Registered Hierarchy-Based Signatures from Bilinear Groups

Jiamian Yan[1], Hong Yang[1], Ziyan Yuan[1]
Borui Chen[1], Liang Zhang[2], and Haibin Kan[1]

[1] Fudan University
25113050230@m.fudan.edu.cn, hongyang1358@gmail.com, zyyuan24@m.fudan.edu.cn
berrychen0w0@gmail.com, hbkan@fudan.edu.cn
[2] Hong Kong University of Science and Technology
scottzhang@ust.hk

Abstract. Registration-based signatures (RBS) remove the trusted key generation center (KGC) of identity-based signatures, resolving the global key escrow problem: users generate their own secret state and publish public registration material, which a curator holding no cryptographic secrets aggregates into compact public parameters. Yet many scenarios need both registration to avoid key escrow across organizations and delegation for trust control within an organization. Delegated Reg-FE (ASIACRYPT'2025) realizes this hybrid trust model for encryption, but only for 2-level hierarchies.

In this work, we introduce registered hierarchy-based signatures (RHBS), which combines registration of root users with hierarchical delegation to non-root users in L-level hierarchies. We construct RHBS from BBG-based hierarchical signatures using three techniques: *cross-term cancellation* to remove the KGC, plus *affine share compression* and *power-of-two groups with version tag* to obtain the properties below.

Our construction satisfies standard compactness, efficiency, and unbounded registration. It further achieves *extra compactness* for the common reference string, each root user's public registration material, and signatures. It also supports non-root key updates without re-delegation and public signature updates without re-signing when new registrations change the public parameters. We prove selective-identity unforgeability under the computational asymmetric $(L+1)$-BDHE assumption in the random-oracle model, even if all other root users are malicious. We implement our construction and compare its computational and storage costs with related schemes.

1 Introduction

Cross-organization collaboration lets users from different organizations share data and resources [1]. It involves two trust domains. *Across organizations*, an entity partially trusts and authorizes external partners [2]. No one is trusted by all. *Within an organization*, members trust their root and act under its delegated authority [3, 4]. Some collaborations need both. In WAVE [5], a property manager delegates to building managers, who authorize outside contractors with limited access. In multi-agent collaboration [6], each agent acts under an authority and works with external peers.

Such collaboration can be achieved by two cryptography mechanisms: Digital signatures authorize actions or access requests, while key delegation derives restricted signing keys for descendant identities. Hierarchical identity-based signatures (HIBS) [7] combine both. A user derives signing keys for descendant identities, and verifiers check signatures using only the signer's identity and public parameters. HIBS has been used for TLS delegation to CDN servers [8], IoT data publication [9], and non-attributable email [10].

However, HIBS depends on a key generation center (KGC) that can issue signing keys for every identity. This suffices within an organization, where members trust their root, but fails across organizations, because a common root would cause global key escrow. Running one HIBS instance per organization, N in total, avoids the key escrow but raises the problem of how to manage the public parameters. A full authenticated directory stores N entries. A verifier must obtain the signing organization's entry. A Merkle accumulator [11] keeps the public parameters compact, but adds an $O(\log N)$-size membership proof to every signature.

Registration-based cryptography [12] removes the KGC. Each user generates its own secret state and publishes public registration material. A curator holding no secrets aggregates those materials into compact public parameters. A new registration changes the parameters, so affected users need to update signing keys with helper keys from the curator. Registration-based signatures (RBS) [13] and registered attribute-based signatures (RABS) [14, 15] bring this paradigm to signatures without hierarchical delegation. Delegated Reg-FE [16] combines public registration of authorities and key issuance within their domains in a 2-level hierarchy. To our knowledge, no existing scheme combines registration with L-level delegation for signatures.

Our RHBS Scheme. We introduce *registered hierarchy-based signatures* (RHBS) for L-level identity hierarchies: organizations register as independent root users, and users within each organization receive signing keys through hierarchical delegation. Beyond the standard compactness (for public parameters and helper keys) and efficiency expected of registration-based cryptography, we seek unbounded registration, and three further properties as follows.

First, *extra compactness*: the common reference string (CRS) and each root user's public registration material are independent of the number of registered root users, and signatures are also compact. Compressing the CRS and the public registration materials is an important goal in registered attribute-based encryption (RABE) [17, 18].

Second, *non-root key updates without re-delegation*: a new registration changes a root's helper key. In RBS, a user updates its signing key from its secret state and the curator's latest helper key. In RHBS, root users keep this property. We require non-root users to update their signing keys locally (even after missing earlier updates), without ancestors re-delegating keys to every descendant. This matters for systems with many non-root users or frequently offline roots such as IoT deployments.

Third, *public signature updates without re-signing*: a new registration changes the public parameters used in verification, so earlier signatures may no longer verify. Naive solutions are to store all past public parameters or to ask signers to re-sign. We require that a user can update an existing signature with the public auxiliary information aux, without re-signing. Each signature needs at most $O(\log N)$ updates. This is practical for audit and authorization systems, whose signatures must be verifiable long after issuance.

1.1 Our Contributions

Formal Definition of RHBS. We formalize RHBS, combining registration of root users with hierarchical key delegation to non-root users. The definition covers completeness, efficiency, and standard compactness for public parameters pp and root helper keys hk. Extra compactness bounds the sizes of the CRS crs, each root user's public registration material pub_i, and signatures σ. We also require non-root key updates without re-delegation and public signature updates without re-signing. We define selective-identity unforgeability with an adaptively chosen root identifier (*semi-adaptive* unforgeability). Only the non-root coordinates are fixed in advance. This implies selective-identity unforgeability even if all other roots are malicious.

A Construction from Bilinear Pairings. We construct RHBS from hierarchical signatures obtained by applying the Naor transformation [43] to the Boneh–Boyen–Goh (BBG) hierarchical identity-based encryption scheme [42]. We use the cross-term cancellation approach of prior work to distribute key generation across root users. Our main technical contribution is *affine share compression* for BBG signing-key shares. Each root uses one secret offset to relate the randomizers of its shares for other roots. This lets the curator derive those shares from one public key share containing $O(L)$ group elements, independently of N. We also adapt prior power-of-two grouping to BBG by using the group level to tag each root's helper key. The tag identifies the helper contribution to replace when updating a delegated signing key or an existing signature. Our construction satisfies all the properties defined above, including semi-adaptive unforgeability under the computational asymmetric bilinear Diffie–Hellman exponent (BDHE) assumption with parameter $L + 1$.

Implementation and Evaluation. We implement RHBS in Go over BLS12-381 and use the fast Fourier transform (FFT) to accelerate helper-key aggregation. Over 65,536 accepted registrations, Reg takes 20–32 ms per root on average across the tested configurations. After the final merge, RHBS uses 128 B of public parameters, compared with 6 MiB for a full HIBS directory. At the same scale, an RHBS signature occupies 145 B, compared with 752 B for HIBS with a Merkle directory.

1.2 Related Work

Registration-based encryption. Registration-based cryptography originated with Garg et al.'s registration-based encryption (RBE) [12], removing the KGC from identity-based encryption. Instead, they introduce a curator, whose job is only to aggregate the public keys of all the registered users and update the short public parameter whenever a new user joins the system. Subsequent work gave constructions under standard assumptions [19], smaller ciphertexts [20], and pairing-based [21,22] and lattice-based schemes [23,24]. Other work studies verifiable registration [25], RBE in the plain model [26],black-box lower bounds [27], and other improvements [28, 29, 30]. This paradigm also extends to RABE [31] and functional encryption.

Delegated Reg-FE [16] uses a 2-level trust model. Independent authorities register their master public keys with a secret-free curator in the public domain and issue decryption keys within their private domains. Each call to Key requires the authority's master secret msk and issues a functional decryption key sk_f. The recipient can decrypt with this key but has no interface for further delegation. Thus, repeated calls by the authority issue keys at the same level and do not directly yield L-level delegation.

Registration-based signatures. RBS applies the RBE paradigm to remove the trusted KGC from identity-based signatures. Existing pairing-based RBS schemes [13] partition users into fixed-capacity vector-commitment blocks (with an optional token-based signature refresh).

Registered attribute-based signatures. In RABS, users can sign under policies satisfied by their registered attributes. The primitive was formalized with pairing-based constructions via predicate encodings [14]. Subsequent schemes hide attributes at registration via authority-certified commitments [33] or support circuit policies via two-layer proofs verified against historical public parameters [15]. However, RABS targets policy-based signing among directly registered users rather than intra-organization key delegation.

Hierarchical identity-based signatures and distributed variants. HIBS schemes [7] support multi-level key delegation along identity paths. However, a trusted root KGC holds the master secret and can forge signatures for any descendant identity. Distributed variants such as HIBEChain's DHIBS [34] split this root secret across multiple nodes using verifiable secret sharing. This approach still relies on a threshold honesty assumption: enough colluding nodes can reconstruct the master secret and forge signatures for any user. It also requires online multiparty interaction during key generation.

2 Preliminaries

Notation. Let λ denote the security parameter, $[n] = \{1, \ldots, n\}$, and log denote the base-2 logarithm. For a prime p, let $\mathbb{Z}_p$ denote the field of integers modulo p, with $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\}$. For a finite set S, $x \xleftarrow{\$} S$ denotes uniform sampling from S. We write PPT for probabilistic polynomial time, $\mathsf{negl}(\lambda)$ for a negligible function in λ, and $\mathsf{poly}(\cdot)$ for a polynomial in its arguments and λ (omitting λ).

2.1 Bilinear Pairings and the BDHE Assumption

On input 1^λ, a PPT algorithm $\mathsf{BPGen}(1^\lambda)$ outputs bilinear-group parameters $\mathcal{BP} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ for a $\Theta(\lambda)$-bit prime p. Here $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$, and $\mathbb{G}_T$ are all multiplicative cyclic groups of order p, with identity elements $1_{\mathbb{G}_1}$, $1_{\mathbb{G}_2}$, and $1_{\mathbb{G}_T}$. The map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ satisfies three properties:

- **Bilinearity:** $e(x^a, y^b) = e(x, y)^{ab}$ for all $x \in \mathbb{G}_1$, $y \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}_p$.
- **Non-degeneracy:** $e(g_1, g_2) \neq 1_{\mathbb{G}_T}$.
- **Computability:** There exists an efficient algorithm to compute $e(x, y)$ for all $x \in \mathbb{G}_1$, $y \in \mathbb{G}_2$.

We use Type-3 asymmetric bilinear groups [35]: $\mathbb{G}_1 \neq \mathbb{G}_2$, and no efficient isomorphism is assumed to exist between $\mathbb{G}_1$ and $\mathbb{G}_2$ in either direction.

Computational asymmetric ℓ-BDHE assumption.

We use the computational asymmetric form of the bilinear Diffie–Hellman exponent (BDHE) assumption [36]. For an integer $\ell \geq 1$, Sample $\mathcal{BP} \leftarrow \mathsf{BPGen}(1^\lambda)$ and $a, s \xleftarrow{\$} \mathbb{Z}_p$. Given the challenge tuple

$$D = \Big(\mathcal{BP},\ S_1 = g_1^s,\ S_2 = g_2^s,\ \big\{P_i = g_1^{a^i}\big\}_{i \in [2\ell] \setminus \{\ell+1\}},\ \big\{\widehat{P}_i = g_2^{a^i}\big\}_{i \in [\ell]}\Big),$$

every PPT adversary $\mathcal{A}$ satisfies

$$\mathsf{Adv}_{\mathcal{A}}^{\ell\text{-BDHE}}(\lambda) := \Pr\big[\mathcal{A}(D) = e(g_1, g_2)^{s a^{\ell+1}}\big] \leq \mathsf{negl}(\lambda).$$

2.2 Schnorr Proof of Knowledge

To prove knowledge of a discrete logarithm $\alpha \in \mathbb{Z}_p$ satisfying $x = g_2^\alpha$ in $\mathbb{G}_2$, we use the Fiat–Shamir Schnorr non-interactive zero-knowledge (NIZK) proof of knowledge [37, 38, 39]. Let $H_{\mathrm{NIZK}} : \{0,1\}^* \to \mathbb{Z}_p$ be a hash function applied to encoding of tuples $(g_2, x, T, \mathsf{ctx})$, where $\mathsf{ctx} \in \{0,1\}^*$ is a public context string.

- $\mathsf{NIZK.Prove}((g_2, x), \alpha, \mathsf{ctx}) \to \pi$:
  Sample $w \xleftarrow{\$} \mathbb{Z}_p$ and set $T = g_2^w$. Compute $c = H_{\mathrm{NIZK}}(g_2, x, T, \mathsf{ctx})$, $z = w + c\alpha$. Output $\pi = (T, z)$.
- $\mathsf{NIZK.Verify}((g_2, x), \pi, \mathsf{ctx}) \to \{0,1\}$:
  Parse π as (T, z). Compute $c = H_{\mathrm{NIZK}}(g_2, x, T, \mathsf{ctx})$. Output 1 if $g_2^z = T x^c$ and 0 otherwise.

When H_{NIZK} is modeled as random oracle, this construction satisfies perfect completeness, computational knowledge soundness [40], and zero-knowledge.

3 Formal Definition of RHBS

We first describe the identities and delegation relation of RHBS, then define its syntax and required properties.

3.1 Identities and Delegation

RHBS uses hierarchical key delegation with wildcards [41]. Let $\mathcal{D}$ be the domain of identity coordinates and $L \geq 1$ the maximum hierarchy depth. An identity is a vector $\mathsf{id} = (I_1, \ldots, I_L) \in \mathcal{D} \times (\mathcal{D} \cup \{\star\})^{L-1}$, where $\star \notin \mathcal{D}$ denotes a wildcard. We call the first coordinate I_1, which is never a wildcard, the *root identifier*. Root user $\mathcal{U}_j$ with root identifier rid_j has root identity $(\mathsf{rid}_j, \star, \ldots, \star)$.

For an identity id, let $\mathsf{Fix}(\mathsf{id})$ and $\mathsf{Free}(\mathsf{id})$ respectively denote the sets of fixed and wildcard coordinate indices:

$$\mathsf{Fix}(\mathsf{id}) = \{q \in [L] : I_q \neq \star\}, \quad \mathsf{Free}(\mathsf{id}) = [L] \setminus \mathsf{Fix}(\mathsf{id}).$$

For identities $\mathsf{id} = (I_1, \ldots, I_L)$ and $\mathsf{id}' = (I'_1, \ldots, I'_L)$, define the delegation relation by

$$\mathsf{id} \preceq \mathsf{id}' \iff I'_q = I_q \quad \forall q \in \mathsf{Fix}(\mathsf{id}).$$

A *non-root identity* satisfies $|\mathsf{Fix}(\mathsf{id})| \geq 2$. When $\mathsf{id} \preceq \mathsf{id}'$, id is an *ancestor* of id'; every identity is an ancestor of itself. A signing-key holder for id can delegate a signing key for any id' satisfying $\mathsf{id} \preceq \mathsf{id}'$ by fixing any subset of wildcard coordinates. Traditional hierarchical identity-based cryptography [7, 42] uses prefix-based delegation, a special case restricting $\mathsf{Fix}(\mathsf{id})$ to prefixes of $[L]$.

Table 1. Notation for RHBS

Symbol	Meaning
crs	Common reference string
$\mathsf{pp}, \mathsf{aux}$	Public parameters and auxiliary information maintained by the curator
$\mathsf{pub}_i, \mathsf{priv}_i$	Public registration material and secret state of root $\mathcal{U}_i$
$\mathsf{hk}_{\mathsf{rid}_i}$	Public helper key for root identifier rid_i
$\mathsf{sk}_{\mathsf{id}}$	Signing key for identity id
m, σ	Message and signature on it
ver	Version tag of a root, determined by pp

3.2 Syntax

RHBS involves three roles: a curator, root users, and non-root users. Our syntax retains the core registration and signing algorithms of RBS and RABS. Besides, we use an explicit Key algorithm to derive and update a root user's signing key, and Delegate for hierarchical key delegation, inspired by delegated Reg-FE [16]. We further introduce KeyUpd for key updates by non-root users, and SigUpd for signature updates without re-signing.

Table 1 summarizes our notation. We write $(\mathsf{pub}, \mathsf{priv})$ for the outputs of Gen, whereas prior works use $(\mathsf{pk}, \mathsf{sk})$ [13, 14] or $(\mathsf{mpk}, \mathsf{msk})$ [16]. Here pub serves only as registration input to Reg, and priv is not a standalone signing key. For instance, RABS passes priv and hk from Upd directly into Sign [14]. We abstract this step into Key, combining priv and hk into an **explicit root signing key** sk for signing and hierarchical delegation.

Definition 3.1 (Registered Hierarchy-Based Signatures). *An RHBS scheme consists of ten algorithms (*Setup, Gen, Reg, Upd, Key, Delegate, KeyUpd, Sign, SigUpd, Verify*) and three public validation algorithms (*CheckSetup, CheckGen, CheckUpd*).*

- $\mathsf{Setup}(1^\lambda, L) \to \mathsf{crs}$*: It takes as input the security parameter* 1^λ *and maximum hierarchy depth* L*, and outputs* crs*. All other algorithms take* crs *as an implicit input.*
- $\mathsf{Gen}(\mathsf{pp}) \to (\mathsf{rid}_i; \mathsf{pub}_i, \mathsf{priv}_i)$*: It takes as input* pp*, selects an unregistered* rid_i*, and outputs* $(\mathsf{rid}_i; \mathsf{pub}_i, \mathsf{priv}_i)$*.*
- $\mathsf{Reg}(\mathsf{rid}_j, \mathsf{pub}_j, \mathsf{pp}, \mathsf{aux}) \to (\mathsf{pp}', \mathsf{aux}')$*: It takes as input an accepted request* $(\mathsf{rid}_i, \mathsf{pub}_i)$ *and* $(\mathsf{pp}, \mathsf{aux})$*, registers* rid_i*, and deterministically outputs* $(\mathsf{pp}', \mathsf{aux}')$*.*
- $\mathsf{Upd}(\mathsf{aux}, \mathsf{rid}_i) \to \mathsf{hk}_{\mathsf{rid}_i}$*: It takes as input* aux *and* rid_i*, and deterministically outputs* $\mathsf{hk}_{\mathsf{rid}_i}$ *with* rid_i*'s tag* ver*.*

- $\mathsf{Key}(\mathsf{rid}_i, \mathsf{priv}_i, \mathsf{hk}_{\mathsf{rid}_i}) \to \mathsf{sk}_{\mathsf{rid}_i}$*: The root user takes as input* rid_i*,* priv_i*, and* $\mathsf{hk}_{\mathsf{rid}_i}$*, and outputs* $\mathsf{sk}_{\mathsf{rid}_i}$ *for* $(\mathsf{rid}_i, \star, \ldots, \star)$ *with the helper's version tag.*
- $\mathsf{Delegate}(\mathsf{id}, \mathsf{sk}_{\mathsf{id}}, \mathsf{id}') \to \mathsf{sk}_{\mathsf{id}'}$*: It takes as input* id*,* $\mathsf{sk}_{\mathsf{id}}$ *(with* ver*), and a non-root* id' *satisfying* $\mathsf{id} \preceq \mathsf{id}'$*, and outputs* $\mathsf{sk}_{\mathsf{id}'}$ *with* ver*.*
- $\mathsf{KeyUpd}(\mathsf{id}, \mathsf{hk}_{\mathsf{rid}_i}, \mathsf{sk}_{\mathsf{id}}) \to \mathsf{sk}'_{\mathsf{id}}$*: It takes as input a non-root* id *with root identifier* rid_i*,* $\mathsf{hk}_{\mathsf{rid}_i}$ *(with target* ver'*), and* $\mathsf{sk}_{\mathsf{id}}$ *(with source* ver*), and outputs* $\mathsf{sk}'_{\mathsf{id}}$ *with* ver'*.*
- $\mathsf{Sign}(\mathsf{id}, \mathsf{sk}_{\mathsf{id}}, m) \to \sigma$*: It takes as input* id*,* $\mathsf{sk}_{\mathsf{id}}$ *(with* ver*), message* m*, and outputs a signature* σ *with* ver*.*
- $\mathsf{SigUpd}(\mathsf{aux}, \mathsf{id}, m, \sigma, \mathsf{ver}') \to \sigma'$ *or* $\perp$*: It takes as input* aux*,* id *(under* rid_i*),* m*,* σ *(with source* ver*), and target* ver'*. It outputs* σ' *on* (id, m) *with* ver'*, or* $\perp$ *if either* ver *or* ver' *is unavailable for* rid_i *in* aux*.*
- $\mathsf{Verify}(\mathsf{id}, \mathsf{m}, \sigma, \mathsf{pp}) \to \{0, 1\}$*: It takes as input* id *with root identifier* rid_i*,* m*,* σ*, and* pp*, and outputs* 1 *if* rid_i *is registered in* pp*,* σ*'s tag equals* rid_i*'s tag in* pp*, and* σ *is valid on* $(\mathsf{id}, \mathsf{m})$ *under* pp*, and* 0 *otherwise.*

The public validation algorithms check crs*, registration requests, and* hk*, respectively:*

- $\mathsf{CheckSetup}(\mathsf{crs}) \to \{0, 1\}$*: It takes as input* crs*, and outputs* 1 *if* crs *is valid and* 0 *otherwise.*
- $\mathsf{CheckGen}(\mathsf{rid}_i, \mathsf{pub}_i, \mathsf{pp}) \to \{0, 1\}$*: It takes as input* rid_i*,* pub_i*, and* pp*. It outputs* 1 *if* rid_i *is available under* pp*, and* pub_i *is valid for* $(\mathsf{rid}_i, \mathsf{pp})$*, and* 0 *otherwise.*
- $\mathsf{CheckUpd}(\mathsf{rid}_i, \mathsf{pub}_i, \mathsf{hk}_{\mathsf{rid}_i}, \mathsf{pp}) \to \{0, 1\}$*: It takes as input* rid_i*,* pub_i*,* $\mathsf{hk}_{\mathsf{rid}_i}$ *(with tag* ver*), and* pp*. It outputs* 1 *if* rid_i *is registered in* pp*,* ver *equals* rid_i*'s version tag, and* $\mathsf{hk}_{\mathsf{rid}_i}$ *is valid for* $(\mathsf{rid}_i, \mathsf{pub}_i, \mathsf{pp})$*, and* 0 *otherwise.*

Workflow of RHBS. System initialization sets $(\mathsf{pp}, \mathsf{aux})$ to the empty state, runs Setup to output crs. A root user $\mathcal{U}_i$ runs Gen, stores priv_i locally, and publishes $(\mathsf{rid}_i, \mathsf{pub}_i)$. The curator checks the request with CheckGen, updates $(\mathsf{pp}, \mathsf{aux})$ deterministically with Reg on acceptance, and returns $\mathsf{hk}_{\mathsf{rid}_i}$ via Upd. Root user $\mathcal{U}_i$ validates $\mathsf{hk}_{\mathsf{rid}_i}$ via CheckUpd, and runs Key to derive or update its $\mathsf{sk}_{\mathsf{rid}_i}$. User holding $\mathsf{sk}_{\mathsf{id}}$ can run Delegate to derive $\mathsf{sk}_{\mathsf{id}'}$ if $\mathsf{id} \preceq \mathsf{id}'$. User signs messages with id and $\mathsf{sk}_{\mathsf{id}}$, while anyone can verify using signer's id and the compact pp (without the potentially large state aux).

Later registrations in Reg may advance rid_i's version tag ver in pp: the curator then supplies the refreshed $\mathsf{hk}_{\mathsf{rid}_i}$ via Upd, $\mathcal{U}_i$ reruns Key, and non-root users under rid_i run KeyUpd locally without re-delegation. To update a prior signature without re-signing, a user runs SigUpd with aux.

Trust model. We assume a publicly readable, append-only bulletin board that gives all users a consistent view of the ordered sequence of accepted registrations $(\mathsf{rid}_i, \mathsf{pub}_i)$. As in registration-based encryption [12], the curator holds no cryptographic secrets. Given this sequence, anyone can reproduce the curator's state and helper keys by running the deterministic algorithms Reg and Upd.

3.3 Required Properties

We define completeness, compactness, efficiency, and unforgeability for RHBS. Let N denote the number of currently registered root users.

Definition 3.2 (Completeness). *An RHBS scheme is complete if the following five conditions hold. In (ii)–(v), fix* crs *with* $\mathsf{CheckSetup}(\mathsf{crs}) = 1$*, and let an honest curator start from the initial state and process* $\mathsf{poly}(\lambda)$ *adaptive (possibly malicious) registration requests via* CheckGen *and* Reg*. Probabilities are over randomness of algorithms:*

(i) Setup correctness. *For* $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, L)$*,*

$$\Pr[\mathsf{CheckSetup}(\mathsf{crs}) = 1] = 1.$$

(ii) Registration correctness. *At any state* $(\mathsf{pp}, \mathsf{aux})$ *permitting registration, for* $(\mathsf{rid}_i; \mathsf{pub}_i, \mathsf{priv}_i) \leftarrow \mathsf{Gen}(\mathsf{pp})$*,*

$$\Pr[\mathsf{CheckGen}(\mathsf{rid}_i, \mathsf{pub}_i, \mathsf{pp}) = 1] \geq 1 - \mathsf{negl}(\lambda).$$

(iii) Helper-key correctness. *For any honestly registered* $(\mathsf{rid}_j, \mathsf{pub}_j)$*, at its registration state and every subsequent state* $(\mathsf{pp}, \mathsf{aux})$ *with* $\mathsf{hk}_{\mathsf{rid}_j} \leftarrow \mathsf{Upd}(\mathsf{aux}, \mathsf{rid}_j)$*,*

$$\Pr[\mathsf{CheckUpd}(\mathsf{rid}_j, \mathsf{pub}_j, \mathsf{hk}_{\mathsf{rid}_j}, \mathsf{pp}) = 1] \geq 1 - \mathsf{negl}(\lambda).$$

(iv) Signature correctness. *Let* $\mathsf{sk_{id}}$ *be derived under an honestly registered* rid_j *via any valid sequence of* Key, Delegate, *and* KeyUpd *calls using helper keys returned by* Upd. *At any state* $(\mathsf{pp}, \mathsf{aux})$ *matching* $\mathsf{sk_{id}}$*'s tag for* rid_j*, every message* m *and* $\sigma \leftarrow \mathsf{Sign}(\mathsf{id}, \mathsf{sk_{id}}, \mathsf{m})$ *satisfy*

$$\Pr[\mathsf{Verify}(\mathsf{id}, \mathsf{m}, \sigma, \mathsf{pp}) = 1] \geq 1 - \mathsf{negl}(\lambda).$$

(v) Signature-update correctness. *Let* $\mathsf{pp}, \mathsf{pp}'$ *be parameters from any two states of the execution assigning tags* $\mathsf{ver}, \mathsf{ver}'$ *(in either order) to an honestly registered* rid_j*, and let* aux *be from any state after both versions appear. For any* id *under* rid_j*, message* m*, and signature* σ*, letting* $\sigma' \leftarrow \mathsf{SigUpd}(\mathsf{aux}, \mathsf{id}, \mathsf{m}, \sigma, \mathsf{ver}')$*,*

$$\Pr\left[\begin{array}{l}\mathsf{Verify}(\mathsf{id}, \mathsf{m}, \sigma, \mathsf{pp}) = 1 \\ \quad \Longrightarrow \big(\sigma' \neq \bot \wedge \\ \qquad \mathsf{Verify}(\mathsf{id}, \mathsf{m}, \sigma', \mathsf{pp}') = 1\big)\end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Prior RBS schemes require only standard compactness; we additionally introduce extra compactness requirements.

Definition 3.3 (Compactness). *Let* $|\cdot|$ *denote bit-length. Each* $\mathsf{hk}_{\mathsf{rid}_i}$, $\mathsf{sk_{id}}$*, and* σ *consists of a body (*$\overline{\mathsf{hk}}_{\mathsf{rid}_i}$, $\overline{\mathsf{sk}}_{\mathsf{id}}$*, and* $\overline{\sigma}$*) and version tag* ver. *An RHBS scheme is* standard-compact *if (i) holds, and* extra-compact *if both (i) and (ii) hold:*

(i) Standard compactness:
- $|\mathsf{ver}| = O(\log \log N)$ *(independent of* λ *and* L*);*
- $|\mathsf{pp}| \leq \mathsf{poly}(L, \log N)$*;*
- $|\overline{\mathsf{hk}}_{\mathsf{rid}_i}| \leq \mathsf{poly}(L, \log N)$.

(ii) Extra compactness:
- $|\mathsf{crs}| \leq \mathsf{poly}(L)$*;*
- $|\mathsf{pub}_i| \leq \mathsf{poly}(L)$*;*
- $|\overline{\mathsf{sk}}_{\mathsf{id}}| \leq \mathsf{poly}(L)$*;*
- $|\overline{\sigma}| \leq \mathsf{poly}(\lambda)$ *(independent of* L *and* N*).*

In registration-based cryptography, pp is used for all users for frequent operations such as encryption or Verify, so pp should remain compact. While aux is maintained solely by the curator for less frequent operations (e.g., Reg, Upd, and SigUpd), therefore we don't require compactness.

Definition 3.4 (Efficiency). *For* $N \geq 1$*, an RHBS scheme is efficient if the following hold:*

(i) Time cost. *Running times satisfy:*
- $\mathsf{poly}(L, \log N)$ *per valid invocation of* Upd, Key, KeyUpd, SigUpd, Verify *and* CheckUpd.
- $\mathsf{poly}(L, \log N)$ *per valid invocation of* Reg*, amortized over* N *registrations.*

(ii) Update frequency. *Invocations of algorithms satisfy:*
- $O(\log N)$ *invocations of* Upd *per registered* rid_j*;*
- $O(\log N)$ *invocations of* Key *per root user;*
- $O(\log N)$ *invocations of* KeyUpd *per non-root user;*
- $O(\log N)$ *invocations of* SigUpd *per signature.*

We model existential unforgeability under adaptive chosen-message attacks, allowing malicious registration, root corruption, and non-ancestor signing-key exposure.

Definition 3.5 (Selective-identity unforgeability with adaptively chosen root identifier). *Given* $(1^\lambda, L)$ *and the public description of* $\mathcal{D}$*, a PPT adversary commits to* $(I_2^*, \ldots, I_L^*)$*, including wildcard positions, before receiving* $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, L)$ *or querying any scheme random oracle. The challenger acts as the honest curator, starting with empty* $(\mathsf{pp}, \mathsf{aux})$ *and logs* $C, \mathcal{E}, Q_{\mathrm{sig}}$*. The adversary reads the entire public state, including helper histories, and adaptively issues the following queries. Out-of-domain queries return* $\bot$ *without changing state or logs. Both registration queries use the next index* $\mathsf{rid} = N + 1$*. They return* $\bot$ *on* CheckGen *failure and otherwise return the state updated by* Reg.

- RegH()*: Run* $\mathsf{Gen}(\mathsf{pp})$ *and process its request. On acceptance, store* $(\mathsf{rid}, \mathsf{priv})$ *as an honest registration. Return* rid *and* pub *along with the outcome.*
- RegA(pub)*: Submit* $(N + 1, \mathsf{pub})$ *for registration.*

– $\mathsf{Corrupt}(\mathsf{rid})$*: For an honestly registered root, log* rid *in* C *and return its stored* priv.
– $\mathsf{Extract}(\mathsf{id})$*: For an identity under an honest registration, obtain the root signing key via* Upd *and* Key *using the stored* priv*. For a non-root identity, delegate directly to* id *with fresh randomness. Log* id *in* $\mathcal{E}$ *and return its signing key.*
– $\mathsf{Sign}(\mathsf{id}, \mathsf{m})$*: Derive the key as in* Extract *without logging an exposure. Log* $(\mathsf{id}, \mathsf{m})$ *in* Q_{sig} *and return a fresh signature under that key.*

Finally, $\mathcal{A}$ *chooses* rid^* *and outputs* $(\mathsf{id}^*, \mathsf{m}^*, \sigma^*)$ *with* $\mathsf{id}^* = (\mathsf{rid}^*, I_2^*, \ldots, I_L^*)$*. It wins if under the final* pp

(i) $\mathsf{Verify}(\mathsf{id}^*, \mathsf{m}^*, \sigma^*, \mathsf{pp}) = 1$;
(ii) rid^* *was registered by* $\mathsf{RegH}()$ *and* $\mathsf{rid}^* \notin C$;
(iii) $\nexists\, \mathsf{id} \in \mathcal{E} : \mathsf{id} \preceq \mathsf{id}^*$;
(iv) $(\mathsf{id}^*, \mathsf{m}^*) \notin Q_{\mathrm{sig}}$.

An RHBS scheme is unforgeable in this game if every PPT adversary $\mathcal{A}$ *satisfies* $\Pr[\mathcal{A}\ \textit{wins}] \le \mathsf{negl}(\lambda)$ *over all game and oracle randomness.*

Remarks on the model. Freshness is per $(\mathsf{id}, \mathsf{m})$ across versions. Even an ancestor may be queried on m^*. Multiplying (σ_0, σ_1) by $(\Phi(\mathsf{id}, \mathsf{m})^{s'}, g_2^{s'})$ publicly rerandomizes a signature at its existing tag, ruling out strong unforgeability. SigUpd also preserves the signed pair. Per-version records can use $(\mathsf{id}, \mathsf{m}, \mathsf{ver})$ as their key.

The honest curator computes the common log's state. Users can replay pp and, given verified pp, check helpers against the registered pub_i with CheckUpd and updated signatures with Verify on the original pair. The board supplies consistency. Availability is outside this game. Distribution-preserving delegation and updates let Extract and Sign simulate multi-hop derivations.

4 Our RHBS Construction

Our RHBS construction builds on BBG HIBS, obtained by applying the Naor transformation [43] to the BBG encryption scheme [42] with wildcard key delegation [41, Section 4.2]. We instantiate $\mathcal{D} = \mathbb{Z}_p^*$, assign sequential positive integers below p as root identifiers, and reserve coordinate $\ell = L + 1$ for message hashing. Section 4.1 explains the three techniques, Section 4.2 presents the full scheme, and Section 4.3 establishes its completeness, compactness, efficiency, and security.

4.1 Technical Overview

Technique 1 (Cross-term cancellation).
In BBG HIBS, crs contains $u, v, h_1, \ldots, h_\ell \in \mathbb{G}_1$, and a KGC holds the master secret u^α for $\alpha \in \mathbb{Z}_p^*$. To issue $\mathcal{U}_j$'s root signing key, it samples $r_j \xleftarrow{\$} \mathbb{Z}_p$ and computes

$$\mathsf{sk}_{\mathsf{rid}_j} = \left(u^\alpha (v h_1^{\mathsf{rid}_j})^{r_j},\ g_2^{r_j},\ \{h_q^{r_j}\}_{q=2}^{\ell}\right).$$

To remove the KGC, we apply the *cross-term cancellation* approach of prior constructions [17,18,31,44,45,46]. Each root user $\mathcal{U}_i$ independently chooses $\alpha_i \in \mathbb{Z}_p^*$ and, for each recipient $\mathcal{U}_j$ (including $\mathcal{U}_i$ itself), samples a fresh $r_{i,j} \xleftarrow{\$} \mathbb{Z}_p$ to compute one BBG signing-key share:

$$\left\{\mathsf{skShare}_{i,j} = \left(u^{\alpha_i} (v h_1^{\mathsf{rid}_j})^{r_{i,j}},\ g_2^{r_{i,j}},\ \{h_q^{r_{i,j}}\}_{q=2}^{\ell}\right)\right\}_{j=1}^{N}.$$

In (1), row i shows the off-diagonal shares published by $\mathcal{U}_i$, while $\circ$ marks the diagonal share $\mathsf{skShare}_{i,i}$ kept private; column j collects other roots' shares for $\mathcal{U}_j$:

$$\begin{array}{c} \\ \mathcal{U}_1 \xrightarrow{\text{generate}} \\ \mathcal{U}_2 \xrightarrow{\text{generate}} \\ \vdots \\ \mathcal{U}_N \xrightarrow{\text{generate}} \end{array} \begin{array}{c} \begin{array}{cccc} \mathcal{U}_1 & \mathcal{U}_2 & \cdots & \mathcal{U}_N \\ \uparrow \text{receive} & \uparrow \text{receive} & & \uparrow \text{receive} \end{array} \\ \begin{pmatrix} \circ & \mathsf{skShare}_{1,2} & \cdots & \mathsf{skShare}_{1,N} \\ \mathsf{skShare}_{2,1} & \circ & \cdots & \mathsf{skShare}_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ \mathsf{skShare}_{N,1} & \mathsf{skShare}_{N,2} & \cdots & \circ \end{pmatrix} \end{array} \tag{1}$$

The curator aggregates $\mathsf{hk}_{\mathsf{rid}_j} = \prod_{i \neq j} \mathsf{skShare}_{i,j}$, and $\mathcal{U}_j$ computes $\mathsf{sk}_{\mathsf{rid}_j} = \mathsf{skShare}_{j,j} \cdot \mathsf{hk}_{\mathsf{rid}_j}$. This yields a valid BBG root key with $\alpha = \sum_{i=1}^{N} \alpha_i$ and $r_j = \sum_{i=1}^{N} r_{i,j}$, preserving BBG delegation without a global KGC. However, two limitations remain: (1) each root publishes $N - 1$ shares of total size $O(NL)$, and (2) each new registration requires every existing root to publish a fresh share for the newcomer.

Technique 2 (Affine share compression). We introduce *affine share compression* to encode each sender's off-diagonal BBG signing-key shares in one public key share. Our starting point is the BBG share structure in (1). Column j collects the shares for recipient $\mathcal{U}_j$. The exponents of v and h_1 in their first components form the points $\{(r_{i,j}, \mathsf{rid}_j r_{i,j})\}_{i=1}^{N}$ on the line $y = \mathsf{rid}_j x$. In Technique 1, each sender independently chooses one point on each recipient line.

Now consider the row generated by $\mathcal{U}_i$. Writing $\Delta_{i,j} = \mathsf{rid}_j - \mathsf{rid}_i$, we can express the first component of $\mathsf{skShare}_{i,j}$ in terms of the sender's own identity base:

$$u^{\alpha_i} (v h_1^{\mathsf{rid}_j})^{r_{i,j}} = \big(u^{\alpha_i} h_1^{\Delta_{i,j} r_{i,j}}\big)(v h_1^{\mathsf{rid}_i})^{r_{i,j}}.$$

To reuse one mask for u^{α_i} across all off-diagonal shares, $\mathcal{U}_i$ chooses a secret $\rho_i \in \mathbb{Z}_p^*$ and sets $\Delta_{i,j} r_{i,j} = \rho_i$ for every $j \neq i$. This fixes the mask at $h_1^{\rho_i}$ and gives $r_{i,j} = \rho_i / \Delta_{i,j}$. In Figure 1, this same condition places the off-diagonal points on the shifted line $y = \mathsf{rid}_i x + \rho_i$, which meets every recipient line except $\mathcal{U}_i$'s own. The sender samples $r_{i,i}$ independently for its private diagonal share.

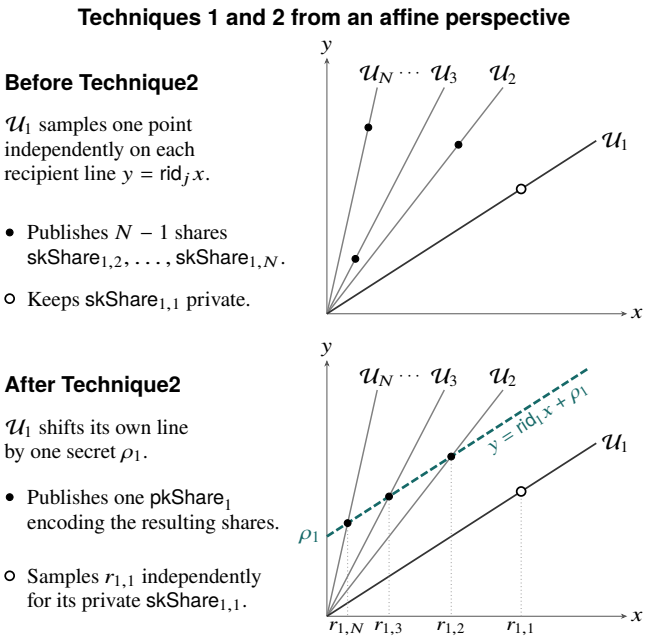

Fig. 1. Share randomizers for $\mathcal{U}_1$ over $\mathbb{Z}_p$ (schematic). In the lower panel, the shifted line fixes all off-diagonal points and leaves the diagonal point independent.

The sender publishes the masked contribution d_i together with the bases needed to derive the remaining factors:

$$\mathsf{pkShare}_i = \big(\underbrace{u^{\alpha_i} h_1^{\rho_i}}_{d_i},\ \underbrace{(v h_1^{\mathsf{rid}_i})^{\rho_i}}_{a_i},\ \underbrace{g_2^{\rho_i}}_{b_i},\ \{\ \underbrace{h_q^{\rho_i}}_{c_{q,i}}\ \}_{q=2}^{\ell}\big).$$

For any $j \neq i$, the curator uses the public inverse $z_{i,j} = \Delta_{i,j}^{-1}$ to recover the complete share

$$\mathsf{skShare}_{i,j} = \big(d_i a_i^{z_{i,j}},\ b_i^{z_{i,j}},\ \{c_{q,i}^{z_{i,j}}\}_{q=2}^{\ell}\big).$$

If $r_{i,j} = \rho_i \Delta_{i,j}$, the public first components A_t at identifier differences $t = 1, 2, 3$ would reveal $u^{\alpha_i} = A_1^3 A_3 / A_2^3$.

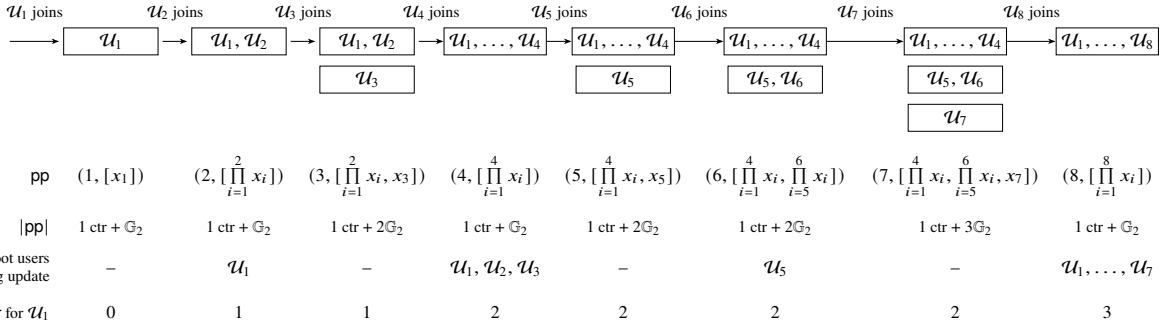

Fig. 2. Power-of-two groups with version tag. The counter and one product of $x_i = g_2^{\alpha_i}$ per group give $|\mathsf{pp}| \le 2 + \lfloor \log_2 N \rfloor$ entries. Each root's version tag changes at most $\lfloor \log_2 N \rfloor$ times over N root registrations. The row "ver for $\mathcal{U}_1$" tracks the version of $\mathcal{U}_1$'s group parameter pp_S.

$$\begin{array}{cc} \mathcal{U}_1 \xrightarrow{\text{generate}} & \begin{pmatrix} \mathsf{skShare}_{1,1} & \mathsf{pkShare}_1 \\ \mathsf{skShare}_{2,2} & \mathsf{pkShare}_2 \\ \vdots & \vdots \\ \mathsf{skShare}_{N,N} & \mathsf{pkShare}_N \end{pmatrix} \\ \mathcal{U}_2 \xrightarrow{\text{generate}} & \\ \vdots & \\ \mathcal{U}_N \xrightarrow{\text{generate}} & \\ & \begin{matrix} \text{private} & \text{public} \end{matrix} \end{array} \tag{2}$$

As (2) shows, each root keeps one diagonal share and publishes one public key share. Its public registration material drops from $O(NL)$ to $O(L)$ group elements and suffices for later recipients. Aggregating all roots into one group, however, still changes pp and every existing root's helper key at each registration.

Technique 3 (Power-of-two groups with version tag).

We adapt the power-of-two groups of prior RBE and RABE schemes [21,31] to track the BBG helper contribution in signing keys and signatures. We retain their equal-size merge rule and use each group's level as a version tag for this contribution (Figure 2).

For each root $\mathcal{U}_j$ in a group S, the curator aggregates off-diagonal shares only within S to form $\mathsf{hk}_{\mathsf{rid}_j}$ with tag $\mathsf{ver} = \log_2 |S|$. Each merge doubles $|S|$ and increases ver, so $(\mathsf{rid}_j, \mathsf{ver})$ uniquely identifies a helper record in aux. In Verify, the binary expansion of N locates rid_j's group S in pp, and checking $\mathsf{ver} = \log_2 |S|$ confirms tag alignment before pairing verification under pp_S. In SigUpd, the source and target tags locate both helper records for rid_j in aux, even across multiple intervening merges.

Moreover, specializing a root helper to an identity id (and to a message m) preserves componentwise products and quotients. Specializing the ratio of target and source root helpers thus yields the exact correction factor to replace the old helper contribution while keeping the secret core intact. Because $\mathsf{sk}_{\mathsf{id}}$ caches its specialized helper components, its holder runs KeyUpd using only the target $\mathsf{hk}_{\mathsf{rid}_j}$. For a signature σ, a user obtains the correction from the two tagged helpers by running SigUpd with aux.

4.2 Full Construction

Initially, $\mathsf{pp} = (0, [])$ and $\mathsf{aux} = \emptyset$. For $N \ge 1$ with binary expansion $N = \sum_{m=1}^{k} 2^{e_m}$ $(e_1 > \cdots > e_k \ge 0)$, let $\mathcal{F} = (S_1, \ldots, S_k)$ partition $[N]$ into consecutive groups of sizes $|S_m| = 2^{e_m}$, where $S_m = \left\{\sum_{r<m} 2^{e_r} + 1, \ldots, \sum_{r \le m} 2^{e_r}\right\}$. Write $\mathsf{pp} = \left(N, (\mathsf{pp}_S)_{S \in \mathcal{F}}\right)$ in group order; each root in S_m has current version tag $\mathsf{ver} = \log_2 |S_m| = e_m$. With $\ell = L + 1$, for any identity $\mathsf{id} = (I_1, \ldots, I_L)$ and message m, we have $F_{\mathsf{id}} = v \prod_{q \in \mathsf{Fix}(\mathsf{id})} h_q^{I_q}$ and $\Phi(\mathsf{id}, \mathsf{m}) = F_{\mathsf{id}} h_\ell^{H_{\mathrm{msg}}(\mathsf{m})}$.

– $\mathsf{Setup}(1^\lambda, L) \to \mathsf{crs}$:
Run $\mathcal{BP} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow \mathsf{BPGen}(1^\lambda)$, and choose hash functions $H_{\mathbb{G}_1} : \{0,1\}^* \to \mathbb{G}_1$ and $H_{\mathrm{msg}} : \{0,1\}^* \to \mathbb{Z}_p^*$. Compute $u = H_{\mathbb{G}_1}(\text{"u"})$, $v = H_{\mathbb{G}_1}(\text{"v"})$, and $h_q = H_{\mathbb{G}_1}(\text{"h}q\text{"})$ for $q \in [\ell]$. Output

$$\mathsf{crs} = \left(\mathcal{BP}, u, v, \{h_q\}_{q \in [\ell]}, H_{\mathbb{G}_1}, H_{\mathrm{msg}}\right).$$

– $\mathsf{Gen}(\mathsf{pp}) \to (\mathsf{rid}_i; \mathsf{pub}_i, \mathsf{priv}_i)$:

Read N from pp (for $N+1 < p$) and set $\mathsf{rid}_i = N+1$. Sample $\alpha_i, \rho_i \xleftarrow{\$} \mathbb{Z}_p^*$ and $r_{i,i} \xleftarrow{\$} \mathbb{Z}_p$, set $x_i = g_2^{\alpha_i}$, and compute $\mathsf{pkShare}_i = (d_i, a_i, b_i, \{c_{q,i}\}_{q=2}^{\ell})$ and $\mathsf{skShare}_{i,i} = (a_{i,i}, b_{i,i}, \{c_{q,i,i}\}_{q=2}^{\ell})$ as

$$\mathsf{pkShare}_i = \big(u^{\alpha_i} h_1^{\rho_i},\ (v h_1^{\mathsf{rid}_i})^{\rho_i},\ g_2^{\rho_i},\ \{h_q^{\rho_i}\}_{q=2}^{\ell}\big),$$
$$\mathsf{skShare}_{i,i} = \big(u^{\alpha_i} (v h_1^{\mathsf{rid}_i})^{r_{i,i}},\ g_2^{r_{i,i}},\ \{h_q^{r_{i,i}}\}_{q=2}^{\ell}\big).$$

Set $\mathsf{ctx}_i = (\mathsf{rid}_i, \mathsf{pkShare}_i)$ and run $\pi_i \leftarrow \mathsf{NIZK.Prove}((g_2, x_i), \alpha_i, \mathsf{ctx}_i)$. Output $(\mathsf{rid}_i; \mathsf{pub}_i, \mathsf{priv}_i)$

$$\mathsf{pub}_i = (x_i, \pi_i, \mathsf{pkShare}_i), \quad \mathsf{priv}_i = (\alpha_i, \mathsf{skShare}_{i,i}).$$

– $\mathsf{Reg}(\mathsf{rid}_j, \mathsf{pub}_j, \mathsf{pp}, \mathsf{aux}) \to (\mathsf{pp}', \mathsf{aux}')$:
Parse $\mathsf{pp} = \big(N, (\mathsf{pp}_S)_{S \in \mathcal{F}}\big)$ and initialize $\mathsf{aux}' \leftarrow \mathsf{aux}$.

(i) *Registration material.* Store $\mathsf{pub}_j = (x_j, \pi_j, \mathsf{pkShare}_j)$ under rid_j in aux'.

(ii) *Group merging.* Initialize $S = \{\mathsf{rid}_j\}$. While $\mathcal{F}$ is nonempty and its rightmost group T satisfies $|T| = |S|$, remove T from $\mathcal{F}$ and set $S \leftarrow T \cup S$. Append S as the rightmost group of $\mathcal{F}$ and store the updated partition in aux'.

(iii) *Public parameters.* Compute $\mathsf{pp}_S = \prod_{\mathsf{rid}_i \in S} x_i$. Form pp' by removing the entries of groups merged into S, appending pp_S, and setting the counter to $N+1$.

(iv) *Helper keys.* Set $\mathsf{ver} = \log_2 |S|$. For each $\mathsf{rid}_t \in S$ and each $\mathsf{rid}_i \in S \setminus \{\mathsf{rid}_t\}$, set $z_{i,t} = (\mathsf{rid}_t - \mathsf{rid}_i)^{-1} \bmod p$ and compute

$$\mathsf{skShare}_{i,t} = \big(d_i a_i^{z_{i,t}},\ b_i^{z_{i,t}},\ \{c_{q,i}^{z_{i,t}}\}_{q=2}^{\ell}\big)$$

from the stored $\mathsf{pkShare}_i = (d_i, a_i, b_i, \{c_{q,i}\}_{q=2}^{\ell})$. Compute the componentwise product (with empty products equal to group identities):

$$\overline{\mathsf{hk}}_{\mathsf{rid}_t} = (\widehat{A}_t, \widehat{B}_t, \{\widehat{C}_{q,t}\}_{q=2}^{\ell}) = \prod_{\mathsf{rid}_i \in S \setminus \{\mathsf{rid}_t\}} \mathsf{skShare}_{i,t}.$$

Set $\mathsf{hk}_{\mathsf{rid}_t} = (\overline{\mathsf{hk}}_{\mathsf{rid}_t}, \mathsf{ver})$. In aux', extend rid_t's helper array through slot ver (leaving skipped slots empty), store $\mathsf{hk}_{\mathsf{rid}_t}$ at slot ver, and update rid_t's latest-record pointer.

Output $(\mathsf{pp}', \mathsf{aux}')$.

– $\mathsf{Upd}(\mathsf{aux}, \mathsf{rid}_j) \to \mathsf{hk}_{\mathsf{rid}_j}$:
Follow rid_j's latest-record pointer in aux and output the stored $\mathsf{hk}_{\mathsf{rid}_j} = (\overline{\mathsf{hk}}_{\mathsf{rid}_j}, \mathsf{ver})$.

– $\mathsf{Key}(\mathsf{rid}_j, \mathsf{priv}_j, \mathsf{hk}_{\mathsf{rid}_j}) \to \mathsf{sk}_{\mathsf{rid}_j}$:
Parse $\mathsf{priv}_j = (\alpha_j, \mathsf{skShare}_{j,j})$ with $\mathsf{skShare}_{j,j} = (a_{j,j}, b_{j,j}, \{c_{q,j,j}\}_{q=2}^{\ell})$, and $\mathsf{hk}_{\mathsf{rid}_j} = (\overline{\mathsf{hk}}_{\mathsf{rid}_j}, \mathsf{ver})$ with $\overline{\mathsf{hk}}_{\mathsf{rid}_j} = (\widehat{A}_j, \widehat{B}_j, \{\widehat{C}_{q,j}\}_{q=2}^{\ell})$. Compute $F_{\mathsf{rid}_j} = v h_1^{\mathsf{rid}_j}$ and output

$$\mathsf{sk}_{\mathsf{rid}_j} = \big(a_{j,j}\widehat{A}_j,\ b_{j,j}\widehat{B}_j,\ \{c_{q,j,j}\widehat{C}_{q,j}\}_{q=2}^{\ell},\ F_{\mathsf{rid}_j},\ \overline{\mathsf{hk}}_{\mathsf{rid}_j},\ \mathsf{ver}\big).$$

– $\mathsf{Delegate}(\mathsf{id}, \mathsf{sk}_{\mathsf{id}}, \mathsf{id}') \to \mathsf{sk}_{\mathsf{id}'}$:
Write $\mathsf{id}' = (I'_1, \ldots, I'_L)$ and $J = \mathsf{Fix}(\mathsf{id}') \setminus \mathsf{Fix}(\mathsf{id})$. Parse $\mathsf{sk}_{\mathsf{id}} = (a, b, \{c_q\}, F_{\mathsf{id}}, \overline{\mathsf{hk}}_{\mathsf{id}}, \mathsf{ver})$ with $\overline{\mathsf{hk}}_{\mathsf{id}} = (\widehat{A}, \widehat{B}, \{\widehat{C}_q\})$, where $q \in \mathsf{Free}(\mathsf{id}) \cup \{\ell\}$.

(i) *Signing-key delegation.* Set $F_{\mathsf{id}'} = F_{\mathsf{id}} \prod_{k \in J} h_k^{I'_k}$. Sample $t \xleftarrow{\$} \mathbb{Z}_p$ and compute

$$a' = a\Big(\prod_{k \in J} c_k^{I'_k}\Big) F_{\mathsf{id}'}^{t}, \quad b' = b g_2^{t}, \quad c'_q = c_q h_q^{t} \ \ \big(q \in \mathsf{Free}(\mathsf{id}') \cup \{\ell\}\big).$$

(ii) *Helper-cache specialization.* Compute

$$\overline{\mathsf{hk}}_{\mathsf{id}'} = \Big(\widehat{A} \prod_{k \in J} \widehat{C}_k^{I'_k},\ \widehat{B},\ \{\widehat{C}_q\}_{q \in \mathsf{Free}(\mathsf{id}') \cup \{\ell\}}\Big).$$

Output $\mathsf{sk}_{\mathsf{id}'} = \big(a', b', \{c'_q\}, F_{\mathsf{id}'}, \overline{\mathsf{hk}}_{\mathsf{id}'}, \mathsf{ver}\big)$.

– $\mathsf{KeyUpd}(\mathsf{id}, \mathsf{hk}_{\mathsf{rid}_j}, \mathsf{sk}_{\mathsf{id}}) \to \mathsf{sk}'_{\mathsf{id}}$:

Write $\mathsf{id} = (I_1, \ldots, I_L)$. Parse $\mathsf{sk}_{\mathsf{id}} = (a, b, \{c_q\}, F_{\mathsf{id}}, \overline{\mathsf{hk}}_{\mathsf{id}}^{\mathrm{old}}, \mathsf{ver})$ for $q \in \mathsf{Free}(\mathsf{id}) \cup \{\ell\}$, and $\mathsf{hk}_{\mathsf{rid}_j} = (\overline{\mathsf{hk}}_{\mathsf{rid}_j}, \mathsf{ver}')$ with $\overline{\mathsf{hk}}_{\mathsf{rid}_j} = (\widehat{A}_j, \widehat{B}_j, \{\widehat{C}_{q,j}\}_{q=2}^{\ell})$.

(i) *Helper-cache specialization.* Compute

$$\overline{\mathsf{hk}}_{\mathsf{id}}^{\mathrm{new}} = \Big(\widehat{A}_j \prod_{k \in \mathsf{Fix}(\mathsf{id}) \setminus \{1\}} \widehat{C}_{k,j}^{I_k},\ \widehat{B}_j,\ \{\widehat{C}_{q,j}\}_{q \in \mathsf{Free}(\mathsf{id}) \cup \{\ell\}}\Big).$$

(ii) *Signing-key update.* Compute the componentwise product and quotient

$$(a', b', \{c'_q\}) = (a, b, \{c_q\}) \cdot \frac{\overline{\mathsf{hk}}_{\mathsf{id}}^{\mathrm{new}}}{\overline{\mathsf{hk}}_{\mathsf{id}}^{\mathrm{old}}}.$$

Output $\mathsf{sk}'_{\mathsf{id}} = \big(a', b', \{c'_q\}, F_{\mathsf{id}}, \overline{\mathsf{hk}}_{\mathsf{id}}^{\mathrm{new}}, \mathsf{ver}'\big)$.

– $\mathsf{Sign}(\mathsf{id}, \mathsf{sk}_{\mathsf{id}}, \mathsf{m}) \to \sigma$:

Read $a, b, c_\ell, F_{\mathsf{id}}$, and ver from $\mathsf{sk}_{\mathsf{id}}$. Compute $\mu = H_{\mathrm{msg}}(\mathsf{m})$, sample $s \xleftarrow{\$} \mathbb{Z}_p$, and output $\sigma = (\overline{\sigma}, \mathsf{ver})$ with

$$\overline{\sigma} = (\sigma_0, \sigma_1) = \big(a c_\ell^{\mu} (F_{\mathsf{id}} h_\ell^{\mu})^s,\ b g_2^s\big).$$

– $\mathsf{SigUpd}(\mathsf{aux}, \mathsf{id}, \mathsf{m}, \sigma, \mathsf{ver}') \to \sigma'$ or $\perp$:

Write $\mathsf{id} = (I_1, \ldots, I_L)$ and set $\mathsf{rid}_j = I_1$. Parse $\sigma = (\overline{\sigma}, \mathsf{ver})$ with $\overline{\sigma} = (\sigma_0, \sigma_1)$. Read the records at slots ver and ver' in rid_j's helper array in aux. Return $\perp$ if either record is missing.

Parse the two stored helper bodies as $\overline{\mathsf{hk}}_{\mathsf{rid}_j} = (\widehat{A}_j, \widehat{B}_j, \{\widehat{C}_{q,j}\}_{q=2}^{\ell})$ and $\overline{\mathsf{hk}}'_{\mathsf{rid}_j} = (\widehat{A}'_j, \widehat{B}'_j, \{\widehat{C}'_{q,j}\}_{q=2}^{\ell})$. Set $\widehat{A} = \widehat{A}'_j / \widehat{A}_j$, $\widehat{B} = \widehat{B}'_j / \widehat{B}_j$, and $\widehat{C}_q = \widehat{C}'_{q,j} / \widehat{C}_{q,j}$ for $2 \le q \le \ell$. Compute $\mu = H_{\mathrm{msg}}(\mathsf{m})$ and output $\sigma' = (\overline{\sigma}', \mathsf{ver}')$ with

$$\overline{\sigma}' = \Big(\sigma_0 \widehat{A} \Big(\prod_{q \in \mathsf{Fix}(\mathsf{id}) \setminus \{1\}} \widehat{C}_q^{I_q} \Big) \widehat{C}_\ell^{\mu},\ \sigma_1 \widehat{B}\Big).$$

– $\mathsf{Verify}(\mathsf{id}, \mathsf{m}, \sigma, \mathsf{pp}) \to \{0, 1\}$:

Write $\mathsf{id} = (I_1, \ldots, I_L)$, and parse $\mathsf{pp} = \big(N, (\mathsf{pp}_S)_{S \in \mathcal{F}}\big)$ and $\sigma = (\overline{\sigma}, \mathsf{ver})$ with $\overline{\sigma} = (\sigma_0, \sigma_1)$. Accept if and only if there is a group $S \in \mathcal{F}$ containing I_1 such that $\mathsf{ver} = \log_2 |S|$ and

$$e(\sigma_0, g_2) = e(u, \mathsf{pp}_S) \cdot e\big(\Phi(\mathsf{id}, \mathsf{m}), \sigma_1\big).$$

– $\mathsf{CheckSetup}(\mathsf{crs}) \to \{0, 1\}$:

Parse $\mathsf{crs} = \big(\mathcal{BP}, u, v, \{h_q\}_{q \in [\ell]}, H_{\mathbb{G}_1}, H_{\mathrm{msg}}\big)$. Recompute $u, v, \{h_q\}_{q \in [\ell]}$ from $H_{\mathbb{G}_1}$ as in Setup. Output 1 if all recomputed elements match those in crs, and 0 otherwise.

– $\mathsf{CheckGen}(\mathsf{rid}_i, \mathsf{pub}_i, \mathsf{pp}) \to \{0, 1\}$:

Read N from pp, and parse $\mathsf{pub}_i = (x_i, \pi_i, \mathsf{pkShare}_i)$ with $\mathsf{pkShare}_i = (d_i, a_i, b_i, \{c_{q,i}\}_{q=2}^{\ell})$. Reject unless $\mathsf{rid}_i = N + 1 < p$ and both x_i and b_i are nonidentity elements of $\mathbb{G}_2$.

(i) *Proof of knowledge.* Set $\mathsf{ctx}_i = (\mathsf{rid}_i, \mathsf{pkShare}_i)$ and check $\mathsf{NIZK.Verify}((g_2, x_i), \pi_i, \mathsf{ctx}_i) = 1$.

(ii) *Share well-formedness.* Sample $\beta_1, \ldots, \beta_\ell \xleftarrow{\$} \mathbb{Z}_p$, set $\widehat{d} = d_i a_i^{\beta_1} \prod_{q=2}^{\ell} c_{q,i}^{\beta_q}$, and check

$$e(\widehat{d}, g_2) = e(u, x_i) \cdot e\Big(h_1 (v h_1^{\mathsf{rid}_i})^{\beta_1} \prod_{q=2}^{\ell} h_q^{\beta_q},\ b_i\Big).$$

Output 1 if both checks pass, and 0 otherwise.

– $\mathsf{CheckUpd}(\mathsf{rid}_j, \mathsf{pub}_j, \mathsf{hk}_{\mathsf{rid}_j}, \mathsf{pp}) \to \{0, 1\}$:

Parse $\mathsf{pp} = \big(N, (\mathsf{pp}_S)_{S \in \mathcal{F}}\big)$, read x_j from pub_j, and parse $\mathsf{hk}_{\mathsf{rid}_j} = (\overline{\mathsf{hk}}_{\mathsf{rid}_j}, \mathsf{ver})$ with $\overline{\mathsf{hk}}_{\mathsf{rid}_j} = (\widehat{A}_j, \widehat{B}_j, \{\widehat{C}_{q,j}\}_{q=2}^{\ell})$.

Find the group $S \in \mathcal{F}$ containing rid_j. Return 0 if no such group exists or if $\mathsf{ver} \neq \log_2 |S|$. Sample $\beta_2, \ldots, \beta_\ell \xleftarrow{\$} \mathbb{Z}_p$ and set $\widetilde{A} = \widehat{A}_j \prod_{q=2}^{\ell} \widehat{C}_{q,j}^{\beta_q}$. Output 1 if the following equality holds, and 0 otherwise:

$$e(\widetilde{A}, g_2) = e(u, \mathsf{pp}_S / x_j) \cdot e\Big(v h_1^{\mathsf{rid}_j} \prod_{q=2}^{\ell} h_q^{\beta_q},\ \widehat{B}_j\Big).$$

4.3 Properties Analysis

Completeness. Our construction satisfies Definition 3.2. With fresh uniform coefficients, CheckGen(ii) accepts a share violating the pairing relations with probability at most $1/p$. Over $q_R = \mathsf{poly}(\lambda)$ adaptive malicious registration attempts, all accepted shares satisfy these relations with probability at least $1 - q_R/p$. On this event, (iii)–(v) hold simultaneously.

(i) *Setup correctness.* CheckSetup recomputes the CRS bases deterministically and outputs 1 with probability 1.
(ii) *Registration correctness.* For $N + 1 < p$, an honest output of Gen satisfies NIZK verification and the pairing relations, so CheckGen outputs 1 with probability 1.
(iii) *Helper-key correctness.* The product of the off-diagonal shares satisfies the helper pairing relations for its group, so CheckUpd accepts the helper from Upd.
(iv) *Signature correctness.* Key, Delegate, and KeyUpd yield valid BBG signing keys under pp_S, so signatures under those keys verify with the group's tag.
(v) *Signature-update correctness.* Once both tags are recorded in aux, SigUpd applies the specialized helper quotient and target tag, taking any valid source signature to a valid signature on the same identity and message under pp'.

Compactness. Our construction is extra-compact (Definition 3.3). The version tag $\mathsf{ver} \in \{0, \ldots, \lfloor \log_2 N \rfloor\}$ uses $O(\log \log N)$ bits. Let $F = |\mathsf{Fix}(\mathsf{id})| \in [1, L]$ denote the number of fixed identity coordinates.

(i) *Standard compactness.*
- $|\mathsf{pp}| \le 1$ ctr $+ (1 + \lfloor \log_2 N \rfloor)\mathbb{G}_2$, where the counter N uses $O(\log N)$ bits;
- $|\overline{\mathsf{hk}}_{\mathsf{rid}_j}| = (L + 1)\mathbb{G}_1 + \mathbb{G}_2$.

(ii) *Extra compactness.*
- $|\mathsf{crs}| = (L + 3)\mathbb{G}_1$;
- $|\mathsf{pub}_t| = (L + 2)\mathbb{G}_1 + 3\mathbb{G}_2 + \mathbb{Z}_p$;
- $|\overline{\mathsf{sk}}_{\mathsf{id}}| = [2(L - F + 2) + 1]\mathbb{G}_1 + 2\mathbb{G}_2$;
- $|\overline{\sigma}| = \mathbb{G}_1 + \mathbb{G}_2$.

Efficiency. Our construction is efficient (Definition 3.4):

(i) *Time cost.*
- With F_{id} in the signing key, Sign takes three exponentiations in $\mathbb{G}_1$, one in $\mathbb{G}_2$, four group multiplications, and one message hash.
- Per valid invocation, Upd, Key, KeyUpd, and SigUpd run in $O(L)$ time (with $O(1)$ helper lookups and no pairings), while CheckUpd and Verify locate S in $O(\log N)$ word operations and use $O(L)$ group/field operations plus 3 pairings;
- For the FFT-supported merge sizes in Section 5.1, Reg takes $O(L \log^2 N)$ group/field operations amortized over N accepted registrations, and $O(NL \log N)$ in the worst case (this happens when merging into a single group of size N).

(ii) *Update frequency.* Since a root's version ver changes only when its group merges (at least doubling $|S|$), each root experiences at most $\lfloor \log_2 N \rfloor$ version changes after registration, requiring at most:
- $1 + \lfloor \log_2 N \rfloor$ invocations of Upd per registered rid_j;
- $1 + \lfloor \log_2 N \rfloor$ invocations of Key per root user (including initial key derivation);
- $\lfloor \log_2 N \rfloor$ invocations of KeyUpd per non-root user;
- $\lfloor \log_2 N \rfloor$ invocations of SigUpd per signature.

Unforgeability. We model $H_{\mathbb{G}_1}$, H_{msg}, and H_{NIZK} as independent programmable random oracles. The coordinate-domain description given before commitment is $\mathcal{BP}$. Fix an a priori bound $1 \le M < p$ on the adversary's registration queries, including rejections. Only the reduction uses M. Let q_R bound malicious registration attempts and q_P all H_{NIZK} queries, including honest ones. Set $Q = q_H + q_S + 1$, where q_H bounds external H_{msg} queries (including local updates) and q_S bounds signing queries.

Theorem 4.1 (Selective-identity unforgeability with adaptively chosen root identifier). *For any PPT adversary winning Definition 3.5 with probability ϵ, a PPT solver for computational asymmetric $(L + 1)$-BDHE succeeds with probability*

$$\eta \ge \frac{\epsilon}{MQ} - \frac{Q(Q-1)}{2(p-1)} - \frac{q_P + q_R + 2}{p}.$$

Fixing I_1^* in advance gives selective-identity unforgeability with loss Q. At $L = 1$, the entire identity is chosen adaptively.

4.4 Security Proof

Completeness. CheckSetup accepts crs outputs by recomputation. Fix any accepted crs. For $N + 1 < p$, Gen selects $\mathsf{rid}_i = N + 1$, its NIZK proof verifies, and bilinearity gives

$$\begin{aligned} e(d_i, g_2) &= e(u, x_i)e(h_1, b_i), \\ e(a_i, g_2) &= e(vh_1^{\mathsf{rid}_i}, b_i), \\ e(c_{q,i}, g_2) &= e(h_q, b_i) \quad (2 \le q \le \ell). \end{aligned} \tag{3}$$

Thus CheckGen accepts the honest request. Both CheckGen and CheckUpd batch pairing relations using fresh, independent coefficients. Write E_0 for the ratio of the two sides of the unweighted relation and E_q for the ratio in each weighted relation. Acceptance requires $E_0 \prod_q E_q^{\beta_q} = 1$. If some $E_q \neq 1$, fixing the other coefficients leaves exactly one accepting $\beta_q \in \mathbb{Z}_p$. If only $E_0 \neq 1$, the check always rejects. Over q_R adaptive malicious registration attempts, the probability that CheckGen accepts even one share violating (3) is at most q_R/p. Outside this event, injectivity of $e(\cdot, g_2)$ forces every accepted public key share to have the form in Gen for its x_i and b_i, even when chosen maliciously.

For the group S selected from the verified pp, write $x_j = g_2^{\alpha_j}$ for the registered root contribution and $\mathsf{pp}_S = g_2^{\alpha_S}$. For a well-typed helper with the target tag, $\widehat{B}_j = g_2^w$ fixes w. By injectivity of $e(\cdot, g_2)$, the individual pairing relations hold exactly for helpers of the form

$$\big(u^{\alpha_S - \alpha_j}(vh_1^{\mathsf{rid}_j})^w,\ g_2^w,\ \{h_q^w\}_{q=2}^{\ell}\big).$$

Outside the registration error, multiplying the off-diagonal shares gives this form, including $w = 0$ for a singleton. Every such helper passes. The same batch argument bounds the probability that CheckUpd accepts an invalid helper in q_U adaptive checks by q_U/p.

For any valid helper, dividing the root key's BBG components by its helper cache recovers the private diagonal share. KeyUpd replaces the cache while preserving this core, and Delegate specializes both tuples while rerandomizing the BBG components. For every valid w, the resulting key has the BBG form for group exponent α_S. A root's group grows whenever it changes, so matching tags in one history identify the same group. With Upd outputs the failure bound is q_R/p, or at most $(q_R + q_U)/p$ when using checked external helpers.

For $(\mathsf{id}, \mathsf{m})$ with $I_1 = \mathsf{rid}_j$, set $\mu = H_{\mathrm{msg}}(\mathsf{m})$ and specialize the root helper as

$$T_0 = \widehat{A}_j \prod_{q \in \mathsf{Fix}(\mathsf{id}) \setminus \{1\}} \widehat{C}_{q,j}^{I_q} \widehat{C}_{\ell,j}^{\mu}, \qquad T_1 = \widehat{B}_j. \tag{4}$$

For a valid helper, the form above and bilinearity give

$$e(T_0, g_2) = e(u, \mathsf{pp}_S/x_j)e(\Phi(\mathsf{id}, \mathsf{m}), T_1). \tag{5}$$

Once both versions are stored, their helpers satisfy this equality outside the registration error. In the target-to-source quotient, x_j cancels. Applying the componentwise helper ratio and the target tag in SigUpd therefore makes any valid source signature, including a previously updated one, verify for the same identity and message under the target pp.

Proof (of Theorem 4.1). Use the query bounds of Theorem 4.1 and let $\ell = L + 1$. A worst-case running-time bound also bounds the number of registration queries, at unit cost per query. The solver $\mathcal{B}$ receives the computational challenge of Section 2.1 and checks query domains before applying abort rules. If $P_1 = 1$ or $S_2 = 1$, it outputs $1_{\mathbb{G}_T}$, solving the cases $a = 0$ or $s = 0$. Otherwise $a, s \neq 0$, with a uniform in $\mathbb{Z}_p^*$.

Setup and random oracles. Give $\mathcal{BP}$ to $\mathcal{A}$, which commits to $(I_2^*, \ldots, I_L^*)$ before the remaining CRS and random-oracle access. Write $\overline{\mathsf{id}} = (\overline{I}_1, \ldots, \overline{I}_L)$ for the identity with wildcards replaced by zero. Fixed coordinates are nonzero, so this encoding is injective. Independently sample $j \xleftarrow{\$} [M]$, $h \xleftarrow{\$} [Q]$, and $\mu^* \xleftarrow{\$} \mathbb{Z}_p^*$, and set

$$\boldsymbol{T}^* = (j, \overline{I}_2^*, \ldots, \overline{I}_L^*, \mu^*).$$

The guess j is the root identifier in the global order of successful registrations, including malicious ones.

Choose independent uniform $\beta, \gamma_0, \ldots, \gamma_\ell \in \mathbb{Z}_p$ and program the setup labels of $H_{\mathbb{G}_1}$ to

$$u = g_1^{\beta} P_\ell, \quad h_q = g_1^{\gamma_q} P_{\ell-q+1}, \quad v = g_1^{\gamma_0} \prod_{q=1}^{\ell} P_{\ell-q+1}^{-T_q^*}. \tag{6}$$

For fixed $a, \boldsymbol{T}^*$, these exponent masks give jointly independent uniform bases. Other inputs receive fresh uniform group elements.

Number distinct messages by their first H_{msg} query, including the adversary's local computations, signing queries, and final verification. Assign μ^* to the hth message and independent uniform elements of $\mathbb{Z}_p^*$ to all others; repeated queries use the table. Maintain the H_{NIZK} table independently.

Simulating registrations. Run the real CheckGen and Reg to reproduce public registration material and helper histories. At the next available slot j, a rejected RegA(pub) leaves j available, and an accepted one causes an abort. Use Gen and retain priv for all honest roots except j. When RegH() assigns identifier j, set $x^* = \widehat{P}_1$.

Sample $r_0 \xleftarrow{\$} \mathbb{Z}_p$, set $b^* = g_2^{r_0}\widehat{P}_1^{-1}$, and abort if $b^* = 1$. Implicitly $\rho^* = r_0 - a$. With $V_* = vh_1^j$, compute

$$\begin{aligned} d^* &= P_1^{\beta-\gamma_1} h_1^{r_0}, \\ a^* &= V_*^{r_0} P_1^{-(\gamma_0+\gamma_1 j)} \prod_{q=2}^{\ell} P_{\ell-q+2}^{T_q^*}, \\ c_q^* &= h_q^{r_0} P_1^{-\gamma_q} P_{\ell-q+2}^{-1} \quad (2 \le q \le \ell). \end{aligned} \tag{7}$$

These are $u^a h_1^{\rho^*}$, $V_*^{\rho^*}$, and $h_q^{\rho^*}$. The missing $a^{\ell+1}$ term cancels in d^*, and $T_1^* = j$ cancels a^ℓ in V_*. All required power indices are at most ℓ. The trial fails with probability $1/p$; otherwise ρ^* is uniform nonzero, giving $(x^*, \mathsf{pkShare}^*)$ its real joint distribution.

Set $\mathsf{ctx}^* = (j, \mathsf{pkShare}^*)$. Choose $c, z \xleftarrow{\$} \mathbb{Z}_p$, set $U = g_2^z (x^*)^{-c}$, and program $H_{\mathrm{NIZK}}(g_2, x^*, U, \mathsf{ctx}^*) = c$. Abort if this input is already in the oracle table. With the table, context, and c fixed, U is uniform, so the abort probability is at most q_P/p. At a fresh input, $z = w + ca$ gives the real Schnorr distribution. The target's private diagonal share need not be computed.

Simulating BBG components. For queries under root j, the same calculation simulates Extract and Sign. Let K assign values to coordinates in $B \subseteq [\ell]$. For Extract, take $B = \mathsf{Fix}(\mathsf{id})$ and $K_q = I_q$; for Sign, take $B = [\ell]$ and $K = (\overline{\mathsf{id}}, \mu)$. Put $\Delta_q = K_q - T_q^*$ for $q \in B$ and $\Delta_q = -T_q^*$ otherwise, and set

$$\Gamma_K = \gamma_0 + \sum_{q \in B} \gamma_q K_q, \qquad F_K = v \prod_{q \in B} h_q^{K_q}.$$

Choose $q_0 \in B$ with $\delta = K_{q_0} - T_{q_0}^* \neq 0$. Sample $t \xleftarrow{\$} \mathbb{Z}_p$ and compute

$$\begin{aligned} A_K &= P_1^{\beta} F_K^t P_{q_0}^{-\Gamma_K/\delta} \prod_{\substack{k \in [\ell] \\ k \neq q_0}} P_{\ell-k+1+q_0}^{-\Delta_k/\delta}, \\ B_K &= g_2^t \widehat{P}_{q_0}^{-1/\delta}, \\ C_{q,K} &= h_q^t P_{q_0}^{-\gamma_q/\delta} P_{\ell-q+1+q_0}^{-1/\delta} \quad (q \notin B). \end{aligned} \tag{8}$$

Substituting $F_K = g_1^{\Gamma_K} \prod_{q=1}^{\ell} P_{\ell-q+1}^{\Delta_q}$ and $r = t - a^{q_0}/\delta$ cancels the unknown power in u^a, yielding $(u^a F_K^r, g_2^r, \{h_q^r\}_{q \notin B})$. The required indices $\ell - q + 1 + q_0$ lie in $[2, 2\ell]$. The missing index occurs only at $q = q_0$, which the product omits and $q \notin B$ excludes. Second-group indices are at most ℓ. Conditional on the transcript and latent secrets, $t \mapsto r$ is a bijection.

Answering queries. Use stored secrets and the real algorithms for honest roots other than j. For root j, abort on corruption or extraction of an ancestor of $(j, I_2^*, \ldots, I_L^*)$. Otherwise an extraction fixes some q_0 with $I_{q_0} \neq T_{q_0}^*$, including when $T_{q_0}^* = 0$. Use (8) with $B = \mathsf{Fix}(\mathsf{id})$, multiply the specialized public helper into the simulated BBG core, and append the helper cache, public F_{id}, and version tag.

For signing, set $\mu = H_{\mathrm{msg}}(\mathsf{m})$, $B = [\ell]$, and $K = (\overline{\mathsf{id}}, \mu)$, so $F_K = \Phi(\mathsf{id}, \mathsf{m})$. Abort if $K = \boldsymbol{T}^*$; otherwise obtain (A_K, B_K) from (8). A proper ancestor of the guessed target differs at a zero-padded coordinate, allowing signatures on the target message. Compute (T_0, T_1) from the root helper by (4) and return $(A_K T_0, B_K T_1, \mathsf{ver})$.

Consistency across registration states. At every level, write $(a_0, b_0, \{c_{q,0}\})$ for the BBG components of $\mathsf{sk}_{\mathsf{id}}$ and $(A, B, \{C_q\})$ for its helper cache. Define normalization on these two tuples by

$$\mathcal{N}(\mathsf{sk}_{\mathsf{id}}) = (a_0/A, b_0/B, \{c_{q,0}/C_q\}_{q \in \mathsf{Free}(\mathsf{id}) \cup \{\ell\}}).$$

For an honest root user indexed by j, the core satisfies

$$\mathcal{N}(\mathsf{sk}_{\mathsf{id}}) = \left(u^{\alpha_j} F_{\mathsf{id}}^r, g_2^r, \{h_q^r\}_{q \in \mathsf{Free}(\mathsf{id}) \cup \{\ell\}}\right). \tag{9}$$

For root j, normalization recovers its fixed diagonal share with $r = r_{j,j}$. KeyUpd multiplies a non-root key and its cache by the same helper ratio, preserving the core. Delegate specializes both tuples to the child identity and rerandomizes only the BBG tuple. Specialization commutes with products and quotients, so updating and multi-hop delegation commute for the same delegation randomness. Previously exposed cores persist. These identities hold for the actual helpers even when malicious shares are invalid.

For signatures, let (T_0, T_1) and (T_0', T_1') be the source and target helper specializations from (4). SigUpd replaces the tag and preserves the normalized signature:

$$(\sigma_0'/T_0', \sigma_1'/T_1') = (\sigma_0/T_0, \sigma_1/T_1).$$

Thus successive or skipped-version updates preserve the same core.

Every allowed extraction under root j is non-root and has normalized randomizer $r_{j,j} + t_{\mathrm{del}}$ with fresh uniform t_{del}. Signing adds fresh uniform s_{sig}, with $t_{\mathrm{del}} = 0$ for root signatures. Conditional on prior answers and the fixed diagonal share, these randomizers are uniform. Coupling them with the bijective simulation randomizers gives the real joint transcript up to abort, including repeated queries, exposed keys, caches, version tags, and the public helper history.

Forgery and extraction. Abort unless the forgery is accepted and $I_1^* = j$. Its tag identifies the final group S containing root j. Compute (T_0, T_1) from the target root's helper for S by (4) at $(\mathsf{id}^*, \mathsf{m}^*)$. Outside the CheckGen error of probability at most q_R/p, dividing the verification equation by (5) yields $\widetilde{\sigma} = (\sigma_0/T_0, \sigma_1/T_1)$ satisfying

$$e(\widetilde{\sigma}_0, g_2) = e(u, x_j)\, e\big(\Phi(\mathsf{id}^*, \mathsf{m}^*), \widetilde{\sigma}_1\big). \tag{10}$$

This normalization uses only public registration material.

On a winning run with correct guesses and no hash collision, the encoded forgery point is $\boldsymbol{T}^*$. Target honesty and non-exposure exclude adversarial registration, corruption, and ancestor extraction at j. A signing query at $\boldsymbol{T}^*$ would violate pair freshness. Outside the registration-simulation bad events, $x_j = g_2^a$ and $\Phi^* = g_1^{\Gamma^*}$ with $\Gamma^* = \gamma_0 + \sum_{q=1}^{\ell} \gamma_q T_q^*$. Let r satisfy $\widetilde{\sigma}_1 = g_2^r$. Non-degeneracy and (10) imply $\widetilde{\sigma}_0 = u^a (g_1^{\Gamma^*})^r$. Without computing r, output

$$Z' = \frac{e(\widetilde{\sigma}_0, S_2)}{e(S_1, \widetilde{\sigma}_1)^{\Gamma^*} e(P_1, S_2)^{\beta}} = e(g_1, g_2)^{s a^{\ell+1}}. \tag{11}$$

Success probability. Couple the simulation to a real execution continued with the latent secrets after any abort. For every fixed (j, h) the reference transcript has the real distribution, so it is independent of the pair (j, h). Let W denote reference success and J the first-query index of its final message (arbitrary on failure). Final verification queries that message, so

$$\Pr[W \wedge j = I_1^* \wedge h = J] = \epsilon/(MQ). \tag{12}$$

In this unconditioned execution, message-hash collisions cost at most $Q(Q-1)/(2(p-1))$, Schnorr programming conflicts at most q_P/p, invalid-share acceptance at most q_R/p, and $b^* = 1$ at most $1/p$. Zero challenge exponents are solved directly. Allowing $1/p$ slack, the solver's success probability satisfies

$$\eta \geq \frac{\epsilon}{MQ} - \frac{Q(Q-1)}{2(p-1)} - \frac{q_P + q_R + 2}{p}. \tag{13}$$

The solver outputs a fixed group element on failure and runs in polynomial time, proving the theorem.

For a precommitted root, set $j = I_1^*$ directly. Only h is guessed in (12), giving Q in place of MQ in (13) with the same registration budget.

5 Implementation and Evaluation

5.1 Implementation and Baselines

We implement RHBS in Go using `gnark-crypto` and `gnark` over BLS12-381 [48]. Compressed elements take 48 B in $\mathbb{G}_1$, 96 B in $\mathbb{G}_2$, and 32 B in $\mathbb{Z}_p$. We instantiate H_{msg} and H_{NIZK} with domain-separated SHA-256, and $H_{\mathbb{G}_1}$ with `HashToG1` using the BLS12-381 suite in RFC 9380 [49, Section 8.8.1]. We ran the benchmarks on a 10-core Apple M4 with 24 GiB of unified memory under macOS 26.5.2. In Table 2, time counts algebraic and hash operations and explicit outputs. RHBS verification also takes $O(\log N)$ word operations to locate the root's group. Storage figures give encoded sizes for up to 65,536 roots. For the 65,536-root Reg benchmark, we run each configuration three times and schedule independent group convolutions in parallel. We set `GOMAXPROCS` to 4 at $L = 1, 4$ and 10 at $L = 16$. Signing and verification also cover up to 65,536 roots. For Tables 3 and 4, we set

Table 2. Comparison of pairing-based signatures and independent HIBS with full and Merkle-authenticated directories.

		RBS		RABS	RHBS		Independent HIBS	
		LYC⁺25 [13] Basic (§3.1)	LYC⁺25 [13] Updatable (§4.2)	ZZZ⁺24 [14]	Ours ($L = 1$)	Ours (L)	Full directory	Merkle accumulator
Storage	|crs|	$O(B^2)$	$O(B^2)$	$O(N^2)$	$O(1)$	$O(L)$	$O(L)$	$O(L)$
	|pp|	$O(\lceil N/B \rceil)$	$O(\lceil N/B \rceil)$	$O(\log N)$	$O(\log N)$	$O(\log N)$	$O(N)$	$O(\log N)$
	|aux|	$O(N)$	$O(N)$	$O(N^2)$	$O(N \log N)$	$O(NL \log N)$	—	$O(N)$
	|pub|	$O(B)$	$O(B)$	$O(N)$	$O(1)$	$O(L)$	$O(1)$	$O(1)$
	|priv|	$O(1)$	$O(1)$	$O(\log N)$	$O(1)$	$O(L)$	$O(1)$	$O(1)$
	|hk|	$O(1)$	$O(1)$	$O(\log N)$	$O(1)$	$O(L)$	—	$O(\log N)$
	|sk|	—	—	—	$O(1)$	$O(L-F+1)$	$O(L-F+1)$	$O(L-F+1)$
	$\|\sigma\|$	1	1	$O(\log N)$	$O(1)$	$O(1)$	$O(1)$	$O(\log N)$
Time	Setup	$O(B^2)$	$O(B^2)$	$O(N^2)$	$O(1)$	$O(L)$	$O(L)$	$O(L)$
	Gen	$O(B)$	$O(B)$	$O(N)$	$O(1)$	$O(L)$	$O(1)$	$O(1)$
	Reg[a]	$O(B)$	$O(B)$	$O(N)$	$O(\log^2 N)$	$O(L \log^2 N)$	$O(1)$	$O(1)$
	Upd	$O(1)$	$O(1)$[b]	$O(\log N)$	$O(1)$	$O(L)$	—	$O(\log N)$
	Key	—	—	—	$O(1)$	$O(L)$	$O(L)$	$O(L)$
	Sign	$O(1)$	$O(1)$	$O(\log N)$	$O(1)$	$O(1)$	$O(1)$	$O(\log N)$[c]
	Verify	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(L)$	$O(L)$	$O(L+\log N)$
Features	Unbounded	×	×	×	✓	✓	✓	✓
	Access Policies	×	×	✓	×	×	×	×
	Delegation	×	×	×	×	✓	✓	✓
	Signature updates	×	✓	×	✓	✓	Not needed	✓

We use N registered users (roots in HIBS/RHBS), maximum hierarchy depth L, LYC⁺25 block capacity $B \in [N]$, and $F = |\mathsf{Fix}(\mathsf{id})|$ fixed coordinates.

Each RHBS helper key, signing key, and signature also carries a version tag ver of $O(\log \log N)$ bits. For $N < 2^{256}$, the tag fits in one byte, so its size is effectively constant in practice.

[a] Reg costs are amortized over N successful registrations, including merges.

[b] Here Upd applies one registration token to one signature.

[c] HIBS-Merkle signing uses $O(1)$ group operations and copies $O(\log N)$ cached hash values into the signature.

GOMAXPROCS to 1 and take the median of seven batch means per run. The remaining timing experiments cover up to 64 roots and report three-run means.

Toeplitz and FFT acceleration. For a merged group S of size $s \geq 2$, the coefficient for contributor i and recipient t is $(t - i)^{-1}$ for $i \neq t$ because root identifiers are consecutive. Setting the diagonal to zero gives a skew-symmetric Toeplitz matrix. Its $2s \times 2s$ circulant embedding reduces aggregation to cyclic convolution in the exponent [50, 51]. We reuse the scalar FFT of the kernel across $L + 1$ vectors in $\mathbb{G}_1$ and one in $\mathbb{G}_2$, applying radix-2 FFTs in Jacobian coordinates to evaluate the group convolutions.

The FFT requires a primitive $2s$th root of unity in $\mathbb{Z}_p$. Batch inversion prepares the kernel in $O(s)$ field operations and one inversion. Each merge costs $O(Ls \log s)$, including $O(Ls)$ writes to archive the helpers. At size 2^k, at most $\lfloor N/2^k \rfloor$ groups form during N registrations, for $O(NLk)$ work at that size. Summing over $1 \leq k \leq \lfloor \log_2 N \rfloor$ and including $O(NL)$ for registration material and singleton helpers gives $O(L \log^2 N)$ amortized work per registration. A merge into one group of size N costs $O(NL \log N)$.

RBS baselines. Table 2 compares Basic (§3.1) and Updatable (§4.2) from Liu et al. [13] (LYC⁺25). We use Updatable in the experiments. We omit Huang et al. [32], who leave the dynamic construction unspecified, and Adaptive (§4.1), which adds a $\mathbb{G}_T$ signature element and a signing pairing. Both compared variants sign with the secret x_i from KeyGen, counted under priv. The registration material is (pk, ξ), with ξ discarded after Reg. Each of the $\lceil N/B \rceil$ blocks contributes one commitment to pp. Verification uses the signer's public key, its opening, and the corresponding CRS slot element.

We adapt the Type-3 BLS12-381 implementation described for Basic [13, §5.1] to Updatable. Commitments, openings, and signatures are in $\mathbb{G}_1$, with an additional $\mathbb{G}_2$ public-key component for verification. We use intra-block CRS offsets and pass integer identifiers directly.

RABS baseline. We use Zhang et al.'s pairing-based RABS [14, §4.1] to compare registered signing on an identity-equality task shared with RHBS at $L = 1$. We register the user identifier as a single attribute, instantiate

the predicate encoding framework (§3.4) with equality, and apply their power-of-two transformation (§5). RABS supports expressive access policies over registered attributes. A policy over hierarchy labels can test a registered signer's subtree membership. Deriving a descendant signing key from a parent key requires a separate delegation mechanism. We use the independent HIBS baselines to compare that capability. The reported RABS costs apply to equality using its general policy machinery. Table 2 also records RABS's support for richer attribute policies.

Independent HIBS baselines. Each organization uses one BBG-based HIBS instance from Section 4. The instances share a transparent CRS and cache F_{id} in each signing key. We count the instance key pair $x_j = g_2^{\alpha_j}$ and α_j as Gen outputs under pub and priv. Key derives the root signing key from α_j.

HIBS-Dir *(full directory).* The verifier looks up x_j in pp, which stores all N instance public keys in a Go `map`. Registration inserts a single entry. Existing signing keys and signatures remain valid.

HIBS-Merkle *(Merkle accumulator).* An append-only Merkle forest [11] merges equally sized subtrees on insertion. pp stores the counter and subtree roots, while aux stores the forest nodes. Upd returns the opening (x_j, π_j), where π_j is the sibling path to the active subtree root. The signer attaches the opening to the BBG signature so the verifier can authenticate x_j. A merge makes only the affected paths stale, so each opening needs at most $\lfloor \log_2 N \rfloor$ refreshes. The refreshed opening can be reused across signatures of the same instance, whose BBG components remain valid.

5.2 Performance Evaluation

Storage Evaluation. RHBS uses 145 B per signature, including a one-byte version tag, throughout Figure 3, versus 144 B for HIBS-Dir. At $N = 2^{16}$, HIBS-Merkle uses 752 B, including the instance public key and its membership proof. At that scale, RHBS uses 81% fewer bytes per signature and avoids a Merkle proof check for resource-constrained verifiers. All three hierarchical schemes share the $48(L + 3)$-byte CRS, which is 912 B at $L = 16$.

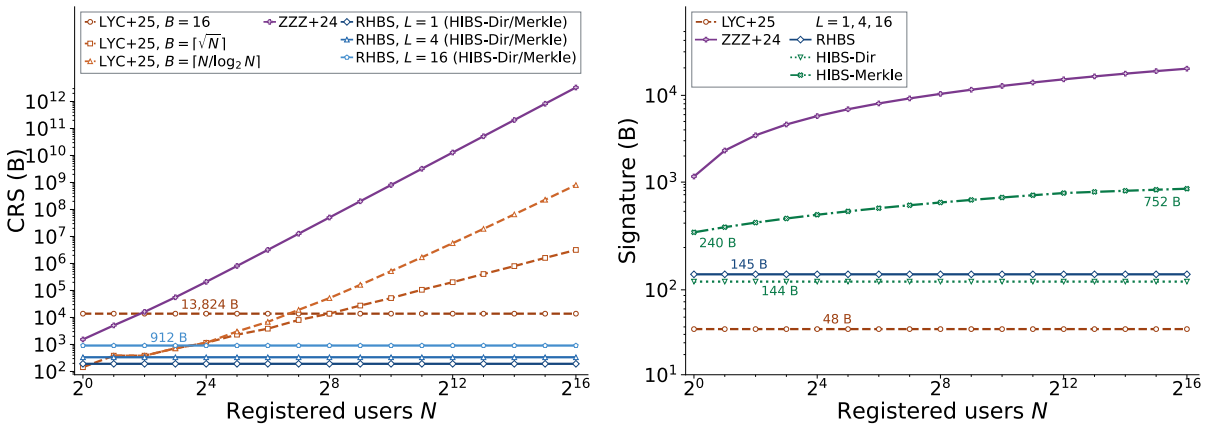

Fig. 3. CRS and signature sizes (bytes, left to right).

Affine share compression keeps each root's public registration material pub independent of N at fixed L. The secret state priv is also independent of N. At $L = 16$, they occupy 1,184 B and 944 B, respectively (Figure 4).

Figure 5 shows a sawtooth pattern in pp for both RHBS and HIBS-Merkle. One entry per active power-of-two group or subtree keeps both sizes logarithmic. Equal-size merges remove entries. The merge into one group at $N = 2^{16}$ drops RHBS pp from 1,568 B to 128 B, while the HIBS-Dir directory occupies 6 MiB at that scale. At $L = 16$, the RHBS curator stores about 588 MiB of aux, including historical helpers for public signature updates.

For fixed L, RHBS root helpers and signing keys have the same sizes for every N (Figure 6). At $L = 16$, a root helper occupies 913 B and a root signing key occupies 1,873 B, compared with 960 B for either HIBS baseline. RHBS stores a helper cache and version tag to support local updates after delegation. With F fixed coordinates, RHBS and HIBS signing keys occupy $96(L - F) + 433$ B and $48(L - F) + 240$ B, respectively.

Computational Evaluation. Figure 7 reports Setup and Gen. The Gen timing covers registration material and secret state generation before root signing-key derivation. Both RHBS algorithms perform $O(L)$ work independently of N. The RBS costs depend on the block size B, which grows with N in two configurations. The RABS costs grow with the configured capacity. We report results through $N = 2^6$, as larger RBS blocks and RABS capacities can make setup and generation expensive. At $N = 64$, raising L from 1 to 16 increases RHBS setup from 0.23 to

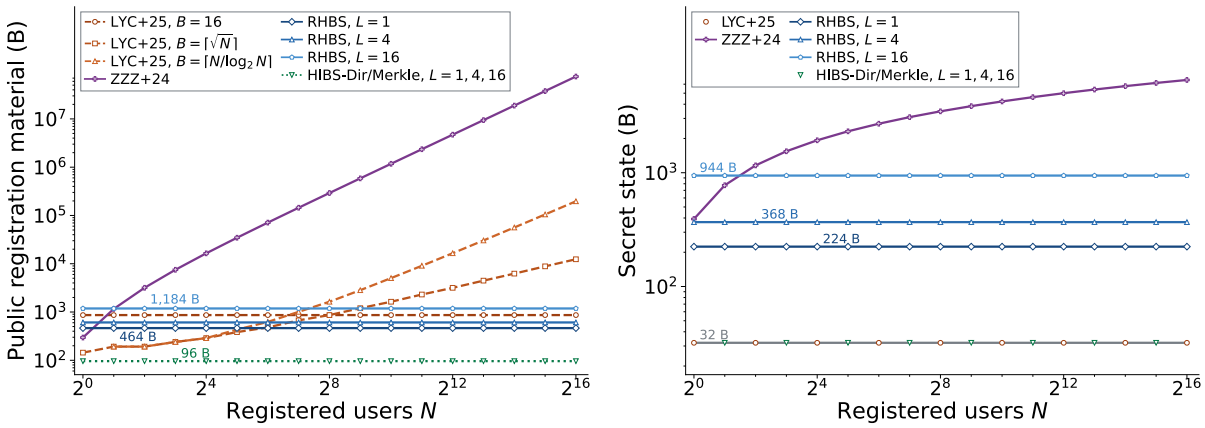

Fig. 4. Sizes of pub and priv (bytes, left to right).

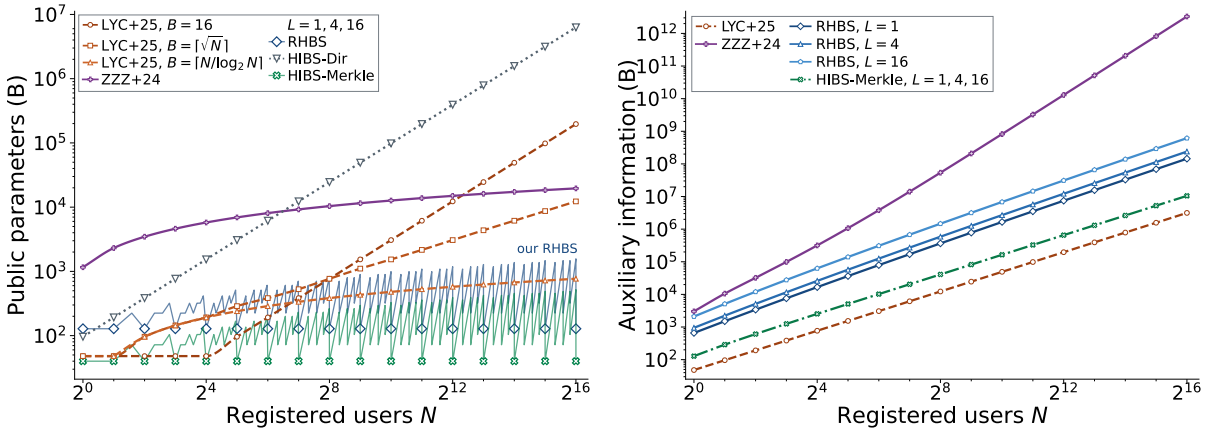

Fig. 5. Sizes of pp and aux (bytes, left to right).

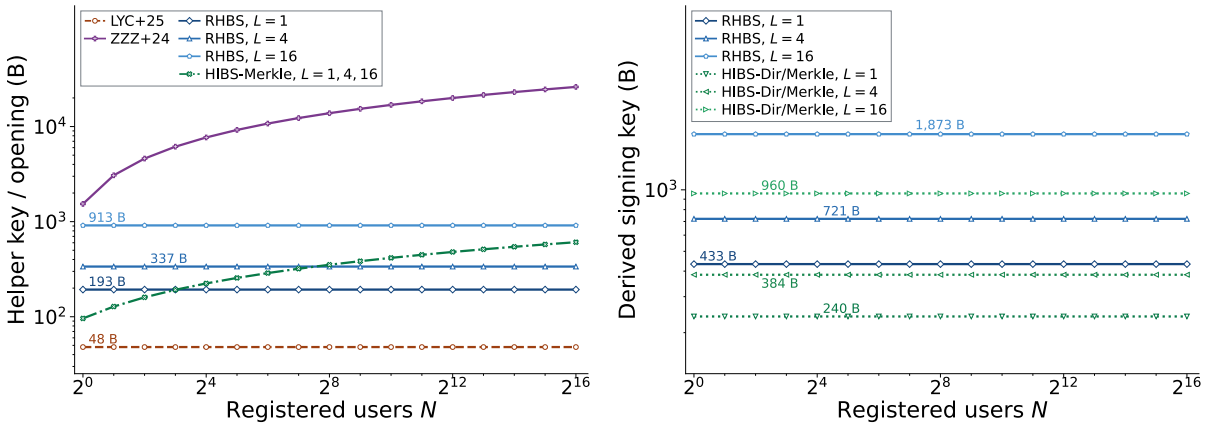

Fig. 6. Helper and root signing key sizes (bytes, left to right). The Merkle opening includes the instance public key.

1.28 ms and generation from 0.79 to 2.60 ms. Both HIBS baselines generate an instance master key pair in about 0.12 ms.

Figure 8 follows 65,536 sequential RHBS registrations. We time group merging, helper computation and archival, and updates to pp. Table 4 reports CheckGen separately. The Nth call forms a group whose size is the largest power of two dividing N. Calls forming groups of the same size fall into horizontal bands, with odd indices forming singletons in the bottom band. Across configurations, the median call takes 0.34–0.57 ms and cumulative time ranges from 1,310 to 2,065 s. The final merge takes 280–444 s, about one fifth of the cumulative time in each configuration.

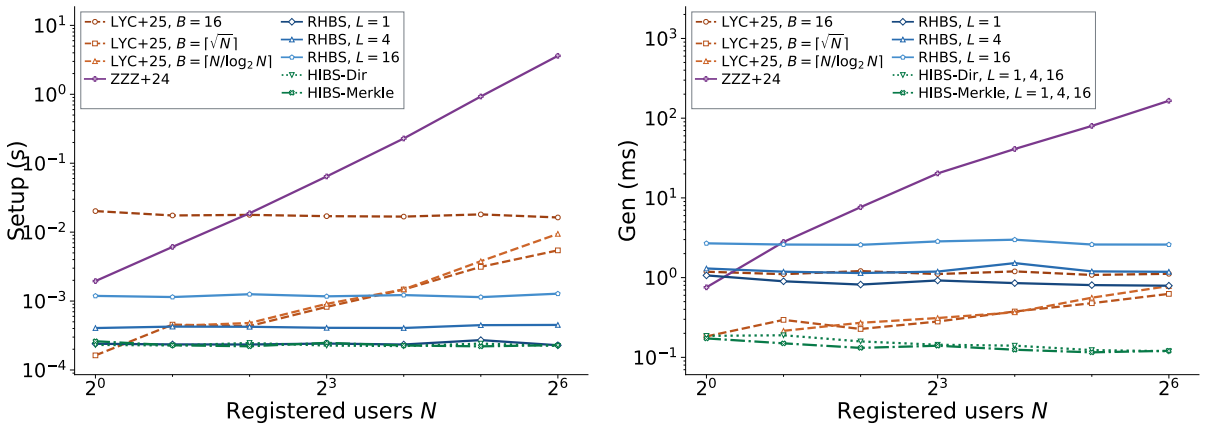

Fig. 7. Time for Setup (left, s) and Gen (right, ms) versus N.

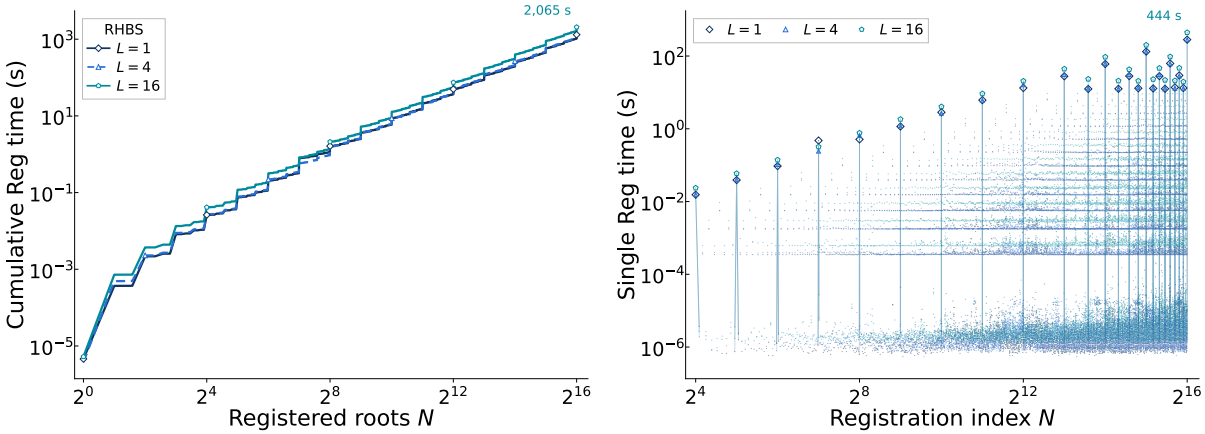

Fig. 8. RHBS registration time. Left, cumulative time for the first N registrations. Right, time per call from $N = 16$, with hollow markers for power-of-two indices and merges of at least 4,096 roots.

In the separate 64-root benchmark, cumulative registration takes 0.41–2.08 s for RHBS across depths, 0.27–0.54 s for LYC$^+$25 across block sizes, and 26.2 s for ZZZ$^+$24. LYC$^+$25 includes consistency checks. The independent HIBS baselines update only their directories and take less than 0.1 ms in total.

Upd retrieves an archived RHBS helper in 0.18–0.45 μs across the 64-root benchmark. The corresponding ZZZ$^+$24 times range from 0.52 to 4.90 μs. For LYC$^+$25, helper retrieval and token-based signature refresh are timed separately.

Table 3 compares signing from the original secret state with any required helper or opening ready. LYC$^+$25 uses its secret exponent and the current block commitment in one Sign call. ZZZ$^+$24 takes the user's secret state and helper key directly in its Sign call. RHBS combines its secret state and helper key in Key. The HIBS baselines derive the root signing key from the instance master secret. Thus we compare native Sign in RBS and RABS with Key+Sign in the hierarchical schemes. The separate Sign rows measure reuse of an already derived key.

For this benchmark, the HIBS baselines register instances sequentially. For RHBS, LYC$^+$25, and ZZZ$^+$24, we prepare equivalent inputs before timing and check them against the original registration or aggregation paths at small sizes. Verification includes group and tag checks in RHBS, directory lookup in HIBS-Dir, and membership proof verification in HIBS-Merkle. At 65,536 roots, RHBS takes 0.282 ms for Sign, 0.299 ms for Key+Sign, and 1.529 ms for verification at $L = 16$.

Table 4 reports RHBS delegation, update, and validation times. At $L = 16$, updating a non-root signing key takes 0.704 ms with the target helper key ready. A public signature update takes 0.791 ms, including both helper lookups, the componentwise quotients, and the message hash.

5.3 Application Recommendations

Curator deployment and verification. Because the curator holds no secrets and runs deterministic algorithms, any party can assume this role in principle. However, maintaining aux requires $O(NL \log N)$ storage and FFT-

Table 3. Signing and verification times (ms).

Signing

Scheme	Operation	Registered users N			
		2^4	2^8	2^{12}	2^{16}
LYC+25 Updatable	Native Sign	0.091	0.089	0.091	0.092
ZZZ+24 RABS	Native Sign	9.557	17.174	24.704	31.601
RHBS	Sign	0.280	0.279	0.283	0.282
	Key+Sign	0.292	0.291	0.294	0.299
HIBS-Dir	Sign	0.287	0.286	0.361	0.283
	Key+Sign	0.586	0.634	0.612	0.583
HIBS-Merkle	Sign	0.282	0.289	0.288	0.285
	Key+Sign	0.582	0.601	0.583	0.585

Verification

Scheme	L	Registered users N			
		2^4	2^8	2^{12}	2^{16}
LYC+25 Updatable	—	0.909	0.898	0.931	0.920
ZZZ+24 RABS	—	6.952	6.045	6.028	6.018
RHBS	1	0.680	0.683	0.707	0.737
	4	0.824	0.825	0.886	0.838
	16	1.519	1.502	1.494	1.529
HIBS-Dir	1	0.752	0.894	1.075	0.775
	4	0.946	1.022	0.905	0.917
	16	1.731	1.889	1.816	1.590
HIBS-Merkle	1	0.769	0.771	0.735	0.755
	4	0.907	0.943	0.931	0.916
	16	1.636	1.668	1.633	1.610

LYC+25 uses $B = 16$. The hierarchical schemes use root identities at $L = 1$ for signing and fully specified identities ($F = L$) for verification. Each tested N yields one RHBS group.

Table 4. RHBS delegation, update, and validation times (ms).

Algorithm	$L = 1$	$L = 4$	$L = 16$
Delegate	—	0.489	1.169
KeyUpd	—	0.147	0.704
SigUpd	0.056	0.195	0.791
CheckSetup	0.173	0.304	0.879
CheckGen	1.424	1.822	3.556
CheckUpd	1.006	1.405	3.097

Delegation fixes one additional coordinate of the root identity. Updates use fully specified identities under root 1, from tag 5 ($N = 32$) to tag 6 ($N = 64$). CheckUpd includes group lookup at $N = 64$. At $L = 1$, only root identities exist.

based merges in Reg, making the role best suited for parties with sufficient compute and storage. For example, a management server in WAVE [5] or a cloud gateway in JEDI [9].

Full verification. Participants with sufficient compute and storage can perform full verification. They read $(\mathsf{rid}_i, \mathsf{pub}_i)$ sequentially from the public bulletin board, check each request by CheckGen, and run Reg on accepted requests to audit or replace the curator.

Lightweight verification. Constrained users avoid recomputing aux. They can still derive pp from the public bulletin board: after validating requests by CheckGen, computing pp in Reg(ii)–(iii) takes $O(1)$ amortized $\mathbb{G}_2$ multiplications per accepted root ($O(N)$ over N roots), bypassing the $O(NL\log^2 N)$ helper-key derivations in Reg(iv) that dominate Reg's cost. Moreover, since aux serves only as input to Upd and SigUpd, users check only their outputs against this verified pp: validating helper keys from Upd by CheckUpd, and updated signatures from SigUpd on (id, m) by Verify—both taking milliseconds in our evaluation (Tables 3 and 4).

RHBS vs. Independent HIBS. Independent HIBS registers new roots through directory insertions and leaves existing signing keys valid. In HIBS-Dir, signatures remain valid. In HIBS-Merkle, affected signatures share a refreshed opening per root. RHBS uses larger registration materials and signing keys, and its curator computes and

archives helper keys. After a group merge, affected signing keys and signatures need updates for the new parameters. These updates can skip intervening versions without contacting a delegator or signer.

Over 65,536 accepted registrations, Reg averages 20–32 ms per root after validation, a cost suited to background enrollment. The final power-of-two merge takes 280–444 s (Figure 8). For a predictable large merge, the curator can compute partial aggregates from already posted shares and incorporate the last root's shares once its request is validated. Alternatively, a deployment can trigger the merge during a maintenance window by registering a reserved root with no application permissions. It occupies a registration slot and triggers the usual updates.

With HIBS-Dir, verifiers keep the $O(N)$ directory or fetch and cache authenticated entries as needed. With RHBS, a verifier can store the authenticated $O(\log N)$ public parameters and verify signatures from all roots in that snapshot offline. With HIBS-Merkle, each signature carries an opening with $O(\log N)$ hashes. RHBS needs 145 B per signature compared with 752 B at 65,536 roots, reducing bandwidth and eliminating path checks. These savings suit constrained verifiers and applications with many verifications between registrations.

Sequential registration and root identifiers. Sequential identifiers $\mathsf{rid}_i = N + 1$ enable binary group lookup and Toeplitz aggregation. Under concurrent registration, a preempted applicant reruns Gen with the updated pp. These identifiers designate organizational namespaces; they do not certify organization names. Non-root coordinates remain freely chosen. For example, a property manager authenticates a contractor and its registered rid, then signs an authorization specifying that identifier and permitted access. A controller that trusts the manager verifies the authorization and the contractor's signature under authenticated pp, checking that the signer's root identifier and requested access match the authorization. No organization-name lookup is needed during verification.

Independent HIBS can support the same workflow by including the contractor's instance public parameters in the signed authorization. RHBS authorizations identify the contractor by its root identifier, and verifiers reuse the same compact pp across organizations.

6 Conclusion

We introduced RHBS for registration across independent organizations and L-level delegation within each organization. Affine share compression keeps each root's public registration material independent of N. Power-of-two groups with version tags enable non-root key updates without re-delegation and public signature updates without re-signing. We proved selective-identity unforgeability with an adaptively chosen root identifier under computational asymmetric $(L + 1)$-BDHE in the random-oracle model, even if all other roots are malicious.

RHBS puts more work on the curator during registration so that verifiers can use compact public parameters across all roots and check short signatures without membership proofs. Over 65,536 registrations, the mean cost per root is in the tens of milliseconds, with peaks at large merges. At that scale and depth 16, verification takes 1.529 ms and each signature occupies 145 B.

Open problems include uncoordinated root identifiers with compact parameters and fast aggregation, adaptive choice of all identity coordinates without exponential depth loss (ideally in the standard model), root revocation, and post-quantum RHBS from lattices.

References

1. K. Puttaswamy *et al.*, "Delta Sharing: An open protocol for cross-platform data sharing," *Proceedings of the VLDB Endowment*, vol. 18, no. 12, pp. 5197–5209, Aug. 2025, doi: 10.14778/3750601.3750637.
2. N. Li, J. C. Mitchell, and W. H. Winsborough, "Design of a role-based trust-management framework," in *Proc. IEEE Symposium on Security and Privacy (S&P)*, 2002, pp. 114–130, doi: 10.1109/SECPRI.2002.1004366.
3. A. Birgisson, J. G. Politz, U. Erlingsson, A. Taly, M. Vrable, and M. Lentczner, "Macaroons: Cookies with contextual caveats for decentralized authorization in the cloud," in *Proc. Network and Distributed System Security Symposium (NDSS)*, 2014, doi: 10.14722/ndss.2014.23212.
4. S. Nath, A. Soneji, J. Baek, T. Bao, A. Doupé, C. Rubio-Medrano, and G.-J. Ahn, "'It's almost like Frankenstein': Investigating the complexities of scientific collaboration and privilege management within research computing infrastructures," in *Proc. IEEE Symposium on Security and Privacy (S&P)*, 2025, pp. 3236–3254, doi: 10.1109/SP61157.2025.00197.
5. M. P. Andersen, S. Kumar, M. AbdelBaky, G. Fierro, J. Kolb, H.-S. Kim, D. E. Culler, and R. A. Popa, "WAVE: A decentralized authorization framework with transitive delegation," in *Proc. USENIX Security Symposium*, 2019, pp. 1375–1392. https://www.usenix.org/conference/usenixsecurity19/presentation/andersen.
6. T. South, S. Marro, T. Hardjono, R. Mahari, C. D. Whitney, A. Chan, and A. Pentland, "Position: AI agents need authenticated delegation," in *Proc. International Conference on Machine Learning (ICML)*, 2025, pp. 82211–82231. https://proceedings.mlr.press/v267/south25a.html.
7. C. Gentry and A. Silverberg, "Hierarchical ID-based cryptography," in *Proc. ASIACRYPT*, 2002, pp. 548–566, doi: 10.1007/3-540-36178-2_34.
8. L. Alber, S. More, and S. Ramacher, "Short-lived forward-secure delegation for TLS," in *Proc. ACM Cloud Computing Security Workshop (CCSW)*, 2020, pp. 119–132, doi: 10.1145/3411495.3421362.
9. S. Kumar, Y. Hu, M. P. Andersen, R. A. Popa, and D. E. Culler, "JEDI: Many-to-many end-to-end encryption and key delegation for IoT," in *Proc. USENIX Security Symposium*, 2019, pp. 1519–1536. https://www.usenix.org/conference/usenixsecurity19/presentation/kumar-sam.
10. M. A. Specter, S. Park, and M. Green, "KeyForge: Non-attributable email from forward-forgeable signatures," in *Proc. USENIX Security Symposium*, 2021, pp. 1755–1773. https://www.usenix.org/conference/usenixsecurity21/presentation/specter-keyforge.
11. L. Reyzin and S. Yakoubov, "Efficient asynchronous accumulators for distributed PKI," in *Proc. Security and Cryptography for Networks (SCN)*, 2016, pp. 292–309, doi: 10.1007/978-3-319-44618-9_16. Full version: https://eprint.iacr.org/2015/718.
12. S. Garg, M. Hajiabadi, M. Mahmoody, and A. Rahimi, "Registration-based encryption: Removing private-key generator from IBE," in *Proc. Theory of Cryptography Conference (TCC)*, 2018, pp. 689–718, doi: 10.1007/978-3-030-03807-6_25. Full version: https://eprint.iacr.org/2018/919.
13. Z. Liu, Y. Yuan, Z. Chen, Y. Han, and B. Wang, "Efficient registration-based signature schemes," *Journal of Systems Architecture*, vol. 166, Art. 103458, 2025, doi: 10.1016/j.sysarc.2025.103458.
14. Y. Zhang, J. Zhao, Z. Zhu, J. Gong, and J. Chen, "Registered attribute-based signature," in *Proc. Public-Key Cryptography (PKC)*, 2024, pp. 133–162, doi: 10.1007/978-3-031-57718-5_5. Full version: https://eprint.iacr.org/2024/293.
15. L. Yang, X. Zhang, Y. Deng, Z. Wu, and X. Zhu, "Anonymous registered attribute-based signature for circuits," *Theoretical Computer Science*, vol. 1063, Art. 115681, 2026, doi: 10.1016/j.tcs.2025.115681.
16. Z. Zhu, J. Zhao, K. Zhang, J. Gong, and H. Qian, "When KGC meets curator: New paradigm of registered ABE and FE," in *Proc. ASIACRYPT*, 2025, pp. 35–67, doi: 10.1007/978-981-95-5119-4_2. Revised full version (2026): https://eprint.iacr.org/2026/1083.
17. R. Garg, G. Lu, B. Waters, and D. J. Wu, "Reducing the CRS size in registered ABE systems," in *Proc. CRYPTO*, 2024, pp. 143–177, doi: 10.1007/978-3-031-68382-4_5.
18. R. Stracovsky, B. Waters, and D. J. Wu, "Pairing-based registered ABE for Boolean formulas with a linear-size CRS," in *Proc. CRYPTO*, 2026, pp. 382–413, doi: 10.1007/978-3-032-35367-2_13. Full version: https://eprint.iacr.org/2026/1062.
19. S. Garg, M. Hajiabadi, M. Mahmoody, A. Rahimi, and S. Sekar, "Registration-based encryption from standard assumptions," in *Proc. Public-Key Cryptography (PKC)*, 2019, pp. 63–93, doi: 10.1007/978-3-030-17259-6_3. Full version: https://eprint.iacr.org/2018/1030.
20. K. Cong, K. Eldefrawy, and N. P. Smart, "Optimizing registration based encryption," in *Proc. IMA International Conference on Cryptography and Coding (IMACC)*, 2021, pp. 129–157, doi: 10.1007/978-3-030-92641-0_7.
21. N. Glaeser, D. Kolonelos, G. Malavolta, and A. Rahimi, "Efficient registration-based encryption," in *Proc. ACM Conference on Computer and Communications Security (CCS)*, 2023, pp. 1065–1079, doi: 10.1145/3576915.3616596.
22. D. Fiore, D. Kolonelos, and P. de Perthuis, "Cuckoo commitments: Registration-based encryption and key-value map commitments for large spaces," in *Proc. ASIACRYPT*, 2023, pp. 166–200, doi: 10.1007/978-981-99-8733-7_6. Full version: https://eprint.iacr.org/2023/1389.
23. N. Döttling, D. Kolonelos, R. W. F. Lai, C. Lin, G. Malavolta, and A. Rahimi, "Efficient laconic cryptography from learning with errors," in *Proc. EUROCRYPT*, 2023, pp. 417–446, doi: 10.1007/978-3-031-30620-4_14. Full version: https://eprint.iacr.org/2023/404.
24. M. Klooß, R. W. F. Lai, J. N. Siemer, and M. Swarnakar, "Scalable registration-based encryption from lattices," in *Proc. IEEE Symposium on Security and Privacy (S&P)*, 2026, pp. 2520–2538, doi: 10.1109/SP63933.2026.00199. Full version: https://eprint.iacr.org/2026/717.

25. R. Goyal and S. Vusirikala, "Verifiable registration-based encryption," in *Proc. CRYPTO*, 2020, pp. 621–651, doi: 10.1007/978-3-030-56784-2_21.
26. J. Dujmovic, G. Malavolta, and W. Qi, "Registration-based encryption in the plain model," in *Proc. Public-Key Cryptography (PKC)*, 2025, pp. 194–226, doi: 10.1007/978-3-031-91820-9_7.
27. M. Hajiabadi, M. Mahmoody, W. Qi, and S. Sarfaraz, "Lower bounds on assumptions behind registration-based encryption," in *Proc. Theory of Cryptography Conference (TCC)*, 2023, pp. 306–334, doi: 10.1007/978-3-031-48618-0_11.
28. M. Mahmoody and W. Qi, "Online mergers and applications to registration-based encryption and accumulators," in *Proc. Conference on Information-Theoretic Cryptography (ITC)*, 2023, pp. 15:1–15:23, doi: 10.4230/LIPIcs.ITC.2023.15.
29. M. Mahmoody, W. Qi, and A. Rahimi, "Lower bounds for the number of decryption updates in registration-based encryption," in *Proc. Theory of Cryptography Conference (TCC)*, 2022, pp. 559–587, doi: 10.1007/978-3-031-22318-1_20.
30. W. Qi, "Tight lower bound on witness update frequency in additive positive accumulators," *IACR Communications in Cryptology*, vol. 2, no. 4, 2026, doi: 10.62056/av7t7ta5v.
31. S. Hohenberger, G. Lu, B. Waters, and D. J. Wu, "Registered attribute-based encryption," in *Proc. EUROCRYPT*, 2023, pp. 511–542, doi: 10.1007/978-3-031-30620-4_17.
32. H.-J. Huang, Z.-Y. Liu, H. Chang, and Y.-F. Tseng, "Efficient registration-based signature," in *Proc. SITAIBA 2025*, 2026, pp. 293–308, doi: 10.1007/978-3-032-24063-7_22.
33. L. Yang, X. Zhang, Y. Deng, X. Zhu, Z. Wu, and Z. Zhang, "Registered attribute-based signature with attribute privacy," in *Proc. Provable and Practical Security (ProvSec)*, 2025, pp. 65–87, doi: 10.1007/978-981-95-2961-2_4.
34. Z. Wan, W. Liu, and H. Cui, "HIBEChain: A hierarchical identity-based blockchain system for large-scale IoT," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 2, pp. 1286–1301, 2023, doi: 10.1109/TDSC.2022.3152797. Full version: https://eprint.iacr.org/2019/1425.
35. S. D. Galbraith, K. G. Paterson, and N. P. Smart, "Pairings for cryptographers," *Discrete Applied Mathematics*, vol. 156, no. 16, pp. 3113–3121, 2008, doi: 10.1016/j.dam.2007.12.010.
36. R. Dubois, A. Guillevic, and M. Sengelin Le Breton, "Improved broadcast encryption scheme with constant-size ciphertext," in *Proc. Pairing 2012*, 2013, pp. 196–202, doi: 10.1007/978-3-642-36334-4_12. Full version: https://eprint.iacr.org/2012/370.
37. C.-P. Schnorr, "Efficient signature generation by smart cards," *Journal of Cryptology*, vol. 4, no. 3, pp. 161–174, 1991, doi: 10.1007/BF00196725.
38. A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Proc. CRYPTO '86*, 1987, pp. 186–194, doi: 10.1007/3-540-47721-7_12.
39. F. Hao, "Schnorr non-interactive zero-knowledge proof," RFC 8235, 2017, doi: 10.17487/RFC8235.
40. T. Attema, S. Fehr, and M. Klooß, "Fiat–Shamir transformation of multi-round interactive proofs (extended version)," *Journal of Cryptology*, vol. 36, no. 4, Art. 36, 2023, doi: 10.1007/s00145-023-09478-y.
41. M. Abdalla, E. Kiltz, and G. Neven, "Generalized key delegation for hierarchical identity-based encryption," in *Proc. European Symposium on Research in Computer Security (ESORICS)*, 2007, pp. 139–154, doi: 10.1007/978-3-540-74835-9_10. Full version: https://eprint.iacr.org/2007/221.
42. D. Boneh, X. Boyen, and E.-J. Goh, "Hierarchical identity based encryption with constant size ciphertext," in *Proc. EUROCRYPT*, 2005, pp. 440–456, doi: 10.1007/11426639_26. Full version: https://crypto.stanford.edu/~dabo/papers/shibe.pdf.
43. D. Boneh and M. Franklin, "Identity-based encryption from the Weil pairing," in *Proc. CRYPTO*, 2001, pp. 213–229, doi: 10.1007/3-540-44647-8_13.
44. Q. Wu, Y. Mu, W. Susilo, B. Qin, and J. Domingo-Ferrer, "Asymmetric group key agreement," in *Proc. EUROCRYPT*, 2009, pp. 153–170, doi: 10.1007/978-3-642-01001-9_9.
45. D. Kolonelos, G. Malavolta, and H. Wee, "Distributed broadcast encryption from bilinear groups," in *Proc. ASIACRYPT*, 2023, pp. 407–441, doi: 10.1007/978-981-99-8733-7_13. Full version: https://eprint.iacr.org/2023/874.
46. B. Waters and D. J. Wu, "Silent threshold cryptography from pairings: Expressive policies in the plain model," in *Proc. EUROCRYPT*, 2026, pp. 517–547, doi: 10.1007/978-3-032-25330-9_18. Full version: https://eprint.iacr.org/2025/1547.
47. H. Wee and D. J. Wu, "Unbounded distributed broadcast encryption and registered ABE from succinct LWE," in *Proc. CRYPTO*, 2025, pp. 204–235, doi: 10.1007/978-3-032-01881-6_7.
48. S. Bowe, "BLS12-381: New zk-SNARK elliptic curve construction," Electric Coin Company, 2017. https://zodl.com/new-snark-curve/.
49. A. Faz-Hernandez, S. Scott, N. Sullivan, R. S. Wahby, and C. A. Wood, "Hashing to elliptic curves," RFC 9380, 2023, doi: 10.17487/RFC9380.
50. G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed. Johns Hopkins University Press, 2013.
51. D. Feist and D. Khovratovich, "Fast amortized KZG proofs," Cryptology ePrint Archive, Paper 2023/033, 2023. https://eprint.iacr.org/2023/033.
52. Y. Sakemi (ed.), S. Kanno, and R. Wahby, "Pairing-friendly curves," IRTF, Internet-Draft draft-irtf-cfrg-pairing-friendly-curves-16, Nov. 16, 2026. https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-pairing-friendly-curves-16.
53. J. H. Cheon, "Discrete logarithm problems with auxiliary inputs," *Journal of Cryptology*, vol. 23, no. 3, pp. 457–476, 2010, doi: 10.1007/s00145-009-9047-0.
54. J. H. Cheon and T. Kim, "A new approach to the discrete logarithm problem with auxiliary inputs," *LMS Journal of Computation and Mathematics*, vol. 19, no. 1, pp. 1–15, 2016, doi: 10.1112/S1461157015000303.