs 1 if the tagged message is not the identity element of $\mathbb{G}_2$.

[16] We exclude the case that the equation has no solution in the categorization of quadratic equations modulo a prime in Appendix D.

– The second, corresponds to the case where the key-tag pair is associated to a representation such that Equation (3) is determined and has one or two solutions, of which one must be the $\log_{\tilde{G}_2} \tilde{X}$. In this case, Equation (3) can be efficiently solved (see Lemma 1) and the reduction can find the single (if $c = 0$ or $(d' - a)^2 + 4c'b = 0$) or two solutions (if $c' \neq 0$ and $(d' - a)^2 + 4c'b \neq 0$). If the equation has two solutions, the reduction chooses the one satisfying $\tilde{G}_2^x = \tilde{X}$. Knowing the value x, which is the challenge secret key in the BBS unforgeability game, the reduction $\mathcal{B}_1$ can trivially win the strong unforgeability experiment for BBS signatures.

$$\Pr[\mathcal{B}_1 \text{ wins}] \geq \Pr[\mathcal{B}_1 \text{ extracts} \wedge \neg \mathsf{Coll}] = \\ \Pr[\neg \mathsf{Coll} | \mathcal{B}_1 \text{ extracts}] \Pr[\mathcal{B}_1 \text{ extracts}],$$

so, by Equation (2), the following holds

$$\Pr[\mathcal{B}_1 \text{ wins}] \geq \Pr[\neg \mathsf{Coll} | \mathcal{B}_1 \text{ extracts}] \cdot \left(\frac{\left(\mathsf{Adv}^{\mathsf{UNF}}_{\mathcal{A}, \mathsf{IHBBS}_\tau} \right)^2}{q_H} + \mathsf{negl}(\lambda) \right). \quad (4)$$

Reduction 2: to SUF-CMA of BBS As for the previous reduction, we describe how $\mathcal{B}_2$ interacts with the adversary $\mathcal{A}$ of the issuer hiding unforgeability, and with the challenger of the SUF-CMA of BBS signatures $\mathcal{C}$. Reduction $\mathcal{B}_2$ is indistinguishable from $\mathcal{B}_1$ for $\mathcal{A}$, but is able to win the SUF-CMA experiment for BBS signatures when the event Coll_2 (which is the analogue of Coll_1) happens.

Reduction $\mathcal{B}_2$ insight. The key observation for the design of reduction $\mathcal{B}_2$ is that, if $\mathcal{B}_2$ extracts a credential from $\mathcal{A}$ which is in the class of one of the credentials given to $\mathcal{A}$, the reduction can find this credential. To this end, it must run the algorithm $\mathsf{VfEquivalence}$ inputting the extracted credential and, one at a time, the credentials $\mathcal{A}$ has issued (and the additional signature used to generate the leakage of $\tilde{X}$). $\mathcal{B}_2$ can additionally learn the scalar that maps one key-signature pair to the other. More specifically, with $\tilde{X}^\dagger, (A^\dagger, e^\dagger), \mathbf{a}^\dagger$ being the extracted credential and $\tilde{X}_i, (A_i, e_i), \mathbf{a}^\star$ the equivalent credential issued by $\tilde{X}_i \in Q_{\mathsf{keys}}$, $\tilde{X}_i \notin pol^\dagger$, $\mathcal{B}_2$ can compute $\epsilon_i \leftarrow \frac{e_i}{e^\dagger}$, the scalar mapping a credential in the other via Adapt[17]. Now, if the reduction knows the secret key sk_i of $\tilde{X}_i = \tilde{G}_2^{sk_i}$, the value ϵ_i such that $\tilde{X}_i = (\tilde{X}^\dagger)^{\epsilon_i}$, the value a such that $\tilde{X}^\dagger = \tilde{X}^\alpha$ then the reduction can compute $\log_{\tilde{G}_2} \tilde{X} = \frac{sk_i}{\alpha \epsilon_i}$ as $\tilde{G}_2^{sk_i} = \tilde{X}_i = (\tilde{X}^\dagger)^{\epsilon_i} = (\tilde{X}^\alpha)^{\epsilon_i}$.

The reduction $\mathcal{B}_2$ simulates the IHAC unforgeability game as follows. Along the description of the reduction, we also ensure carefully that $\mathcal{B}_2$ creates a simulation of the IHAC unforgeability game that is identically distributed to that of $\mathcal{B}_1$. This way, the probability of failure events is identical for both reduction, simplifying the final combination of both reductions.

[17] Note that with overwhelming probability all the e_i in the signatures given by $\mathcal{B}_2$ to $\mathcal{A}$ are non-zero.

Setup phase.

- $\mathcal{B}_2$ receives from $\mathcal{C}$ the following information:

$$(p, \mathbf{e}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, G_1, \tilde{G}_2), (H_1, \ldots, H_m), \tilde{X},$$

  and forwards to $\mathcal{A}$

$$(p, \mathbf{e}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, G_1, \tilde{G}_2), (H_1, \ldots, H_m).$$

- $\mathcal{B}_2$ computes a pair of elements $(U_1, U_2) \in \mathbb{G}_1^2$ s.t. $\mathbf{e}(U_1, \tilde{X}) = \mathbf{e}(U_2, \tilde{G}_2)$. To do so, it samples a random vector of attributes $\mathbf{a}^* \leftarrow\!\!\$\ \mathbb{Z}_p^m$ and sends a sign query to $\mathcal{C}$ for $\mathbf{a}^*$ obtaining (A^*, e^*), valid under $\tilde{X}$. From these values it can compute $U_1 \leftarrow A^*$ and $U_2 \leftarrow C(\mathbf{a}^*)(A^*)^{-e^*}$ which satisfies $\mathbf{e}(U_1, \tilde{X}) = \mathbf{e}(U_2, \tilde{G}_2)$ by the verification equation of BBS signatures. Note that (U_1, U_2) are distributed uniformly random in the set of pairs of elements in $\mathbb{G}_1$ satisfying $\mathbf{e}(U_1, \tilde{X}) = \mathbf{e}(U_2, \tilde{G}_2)$.
  Note that $\mathcal{B}_1$ performs the same operation.

$\mathcal{O}_{\mathsf{RO}}$: Upon reception of a query for m to $\mathcal{O}_{\mathsf{RO}}$, if the input was already added to Q_{RO}, return $Q_{\mathsf{RO}}[m]$, otherwise, sample uniformly at random $d \leftarrow\!\!\$\ \mathbb{Z}_p$, set $Q_{\mathsf{RO}}[m] \leftarrow d$ and return d.

$\mathcal{O}_{\mathsf{KGen}}$: When $\mathcal{A}$ sends the i-th query for a new issuer public key, sample $u_i \leftarrow\!\!\$\ \{0, 1\}$ and $\alpha_i \leftarrow\!\!\$\ \mathbb{Z}_p^*$ and set
- if $u_i = 1$, $\tilde{X}_i \leftarrow \tilde{X}^{\alpha_i}$.
- If $u_i = 0$, $\tilde{X}_i \leftarrow \tilde{G}_2^{\alpha_i}$.

  The public key is added to Q_{keys}. All such public keys are distributed uniformly at random, as for $\mathcal{B}_1$.

$\mathcal{O}_{\mathsf{Corrupt}}$: When $\mathcal{A}$ asks for the issuance and the corruption of a credential from $\tilde{X}_i$ for $\mathbf{a}$, we consider two cases.
- If $u_i = 1$, send a sign query to the BBS signature unforgeability game for $\mathbf{a}$, receive (A, e), and update the signature computing $(\tilde{X}_i, (A', e')) \leftarrow \mathsf{Adapt}(\tilde{X}, (A, e), \alpha_i)$. Return (A', e').
- If $u_i = 0$, sign $\mathbf{a}$ with the secret key α_i corresponding to $\tilde{X}_i$.

  The credential is added to Q_{cor}. In both cases the value e of the signature is sampled uniformly at random either by the BBS unforgeability game or by $\mathcal{B}_2$ and A is uniquely determined, therefore the signatures have the right distribution, as for $\mathcal{B}_1$.

$\mathcal{O}_{\mathsf{SetPolicy}}$: Upon the i-th query for the generation of a policy public key for pol_i, execute $\mathsf{SetPolicy}(pol_i)$ honestly and return the generated policy key. The policy public key is then added to Q_{pols}. The generated policy keys are trivially well-distributed, as for $\mathcal{B}_1$.

$\mathcal{O}_{\mathsf{Present}}$: Upon a query for the issuance and the presentation under a policy (pol, pk_{pol}) of the i_{cred}-th credential generated by $\mathcal{O}_{\mathsf{Issue}}$ for the attribute indices in $\mathcal{D}$ and nonce $nonce$, we verify pk_{pol} and, if it is not valid, returns $\perp$. Otherwise, simulate the presentation as the reduction $\mathcal{B}_1$ does, failing with probability $\leq \frac{q_H + q_P}{p^2}$. If this is the case, $\mathcal{B}_2$ aborts. Else return a presentation $(\tilde{X}^\star, \tilde{T}^\star, \pi^\star)$, which is indistinguishable from a real presentation and is added to Q_{pres}.

We define the event '$\mathcal{B}_2$ $\mathsf{simulates}$' as the event that $\mathcal{B}_2$ does not abort during any of the presentation queries, which happens with the same overwhelming probability as '$\mathcal{B}_1$ $\mathsf{simulates}$', namely $\Pr[\mathcal{B}_2 \text{ } \mathsf{simulates}] \geq 1 - \frac{(q_H+q_P)^2}{p^2}$.

Forgery Phase. Eventually, the adversary produces a forgery attempt

$$(pol^\dagger, pk_{pol^\dagger}, \mathcal{D}^\dagger, (a_i^\dagger)_{i\in\mathcal{D}^\dagger}, nonce^\dagger, (\tilde{X}^\dagger, \tilde{T}^\dagger, \pi^\dagger)).$$

The reduction can extract a valid BBS credential for the presented attributes valid under the public key $\tilde{X}^\dagger$ (as $\mathcal{B}_1$ does), rewinding $\mathcal{A}$, and we call '$\mathcal{B}_2$ $\mathsf{extracts}$' the event that $\mathcal{B}_2$ extracts such a credential. Note that $\mathcal{B}_2$ receives a valid forgery from $\mathcal{A}$ with probability

$$\begin{aligned}\epsilon_2 = \Pr\left[\mathcal{A}^{\mathcal{B}_2}\mathsf{wins} \wedge \mathcal{B}_2 \text{ } \mathsf{simulates}\right] &= \Pr\left[\mathcal{A}^{\mathcal{B}_2}\mathsf{wins} | \mathcal{B}_2 \text{ } \mathsf{simulates}\right] \Pr[\mathcal{B}_2 \text{ } \mathsf{simulates}] \\ &\geq \mathsf{Adv}^{\mathsf{UNF}}_{\mathcal{A},\mathsf{IHBBS}_\tau} \cdot \left(1 - \frac{(q_H+q_P)^2}{p^2}\right),\end{aligned}$$

then $\mathcal{B}_2$ can extract such a credential with probability

$$\begin{aligned}\Pr[\mathcal{B}_2 \text{ } \mathsf{extracts}] \geq \frac{\epsilon_2^2}{q_H} - \frac{\epsilon_2}{p} \geq &\left(\mathsf{Adv}^{\mathsf{UNF}}_{\mathcal{A},\mathsf{IHBBS}_\tau} \cdot \left(1 - \frac{(q_H+q_P)^2}{p^2}\right)\right)^2 / q_H \\ &- \left(\mathsf{Adv}^{\mathsf{UNF}}_{\mathcal{A},\mathsf{IHBBS}_\tau} \cdot \left(1 - \frac{(q_H+q_P)^2}{p^2}\right)\right) / p.\end{aligned}$$

by rewinding the adversary. We can rewrite the inequality above as

$$\Pr[\mathcal{B}_2 \text{ } \mathsf{extracts}] \geq \frac{\left(\mathsf{Adv}^{\mathsf{UNF}}_{\mathcal{A},\mathsf{IHBBS}_\tau}\right)^2}{q_H} + \mathsf{negl}(\lambda). \tag{5}$$

where the negligible terms are collected in the function $\mathsf{negl}(\lambda)$ which is the same as for the reduction $\mathcal{B}_1$.

If $\neg\mathcal{B}_2$ extracts happens, $\mathcal{B}_2$ aborts.

Let $\mathbf{a}^\dagger, (A^\dagger, e^\dagger)$ be the credential extracted by $\mathcal{B}_2$ valid under $\tilde{X}^\dagger$, and let 'Coll_2' be the event defined as for $\mathcal{B}_1$. In particular, define 'Coll_2' as the event that:

- the extracted credential $((A^\dagger, e^\dagger), \tilde{X}^\dagger) \in [(A, e), \tilde{X}_j]$ for some credential (A, e) on the same attributes $\mathbf{a}^\dagger$ issued to $\mathcal{A}$ on behalf of $\tilde{X}_j$ over the experiment execution.
- $\mathbf{a}^* = \mathbf{a}^\dagger$ and $((A^\dagger, e^\dagger), \tilde{X}^\dagger) \in [(A*, e*), \tilde{X}]$, where $\mathbf{a}^*$ are the random attributes queried to $\mathcal{C}$ (receiving the signature $(A*, e*)$) to generate U_1, U_2.

Note that the reduction can check it using the efficient algorithm $\mathsf{VfEquivalence}$ with input $(\tilde{X}^\dagger, (A^\dagger, e^\dagger))$, the extracted credentials issued to $\mathcal{A}$, and the random credential (A^*, e^*) issued by $\tilde{X}$, for $\mathbf{a}^*$.

If $\neg\mathsf{Coll}_2$ happens, $\mathcal{B}_2$ aborts.

Let (A, e) be such a signature issued by $\tilde{X}_j$ for the same extracted attributes $\mathbf{a}\dagger$ such that $((A^\dagger, e^\dagger), \tilde{X}^\dagger) \in [(A, e), \tilde{X}_j]$. Then we define $\epsilon \leftarrow \frac{e}{e^\dagger}$, and observe that $\tilde{X}_j = (\tilde{X}^\dagger)^\epsilon$. Also, since the adversary is algebraic, it provides $\mathcal{B}_2$ with a representation of $\tilde{X}^\star$ with respect to the elements in $\mathbb{G}_2$ it has previously seen. As for $\mathcal{B}_1$, also $\mathcal{B}_2$ knows the representation of every element in $\mathbb{G}_2$ given to $\mathcal{A}$ w.r.t. $\tilde{X}$ and $\tilde{G}_2$: $\tilde{X}^\dagger = \tilde{X}^a \tilde{G}_2^b$. This means that $\mathcal{B}_2$ can compute a representation of $\tilde{X}_j = \tilde{X}^{\epsilon a} \tilde{G}_2^{\epsilon b}$.

Now we observe that whenever $b = 0$ and also the bit $u_j = 0$ (sampled while $\mathcal{B}_2$ was generating $\tilde{X}_j$) $\mathcal{B}_2$ knows $\tilde{G}_2^{\alpha_j} = \tilde{X}^{\epsilon a}$, i.e. $x = \frac{\alpha_j}{\epsilon a}$, then $\mathcal{B}_2$ can trivially win the strong unforgeability experiment for BBS signature as it has extracted the secret counterpart of $\tilde{X}$.

Similarly, whenever $b \neq 0$ and the bit $u_j = 1$, (sampled while $\mathcal{B}_2$ was generating $\tilde{X}_j$), $\mathcal{B}_2$ knows $\tilde{X}_2^{\alpha_j} = \tilde{X}^{\epsilon a} \tilde{G}_2^{\epsilon b}$, i.e. $(\alpha_j - \epsilon a)x = \epsilon b \neq 0$, i.e., $(\alpha_j - \epsilon a) \neq 0$ and most importantly $x = \frac{\epsilon b}{\alpha_j - \epsilon a}$. Then $\mathcal{B}_2$ can trivially win the strong unforgeability experiment for BBS signature as it has extracted the secret counterpart of $\tilde{X}$. We include in this case of the analysis also the event that the credential that collides is (A^*, e^*) issued by $\tilde{X}$ at the beginning of the experiment to generate the leakage for $\tilde{X}$ (i.e. $\tilde{X}_j = \tilde{X}$, and $\alpha_j = 1$).

Note that the value of the bit u_j is independent of the adversary's view, so

$$
\begin{aligned}
&\Pr[\mathcal{B}_2 \text{ wins}] \geq \\
&\Pr[\mathcal{B}_2 \text{ extracts} \wedge \mathsf{Coll}_2 \wedge b = 0 \wedge u_j = 0] + \Pr[\mathcal{B}_2 \text{ extracts} \wedge \mathsf{Coll}_2 \wedge b \neq 0 \wedge u_j = 1] = \\
&\frac{1}{2}\left(\Pr[\mathcal{B}_2 \text{ extracts} \wedge \mathsf{Coll}_2 \wedge b = 0] + \Pr[\mathcal{B}_2 \text{ extracts} \wedge \mathsf{Coll}_2 \wedge b \neq 0]\right) = \\
&\frac{1}{2}\left(\Pr[\mathcal{B}_2 \text{ extracts} \wedge \mathsf{Coll}_2]\right) = \frac{1}{2}\Pr[\mathsf{Coll}_2 | \mathcal{B}_2 \text{ extracts}] \Pr[\mathcal{B}_2 \text{ extracts}]
\end{aligned}
$$

so, by equation Equation (5) the following holds

$$
\Pr[\mathcal{B}_2 \text{ wins}] \geq \frac{1}{2}\Pr[\mathsf{Coll}_2 | \mathcal{B}_2 \text{ extracts}] \cdot \left(\frac{\left(\mathsf{Adv}^{\mathsf{UNF}}_{\mathcal{A},\mathsf{IHBBS}_\tau}\right)^2}{q_H} + \mathsf{negl}(\lambda) \right). \tag{6}
$$

Combining the Reductions $\mathcal{B}_1$ and $\mathcal{B}_2$. We define the reduction $\mathcal{B}$ to the strong unforgeability of BBS signatures as the reduction that samples $i \leftarrow\!\!\$\ \{1, 2\}$ and runs $\mathcal{B}_i$ interacting with the adversary $\mathcal{A}$. Observe:

- Both $\mathcal{B}_1$ and $\mathcal{B}_2$ perfectly simulate the issuer-hiding unforgeability experiment (or abort with identical negligible probability if a presentation query requires overwriting the hash table Q_{RO}), therefore, under $\mathcal{A}$'s view, interactions with $\mathcal{B}_1$ and $\mathcal{B}_2$ are identical, and the probability for either reductions to extract a credential for the randomized public key included in $\mathcal{A}$'s forgeries is the same.

– As the events Coll_1 and Coll_2 are triggered under the same conditions, namely if the extracted credential (uniquely determined by the forgery produced by $\mathcal{A}$) is in the same class of the signatures that $\mathcal{B}_1$ and $\mathcal{B}_2$ have sent to $\mathcal{A}$ as a response of a corrupt query, (or in the same class of $\tilde{X}, (A^*, e^*)$ queried at the beginning of the experiment to generate the randomness associated to the public key $\tilde{X}$), we can state that

$$\Pr[\mathsf{Coll}_1|\mathcal{B}_1 \text{ extracts}] = \Pr[\mathsf{Coll}_2|\mathcal{B}_2 \text{ extracts}]. \tag{7}$$

We can compute the probability of success of $\mathcal{B}$ in winning the strong unforgeability experiment for BBS signatures as follows.

$$\begin{aligned}\Pr[\mathcal{B} \text{ wins}] &= \Pr[\mathcal{B} \text{ wins} \wedge k = 1] + \Pr[\mathcal{B} \text{ wins} \wedge k = 2] \\ &\geq \frac{1}{2}\left(\Pr[\mathcal{B} \text{ wins}|k = 1] + \Pr[\mathcal{B} \text{ wins}|k = 2]\right) \\ &= \frac{1}{2}\left(\Pr[\mathcal{B}_1 \text{ wins}] + \Pr[\mathcal{B}_2 \text{ wins}]\right)\end{aligned}$$

then, by Equation (4) and Equation (6) we obtain

$$\begin{aligned}\Pr[\mathcal{B} \text{ wins}] \geq \frac{1}{2}\Pr[\neg\mathsf{Coll}_1|\mathcal{B}_1 \text{ extracts}] \cdot \left(\frac{\left(\mathsf{Adv}^{\mathsf{UNF}}_{\mathcal{A},\mathsf{IHBBS}_\tau}\right)^2}{q_H} + \mathsf{negl}(\lambda)\right) + \\ \frac{1}{4}\Pr[\mathsf{Coll}_2|\mathcal{B}_2 \text{ extracts}] \cdot \left(\frac{\left(\mathsf{Adv}^{\mathsf{UNF}}_{\mathcal{A},\mathsf{IHBBS}_\tau}\right)^2}{q_H} + \mathsf{negl}(\lambda)\right) \geq \\ \frac{1}{4} \cdot \left(\frac{\left(\mathsf{Adv}^{\mathsf{UNF}}_{\mathcal{A},\mathsf{IHBBS}_\tau}\right)^2}{q_H} + \mathsf{negl}(\lambda)\right) \cdot \\ \cdot \left(\Pr[\neg\mathsf{Coll}_1|\mathcal{B}_1 \text{ extracts}] + \Pr[\mathsf{Coll}_2|\mathcal{B}_2 \text{ extracts}]\right)\end{aligned}$$

but by Equation (7),

$$\begin{aligned}&\Pr[\neg\mathsf{Coll}_1|\mathcal{B}_1 \text{ extracts}] + \Pr[\mathsf{Coll}_2|\mathcal{B}_2 \text{ extracts}] = \\ &\Pr[\neg\mathsf{Coll}_1|\mathcal{B}_1 \text{ extracts}] + \Pr[\mathsf{Coll}_1|\mathcal{B}_1 \text{ extracts}] = 1,\end{aligned}$$

which implies

$$\Pr[\mathcal{B} \text{ wins}] \geq \frac{1}{4} \cdot \left(\frac{\left(\mathsf{Adv}^{\mathsf{UNF}}_{\mathcal{A},\mathsf{IHBBS}_\tau}\right)^2}{q_H} + \mathsf{negl}(\lambda)\right) \tag{8}$$

D Quadratic Equations Modulo A Prime

The following lemma supports reduction $\mathcal{B}_1$ in the proof of Theorem 2.

Lemma 1. *For any $a, b, c \in \mathbb{Z}_p$, $a \neq 0 \mod p$, the quadratic equation*

$$ax^2 + bx + c = 0 \mod p$$

has two solutions whenever $b^2 - 4ac$ is a quadratic residue, it has a solution when $b^2 - 4ac = 0$, otherwise it has no solutions.

Proof. We can rewrite $ax^2 + bx + c = 0$ as $4a^2x^2 + 4abx + 4ac + b^2 - b^2 = 0$, therefore as

$$(2ax + b)^2 = b^2 - 4ac \mod p$$

If $b^2 - 4ac$ is a quadratic residue, which can be checked using Euler's Criterion verifying if $(b^2 - 4ac)^{\frac{p-1}{2}} = 1$, we can compute the square root s of this value[18]. At this point, we can write $(2ax + b)^2 - s^2 = 0 \mod p$ which implies

$$(2ax + b - s)(2ax + b + s) = 0 \mod p,$$

which holds only if one of the two factors is 0 an which implies that there are two solutions to the equation which are $x = \frac{s-b}{2a} \mod p$ and $x = \frac{-s-b}{2a} \mod p$. If $b^2 - 4ac = 0$, $(2ax + b)^2 = 0 \mod p$ so $x = -b(2a)^{-1} \mod p$ is the only solution.

E $\mathsf{IHBBS}_{\tau 2}$: Turning IHBBS_{τ} into a Type-2 IHAC Scheme

Our IHBBS_{τ} scheme presented in Section 5 can easily be adapted into a Type-2 scheme if a verifier prefers not to trust the TTP, to generate policies and policy public keys itself and send them directly to the user. In this case, we can further apply the following optimizations to algorithms $\mathsf{SetPolicy}, \mathsf{VfPolicy}, \mathsf{Present}$ to obtain a Type-2 scheme we name $\mathsf{IHBBS}_{\tau 2}$:

SetPolicy: as the verifier will be the one generating the policy key, it can just send to the user the tags of the public keys in the policy without the public key of the tag, together with a NIZKP of $\pi \leftarrow\!\$\ \mathsf{NIZKP}\{\alpha \in \mathbb{Z}_p : \tilde{T}_i = \tilde{X}_i^{\alpha}, \forall \tilde{X}_{i \in pol}\} \in \mathbb{Z}_p^2$.

VfPolicy: verification just checks π.

Present: the user must compute the tag and use it to compute the challenge of the NIZKP of knowledge of a credential for the randomized public key $\tilde{X}^{\star}$, but they can avoid sending it as part of the presentation as the verifier knows the tag secret key and can recompute it. During verification, the verifier is assured that the user knew the associated tag as they will use it as an input to the random oracle for the computation of the challenge.

Our scheme applying these optimization yields a mild efficiency improvement over [KS25]. We also expect that our AGM-based security analysis carries over, thus improving also the required idealized model from GGM used in [KS25]. The

[18] This can be done using a direct formula if $p = 3 \mod 4$ which is the case of BLS12-381, or Tonelli-Shanks if $p = 1 \mod 4$.

comparison is shown in Table 3, considering in the communication overhead the policy public key that the verifier must send to the user, and in the computational complexity the cost of verifying the policy public key. We refer to Section 6 for the description of the methodology used for our comparison. The concrete numbers have been collected using the same setting used for the data in Table 2

Table 3. Performance comparison between Type-2 constructions $\mathsf{IHBBS}_{\tau 2}$ and [KS25]. The computational overhead (for the prover) is measured in terms of exponentiations $\mathsf{e}_{\mathbb{G}}$, where the cost of verifying a policy key is shown in gray. The communication overhead considers the policy public key in gray (not including the issuer keys in *pol*), and the proof size.

Scheme - model	**comput. OH**	**commun. OH**
[KS25] - GGM	$(3+2\lvert pol\rvert)\mathsf{e}_{\mathbb{G}_1} + 1\mathsf{e}_{\mathbb{G}_2}$	$(\lvert pol\rvert+1)\mathbb{G}_2 + 3\mathbb{Z}_p + 1\mathbb{G}_2$
concrete numbers	22 ms	$96\lvert pol\rvert + 288$ B
$\mathsf{IHBBS}_{\tau 2}$ - AGM	$(1+2\lvert pol\rvert)\mathsf{e}_{\mathbb{G}_1} + 1\mathsf{e}_{\mathbb{G}_1} + 2\mathsf{e}_{\mathbb{G}_2}$	$(\lvert pol\rvert)\mathbb{G}_2 + 2\mathbb{Z}_p + 1\mathbb{G}_2$
concrete numbers	22 ms	$96\lvert pol\rvert + 160$B