Silent Threshold Cryptography from Pairings: Expressive Policies in the Plain Model

Brent Waters
UT Austin and NTT Research
bwaters@cs.utexas.edu

David J. Wu
UT Austin
dwu4@cs.utexas.edu

Abstract

Threshold cryptography is a standard technique for distributing trust by splitting cryptographic keys into multiple shares held by different parties. The classic model of threshold cryptography assumes either that a *trusted* dealer distributes the shares to the different parties (and in doing so, also knows the overall secret) or that the users participate in an *interactive* distributed key-generation protocol to derive their individual shares. In recent years, several works have proposed a new model where users can independently choose a public key, and there is a public and deterministic function that derives the joint public key associated with a group of users from their individual keys. Schemes with this *silent* (i.e., non-interactive) setup procedure allow us to take advantage of the utility of threshold cryptography without needing to rely on a trusted dealer or an expensive interactive setup phase.

Existing works have primarily focused on threshold policies. This includes notions like threshold signatures (resp., encryption) with silent setup (where only quorums with at least T users can sign (resp., decrypt) a message) and distributed broadcast encryption (a special case of threshold encryption where the threshold is 1). Currently, constructions that support general threshold policies either rely on strong tools such as indistinguishability obfuscation and witness encryption, or analyze security in idealized models like the generic bilinear group model. The use of idealized models is due to the reliance on techniques for constructing succinct non-interactive arguments of knowledge (SNARKs).

In this work, we introduce a new pairing-based approach for constructing threshold signatures and encryption schemes with silent setup. On the one hand, our techniques directly allow us to support expressive policies like monotone Boolean formulas in addition to thresholds. On the other hand, we only rely on basic algebraic tools (i.e., a simple cross-term cancellation strategy), which yields constructions with shorter signatures and ciphertexts compared to previous pairing-based constructions. As an added bonus, we can also prove (static) security under q-type assumptions in the plain model. Concretely, the signature size in our distributed threshold signature scheme is 3 group elements and the ciphertext size in our distributed threshold encryption scheme is 4 group elements (together with a short tag).

1 Introduction

Threshold cryptography [Des87, Fra89, DF89, DDFY94] is a general paradigm for distributing trust in cryptography. For instance, in a threshold signature scheme, there is a distributed signing process where at least T out of N participants must come together to produce a signature on a message. Similarly, in threshold encryption, a group of at least T out of N parties must come together in order to decrypt a ciphertext. In the classic setting of threshold cryptography, there is a central authority (also called a trusted dealer) that generates shares of the signing key (or decryption key) and issues the individual shares to the different participants. The trusted dealer in this case would also know the "master" signing key (or decryption key) for the underlying system. One way to remove the central trusted authority is to have individual parties run an *interactive* distributed key-generation protocol to jointly derive their individual shares. While the distributed key-generation protocol ensures that no single entity knows the master secret key for the cryptosystem, the protocol itself often requires high communication or computational costs [TCZ+20]. Moreover, when the universe of users in the threshold policy is dynamic or we want to support more general policies beyond thresholds (e.g., a signing or decryption policy described by a Boolean formula), we often need to re-run the distributed key-generation protocol each time the set of users or the policy changes.

Distributed cryptography. A natural question then is whether users can *independently* and *non-interactively* generate their public and private keys (for either a signature scheme or an encryption scheme) and then simply publish their public key to a directory. For instance, in a distributed monotone-policy signature scheme, we want the ability to take a tuple of verification keys $(\mathsf{vk}_1, \ldots, \mathsf{vk}_N)$ for an arbitrary set of N users along with a signing policy φ and aggregate them into a succinct verification key vk_φ for the set of users and the policy. Individual users can generate "partial signatures" on messages of their choosing using their individual signing key (as in a vanilla digital signature scheme). Moreover, given a collection of signatures $\{\sigma_i\}_{i \in S}$ on a message m from a set of parties $S \subseteq [N]$ that satisfy the policy φ, there is a public aggregation algorithm that outputs an aggregate signature σ_{agg} on the message m. Both the verification key vk_φ and the aggregate signature σ_{agg} should be short (i.e. have size that is independent of the number of users N or the complexity of the policy φ). Security says that any unauthorized set of users S cannot produce a signature on any message m that verifies with respect to vk_φ. When specialized to the setting of threshold policies, this is the notion of threshold signatures with *silent* setup [MRV+21, DCX+23, GJM+24].

We can consider an analogous notion for the case of encryption. In this case, we want the ability to take a tuple of public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$ for an arbitrary set of N users along with a decryption policy φ and aggregate them into a succinct encryption key ek_φ associated with the set of users and the policy. Anyone can encrypt a message with respect to ek_φ with a ciphertext whose size is independent of N or the complexity of the policy φ. Thereafter, any subset of users $S \subseteq [N]$ who satisfies the decryption policy φ and use their secret keys to generate a "partial decryption" σ. It is possible to recover the message given partial decryptions σ_i for a set of users $S \subseteq [N]$ that satisfy the policy φ. When φ is a threshold policy, this corresponds to threshold encryption with silent setup [GKPW24] and for more general policies, it is called distributed monotone-policy encryption [DJWW25]. An important special case of threshold encryption with silent setup is the setting of distributed (or flexible) broadcast encryption [WQZD10, BZ14, FWW23]. This corresponds to the setting where the decryption policy φ is a threshold policy with threshold 1.

Constructing distributed cryptography. Thus far, much of the work on constructing distributed cryptographic notions have centered on the special case of distributed broadcast encryption, and currently, we have many realizations from pairing-based assumptions [WQZD10, KMW23], lattice-based assumptions [CW24, CHW25, WW25], or general tools such as indistinguishability obfuscation [BZ14] and witness encryption [FWW23]. However, distributed broadcast encryption corresponds to the special case where the set of decryptors has size 1. Generalizing beyond distributed broadcast encryption has been challenging, and existing constructions have either relied on strong tools such as indistinguishability obfuscation or witness encryption [RSY21, ADM+24, DJWW25] or idealized models such as the random oracle model [MRV+21] or the algebraic/generic group model [DCX+23, GJM+24, GKPW24, BCF+25]. The latter constructions in idealized models all rely on techniques from succinct non-interactive arguments of knowledge (SNARKs) and it seems challenging to prove security of these schemes in the plain model. Such a scheme must minimally overcome barriers such as [GW11] and moreover, if witness extraction is essential to the security analysis, then idealized models are *necessary* to extract a witness from a succinct proof. Moreover, the use of SNARKs introduces additional complexity and computational costs to the scheme itself. Finally, existing constructions only consider simple policies such as (weighted) threshold policies [RSY21, MRV+21, DCX+23, ADM+24, GJM+24, GKPW24] or policy families that can be computed by a read-once bounded-space Turing machine [DJWW25].

This work. In this work, we introduce a new approach for constructing distributed monotone-policy signatures and encryption schemes from pairings. Our approach is "low tech" and directly leverages the algebraic structure of the Boneh-Boyen signature and identity-based encryption (IBE) scheme [BB04a].[1] We achieve two main sets of results:

- **Distributed monotone-policy signatures:** First, we obtain a distributed monotone-policy signature scheme that supports any policy family captured by a linear secret sharing scheme (e.g., this include threshold policies [Sha79] as well as monotone Boolean formulas [BL88]) where the aggregate signature just consists of 3 group elements (1 more than a vanilla Boneh-Boyen signature). This is $\approx 3\times$ shorter than the pairing-based scheme from [DCX+23, GJM+24] and in fact, exactly matches the level of succinctness obtained using the most succinct pairing-based SNARK for NP [Gro16]; see Table 1 for a comparison. All of these prior constructions rely on SNARK machinery whereas our construction avoids it altogether.

[1] Throughout this work, whenever we refer to the "Boneh-Boyen signature scheme," we specifically mean the signature scheme derived from the identity-based encryption (IBE) scheme in [BB04a], and *not* the short signature scheme from [BB04b].

Scheme	Policy	Assumption	$\lvert\mathsf{crs}\rvert$	$\lvert\mathsf{vk}\rvert$	$\lvert\mathsf{vk}_\varphi\rvert$	$\lvert\sigma\rvert$	$\lvert\sigma_{\mathsf{agg}}\rvert$	AD	$\frac{\text{NO}}{\text{RO}}$
Generic (SNARK)*	Boolean circuit	generic group	N	1	1	$\lvert\mathbb{G}\rvert$	$3\lvert\mathbb{G}\rvert$	✓	✓†
Generic (BARG)*	Boolean circuit	k-Lin	N	1	1	$\lvert\mathbb{G}\rvert$	$\mathsf{poly}(\lambda)\lvert\mathbb{G}\rvert$	✗	✓†
Cor. 3.19	Boolean formula‡	q-type	N^2	N	1	$2\lvert\mathbb{G}\rvert$	$3\lvert\mathbb{G}\rvert$	✗	✓
[GJM+24]	weighted threshold	generic group	N	N	1	$\lvert\mathbb{G}\rvert$	$9\lvert\mathbb{G}\rvert + 5\lvert\mathbb{F}\rvert$	✓	✗
[DCX+23]	weighted threshold	algebraic group	N	N	1	$\lvert\mathbb{G}\rvert$	$8\lvert\mathbb{G}\rvert$	✓	✗
Cor. 5.6	threshold	q-type	$N \log N$	N	$\log N$	$2\lvert\mathbb{G}\rvert$	$3\lvert\mathbb{G}\rvert$	✗	✓

*These refer to generic constructions either based on SNARKs and instantiated with the pairing-based scheme of [Gro16] or based on monotone-policy batch arguments (BARG) [BBK+23, BCJP24] and instantiated with the pairing-based monotone-policy BARG obtained via [WW22, KLVW23, NWW24]. We report metrics based on applying these techniques with the signature scheme from [BLS01]. Note that difference choices of the underlying proof systems and signature scheme will yield different asymptotics; we picked the options that minimizes the signature sizes. We refer to Section 1.3 for more details on these generic approaches.

†As written, these instantiations do rely on the random oracle for the [BLS01] signature scheme. However, we could alternatively instantiate the underlying signature scheme with a construction that does not rely on random oracles. For instance, an instantiation with the [BB04b] signature scheme would yield a construction without random oracles and where the size of a user's individual signature σ is $\lvert\mathbb{G}\rvert + \lvert\mathbb{F}\rvert$. The other metrics are unaffected.

‡Assuming a monotone Boolean formula of size at most N (and each variable appears once).

Table 1: Comparison of our distributed monotone-policy signature scheme with previous constructions. For each scheme, we report the policy family the scheme supports (assumed to be monotone), the underlying cryptographic assumption on which they rely, the sizes of the common reference string crs, an individual user's verification key vk (including the aggregation hint), the aggregate verification key vk_φ associated with a policy, a user's individual signature σ, the final aggregated signature σ_{agg}, as well as whether the scheme is proven to be adaptively secure (AD), and if the scheme can be instantiated *without* relying on the random oracle heuristic ($\frac{\text{NO}}{\text{RO}}$). We assume each scheme is instantiated to support a policy over N users (this is a restriction on the number of inputs to φ, not the number of users that can join the system). We suppress $\mathsf{poly}(\lambda)$ factors when reporting parameter sizes, except when reporting signature sizes, in which case, we give the *exact* breakdown in terms of the number of group elements ($\mathbb{G}$) and number of field elements ($\mathbb{F}$). All of the threshold schemes support *dynamic* thresholds where the threshold can be chosen at verification time.

- **Distributed monotone-policy encryption:** The same techniques also give a distributed monotone-policy encryption scheme for the same policy family where the ciphertext size consists of 4 group elements together with a λ-bit tag. The previous scheme of [GKPW24] needed 10 group elements. Other constructions of distributed monotone-policy encryption relied on strong tools like obfuscation or witness encryption [RSY21, ADM+24, DJWW25]; see Table 2.

In both cases, the direct algebraic approach we take enables (1) shorter signatures (or ciphertexts) and reduced computational costs compared to previous pairing-based constructions; (2) support for general policy families (beyond threshold policies); and (3) a security reduction in the plain model. In particular, we prove *static* security of our schemes from a q-type assumption in the plain model, where q roughly grows with the bound N on the maximum number of users associated with a policy. The proof strategy relies on a standard partitioning strategy. Beyond the functionality, computational assumption, and efficiency improvements achieved by our construction, we believe an important conceptual message of this work is highlighting the versatility of classic pairing-based techniques for realizing distributed cryptography without needing to bring in more complex machinery such as SNARKs, witness encryption, or indistinguishability obfuscation.

1.1 Our Results

In this work, we put forward a new approach for building distributed monotone-policy signatures and encryption from pairings. Our approach is general and allows us to support any policy family that can be described by a linear secret sharing scheme (which includes threshold policies as a special case). We start with a summary of our main results and follow with a technical overview of our approach in Section 1.2.

Distributed monotone-policy signatures. Our first result is a construction of a *distributed monotone-policy signature* scheme; this is a generalization of threshold signatures with silent setup to more general signing policies φ. Here, we provide two sets of results (one for general policies and one tailored for threshold policies):

- **General policies:** Our first result (Corollary 3.19) is a construction of distributed monotone-policy signatures that can support arbitrary monotone Boolean formulas. To support a Boolean formula with up to N users (and size N), we require a CRS of size $O(N^2)$. Each user's public key has size $O(N)$ and the aggregate verification key contains 6 group elements. A user's individual signature consists of just 2 group elements and the final aggregate signature contains 3 group elements. Verifying the final signature requires just 3 pairing operations. Security relies on a q-type assumption over prime-order pairing groups (where $q = O(N)$).

  More broadly, our scheme supports any monotone policy that can be described by a linear secret sharing scheme. An example of a policy in this class is a conjunction of thresholds. In this example, we partition the signers into k different groups, and the aggregation policy is the aggregator must have signatures from a majority of users from *each* group. This naturally captures settings where multiple quorums must independently sign off on a policy or directive; for instance, this is the case in many legislative bodies where bills must be passed by majorities in two different bodies before they become law.

- **Threshold policies:** For the special case of threshold policies, we can adapt our construction (Corollary 5.6) to obtain one where the CRS size is $O(N \log N)$ and the aggregate verification key contains $O(\log N)$ group elements. The sizes of each user's individual signature and the aggregate signature are unchanged. Moreover, like [DCX+23, GJM+24] this scheme supports *dynamic* thresholds, where the threshold can be chosen at verification time (rather than fixed at preprocessing time).

Previously, the works of [DCX+23, GJM+24] show how to construct a threshold signature with silent setup using pairings. Their scheme relies on a combination of an aggregate signature scheme (i.e., [BLS01]) together with a specialized SNARK (e.g., inner product arguments or the Plonk proof system [GWC19]). Signature aggregation in these schemes essentially corresponds to running the aggregation algorithm of an underlying aggregate signature scheme and then giving a SNARK proof that the aggregation was performed correctly. Likewise, verification requires checking the SNARK proof together with running the verification algorithm of the underlying aggregate signature scheme. The use of SNARKs has a number of consequences in terms of functionality, efficiency, and security:

- **Support for weighted thresholds:** An appealing feature of [DCX+23, GJM+24] is they support *weighted* thresholds. In contrast, our scheme only supports unweighted thresholds, and we leave as an open problem how to extend our techniques to support weighted thresholds with minimal overhead.

- **Longer signatures:** Because the aggregate signature contains a SNARK proof, signatures in [DCX+23, GJM+24] are longer (see Table 1). In our scheme, the aggregate signature contains just 3 group elements. Concretely, if we instantiate our scheme over the standard BLS-381 pairing curve [BLS02] (see Remark 5.7), we obtain a scheme where the aggregate signature is 192 bytes. This is 2.8–3× shorter than previous schemes based on aggregate signatures and SNARKs (536 bytes in the case of [DCX+23] and 592 bytes in the case of [GJM+24]). Moreover, we believe the concrete aggregation and verification costs will be higher for the SNARK-based schemes due to the need to generate and validate the SNARK proof (on top of the costs of the underlying aggregate signature scheme). The performance of our scheme is comparable to that of a plain pairing-based multi-signature (specifically, the [LOS+06] multi-signature based on the Boneh-Boyen signature scheme).

- **Idealized models:** Due to the extensive reliance on SNARK techniques, the security analysis in [DCX+23, GJM+24] critically relies on the generic (or algebraic) bilinear group model [Sho97, BBG05, FKL18]. In contrast, we are able to prove security from a q-type assumption in the *plain* model. On the flip side, working in the generic group model allows previous works to argue adaptive security. Note that this itself is not unexpected, as working in idealized models often enables direct proofs of adaptive security even when no such proofs are known in the plain model (see [RW22] for an example in the setting of attribute-based encryption).

A key conceptual contribution of our work is we show how to directly extend the classic Boneh-Boyen signature scheme [BB04a] into a distributed monotone-policy signature scheme. We take a simple algebraic approach that avoids

Scheme	Policy	Assumption	$\lvert\mathsf{crs}\rvert$	$\lvert\mathsf{pk}\rvert$	$\lvert\mathsf{ek}_\varphi\rvert$	$\lvert\mathsf{ct}\rvert$	$\lvert\sigma\rvert$	AD	$\frac{\text{NO}}{\text{RO}}$
[DJWW25]*	S-space read-once TM	$i\mathcal{O}$ + SSB	1	1	–	2^S	1	✗	✓
Cor. 4.9	Boolean formula†	q-type	N^2	N	1	$3\lvert\mathbb{G}\rvert + \lvert\mathbb{G}_T\rvert + \lvert\mathbb{F}\rvert$	$2\lvert\mathbb{G}\rvert$	✗	✓
[RSY21]*	threshold	$i\mathcal{O}$ + OWF	–	1	–	1	1	✗	✓
[ADM+24]*	threshold	$i\mathcal{O}$ + SSB	–	1	–	1	1	✗	✓
[GJM+24]	weighted threshold	generic group	N	N	1	$9\lvert\mathbb{G}\rvert + \lvert\mathbb{G}_T\rvert$	$\lvert\mathbb{G}\rvert$	✓	✗
Cor. 5.15	threshold	q-type	$N \log N$	N	$\log N$	$3\lvert\mathbb{G}\rvert + \lvert\mathbb{G}_T\rvert + \lvert\mathbb{F}\rvert$	$2\lvert\mathbb{G}\rvert$	✗	✓

*These schemes operate in a model where the encryption algorithm take the set of public keys as input, and do not define a separate preprocessing phase to produce an aggregate encryption key ek_φ.

†Assuming a monotone Boolean formula of size at most N (and each variable appears once).

Table 2: Comparison of our distributed monotone-policy encryption scheme with prior results. For each scheme, we report the policy family the scheme supports (assumed to be monotone), the underlying cryptographic assumption on which they rely, the sizes of the common reference string crs, an individual user's public key pk (including the aggregation hint), the aggregate encryption key ek_φ associated with a policy, a ciphertext ct, and a partial decryption σ, as well as whether the scheme is proven to be adaptively secure (AD), and whether the scheme can be instantiated *without* relying on the random oracle heuristic ($\frac{\text{NO}}{\text{RO}}$). We assume each scheme is instantiated to support policies over a maximum of N users (this is a restriction on the number of inputs to φ, not the number of users that can join the system). We suppress $\mathsf{poly}(\lambda)$ factors when reporting parameter sizes, except when reporting the ciphertext size and the partial decryption size for a pairing-based construction, in which case, we give the *exact* breakdown in terms of the number of base group elements ($\mathbb{G}$), target group elements ($\mathbb{G}_T$), and the number of field elements ($\mathbb{F}$). We write TM to denote a Turing machine, $i\mathcal{O}$ to denote indistinguishability obfuscation [BGI+01], OWF to denote a one-way function, and SSB to denote a somewhere-statistically-binding hash function [HW15]. The threshold schemes all support *dynamic* thresholds where the threshold is specified at encryption time.

SNARK machinery and instead relies on a simple "cross-term cancellation" strategy. This is a common strategy that has featured in numerous pairing-based constructions, including vector commitments [CF13], batch arguments [WW22], (distributed) broadcast encryption [BGW05, KMW23], and (registered) attribute-based encryption [Wat11, HLWW23]. We provide a concrete comparison with previous approaches in Table 1.

Distributed monotone-policy encryption. The same techniques we use to obtain our distributed monotone-policy signature scheme immediately gives a distributed monotone-policy encryption scheme for the same policy family. This is unsurprising given that the Boneh-Boyen signature scheme was derived from the Boneh-Boyen identity-based encryption (IBE) scheme.[2] Previously, constructions of distributed monotone-policy encryption for policies beyond threshold policies required strong tools like indistinguishability obfuscation or witness encryption [ADM+24, DJWW25]. Even for the special case of threshold policies [GKPW24], security relies on the generic bilinear group model and makes use of techniques from proof systems (e.g., sumchecks and low-degree checks). As was the case with our signature scheme, our approach gives a direct algebraic instantiation that avoids any additional machinery. We provide a comparison with previous constructions in Table 2. Finally, we also note that we can view our distributed monotone-policy encryption scheme as a "signature-based witness encryption scheme" [DHMW23, ADM+24], where the associated signature scheme is the Boneh-Boyen signature scheme (see Remark 4.8).

Application to flexible broadcast encryption. An immediate consequence of our distributed monotone-policy encryption scheme is a new construction of flexible broadcast encryption [FWW23]. Flexible broadcast encryption is a generalization of distributed broadcast encryption [WQZD10, BZ14] where users can post their public keys to a public key directory. Thereafter, anyone can encrypt a message to an arbitrary subset of recipients with a short ciphertext.

[2]In fact, our distributed monotone-policy scheme is technically a distributed monotone-policy IBE scheme (see Remark 5.16) and can be viewed as a distributed version of the threshold Boneh-Boyen IBE scheme from [BBH06].

Distributed broadcast encryption schemes have an additional restriction where each user's public key is bound to an index, and moreover, the broadcaster can only encrypt to a set of public keys that occupy *distinct* indices. Thus, some degree of coordination is necessary among users when generating their public key. Flexible broadcast encryption eliminates this constraint and allows the broadcaster to encrypt to any collection of public keys. Previously, the work of [GLWW23] gives a generic approach to lift a distributed broadcast encryption scheme to a flexible scheme with $\omega(\log \lambda)$ overhead in the size of user public keys. On the other hand, the work of [GKPW24] gives the first direct construction, but relies on the generic group model. Our work gives a new construction of flexible broadcast encryption in the plain model where the size of each user's public key and the size of the ciphertext is essentially the same as the best pairing-based distributed broadcast encryption scheme [KMW23] (see Remark 5.17 for the breakdown). Both our scheme and [KMW23] achieve selective security from q-type assumptions in the plain model.

An open problem: weighted thresholds. A notable feature of previous constructions of threshold signatures (as well as encryption) with silent setup [DCX+23, GJM+24, GKPW24] is they support *weighted* threshold policies. The schemes we present in this work support *unweighted* threshold policies. While there is a generic approach to support weighted thresholds via share virtualization, this incurs an overhead proportional to the magnitude of the weights. An interesting open problem is to design a distributed monotone-policy signature (and encryption) scheme that supports weighted thresholds with the same efficiency as our current constructions.

1.2 Technical Overview

In this section, we provide a general overview of our main constructions. We start by describing the basic syntax of a distributed monotone-policy signature scheme:

- **Setup:** In a distributed monotone-policy signature scheme, there is a common reference string crs that users refer to when generating their individual verification key vk and signing key sk. Each user can generate their keys independently of all other users. As in a standard signature scheme, users can sign messages using their signing key sk and the resulting signatures verify with respect to their verification key vk.
- **Preprocessing:** Next, there is a preprocessing algorithm that takes the verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_N$ for an arbitrary set of users together with a signing policy φ. The preprocessing algorithm outputs a succinct verification key vk_φ and an aggregation key ak_φ associated with the policy φ. The policy φ takes as input a subset of users $S \subseteq [N]$ and outputs 1 if the set is authorized and 0 if not. In this work, we focus exclusively on *monotone* policies (i.e., if $\varphi(S) = 1$, then $\varphi(T) = 1$ for all supersets $T \supseteq S$).
- **Signature aggregation:** Given a collection of signatures $\{\sigma_i\}_{i \in S}$ on a common message m that satisfy the aggregation policy (e.g., $\varphi(S) = 1$), the aggregation algorithm uses the aggregation key ak_φ to publicly derive an aggregate signature σ_{agg} on m that verifies with respect to the aggregated verification key vk_φ.

The security and succinctness requirements on a distributed monotone-policy signature scheme are as follows:

- **Security:** The security requirement is the usual notion of unforgeability for aggregate signatures [BGLS03]. The requirement essentially says that an efficient adversary who chooses the signing keys for an *unauthorized* set of users S (i.e., a set $S \subseteq [N]$ where $\varphi(S) = 0$) cannot produce a valid signature on a message m^* that verifies with respect to vk_φ. As usual, the adversary can also request signatures on non-challenge messages $m \neq m^*$ from the honest users.
- **Succinctness:** The efficiency requirement is that the size of the aggregated verification key vk_φ and the size of the aggregate signature σ_{agg} be succinct. Namely, their size should be independent of the number of users in the set S, the total number of users N, or the complexity of the policy φ. Similarly, the running time of the verification algorithm should not depend on the number of users N or the complexity of the policy φ.

We provide the formal syntax and security requirements in Definition 3.1.

Since we do not allow the verification time to grow with the description length of the policy, the aggregation policy must be fixed at preprocessing time (and the policy φ is succinctly embedded into the aggregated verification

key vk_φ). For certain policy families, the policies have a short description (e.g., threshold policies). In such settings, we can consider a *dynamic* setting where the preprocessing algorithm only takes as input the verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_N$ of the users and the policy is determined at verification time. This is the standard setting for the special case of threshold policies [DCX+23, GKPW24]. Our main construction in this work applies for general policies φ, so we fix the policy at preprocessing time. However, if we focus specifically on threshold policies, it is straightforward to adapt our scheme to support dynamic policies. We provide more discussion in Remark 3.2.

Starting point: Boneh-Boyen signatures. Our starting point is the Boneh-Boyen pairing-based signature scheme [BB04a] (derived from the Boneh-Boyen IBE scheme). We start by recalling the construction when instantiated over a prime-order symmetric pairing group. First, a symmetric pairing group consists of two cyclic groups $\mathbb{G}$ and $\mathbb{G}_T$, each of prime order p, and equipped with an efficiently-computable non-degenerate bilinear map $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. In particular, for all $g \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$, it holds that $e(g^a, g^b) = e(g, g)^{ab}$. In the following, we write g to denote a generator of $\mathbb{G}$. We now recall the Boneh-Boyen signature scheme:

- **Key-generation:** The key-generation algorithm samples two group elements $u, v \xleftarrow{R} \mathbb{G}$ and an exponent $\alpha \xleftarrow{R} \mathbb{Z}_p$. The verification key is $\mathsf{vk} = (u, h, e(g,g)^\alpha)$ and the signing key is $\mathsf{sk} = \alpha$.
- **Signing:** A signature on a message $m \in \mathbb{Z}_p$ is a pair $\sigma = (g^r, g^\alpha (u^m h)^r)$, where $r \xleftarrow{R} \mathbb{Z}_p$ is the signing randomness.
- **Verification:** To verify a signature $\sigma = (\sigma_1, \sigma_2)$ on m with respect to verification key $\mathsf{vk} = (u, h, A)$, the verifier checks the following relation:

$$A \stackrel{?}{=} \frac{e(g, \sigma_2)}{e(\sigma_1, u^m h)}. \tag{1.1}$$

  Correctness follows from bilinearity:

$$e(g, \sigma_2) = e(g, g^\alpha (u^m h)^r) = e(g,g)^\alpha \cdot e(g, (u^m h)^r) = A \cdot e(g^r, u^m h) = A \cdot e(\sigma_1, u^m h).$$

It is easy to derive a threshold version of the Boneh-Boyen signature scheme by secret sharing the signing key $\alpha \in \mathbb{Z}_p$. Specifically, let $(\alpha_1, \ldots, \alpha_N)$ be an additive secret sharing of α (i.e., $\alpha = \sum_{\ell \in [N]} \alpha_\ell$). User ℓ holds $\mathsf{sk}_\ell = \alpha_\ell$. The public verification key for the signature scheme is $\mathsf{vk} = (u, h, e(g,g)^\alpha)$. A "partial signature" from User ℓ on a message m is a Boneh-Boyen signature with $\mathsf{sk}_\ell = \alpha_\ell$: namely, $\sigma_\ell = (g^{r_\ell}, g^{\alpha_\ell}(u^m h)^{r_\ell})$, where $r_\ell \xleftarrow{R} \mathbb{Z}_p$. Given N partial signatures $\sigma_1 = (\sigma_{1,1}, \sigma_{1,2}), \ldots, \sigma_N = (\sigma_{N,1}, \sigma_{N,2})$ from the N users, the aggregate signature σ_{agg} on m is simply the product

$$\sigma_{\mathsf{agg}} = \left(\prod_{\ell \in [N]} \sigma_{\ell,1} \,,\, \prod_{\ell \in [N]} \sigma_{\ell,2} \right) = \left(\prod_{\ell \in [N]} g^{r_\ell} \,,\, \prod_{\ell \in [N]} g^{\alpha_\ell} (u^m h)^{r_\ell} \right) = \left(g^{\sum_{\ell \in [N]} r_\ell} \,,\, g^{\sum_{\ell \in [N]} \alpha_\ell} (u^m h)^{\sum_{\ell \in [N]} r_\ell} \right).$$

Since $\sum_{\ell \in [N]} \alpha_\ell = \alpha$, σ_{agg} is a Boneh-Boyen signature on m with respect to the verification key $\mathsf{vk} = (u, h, e(g,g)^\alpha)$ and randomness $\sum_{\ell \in [N]} r_\ell$. This aggregation property is the basis of the [LOS+06] multi-signature.

This approach directly extends to support any policy family that can be described by a linear secret sharing scheme [Bei96]. Formally, a linear secret sharing scheme for a policy over N parties consists of a "share-generating" matrix $\mathbf{M} \in \mathbb{Z}_p^{N \times W}$ where User ℓ is associated with the ℓ^{th} row $\mathbf{m}_\ell^\intercal$ of $\mathbf{M}$.[3] Next, to secret share a value $\alpha \in \mathbb{Z}_p$, the dealer samples a vector $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^W$ where $s_1 = \alpha$ and computes the shares $\boldsymbol{\alpha} = \mathbf{M}\mathbf{s} \in \mathbb{Z}_p^N$. The ℓ^{th} user's share is $\alpha_\ell = \mathbf{m}_\ell^\intercal \mathbf{s}$. The correctness property of a linear secret sharing scheme says that for any authorized set of parties $S \subseteq [N]$, there exists coefficients $\omega_\ell \in \mathbb{Z}_p$ such that $\sum_{\ell \in S} \omega_\ell \alpha_\ell = \alpha$. General threshold policies (e.g., T-out-of-N policies) [Sha79] and monotone Boolean formulas [BL88, LW11] can all be described by a linear secret sharing scheme (see Remark 2.3).

It is easy to extend Boneh-Boyen signatures to any policy with a linear secret sharing scheme by secret sharing the signing key α according to the share-generation matrix $\mathbf{M}$. Concretely, let $\alpha_1, \ldots, \alpha_N$ be the shares of α under $\mathbf{M}$. We associate User ℓ with the signing key α_ℓ. The verification key for the overall signature scheme is $\mathsf{vk} = (u, h, e(g,g)^\alpha)$.

[3] Technically, this is a special case of a "one-use" linear secret sharing scheme (see Definition 2.1). We refer to Remark 3.10 for discussion on how to generalize to many-use linear secret sharing schemes.

As before, a partial signature under User ℓ's signing key is $\sigma_\ell = (g^{r_\ell}, g^{\alpha_\ell}(u^m h)^{r_\ell})$. Given a collection of partial signatures $\{\sigma_\ell\}_{\ell \in S}$ on a message m and an authorized set $S \subseteq [N]$, the aggregator can combine them by computing the linear reconstruction coefficients $\{\omega_\ell\}_{\ell \in S}$ where $\sum_{\ell \in [N]} \omega_\ell \alpha_\ell = \alpha$. The aggregate signature σ_{agg} on m is then

$$\sigma_{\mathsf{agg}} = \left(\prod_{\ell \in S} \sigma_{\ell,1}^{\omega_\ell} \ , \ \prod_{\ell \in S} \sigma_{\ell,2}^{\omega_\ell} \right) = \left(\prod_{\ell \in S} g^{\sum_{\ell \in S} \omega_\ell r_\ell} \ \cdot \ \prod_{\ell \in S} g^{\sum_{\ell \in S} \omega_\ell \alpha_\ell} (u^m h)^{\omega_\ell r_\ell} \right) = (g^r \ , \ g^\alpha (u^m h)^r) \, , \tag{1.2}$$

where $r = \sum_{\ell \in S} \omega_\ell r_\ell$. Once again, this is a Boneh-Boyen signature on m with respect to the verification key $\mathsf{vk} = (u, h, e(g,g)^\alpha)$ and randomness r.

Distributed monotone-policy signatures. While the above approach supports any policy family with a linear secret sharing scheme, it is in the centralized model where a trusted dealer generates the signing key α and distributes the shares $\alpha_1, \ldots, \alpha_N$ to the N parties. In the distributed setting, each user should have the ability to sample their own share $\alpha_i \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ (independently of other users). In this case, it will no longer be the case that any satisfying subset of shares $\{\alpha_\ell\}_{\ell \in S}$ would reconstruct to the *same* target $\alpha = \sum_{\ell \in S} \omega_\ell \alpha_\ell$. Nonetheless, we can still make use of the aggregation structure from Eq. (1.2) as follows:

- The CRS will contain the group elements (u, h) of the Boneh-Boyen signature scheme.
- Each user independently samples their signing key $\mathsf{sk}_\ell = \alpha_\ell \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and publishes $\mathsf{vk}_\ell = e(g,g)^{\alpha_\ell}$ as their public verification component. We can view $(u, h, e(g,g)^{\alpha_\ell})$ as a Boneh-Boyen verification key for User ℓ.
- Let $\mathbf{M}$ be the share-generation matrix for a policy, and suppose that the aggregator has partial signatures $\{\sigma_\ell\}_{\ell \in S}$ on a message m for an accepting set of users. Then, using Eq. (1.2), the aggregator can derive an aggregate signature $\sigma_{\mathsf{agg}} = (\sigma_{\mathsf{agg},1}, \sigma_{\mathsf{agg},2})$ on m with respect to the aggregated verification key $\mathsf{vk}_{\mathsf{agg}} = e(g,g)^{\sum_{\ell \in S} \omega_\ell \alpha_\ell}$.
- The question is how the verifier checks σ_{agg} when it does not know the aggregated verification key $\mathsf{vk}_{\mathsf{agg}} = e(g,g)^{\sum_{\ell \in S} \omega_\ell \alpha_\ell}$. In the centralized setting, the aggregated key always interpolates to the master verification key $e(g,g)^\alpha$ as long as $\{\omega_\ell\}_{\ell \in S}$ is a valid collection of reconstruction coefficients. This is no longer the case in the distributed setting since the α_ℓ's are sampled independently. One approach is to have the the aggregator include the aggregate verification key $\mathsf{vk}_{\mathsf{agg}}$ as part of the aggregate signature. Now, the verifier can validate the aggregate signature as before using Eq. (1.1):

$$\mathsf{vk}_{\mathsf{agg}} = e(g,g)^{\sum_{\ell \in S} \omega_\ell \alpha_\ell} \stackrel{?}{=} \frac{e(g, \sigma_{\mathsf{agg},2})}{e(\sigma_{\mathsf{agg},1}, u^m h)}. \tag{1.3}$$

- This basic approach gives too much power to the aggregator. Namely, it can always choose $\mathsf{vk}_{\mathsf{agg}}$ to satisfy Eq. (1.3). Thus, for security we need to *constrain* the aggregator to only choose $\mathsf{vk}_{\mathsf{agg}}$ that correspond to an aggregate verification key that is derived from an honest subset of verification keys $\{\mathsf{vk}_\ell\}_{\ell \in S}$. The general blueprint from prior works [DCX+23, GJM+24] is to do the following:
  - First, the preprocessing algorithm takes the verification keys $(\mathsf{vk}_1, \ldots, \mathsf{vk}_N)$ for a group of N users and outputs a short digest. The digest is a succinct commitment to the users' verification keys. The digest is part of the verification key associated with the group of users.
  - After the aggregator computes $\mathsf{vk}_{\mathsf{agg}}$, it includes a SNARK proof that $\mathsf{vk}_{\mathsf{agg}}$ was correctly derived from a subset $\{\mathsf{vk}_\ell\}_{\ell \in S}$ of the verification keys (with respect to the digest) and moreover, the set S satisfies the aggregation policy φ.
  - The overall aggregate signature now contains the aggregated verification key $\mathsf{vk}_{\mathsf{agg}}$, the SNARK proof π that $\mathsf{vk}_{\mathsf{agg}}$ was correctly computed, and finally, an aggregate signature σ_{agg} with respect to $\mathsf{vk}_{\mathsf{agg}}$.

  To verify the signature, the verifier first checks that $\mathsf{vk}_{\mathsf{agg}}$ is correctly computed (with respect to the digest) and that σ_{agg} is a valid aggregate signature.

The previous works rely on some type of SNARK to prove that $\mathsf{vk}_{\mathsf{agg}}$ is correctly constructed. This both introduces extra complexity into the system and seemingly necessitates the use of idealized models (or knowledge assumptions) in the security analysis. A main contribution in this work is an algebraic realization of the general template above that fully avoids the need for SNARKs. In particular, we algebraically certify the validity of a purported aggregation key using a cross-term strategy that has proven useful in many pairing-based cryptographic constructions. This leads to a scheme with shorter aggregate signatures and allows us to prove security in the plain model.

In more detail, our goal is to design a mechanism that achieves two goals: (1) certify that an aggregated verification key $\mathsf{vk}_{\mathsf{agg}}$ is correctly computed from $\{\mathsf{vk}_\ell\}_{\ell\in S}$; and (2) the set S satisfies the policy (defined by $\mathbf{M}$). We describe each of these mechanisms below:

- **Commitment to the set of users:** First, we need a mechanism for the aggregator to certify that the aggregated verification key $\mathsf{vk}_{\mathsf{agg}}$ is correctly derived from a subset $S \subseteq [N]$ of the verification keys $\mathsf{vk}_1 = e(g,g)^{\alpha_1}, \ldots, \mathsf{vk}_N = e(g,g)^{\alpha_N}$ associated with the users in the policy. To do this, we first compute a succinct (vector) commitment to $(g^{\alpha_1}, \ldots, g^{\alpha_N})$. Namely, the commitment is

$$z = g^{\sum_{\ell\in[N]} c_\ell \alpha_\ell} \in \mathbb{G} \quad \text{where} \quad c_i \xleftarrow{\mathrm{R}} \mathbb{Z}_p^*.$$

  Note that the commitment z is an element in the base group $\mathbb{G}$, *not* the target group $\mathbb{G}_T$.[4] The element z is part of the verification key associated with the group of users.

- **Certifying an aggregate verification key.** As before, the aggregate verification key associated with a subset $S \subset [N]$ (and reconstruction coefficients $\{\omega_\ell\}_{\ell\in S}$) is $\mathsf{vk}_{\mathsf{agg}} = e(g,g)^{\sum_{\ell\in S}\omega_\ell\alpha_\ell}$. We can certify that $\mathsf{vk}_{\mathsf{agg}}$ is well-formed using a pairing-check. Specifically, suppose we include the elements g^{1/c_ℓ} for all $\ell \in [N]$ as part of the aggregation key. Then, the aggregator can compute

$$\prod_{\ell\in S} e(z, g^{\omega_\ell/c_\ell}) = \prod_{\ell\in S}\prod_{i\in[N]} e(g,g)^{c_i\alpha_i\cdot\omega_\ell/c_\ell} = \prod_{\ell\in S} e(g,g)^{\omega_\ell\alpha_\ell} \cdot \prod_{\ell\in S}\prod_{\substack{i\in[N]\\ i\neq\ell}} e(g,g)^{(c_i/c_\ell)\alpha_i\omega_\ell}. \tag{1.4}$$

  Next, suppose the aggregation key also contains the *cross-terms* $g^{c_i\alpha_i/c_\ell}$ for all $i \neq \ell$. Then the aggregator can compute

$$\gamma_1 = \prod_{\ell\in S}(g^{1/c_\ell})^{\omega_\ell} \quad \text{and} \quad \gamma_2 = \prod_{\ell\in S}\prod_{\substack{i\in[N]\\ i\neq\ell}} (g^{(c_i\alpha_i/c_\ell)})^{\omega_\ell}. \tag{1.5}$$

  Then, Eq. (1.4) becomes

$$e(z,\gamma_1) = \prod_{\ell\in S} e(g,g)^{\omega_\ell\alpha_\ell} \cdot e(g,\gamma_2) = \mathsf{vk}_{\mathsf{agg}} \cdot e(g,\gamma_2).$$

  We can view (γ_1,γ_2) as a proof that $\mathsf{vk}_{\mathsf{agg}}$ is of the form $\mathsf{vk}_{\mathsf{agg}} = e(g,g)^{\sum_{\ell\in S}\omega_\ell\alpha_\ell}$ with respect to some set $S \subseteq [N]$ and reconstruction coefficients $\{\omega_\ell\}_{\ell\in S}$. It remains to show that this is a valid set of reconstruction coefficients.

- **Policy check:** The remaining ingredient is for the aggregator to certify that the set $S \subseteq [N]$ and the reconstruction coefficients $\{\omega_\ell\}_{\ell\in S}$ are valid (with respect to the share-generating matrix $\mathbf{M}$). To do so, we also embed shares of a random element $s \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ (with respect to $\mathbf{M}$) in the digest z and publish $e(g,g)^s$ as part of the verification key. Specifically, let $s_1, \ldots, s_N \in \mathbb{Z}_p$ be a secret sharing of s with respect to $\mathbf{M}$. Then, we define $z = g^{\sum_{\ell\in[N]} c_\ell(s_\ell+\alpha_\ell)}$. With this modification, Eq. (1.4) becomes

$$\begin{aligned}\prod_{\ell\in S} e(z, g^{\omega_\ell/c_\ell}) &= e(g,g)^{\sum_{\ell\in S}\omega_\ell\alpha_\ell} \cdot e(g,g)^{\sum_{\ell\in S}\omega_\ell s_\ell} \cdot \prod_{\ell\in S}\prod_{\substack{i\in[N]\\ i\neq\ell}} e(g,g)^{(c_i/c_\ell)(s_i+\alpha_i)\omega_\ell} \\ &= \mathsf{vk}_{\mathsf{agg}} \cdot e(g,g)^s \cdot \prod_{\ell\in S}\prod_{\substack{i\in[N]\\ i\neq\ell}} e(g,g)^{(c_i/c_\ell)(s_i+\alpha_i)\omega_\ell}.\end{aligned} \tag{1.6}$$

[4] In the construction, the CRS will contain the elements $g^{c_1}, \ldots, g^{c_\ell}$, and User ℓ will publish $g^{c_i\alpha_\ell}$ for all $i \in [N]$ as part of their aggregation hint. The preprocessing algorithm can the use the components of the aggregation hint to construct the commitment z.

The aggregation key will also include the cross-terms $g^{c_i s_i / c_\ell}$ for all $i \neq \ell$. Using the analogous relations as Eq. (1.5), the aggregator can now compute $\gamma_1, \gamma_2 \in \mathbb{G}$ such that

$$e(z, \gamma_1) = \mathsf{vk}_{\mathsf{agg}} \cdot e(g, g)^s \cdot e(g, \gamma_2). \tag{1.7}$$

We can now interpret (γ_1, γ_2) as a proof that $\mathsf{vk}_{\mathsf{agg}}$ is of the form $\mathsf{vk}_{\mathsf{agg}} = e(g, g)^{\sum_{\ell \in S} \omega_\ell \alpha_\ell}$ where $S \subseteq [N]$ and $\{\omega_\ell\}_{\ell \in S}$ is a set of *satisfying* reconstruction coefficients. Once $\mathsf{vk}_{\mathsf{agg}}$ has been certified, the verifier can just check the original Boneh-Boyen aggregate signature (Eq. (1.2)). In the actual construction (described below), we can collapse the signature to just three group elements. This is because the verification key $\mathsf{vk}_{\mathsf{agg}}$ can be computed from (γ_1, γ_2) using Eq. (1.7) so it does not need to be explicitly included as part of the aggregate signature. Moreover, we can merge γ_2 with the component $\sigma_{\mathsf{agg},1}$ from the aggregate signature (since both of these components are paired with g in the verification relation).

We now describe the full construction:

- **Common reference string:** The common reference string consists of the following components:
  - The public components (u, h) for a Boneh-Boyen signature scheme.
  - The elements g^{c_i}, g^{1/c_i} for all $i \in [N]$ and the cross-terms g^{c_i / c_j} for all $i \neq j$. These are used by the aggregator to certify the aggregate verification key.
  - The elements $B = e(g, g)^{s_1}$, $g^{c_i \mathbf{s}}$ for all $i \in [N]$, and the cross terms $g^{c_i \mathbf{s} / c_j}$ where $\mathbf{s} \xleftarrow{\mathrm{R}} \mathbb{Z}_p^W$ and $i \neq j$. This is used to implement the policy check (specifically, if $\mathbf{M}$ is a share-generating matrix, then $\mathbf{m}_\ell^\intercal \mathbf{s}$ is the ℓ^{th} share of s_1 with respect to $\mathbf{M}$).
- **User keys:** Each user samples a Boneh-Boyen signing key $\alpha \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and their verification key is $e(g, g)^\alpha$. Each user also publishes an aggregation hint $g^{c_i \alpha}$ for all $i \in [N]$ and $g^{c_i \alpha / c_j}$ for all $i \neq j$. The user computes the aggregation hint using the g^{c_i} and g^{c_i / c_j} from the CRS.
- **Partial signatures:** Partial signatures in our scheme are vanilla Boneh-Boyen signatures. Namely, to sign a message m using signing key α, the user samples $r \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and outputs $\sigma = (g^r, g^\alpha (u^m h)^r)$.
- **Aggregate verification key:** Suppose we want to associate verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_N$ with share-generating matrix $\mathbf{M}$. Write $\mathsf{vk}_\ell = e(g, g)^{\alpha_\ell}$. The aggregated verification key consists of the group element z where

$$z = \prod_{\ell \in [N]} g^{c_\ell (\mathbf{m}_\ell^\intercal \mathbf{s} + \alpha_\ell)}, \tag{1.8}$$

  This can be computed using $g^{c_\ell \mathbf{s}}$ from the CRS and $g^{c_\ell \alpha_\ell}$ from each user's aggregation hint.
- **Signature aggregation:** Suppose the aggregator has a collection of signatures $\{\sigma_\ell\}_{\ell \in S}$ on a message m for a set $S \subseteq [N]$ that satisfies the policy. Let $\{\omega_\ell\}_{\ell \in S}$ be the associated set of reconstruction coefficients. The aggregate signature σ_{agg} then consists of the Boneh-Boyen aggregate signature with respect to the aggregate verification key $\mathsf{vk}_{\mathsf{agg}} = e(g, g)^{\sum_{\ell \in S} \omega_\ell \alpha_\ell}$ together with the certificate (γ_1, γ_2) that $\mathsf{vk}_{\mathsf{agg}}$ was correctly computed (see Eq. (1.7)). As mentioned above, we can coalesce γ_2 with the Boneh-Boyen aggregate signature. Thus, the aggregator computes

$$\sigma_{\mathsf{agg},1} = \prod_{\ell \in S} \sigma_{\ell,1}^{\omega_\ell} \quad \text{and} \quad \sigma_{\mathsf{agg},2} = \prod_{\ell \in S} \sigma_{\ell,2}^{\omega_\ell} w_\ell^{\omega_\ell} \quad \text{and} \quad \sigma_{\mathsf{agg},3} = \prod_{\ell \in S} (g^{1/c_\ell})^{\omega_\ell}, \tag{1.9}$$

  where $w_\ell = \prod_{i \neq \ell} g^{c_i / c_\ell (\mathbf{m}_i^\intercal \mathbf{s} + \alpha_i)}$ are the cross-terms from Eq. (1.6). Note that the components of w_ℓ can be computed from the CRS and each user's aggregation hint. The final aggregate signature is then $\sigma_{\mathsf{agg}} = (\sigma_{\mathsf{agg},1}, \sigma_{\mathsf{agg},2}, \sigma_{\mathsf{agg},3})$.

- **Signature verification:** The verification key associated with $\mathsf{vk}_1, \ldots, \mathsf{vk}_N$ and the policy $\mathbf{M}$ is the tuple $(u, h, z, e(g,g)^{s_1})$. The verification relation is a combination of the $\mathsf{vk}_{\mathrm{agg}}$ validation check (Eq. (1.7)) with the Boneh-Boyen aggregate signature verification (Eq. (1.3)):

$$\frac{e(z, \sigma_{\mathrm{agg},3})}{e(g,g)^{s_1}} = \frac{e(g, \sigma_{\mathrm{agg},2})}{e(\sigma_{\mathrm{agg},1}, u^m h)}. \tag{1.10}$$

We provide the formal description in Section 3.1.

Proving security. Security of our scheme relies on a q-type assumption (where the size of the assumption scales with the bound on the number of users N appearing in a policy). Our assumption is a generalization of the standard decisional bilinear Diffie-Hellman assumption [Jou00, BF01] which asserts that $e(g,g)^{abt}$ is pseudorandom given (g, g^a, g^b, g^t), where $a, b, t \xleftarrow{R} \mathbb{Z}_p$. In our setting, we define the N-extended bilinear Diffie-Hellman assumption which asserts that $e(g,g)^{abt}$ is pseudorandom given

$$\left(g, g^a, g^b, g^t, \{g^{c_i}, g^{1/c_i}, g^{abc_i}, g^{tc_i}\}_{i\in[N]}, \{g^{c_i/c_j}, g^{abc_i/c_j}\}_{i\neq j\in[N]}\right),$$

where the exponents $c_1, \ldots, c_N \xleftarrow{R} \mathbb{Z}_p^*$. We show that this assumption holds in the generic bilinear group model [Sho97, BBG05] in Appendix A. For proving security of our signature scheme, it is more convenient to use a "search" version of the assumption which stipulates that given

$$\left(g, g^a, g^b, \{g^{c_i}, g^{1/c_i}, g^{abc_i}\}_{i\in[N]}, \{g^{c_i/c_j}, g^{abc_i/c_j}\}_{i\neq j\in[N]}\right), \tag{1.11}$$

it is computationally infeasible to find $\tau_0, \tau_1, \ldots, \tau_N \in \mathbb{G}$ such that

$$e(g,g)^{ab} = e(g, \tau_0) \cdot \prod_{i\in[N]} e(g^{c_i}, \tau_i). \tag{1.12}$$

This can be viewed as a souped-up version of the computational Diffie-Hellman assumption in $\mathbb{G}$. It is not difficult to show that the decisional assumption implies the search assumption (see Lemma 3.8). Our assumption formulation is similar to the decisional parallel bilinear Diffie-Hellman exponent assumption from [Wat11] which was used to construct ciphertext-policy attribute-based encryption.

We prove security of our distributed monotone-policy signature scheme from the N-extended bilinear Diffie-Hellman assumption against *static* adversaries. In this model, the adversary has to commit to the aggregation policy $\mathbf{M}$, the challenge message m^*, and the set of corrupted users $S \subseteq [N]$ at the beginning of the security game. Importantly, however, the adversary can still choose the verification keys vk_ℓ for the corrupted users $\ell \in S$ *after* seeing the common reference string crs and the verification keys vk_ℓ for the honest users $\ell \notin S$. This is essential for capturing "rogue key" attacks [RY07, BDN18]. The adversary can also request signatures on any message $m \neq m^*$ under the keys for the honest users. For simplicity, we assume that the adversary can only choose keys that are in the support of the honest key-generation algorithm; security against such "semi-malicious" adversaries can be lifted to security against malicious adversaries using standard non-interactive zero-knowledge proofs of knowledge (c.f., [RY07]).

The security proof relies on a partitioning or a cancellation proof strategy where we program the challenge message m^* into the public parameters and the challenge policy $\mathbf{M}$ into the verification keys of the honest users. We provide a quick sketch of the general reduction strategy here:

- The reduction takes as input the challenge for the search N-extended Diffie-Hellman assumption from Eq. (1.11). The static adversary also commits to the policy $\mathbf{M}$, the challenge message m^*, and the indices $C \subseteq [N]$ of the corrupted users.
- When generating the CRS, the reduction makes the following implicit assignments:
  - The reduction sets $g^{c_i}, g^{1/c_i}, g^{c_i/c_j}$ in the CRS to be the corresponding terms from the challenge.

- Next, the reduction algorithm implicitly sets $\mathbf{s} = \tilde{\mathbf{s}} + ab \cdot \tilde{\mathbf{w}}$ where $\tilde{\mathbf{s}} \xleftarrow{R} \mathbb{Z}_p^W$ is random and $\tilde{\mathbf{w}}$ is a vector where $\tilde{w}_1 = 1$ and $\mathbf{m}_\ell^\top \tilde{\mathbf{w}} = 0$ for all $\ell \in C$. Such a vector $\tilde{\mathbf{w}}$ always exists whenever C does not satisfy the policy associated with $\mathbf{M}$. Then the reduction computes $B = e(g,g)^{s_1} = e(g,g)^{\tilde{s}_1} \cdot e(g^a, g^b)$. It computes the terms of the form $g^{c_i \mathbf{s}}$ and $g^{c_i \mathbf{s}/c_j}$ using the g^{abc_i} and g^{abc_i/c_j} terms from the challenge.
- To simulate the verification keys of the honest users $\ell \in [N] \setminus C$, the reduction algorithm implicitly sets $\alpha_\ell = \tilde{\alpha}_\ell - ab\mathbf{m}_\ell^\top \tilde{\mathbf{w}}$, where $\tilde{\alpha}_\ell \xleftarrow{R} \mathbb{Z}_p$. The challenger can simulate the user's verification key by computing $e(g,g)^{\alpha_\ell} = e(g,g)^{\tilde{\alpha}_\ell} \cdot e(g^a, g^b)^{\mathbf{m}_\ell^\top \tilde{\mathbf{w}}}$. The reduction can simulate the components of the aggregation hint $g^{c_i \alpha_\ell}$ and $g^{c_i \alpha_\ell / c_j}$ using the g^{abc_i} and g^{abc_i/c_j} terms from the challenge.
- Finally, the reduction algorithm sets the Boneh-Boyen signature components as $u = (g^b) \cdot g^{\tilde{u}}$ and $h = g^{\tilde{h}}/(g^b)^{m^*}$, where $\tilde{u}, \tilde{h} \xleftarrow{R} \mathbb{Z}_p$. This is the same programming strategy as in the original security proof from [BB04a].

- To answer signing queries on a message $m \neq m^*$ under a verification key vk_ℓ where $\ell \notin C$, the reduction uses the classic Boneh-Boyen simulation strategy [BB04a]. Normally, a signature on m under vk_ℓ would be a pair

$$\sigma_\ell = (\sigma_{\ell,1}, \sigma_{\ell,2}) = (g^r, g^{\alpha_\ell}(u^m h)^r) = (g^r, g^{\tilde{\alpha}_\ell} g^{-ab\mathbf{m}_\ell^\top \tilde{\mathbf{w}}} g^{r(\tilde{u}m + \tilde{h} + b(m-m^*))}).$$

  The trouble of course is that the reduction algorithm does not know g^{ab}. To simulate this, the reduction will set the randomness r so as to *cancel* out this problematic term. In particular, the reduction implicitly sets the randomness $r = \tilde{r} + a(m-m^*)^{-1} \mathbf{m}_\ell^\top \tilde{\mathbf{w}}$ where $\tilde{r} \xleftarrow{R} \mathbb{Z}_p$. In this case, consider the exponent for $\sigma_{\ell,2}$:

$$\tilde{\alpha}_\ell - ab\mathbf{m}_\ell^\top \tilde{\mathbf{w}} + r(\tilde{u}m + \tilde{h} + b(m-m^*)) = \tilde{\alpha}_\ell + r(\tilde{u}m + \tilde{h}) + \tilde{r}b(m-m^*).$$

  Since the reduction knows the exponents $\tilde{\alpha}_\ell, \tilde{u}, \tilde{h}, \tilde{r}$ as well as the messages m, m^*, it can now simulate the signature as

$$\sigma_1 = g^r = g^{\tilde{r}} \cdot (g^a)^{(m-m^*)^{-1} \mathbf{m}_\ell^\top \tilde{\mathbf{w}}}$$
$$\sigma_2 = g^{\alpha_\ell}(u^m h)^r = g^{\tilde{\alpha}_\ell} \cdot (g^b)^{\tilde{r}(m-m^*)} \cdot \sigma_1^{\tilde{u}m + \tilde{h}}.$$

- At the end of the game, the adversary outputs the signing keys $\alpha_\ell \in \mathbb{Z}_p$ for the corrupted parties (recall that we work in the semi-malicious model where the adversary must declare the randomness used to generate the keys for the corrupted users) and an aggregate signature $\sigma_{\mathrm{agg}} = (\sigma_{\mathrm{agg},1}, \sigma_{\mathrm{agg},2}, \sigma_{\mathrm{agg},3})$ on the message m^*. If the aggregator is successful, this means

$$e(g,g)^{s_1} \cdot e(g, \sigma_{\mathrm{agg},2}) = e(\sigma_{\mathrm{agg},1}, u^{m^*} h) \cdot e(z, \sigma_{\mathrm{agg},3}), \tag{1.13}$$

  where $z = \prod_{\ell \in [N]} g^{c_\ell (\mathbf{m}_\ell^\top \mathbf{s} + \alpha_\ell)}$. Now, for the parameters chosen by the reduction, we observe the following:

  - $s_1 = \tilde{s}_1 + ab\tilde{w}_1 = \tilde{s}_1 + ab$ since $\tilde{w}_1 = 1$.
  - $u^{m^*} h = g^{m^*(b+\tilde{u}) + \tilde{h} - m^* b} = g^{\tilde{u}m^* + \tilde{h}}$.
  - For the corrupted users $\ell \in C$, we have that $\mathbf{m}_\ell^\top \tilde{\mathbf{w}} = 0$, which means

$$c_\ell(\mathbf{m}_\ell^\top \mathbf{s} + \alpha_\ell) = c_\ell(\mathbf{m}_\ell^\top(\tilde{\mathbf{s}} + ab\tilde{\mathbf{w}}) + \alpha_\ell) = c_\ell(\mathbf{m}_\ell^\top \tilde{\mathbf{s}} + \tilde{\alpha}_\ell),$$

    where we define $\tilde{\alpha}_\ell = \alpha_\ell$.
  - For the honest users $\ell \in [N] \notin C$, the reduction sets $\alpha_\ell = \tilde{\alpha}_\ell - ab\mathbf{m}_\ell^\top \tilde{\mathbf{w}}$ so

$$c_\ell(\mathbf{m}_\ell^\top \mathbf{s} + \alpha_\ell) = c_\ell(\mathbf{m}_\ell^\top(\tilde{\mathbf{s}} + ab\tilde{\mathbf{w}}) + \tilde{\alpha}_\ell - ab\mathbf{m}_\ell^\top \tilde{\mathbf{w}}) = c_\ell(\mathbf{m}_\ell^\top \tilde{\mathbf{s}} + \tilde{\alpha}_\ell).$$

  Observe then that we can rewrite Eq. (1.13) as

$$e(g,g)^{ab} \cdot e(g,g)^{\tilde{s}_1} \cdot e(g, \sigma_{\mathrm{agg},2}) = e(\sigma_{\mathrm{agg},1}, g^{\tilde{u}m^* + \tilde{h}}) \cdot \prod_{\ell \in [N]} e(g^{c_\ell}, \sigma_{\mathrm{agg},3}^{\mathbf{m}_\ell^\top \tilde{\mathbf{s}} + \tilde{\alpha}_\ell}).$$

  Since the reduction algorithm knows the exponents $\tilde{u}, \tilde{h}, \tilde{\mathbf{s}}, \tilde{\mathbf{w}}, \tilde{\alpha}_\ell, \mathbf{m}_\ell, m^*$, it can rearrange the above relation to obtain a solution to Eq. (1.12).

We refer to the proof of Theorem 3.7 for the formal description.

Reducing CRS size using powers. One drawback of the above distributed monotone-policy signature scheme is the CRS has size $O(N^2W)$, where N is the maximum number of users in the policy and W is the width of the share-generating matrix associated with a policy. The quadratic dependence on N is due to the cross terms g^{c_i/c_j} in the CRS. A standard approach to reduce the CRS size is to set $c_i = c^i$. In this case, $g^{c_i/c_j} = g^{c^{i-j}}$. Instead of publishing $O(N^2)$ cross-terms in the CRS, we now only need $O(N)$ such terms. Security in this case would reduce to a variant of the N-extended bilinear Diffie-Hellman assumption where the c_i's are replaced by powers (see Assumption 3.12).[5] Using powers, we obtain a construction where the size of the CRS is $O(NW)$. We describe this construction and the security analysis in Section 3.2. Note that for many policies of interest (including threshold policies), $W = O(N)$, so the overall CRS size is still quadratic. Later, we describe how to specialize our construction to the specific setting of threshold policies to obtain a construction with a linear-size CRS.

Distributed monotone-policy encryption. The approach we described thus far immediately generalizes to a distributed monotone-policy *encryption* scheme [GKPW24, DJWW25]. In this setting, users can independently generate a public/secret key-pair $(\mathsf{pk}, \mathsf{sk})$ and post the public key pk to a directory. Thereafter, an encrypter can encrypt to an arbitrary set of user public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$ together with an encryption policy. Only an authorized set of users can come together and decrypt the ciphertext. To derive the encryption scheme from our signature scheme, we proceed as follows:

- **Common reference string:** The CRS is the same as that of the distributed monotone-policy signature scheme.
- **User keys:** User key generation is also the same as in the signature scheme. The public key is the verification key while the secret key is the signing key.
- **Aggregated encryption key:** Given a set of users with public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$ and an encryption policy defined by a share-generating matrix $\mathbf{M}$, the preprocessing algorithm constructs the aggregated verification key z as in Eq. (1.8). The public encryption key is then (u, h, B, z), where (u, h) is the Boneh-Boyen verification key and $B = e(g, g)^{s_1}$ from the CRS.
- **Ciphertext:** To encrypt a message $\mu \in \mathbb{G}_T$, the encrypter samples a tag $\tau \xleftarrow{R} \mathbb{Z}_p$ and encryption randomness $t \xleftarrow{R} \mathbb{Z}_p$. The ciphertext is
$$\mathsf{ct} = (\tau\ ,\ B^t \cdot \mu\ ,\ g^t\ ,\ (u^\tau h)^t\ ,\ z^t),$$
- **Partial decryption:** The partial decryption from User ℓ on ct is simply a signature σ_ℓ on the tag τ.
- **Decryption:** Given the partial decryptions $\{\sigma_i\}_{i \in S}$ for an accepting subset $S \subseteq [N]$, we can aggregate the signatures on the tag τ to obtain an aggregate signature $\sigma_{\mathsf{agg}} = (\sigma_{\mathsf{agg},1}, \sigma_{\mathsf{agg},2}, \sigma_{\mathsf{agg},3})$ that satisfies Eq. (1.10):
$$B = e(g, g)^{s_1} = \frac{e(z, \sigma_{\mathsf{agg},3}) \cdot e(\sigma_{\mathsf{agg},1}, u^\tau h)}{e(g, \sigma_{\mathsf{agg},2})}.$$
By bilinearity, this means that
$$B^t = \frac{e(z^t, \sigma_{\mathsf{agg},3}) \cdot e(\sigma_{\mathsf{agg},1}, (u^\tau h)^t)}{e(g^t, \sigma_{\mathsf{agg},2})}.$$
This immediately provides a mechanism to recover the message μ from ct and σ_{agg}.

The relationship between our encryption scheme and signature scheme is analogous to the relationship between the Boneh-Boyen IBE scheme and the Boneh-Boyen signature scheme. Indeed, we can also view our encryption scheme as a distributed version of the threshold Boneh-Boyen IBE scheme [BBH06] (see Remark 5.16). Here, the tags in the ciphertext would correspond to the identity.

We can also view our scheme as a "signature-based witness encryption" scheme [DHMW23, GKPW24, ADM+24] for the Boneh-Boyen signature scheme (see Remark 4.8). Namely, we can view our ciphertext as a witness encryption [GGSW13] of the message μ where the associated decryption key is an aggregate Boneh-Boyen signature on the tag τ associated with the ciphertext. Security of our scheme relies on the same N-extended bilinear Diffie-Hellman assumption. We refer to Section 4 for the formal definition, construction, and analysis of our distributed monotone-policy encryption scheme.

[5] Technically, there are several additional modifications we need to make to obtain a viable assumption (see Remark 3.13).

Specializing to threshold policies. Our construction of distributed monotone-policy signatures and encryption capture threshold policies as a special case (since threshold policies can be described by a simple linear secret sharing scheme). However, if we focus exclusively on the special case of threshold policies, we can obtain constructions that have the following stronger functionality and efficiency properties:

- **Quasi-linear-size CRS:** Instantiating the constructions above for the case of threshold policies leads to a scheme where the size of the CRS grows *quadratically* with the number of users (since the width W of the share-generating matrix for a general threshold is $W = N$). For the special case of threshold policies, it suffices to consider a scheme that only supports a *single* fixed policy which we hard-code into the CRS. This allows us to obtain schemes where the size of the CRS scales *quasi-linearly* with the maximum number of parties N.
- **Dynamic thresholds:** In the constructions thus far, we have considered the setting where the preprocessing algorithm outputs a verification key or encryption key for a *fixed* policy $\mathbf{M}$ over a group of users. For the special case of threshold policies, we can consider a more general notion where the threshold T is determined *dynamically* at verification time (in the setting of signatures) or at encryption time (in the setting of encryption). A trivial way to achieve this is to have the preprocessing algorithm output a key for each of the N possible thresholds $T = 1, \ldots, N$. The goal is to support dynamic thresholds without incurring an $O(N)$ blowup in the size of the verification or encryption key. Efficient support for dynamic thresholds is a key requirement of the notion of threshold signatures with silent setup [DCX+23, GJM+24] and threshold encryption with silent setup [GKPW24].

We now describe how we adapt our approach to support dynamic threshold policies with a quasi-linear-size CRS. For ease of exposition, we describe our approach for the case of signatures, though the same approach works for threshold encryption. As usual, let N be the maximum number of users appearing in a policy.

- **Fixing the policy as part of the CRS.** In the distributed monotone-policy signature scheme, the preprocessed verification key for a group of N users (with secret keys $\alpha_1, \ldots, \alpha_N$) and a policy $\mathbf{M}$ is the element $z = \prod_{\ell \in [N]} g^{c_\ell (\mathbf{m}_\ell^\top \mathbf{s} + \alpha_\ell)}$. In order to support an *arbitrary* policy, the CRS must include $g^{c_i \mathbf{s}}$, which leads to a CRS size that is $\Omega(NW)$, where W is the width of the policy. Suppose instead however that the policy is *known* at setup time. Then, the CRS could instead contain the *precomputed* component $z_0 = \prod_{\ell \in [N]} g^{c_\ell \mathbf{m}_\ell^\top \mathbf{s}}$. In this case, once the group has been chosen, the preprocessing algorithm can compute

$$z = z_0 \cdot \prod_{\ell \in [N]} g^{c_\ell \alpha_\ell} = \prod_{\ell \in [N]} g^{c_\ell (\mathbf{m}_\ell^\top \mathbf{s} + \alpha_\ell)}.$$

  Similarly, the setup algorithm can also precompute the cross-terms $w_{\ell,0} = \prod_{i \neq \ell} g^{c_i \mathbf{m}_i^\top \mathbf{s} / c_\ell}$ and include these as part of the CRS; these are used to compute the w_ℓ terms in Eq. (1.9). With this modification, the size of the CRS no longer depends on the width W of the share-generating matrix.

- **Introducing dummy users.** To support dynamic thresholds involving up to N users, we generate the CRS for a *fixed* N-out-of-$(2N-1)$ policy. Specifically, let $\mathbf{M} \in \mathbb{Z}_p^{(2N-1) \times N}$ be the share-generation matrix associated with this threshold policy (see Eq. (2.1) for the explicit description). The first N rows are associated with the N users in the group (with signing keys $\alpha_1, \ldots, \alpha_N$), whereas the remaining $N-1$ rows correspond to *dummy* users. The setup algorithm associates a random (and secret) signing key $\gamma_\ell \xleftarrow{R} \mathbb{Z}_p$ with each dummy user $\ell \in [N+1, 2N-1]$ and includes their verification key $\mathsf{vk}_\ell = e(g,g)^{\gamma_\ell}$ as well as the dummy user's aggregation hint in the CRS (i.e., the values $g^{c_i \gamma_\ell}$ and $g^{c_i \gamma_\ell / c_j}$) as part of the CRS. Now, take any $K \in [0, N-1]$ and suppose the preprocessing algorithm computes the verification component as

$$z = z_0 \cdot \prod_{\ell \in [N]} g^{c_\ell \alpha_\ell} \cdot \prod_{\ell \in [N+1, N+K]} g^{c_\ell \gamma_\ell} = \prod_{\ell \in [2N-1]} g^{c_\ell (\mathbf{m}_\ell^\top \mathbf{s} + \zeta_\ell)} \quad \text{where} \quad \zeta_\ell = \begin{cases} \alpha_\ell & \ell \in [N] \\ \gamma_\ell & \ell \in [N+1, N+K] \\ 0 & \ell > N+K. \end{cases}$$

  By definition, z can be viewed as the verification key associated with the $2N-1$ users whose signing keys are $(\alpha_1, \ldots, \alpha_N, \gamma_{N+1}, \ldots, \gamma_{N+K}, 0, \ldots, 0)$. Essentially, the signing keys for the $(N-K-1)$ users $\ell > N+K$ are

public (and set to 0). This means the aggregator essentially gets signatures from these users for free. Since the threshold is fixed to N and $(N-K-1)$ signatures are given out for free, we can view z as enforcing a threshold policy for threshold $T = N - (N-K-1) = K+1$. Thus, for each threshold T, the verifier itself can derive the verification key z associated with T. To implement this, we include $g^{c_\ell \gamma_\ell}$ in the verification key and have the verifier compute the desired z based on the threshold. Unfortunately, implementing this naïvely leads to a verification key of size $O(N)$ since we need an element for each dummy user.

- **Coalescing dummy users using powers-of-two.** Instead of giving out the keys for the dummy users $g^{c_\ell \gamma_\ell}$ individually (which leads to a verification key of size $O(N)$), we instead give out pre-multiplied sets. Specifically, define the interval $X_j = [N + 2^{j-1}, N + 2^j - 1]$ of size 2^j. For each interval, we precompute the element $y_j = \prod_{k \in X_j} g^{c_k \gamma_k}$. For a threshold $T = K+1$, the verifier needs to compute a verification key that includes exactly K dummy keys. This corresponds to taking a product of an appropriate subset of the y_j's (based on the binary representation of K). By construction, we only need to publish $\lceil \log N \rceil$ group elements to support an arbitrary threshold $T \in [N]$. For correctness, we also need to give out the aggregation hints (i.e., cross terms) associated with each set of dummy users. When the c_i's are powers, this adds an additional $O(N \log N)$ elements to the CRS (the aggregation hint for each block of dummy users is $O(N)$). This leads to a scheme that supports dynamic thresholds with a quasi-linear-size CRS.

We provide the full description and analysis of our threshold signature scheme with silent setup in Section 5.1. Then, in Section 5.2, we show how the same idea applies equally well to obtain a threshold encryption scheme with silent setup.

1.3 Additional Related Work

The goals of distributed monotone-policy signatures and distributed monotone-policy encryption are to remove trust assumptions from advanced cryptographic notions. This has been the focus of a number of recent works studying notions like registration-based encryption [GHMR18, GHM+19, GV20, CES21, GKMR23, DKL+23, FKdP23], registered attribute-based encryption [HLWW23, FWW23, ZZGQ23, AT24, GLWW24, LWW25, CHW25, WW25, ZZC+25, GHK+25], registered functional encryption [FFM+23, DPY24], and distributed broadcast encryption [WQZD10, BZ14, KMW23, FWW23, GKPW24, CW24, CHW25, WW25].

Generic approaches for constructing distributed monotone-policy signatures. Earlier works [DCX+23, GJM+24] observed a generic approach for constructing threshold signatures with silent setup (and more generally, distributed monotone-policy signatures) using SNARKs for NP. The idea is simple: each user publishes their verification key, and the aggregated verification key for a set of verification key $\mathsf{vk}_1, \ldots, \mathsf{vk}_N$ is a vector commitment com to $(\mathsf{vk}_1, \ldots, \mathsf{vk}_N)$. Given a collection of signatures $\{\sigma_i\}_{i \in S}$ on a message m for some subset of users $S \subseteq [N]$, the aggregate signature is a SNARK proof of knowledge of the following:

- a set $S \subseteq [N]$ where $\varphi(S) = 1$; and
- for all $i \in S$, a verification key vk_i and a signature σ_i such that σ_i is a valid message on m with respect to the verification vk_i and vk_i is the i^{th} verification key associated with the commitment com.

The drawback of using a general-purpose SNARK is that it relies on strong non-falsifiable assumptions or idealized models, and moreover, introduces substantial concrete costs into the system (specifically, the aggregation cost is expensive due to having to generate SNARK proofs). On the flip side, an appealing feature of these constructions is they have very short aggregate signatures (e.g., just 3 group elements if we instantiate with [Gro16]). Previous works on threshold signatures with silent setup [DCX+23, GJM+24] designed a specialized SNARK tailored to threshold policies to obtain a scheme with faster aggregation, but in exchange, longer signatures.

The work of [BCJP24] effectively shows that one can replace the SNARK for NP in the above template with a monotone-policy batch argument (BARG) [BBK+23] instead. In a monotone-policy BARG for an NP relation $\mathcal{R}$ and a monotone predicate $\varphi : \{0,1\}^N \to \{0,1\}$, the prover can cerifty that it knows witnesses $w_1, \ldots, w_N$ associated with a batch of statements $x_1, \ldots, x_N$ such that $\varphi(\mathcal{R}(x_1, w_1), \ldots, \mathcal{R}(x_N, w_N)) = 1$. The size of the proof scales polylogarithmically with the number of instances N (and polynomially with the size of the Boolean circuit computing the relation

$\mathcal{R}$). Whereas a vanilla batch argument [BHK17, KPY19] requires the prover certify that all N statements are true, a monotone-policy batch argument only requires the prover to certify that a certain subset $S \subseteq [N]$ of the statements are true, so long as the subset S satisfies the predicate φ. A line of recent work [BBK+23, NWW24, NWW25] has shown how to build monotone-policy BARGs from most standard number-theoretic assumptions. In particular, this approach provides another pairing-based approach for building distributed monotone-policy signatures. We include a comparison of the asymptotics of this scheme to other schemes in Table 1. Notably, the size of the aggregate signature in schemes following this template necessarily contains a $\mathsf{poly}(\lambda)$ number of group elements. In contrast, the schemes we develop in this work have signatures that contain a constant number of group elements (in fact, the same signature size as the scheme obtained via general-purpose SNARKs).

2 Preliminaries

We write λ to denote the security parameter. For an integer $n \in \mathbb{N}$, we write $[n]$ to denote the set $\{1, \ldots, n\}$. For a set S, we write 2^S to denote the power set of S (i.e., the set containing all subsets of S). For a set S, we write $\{x_i\}_{i \in S}$ to denote the set of tuples $\{(i, x_i) : i \in S\}$; in other words, each index $i \in S$ is mapped to a value x_i. For a finite set S, we write $x \xleftarrow{R} \mathcal{S}$ to denote a uniform random sample from S. For a distribution $\mathcal{D}$, we write $x \leftarrow \mathcal{D}$ to denote a sample from $\mathcal{D}$. For a randomized algorithm Alg, we write $\mathsf{Alg}(x; r)$ to denote the output of Alg on input x with randomness r.

We write $\mathsf{poly}(\lambda)$ to denote a function that is bounded by a fixed polynomial in λ. We write $\mathsf{negl}(\lambda)$ to denote a negligible function (i.e., a function that is $o(\lambda^{-c})$ for all $c \in \mathbb{N}$). We say an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input. We say two distributions $\mathcal{D}_0 = \{\mathcal{D}_{0,\lambda}\}_{\lambda \in \mathbb{N}}$ and $\mathcal{D}_1 = \{\mathcal{D}_{1,\lambda}\}_{\lambda \in \mathbb{N}}$ are computationally indistinguishable if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[\mathcal{A}(1^\lambda, x) = 1 : x \leftarrow \mathcal{D}_{0,\lambda}] - \Pr[\mathcal{A}(1^\lambda, x) = 1 : x \leftarrow \mathcal{D}_{1,\lambda}]| = \mathsf{negl}(\lambda).$$

We say that $\mathcal{D}_0$ and $\mathcal{D}_1$ are statistically indistinguishable if their statistical distance is bounded by $\mathsf{negl}(\lambda)$.

Linear secret sharing and access policies. Let S be a set. An access policy over S is a predicate $\varphi : 2^S \to \{0, 1\}$. We say a set $T \subseteq S$ is authorized if $\varphi(T) = 1$. We say the policy is monotone if for all $T_1 \subseteq T_2$, whenever $\varphi(T_1) = 1$, it holds that $\varphi(T_2) = 1$. Next, we recall the notion of a linear secret sharing scheme for a policy φ.

Definition 2.1 (Linear Secret Sharing Scheme [Bei96, adapted])**.** Let S be a set and $\varphi : 2^S \to \{0, 1\}$ be an access policy over S. A linear secret sharing scheme for φ over $\mathbb{Z}_p$ is a pair $(\mathbf{M}, \rho)$, where $\mathbf{M} \in \mathbb{Z}_p^{N \times W}$ is the share-generating matrix and $\rho : [N] \to S$ is a row-labeling function. We refer to W as the width of the share-generation matrix. The pair $(\mathbf{M}, \rho)$ satisfy the following properties:

- **Share generation:** To share a secret $s \in \mathbb{Z}_p$, sample $v_2, \ldots, v_N \xleftarrow{R} \mathbb{Z}_p$ and define the vector $\mathbf{v} = [s, v_2, \ldots, v_N]^\intercal$. Then, $\mathbf{u} = \mathbf{M}\mathbf{v}$ is the vector of shares where $u_i \in \mathbb{Z}_p$ belongs to party $\rho(i)$ for each $i \in [N]$.
- **Share reconstruction:** For every $T \subseteq S$ where $\varphi(T) = 1$, there exists a vector $\boldsymbol{\omega} \in \mathbb{Z}_p^N$, where $\omega_i = 0$ whenever $\rho(i) \notin T$ such that $\boldsymbol{\omega}^\intercal \mathbf{M} = \mathbf{e}_1^\intercal$. Here, $\mathbf{e}_1^\intercal = [1, 0, \ldots, 0]$ is the first elementary basis vector.
- **Perfect hiding for unauthorized sets:** For every $T \subseteq S$ where $\varphi(T) = 0$, there exists a vector $\tilde{\mathbf{w}} \in \mathbb{Z}_p^N$ such that $\mathbf{e}_1^\intercal \tilde{\mathbf{w}} = 1$ and $\mathbf{m}_i^\intercal \tilde{\mathbf{w}} = \mathbf{0}$ whenever $\rho(i) \in T$, and $\mathbf{m}_i^\intercal$ denotes the i^{th} row of $\mathbf{M}$.

Definition 2.2 (One-Use Linear Secret Sharing Scheme)**.** Let $\varphi : 2^S \to \{0, 1\}$ be an access policy with an associated linear secret sharing scheme $(\mathbf{M}, \rho)$ over $\mathbb{Z}_p$. We say φ is a one-use linear secret sharing scheme if the function ρ is injective. Without loss of generality, when considering one-use linear secret sharing schemes, we take $S = [N]$ (and assume a canonical bijection from S to $[N]$), the share-generation matrix to be $\mathbf{M} \in \mathbb{Z}_p^{N \times W}$, and the row-labeling function $\rho : [N] \to [N]$ to be the identity map. We say φ is K-use if every element of S has at most K preimages under φ (i.e., each element of S is associated with at most K rows of $\mathbf{M}$).

Remark 2.3 (Policy Families with Linear Secret Sharing Schemes)**.** We recall two important families of policies that have linear secret sharing schemes.

- **Monotone Boolean formulas:** The work of [BL88] gives an implicit linear secret sharing scheme over any field $\mathbb{Z}_p$ for monotone Boolean formula policies. We refer to [LW11, Appendix G] for an explicit description of the share-generating matrix. Moreover, if every variable appears at most once in the Boolean formula, then the resulting linear secret sharing scheme is one-use.
- **Threshold policies:** The classic Shamir secret sharing [Sha79] for threshold policies can also be described by a one-use linear secret sharing scheme over a field $\mathbb{Z}_p$ where $p > N$ (and N is the number of parties). In this case, the share-generation matrix corresponds to a Vandermonde matrix. Concretely, a T-out-of-N threshold policy can be described by the following share-generation matrix:

$$\mathbf{M} = \begin{bmatrix} 1 & \omega_1 & \omega_1^2 & \cdots & \omega_1^{T-1} \\ 1 & \omega_2 & \omega_2^2 & \cdots & \omega_2^{T-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_N & \omega_N^2 & \cdots & \omega_N^{T-1} \end{bmatrix} \in \mathbb{Z}_p^{N \times T}, \tag{2.1}$$

  where $\omega_1, \ldots, \omega_N \in \mathbb{Z}_p$ are distinct non-zero values. By default, we take $\omega_i = i$.

Prime order pairing groups. Next, we recall the notion of a prime-order pairing group.

Definition 2.4 (Prime-Order Bilinear Group)**.** A symmetric prime-order bilinear group generator is an efficient algorithm GroupGen that takes as input the security parameter 1^λ and outputs the description $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, e)$ of a bilinear group where $\mathbb{G}$, $\mathbb{G}_T$ are cyclic groups of prime order $p \geq 2^\lambda$ and $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a non-degenerate bilinear map. The group operation in $\mathbb{G}$ and $\mathbb{G}_T$ as well as the pairing e must be efficiently-computable. Moreover, we assume there is an efficient algorithm to sample a random element from $\mathbb{G}$. Finally, there is a function $p(\lambda)$ such that for all $\lambda \in \mathbb{N}$, the GroupGen(1^λ) algorithm outputs a group of prime order $p = p(\lambda)$. In addition, there is an efficient algorithm that takes as input 1^λ and outputs $p(\lambda)$.

Vector notation. For a vector $\mathbf{v} = (v_1, \ldots, v_n)$ and a group element $g \in \mathbb{G}$, we write $g^{\mathbf{v}} \in \mathbb{G}^n$ to denote the vector of group elements $(g^{v_1}, \ldots, g^{v_n})$. Given a vector $\mathbf{u} \in \mathbb{Z}_p^n$ and $g^{\mathbf{v}} \in \mathbb{G}^n$, we can efficiently compute $g^{\mathbf{u}^\intercal \mathbf{v}} \coloneqq \prod_{i \in [n]} (g^{v_i})^{u_i}$.

3 Distributed Monotone-Policy Signatures

In this section, we show how to construct a distributed monotone-policy signature scheme that supports any policy family that has a one-use linear secret-sharing scheme (e.g., which includes threshold policies and monotone Boolean formulas; see Remark 2.3).[6]

Defining distributed monotone-policy signatures. We start with the formal definition of a distributed monotone-policy signature scheme. In this setting, users independently sample their own signing key sk_i and verification key vk_i. The users can publish their verification keys to a public bulletin board. Thereafter, anyone can publicly aggregate a set of verification keys $\{\mathsf{vk}_i\}_{i \in S}$ together with an access policy $\varphi\colon 2^S \to \{0,1\}$ into an aggregation key ak_φ together with a short verification key vk_φ. Given a message m, a set of signatures $\{\sigma_i\}_{i \in T}$ where σ_i is a valid signature on m with respect to vk_i for all $i \in T$ and moreover, $\varphi(T) = 1$, an aggregator can derive an aggregate signature σ_{agg} on m that verifies with respect to vk_φ. Importantly, both the size of the aggregated verification key vk_φ as well as the size of the aggregate signature σ_{agg} should be short (independent of the description size of the policy φ). Our notion generalizes the earlier notion of threshold signatures with silent setup [DCX+23, GJM+24], which corresponds to the special case of (weighted) threshold policies.[7]

A key feature in distributed monotone-policy signatures is that users choose their verification keys independently of other users. Thus, when defining correctness and security for such a scheme, the most natural notion would require

[6] As we discuss in Remark 3.10, we can relax the one-use restriction using standard virtualization techniques [LW11]. Virtualization increases user public keys and the size of individual (pre-aggregated) signatures by a factor of K where K is a bound on the number of times a user's share appears in the policy. However, the final signature size is unaffected.

[7] The work of [GJM+24] also describes an extension to general access policies using general-purpose SNARKs for NP.

correctness and security to hold even if keys are adversarially-chosen. To simplify the exposition, in this work, we give our definitions in a simpler *semi-malicious* model, where we require correctness and security to hold when all of the user-chosen verification keys are in the support of the honest key-generation algorithm. This type of model is sometimes referred to as the *registered key model* [RY07]. As we discuss more in Remark 3.2, it is straightforward to upgrade a scheme with semi-malicious correctness and security to one that that is correct and secure against adversarially-chosen keys using non-interactive zero-knowledge proofs [GMR85, BFM88].

Definition 3.1 (Distributed Monotone-Policy Signatures)**.** Let λ be a security parameter and κ be a policy-family parameter. Let $\Phi = \{\Phi_\kappa\}_{\kappa\in\mathbb{N}}$ be a family of monotone policies. Let $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ be a message space. A distributed monotone-policy signature scheme for policy-space Φ and message-space $\mathcal{M}$ consists of a tuple of efficient algorithms $\Pi_{\mathsf{DMPS}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PartialVerify}, \mathsf{Preprocess}, \mathsf{Aggregate}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^\kappa) \to \mathsf{crs}$: On input the security parameter λ and the policy-family parameter κ, the setup algorithm outputs the common reference string crs.
- $\mathsf{KeyGen}(\mathsf{crs}) \to (\mathsf{vk}, \mathsf{ht}, \mathsf{sk})$: On input the common reference string crs, the key-generation algorithm outputs the verification key vk, an aggregation hint ht, and a signing key sk.
- $\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$: On input a signing key sk and a message $m \in \mathcal{M}$, the signing algorithm outputs a partial signature σ.
- $\mathsf{PartialVerify}(\mathsf{vk}, m, \sigma) \to b$: On input a verification key vk, a message $m \in \mathcal{M}$, and a signature σ, the partial-verification algorithm outputs a bit $b \in \{0,1\}$. This algorithm is deterministic.
- $\mathsf{Preprocess}(\mathsf{crs}, \varphi, \{(\mathsf{vk}_\ell, \mathsf{ht}_\ell)\}_{\ell\in[N]}) \to (\mathsf{vk}_\varphi, \mathsf{ak}_\varphi)$: On input the common reference string crs, a policy $\varphi\colon 2^{[N]} \to \{0,1\}$, and N verification keys vk_ℓ together with their aggregation hints ht_ℓ, the preprocessing algorithm outputs an aggregated verification key vk_φ and an aggregation key ak_φ. This algorithm is deterministic.
- $\mathsf{Aggregate}(\mathsf{ak}_\varphi, \{\sigma_\ell\}_{\ell\in S}) \to \sigma_{\mathsf{agg}}$: On input the aggregation key ak_φ and a set of signatures σ_ℓ, the aggregation algorithm outputs an aggregate signature σ_{agg}. This algorithm is deterministic.
- $\mathsf{Verify}(\mathsf{vk}_\varphi, m, \sigma_{\mathsf{agg}}) \to b$: On input an aggregate verification key vk_φ, a message $m \in \mathcal{M}$, and an aggregate signature σ_{agg}, the verification algorithm outputs a bit $b \in \{0,1\}$. This algorithm is deterministic.

We require Π_{DMPS} satisfy the following properties:

- **Signing correctness:** For all $\lambda, \kappa \in \mathbb{N}$ and all messages $m \in \mathcal{M}_\lambda$,

$$\Pr\left[\mathsf{PartialVerify}(\mathsf{crs}, m, \sigma) = 1 : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa) \\ (\mathsf{vk}, \mathsf{ht}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{crs}) \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m) \end{array}\right] = 1.$$

- **Aggregation correctness:** For all $\lambda, \kappa \in \mathbb{N}$, all messages $m \in \mathcal{M}_\lambda$, all policies $\varphi \in \Phi_\kappa$ where $\varphi\colon 2^{[N]} \to \{0,1\}$, all sets $S \subseteq [N]$ where $\varphi(S) = 1$, all crs in the support of $\mathsf{Setup}(1^\lambda, 1^\kappa)$, all $\{(\mathsf{vk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell)\}_{\ell\in[N]}$ where $(\mathsf{vk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell)$ is in the support of $\mathsf{KeyGen}(\mathsf{crs})$ for all $\ell \in [N]$, and all signatures $\{\sigma_\ell\}_{\ell\in S}$ where $\mathsf{PartialVerify}(\mathsf{crs}, m, \sigma_\ell) = 1$ for all $\ell \in S$,

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}_\varphi, m, \sigma_{\mathsf{agg}}) = 1 : \begin{array}{c} (\mathsf{vk}_\varphi, \mathsf{ak}_\varphi) = \mathsf{Preprocess}(\mathsf{crs}, \varphi, \{(\mathsf{vk}_\ell, \mathsf{ht}_\ell)\}_{\ell\in[N]}) \\ \sigma_{\mathsf{agg}} \leftarrow \mathsf{Aggregate}(\mathsf{ak}_\varphi, \{\sigma_\ell\}_{\ell\in S}) \end{array}\right] = 1.$$

- **Static unforgeability:** For a security parameter λ and an adversary $\mathcal{A}$, we define the static unforgeability game as follows:

  1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the policy parameter 1^κ together with a policy $\varphi \in \Phi_\kappa$. In addition, algorithm $\mathcal{A}$ commits to a set of corrupted users $C \subseteq [N]$ and a challenge message $m^* \in \mathbb{Z}_p$.

2. The challenger checks that $\varphi(C) = 0$. If not, the challenger halts with output $b = 0$.
3. The challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa)$. Then, for each index $\ell \in [N] \setminus C$, the challenger samples a key $(\mathsf{vk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell) \leftarrow \mathsf{KeyGen}(\mathsf{crs})$. It gives crs together with $\{(\mathsf{vk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [N] \setminus C}$ to $\mathcal{A}$.
4. Algorithm $\mathcal{A}$ can now make signing queries by specifying an index $\ell \in [N] \setminus C$ and a message $m \neq m^*$. The challenger replies to each query with $\mathsf{Sign}(\mathsf{sk}_\ell, m)$.
5. Once $\mathcal{A}$ is finished making queries, it specifies the key-generation randomness $\rho_\ell \in \{0,1\}^*$ used to generate the keys for each of the corrupted users $\ell \in C$. The adversary also outputs its forgery σ_{agg}.
6. For each $\ell \in C$, the challenger computes $(\mathsf{vk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell) \leftarrow \mathsf{KeyGen}(\mathsf{crs}; \rho_\ell)$. Then, it computes $(\mathsf{vk}_\varphi, \mathsf{ak}_\varphi) = \mathsf{Preprocess}(\mathsf{crs}, \varphi, \{(\mathsf{vk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [N]})$.[8] The output of the experiment is $b = \mathsf{Verify}(\mathsf{vk}_\varphi, m^*, \sigma_{\mathsf{agg}})$.

We say Π_{DMPS} satisfies static unforgeability in the registered-key model if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\Pr[b = 1] = \mathsf{negl}(\lambda)$ in the static unforgeability game defined above. We say that Π_{DMPS} satisfies static unforgeability in the registered-key model for policies with $N = N(\lambda)$ users if the above holds for all efficient adversaries $\mathcal{A}$ that outputs policies φ on exactly N users.

- **Succinctness:** There exists a universal polynomial $\mathsf{poly}(\cdot, \cdot)$ such that for all $\lambda, \kappa \in \mathbb{N}$, all crs in the support of $\mathsf{Setup}(1^\lambda, 1^\kappa)$, all policies $\varphi \in \Phi_\kappa$ over $N = N(\kappa)$ parties, all $(\mathsf{vk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell)$ in the support of $\mathsf{KeyGen}(\mathsf{crs})$, and setting $(\mathsf{vk}_\varphi, \mathsf{ak}_\varphi) = \mathsf{Preprocess}(\mathsf{crs}, \varphi, \{(\mathsf{vk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [N]})$, the following hold:
  - The size of the aggregated verification key vk_φ is $\mathsf{poly}(\lambda, \log N)$.
  - The size of the aggregate signatures output by $\mathsf{Aggregate}(\mathsf{ak}_\varphi, \cdot)$ is $\mathsf{poly}(\lambda, \log N)$.

  Taken together, these two properties imply that the running time of $\mathsf{Verify}(\mathsf{vk}_\varphi, \cdot, \cdot)$ is also $\mathsf{poly}(\lambda, \log N)$.

Remark 3.2 (Comparison with [GJM+24])**.** The work of [GJM+24] provide a formal definition of threshold signatures with silent setup, which corresponds to the case of distributed monotone-policy signatures for threshold policies. Our notion differs from the [GJM+24] in a few respects:

- **Joint key-generation:** For ease of exposition, we have a single key-generation algorithm KeyGen that outputs the verification key vk and the aggregation hint ht. The syntax of [GJM+24] decompose this into a key-generation algorithm (that outputs the verification key vk and the signing key sk) and a separate hint-generation algorithm (that outputs the ht). This is purely a syntactic distinction and we choose to combine the algorithms for ease of presentation.
- **Policy-specific preprocessing:** The work of [GJM+24] focuses on threshold policies. In the threshold setting, it is natural to consider an extension where the verification keys are fixed at preprocessing time, but the desired threshold is determined at *verification* time. This is often referred to as a *dynamic* policy. In Definition 3.1, the policy φ is fixed at preprocessing time. This is inherent if we seek to support general policies while simultaneously requiring the running time of Verify to be polylogarithmic in the number of users (and thus, the size of the policy). This is because if the policy φ is determined at verification time, the verification algorithm necessarily needs to read the description of φ, which in general, can have size that is linear in the number of users. Supporting dynamic policies while retaining fast verification is feasibly when the policies have a short description, such as the case of threshold policies. Since our focus here is on supporting general policies while retaining fast verification, we fix the policy at preprocessing time rather than verification time. In Section 5, we show that if we restrict the policy family to the special case of threshold policies, then our techniques readily extend to support *dynamic* thresholds (as in [DCX+23, GJM+24]).
- **Semi-malicious security:** As noted above, we consider a semi-malicious model where we assume all of the users' verification keys lie in the support of the honest KeyGen algorithm. The work of [GJM+24] consider the setting where the keys can be arbitrarily chosen by an adversary and there is an efficient public algorithm to determine whether a verification key (and associated aggregation hint) is well-formed or not. An easy way to

[8] Note that because Preprocess is *deterministic*, the adversary can compute $(\mathsf{vk}_\varphi, \mathsf{ak}_\varphi)$ on its own.

compile a scheme that is correct and/or secure is to have each user include a (simulation-sound) non-interactive zero-knowledge (NIZK) proof of knowledge of their secret key as part of their verification key. As in [GJM+24], the preprocessing algorithm can then effectively discard any keys that are invalid. Thus, this strategy allows us to generically upgrade a scheme satisfying our semi-malicious security notions into one that allows the adversary to choose its verification keys arbitrarily. For this reason, we elect to focus on the simpler semi-malicious definition in this work. An analogous strategy was also taken in the recent work on registered multi-authority registered ABE [LWW25] where the authors first design a scheme with semi-malicious keys and then transform the scheme to support fully malicious key registration using simulation-sound NIZK proofs of knowledge.

- **Static vs. adaptive security.** In this work, we prove security of our schemes in a *static* security model where the adversary has to declare the challenge policy and the set of corrupted users at the beginning of the game. The work of [GJM+24] consider the stronger setting of adaptive security where the adversary can adaptively corrupt users in the security game. Our current proof techniques are based on a partitioning argument (where the challenge policy is programmed into the public parameters) which limits us to static security. The work of [GJM+24] prove adaptive security of their scheme in the *generic group model*. We prove security from a q-type assumption in the *plain* model.

3.1 Constructing Distributed Monotone-Policy Signatures

In this section, we give our construction of a distributed monotone-policy signatures using pairing groups. Security of our construction relies on a q-type assumption that we call the N-extended bilinear Diffie-Hellman assumption we use in this work. This assumption shares a similar structure as the decisional parallel bilinear Diffie-Hellman exponent assumption from [Wat11]. We state the assumption below:

Assumption 3.3 (N-Extended Bilinear Diffie-Hellman)**.** Let GroupGen be a prime-order bilinear group generator. For a security parameter λ and a bit $b \in \{0,1\}$, we define the distribution $\mathcal{D}_{b,\lambda}$ as follows:

- Sample $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$, and sample a random generator $g \xleftarrow{R} \mathbb{G} \setminus \{1\}$. Next, sample exponents $a, b, t \xleftarrow{R} \mathbb{Z}_p$ and $c_1, \ldots, c_N \xleftarrow{R} \mathbb{Z}_p^*$ and define

$$\mathsf{params} = \left(1^\lambda, \mathcal{G}, g, g^a, g^b, g^t, \{g^{c_i}, g^{1/c_i}, g^{abc_i}, g^{tc_i}\}_{i\in[N]}, \{g^{c_i/c_j}, g^{abc_i/c_j}\}_{i\neq j\in[N]}\right).$$

- If $b = 0$, let $T = e(g,g)^{abt}$. If $b = 1$, sample $T \xleftarrow{R} \mathbb{G}_T$. Output (params, T).

We say the decisional N-extended bilinear Diffie-Hellman assumption holds with respect to GroupGen if the distributions $\mathcal{D}_0 = \{\mathcal{D}_{0,\lambda}\}_{\lambda\in\mathbb{N}}$ and $\mathcal{D}_1 = \{\mathcal{D}_{1,\lambda}\}_{\lambda\in\mathbb{N}}$ are computationally indistinguishable.

In Appendix A.1 (Theorem A.3), we show that the N-extended bilinear Diffie-Hellman assumption is hard in the generic bilinear group model [Sho97, BBG05] for all $N = \mathsf{poly}(\lambda)$ and against all adversaries making at most $q = \mathsf{poly}(\lambda)$ generic group queries.

Distributed monotone-policy signature construction. We now give our construction of a distributed monotone-policy signature scheme that supports any policy family that can be described by a one-use linear secret sharing scheme (see Remark 2.3).

Construction 3.4 (Distributed Monotone-Policy Signatures)**.** Let λ be a security parameter and κ be a policy-family parameter. Let GroupGen be a prime-order bilinear group generator and let $p = p(\lambda)$ be the order of the group output by GroupGen. Let $\Phi = \{\Phi_\kappa\}_{\kappa\in\mathbb{N}}$ be a family of monotone policies over $N = N(\kappa)$ users and which may be described by a one-use linear secret-sharing scheme with width $W = W(\kappa)$. We represent each policy $\varphi \in \Phi$ with a matrix $\mathbf{M} \in \mathbb{Z}_p^{N\times W}$, and write $\mathbf{m}_i^\intercal$ to denote the i^{th} row of $\mathbf{M}$. We construct a distributed monotone-policy signature scheme $\Pi_{\mathsf{DMPS}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PartialVerify}, \mathsf{Preprocess}, \mathsf{Aggregate}, \mathsf{Verify})$ for the policy family Φ and message space $\mathcal{M} = \{\mathbb{Z}_{p(\lambda)}\}_{\lambda\in\mathbb{N}}$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^\kappa)$: On input the security parameter λ and the policy-family parameter κ, the setup algorithm samples $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$. Let $N = N(\kappa)$ and $W = W(\kappa)$. The setup algorithm samples $g \xleftarrow{R} \mathbb{G} \setminus \{1\}$. $u, h \xleftarrow{R} \mathbb{G}$, $c_i \xleftarrow{R} \mathbb{Z}_p^*$ for all $i \in [N]$, and $\mathbf{s} \xleftarrow{R} \mathbb{Z}_p^W$. It computes $B = e(g,g)^{s_1}$ and outputs the common reference string
$$\mathsf{crs} = \left(\mathcal{G}, g, B, u, h, \{g^{c_i}, g^{1/c_i}, g^{c_i \mathbf{s}}\}_{i\in[N]}, \{g^{c_i/c_j}, g^{(c_i/c_j)\mathbf{s}}\}_{i\neq j\in[N]}\right) \tag{3.1}$$

- $\mathsf{KeyGen}(\mathsf{crs})$: On input the common reference string crs (with components parsed according to Eq. (3.1)), the key-generation algorithm samples $\alpha \xleftarrow{R} \mathbb{Z}_p$ and computes
$$\forall i \in [N] : v_i = (g^{c_i})^\alpha$$
$$\forall i \neq j \in [N] : w'_{i,j} = (g^{c_i/c_j})^\alpha.$$
The algorithm defines $A = e(g,g)^\alpha$ and outputs
$$\mathsf{vk} = (\mathcal{G}, g, u, h, A) \quad \text{and} \quad \mathsf{ht} = \left(\{v_i\}_{i\in[N]}, \{w'_{i,j}\}_{i\neq j\in[N]]}\right)$$
The algorithm outputs the verification key vk, the aggregation hint ht and the signing key $\mathsf{sk} = (\mathsf{vk}, \alpha)$.

- $\mathsf{Sign}(\mathsf{sk}, m)$: On input the signing key $\mathsf{sk} = (\mathsf{vk}, \alpha)$ where $\mathsf{vk} = (\mathcal{G}, g, u, h, A)$ and a message $m \in \mathbb{Z}_p$, the signing algorithm samples $r \xleftarrow{R} \mathbb{Z}_p$ and outputs the partial signature $\sigma = (g^r, g^\alpha (u^m h)^r)$.

- $\mathsf{PartialVerify}(\mathsf{vk}, m, \sigma)$: On input a verification key $\mathsf{vk} = (\mathcal{G}, g, u, h, A)$, a message $m \in \mathbb{Z}_p$, and a signature $\sigma = (\sigma_1, \sigma_2)$, the partial verification algorithm outputs 1 if
$$A \cdot e(\sigma_1, u^m h) = e(g, \sigma_2).$$

- $\mathsf{Preprocess}(\mathsf{crs}, \varphi, \{(\mathsf{vk}_\ell, \mathsf{ht}_\ell)\}_{\ell\in[N]})$: On input the common reference string crs (with components parsed according to Eq. (3.1)), a policy $\varphi = \mathbf{M} \in \mathbb{Z}_p^{N\times W}$, and a collection of verification keys vk_ℓ and hints $\mathsf{ht}_\ell = (\{v_{\ell,i}\}_{i\in[N]}, \{w'_{\ell,i,j}\}_{i\neq j\in[N]})$, the preprocessing algorithm computes
$$z = \prod_{\ell\in[N]} g^{\mathbf{m}_\ell^\intercal (c_\ell \mathbf{s})} v_{\ell,\ell}.$$
Note that the preprocessing algorithm computes z using knowledge of $\mathbf{m}_\ell^\intercal$ and $g^{c_\ell \mathbf{s}}$ from crs. Then, for each $\ell \in [N]$, it computes
$$w_\ell = \prod_{\substack{i\in[N]\\ i\neq\ell}} w'_{i,i,\ell} \cdot g^{\mathbf{m}_i^\intercal ((c_i/c_\ell))\mathbf{s}},$$
using knowledge of $\mathbf{m}_i^\intercal$ and $g^{(c_i/c_\ell)\mathbf{s}}$ from crs. Then, it sets
$$\mathsf{vk}_\varphi = (\mathcal{G}, g, u, h, B, z) \quad \text{and} \quad \mathsf{ak}_\varphi = (\mathbf{M}, z, \{g^{1/c_\ell}, w_\ell\}_{\ell\in[N]}).$$
Finally, the preprocessing algorithm outputs the verification key vk_φ and the aggregation key ak_φ.

- $\mathsf{Aggregate}(\mathsf{ak}_\varphi, \{\sigma_\ell\}_{\ell\in S})$: On input the aggregation key $\mathsf{ak}_\varphi = (\mathbf{M}, z, \{g^{1/c_\ell}, w_\ell\}_{\ell\in[N]})$, and a collection of signatures $\sigma_\ell = (\sigma_{\ell,1}, \sigma_{\ell,2})$ for $\ell \in S \subseteq [N]$, the aggregation algorithm proceeds as follows:
  - First, it checks if S satisfies the policy $\mathbf{M}$. If not, the aggregation algorithm outputs $\perp$. Otherwise, let $\boldsymbol{\omega} \in \mathbb{Z}_p^N$ be a reconstruction vector where $\boldsymbol{\omega}^\intercal \mathbf{M} = \mathbf{e}_1^\intercal$, and moreover, $\omega_\ell = 0$ for all $\ell \in [N] \setminus S$.
  - Compute
$$\sigma_{\mathsf{agg},1} = \prod_{\ell\in S} \sigma_{\ell,1}^{\omega_\ell} \quad \text{and} \quad \sigma_{\mathsf{agg},2} = \prod_{\ell\in S} \sigma_{\ell,2}^{\omega_\ell} w_\ell^{\omega_\ell} \quad \text{and} \quad \sigma_{\mathsf{agg},3} = \prod_{\ell\in S} \left(g^{1/c_\ell}\right)^{\omega_\ell},$$
    Output the signature $\sigma_{\mathsf{agg}} = (\sigma_{\mathsf{agg},1}, \sigma_{\mathsf{agg},2}, \sigma_{\mathsf{agg},3})$.

- $\mathsf{Verify}(\mathsf{vk}_\varphi, m, \sigma_{\mathsf{agg}})$: On input the verification key $\mathsf{vk}_\varphi = (\mathcal{G}, g, u, h, B, z)$, a message $m \in \mathbb{Z}_p$, and a signature $\sigma_{\mathsf{agg}} = (\sigma_{\mathsf{agg},1}, \sigma_{\mathsf{agg},2}, \sigma_{\mathsf{agg},3})$, the verification algorithm outputs 1 if

$$B \cdot e(g, \sigma_{\mathsf{agg},2}) = e(\sigma_{\mathsf{agg},1}, u^m h) \cdot e(z, \sigma_{\mathsf{agg},3}) \tag{3.2}$$

  If Eq. (3.2) does not hold, the verification algorithm outputs 0.

Theorem 3.5 (Signing Correctness)**.** *Construction 3.4 satisfies signing correctness.*

Proof. Take any $\lambda, \kappa \in \mathbb{N}$ and message $m \in \mathbb{Z}_p$. Let $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa)$, $(\mathsf{vk}, \mathsf{ht}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{crs})$ and $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$. By construction, we can write

$$\begin{aligned}
\mathsf{crs} &= \left(\mathcal{G}, g, B, u, h, \{g^{c_i}, g^{1/c_i}, g^{c_i \mathbf{s}}\}_{i \in [N]}, \{g^{c_i/c_j}, g^{(c_i/c_j)\mathbf{s}}\}_{i \neq j \in [N]}\right) \\
\mathsf{vk} &= (\mathcal{G}, g, u, h, A) \\
\sigma &= (\sigma_1, \sigma_2) = (g^r, g^\alpha (u^m h)^r),
\end{aligned}$$

where $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, e)$ and $A = e(g, g)^\alpha$. By bilinearity, we now have

$$e(g, \sigma_2) = e(g, g^\alpha (u^m h)^r) = e(g, g)^\alpha \cdot e(g^r, u^m h) = A \cdot e(\sigma_1, u^m h).$$

We conclude that $\mathsf{PartialVerify}(\mathsf{vk}, m, \sigma)$ outputs 1, as required. □

Theorem 3.6 (Aggregation Correctness)**.** *Construction 3.4 satisfies aggregation correctness.*

Proof. Take any $\lambda, \kappa \in \mathbb{N}$, message $m \in \mathbb{Z}_p$, policy $\varphi \in \Phi_\kappa$ (with associated matrix $\mathbf{M} \in \mathbb{Z}_p^{N \times W}$), any set $S \subseteq [N]$ where $\varphi(S) = 1$, any crs in the support of $\mathsf{Setup}(1^\lambda, 1^\kappa)$, any $\{(\mathsf{vk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell)\}_{\ell \in [N]}$ where $(\mathsf{vk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell)$ is in the support of $\mathsf{KeyGen}(\mathsf{crs})$ for all $\ell \in [N]$, and any collection of signatures $\{\sigma_\ell\}_{\ell \in S}$ where $\mathsf{PartialVerify}(\mathsf{crs}, m, \sigma_\ell) = 1$ for all $\ell \in S$. By construction, we can now write

$$\begin{aligned}
\mathsf{crs} &= \left(\mathcal{G}, g, B, u, h, \{g^{c_i}, g^{1/c_i}, g^{c_i \mathbf{s}}\}_{i \in [N]}, \{g^{c_i/c_j}, g^{(c_i/c_j)\mathbf{s}}\}_{i \neq j \in [N]}\right) \\
\mathsf{vk}_\ell &= (\mathcal{G}, g, u, h, A_\ell) \\
\mathsf{ht}_\ell &= \left(\{v_{\ell,i}\}_{i \in [N]}, \{w'_{\ell,i,j}\}_{i \neq j \in [N]}\right) \\
\sigma_\ell &= (\sigma_{\ell,1}, \sigma_{\ell,2}),
\end{aligned}$$

where $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, e)$ and $B = e(g, g)^{s_1}$. By construction of KeyGen, we have for all $i \neq j \in [N]$,

$$A_\ell = e(g, g)^{\alpha_\ell} \quad \text{and} \quad v_{\ell,i} = g^{\alpha_\ell c_i} \quad \text{and} \quad w'_{\ell,i,j} = g^{\alpha_\ell c_i / c_j}.$$

Moreover, since $\mathsf{PartialVerify}(\mathsf{crs}, m, \sigma_\ell) = 1$ for all $\ell \in S$, this means

$$\forall \ell \in S : e(g, g)^{\alpha_\ell} \cdot e(\sigma_{\ell,1}, u^m h) = e(g, \sigma_{\ell,2}). \tag{3.3}$$

Let

$$\begin{aligned}
(\mathsf{vk}_\varphi, \mathsf{ak}_\varphi) &= \mathsf{Preprocess}(\mathsf{crs}, \mathbf{M}, \{(\mathsf{vk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [N]}) \\
\sigma_{\mathsf{agg}} = (\sigma_{\mathsf{agg},1}, \sigma_{\mathsf{agg},2}, \sigma_{\mathsf{agg},3}) &= \mathsf{Aggregate}(\mathsf{crs}, \mathsf{ak}_\varphi, \{\sigma_\ell\}_{\ell \in S}).
\end{aligned}$$

By construction, the preprocessing algorithm first computes

$$z = \prod_{\ell \in [N]} g^{\mathbf{m}_\ell^\intercal (c_\ell \mathbf{s})} v_{\ell,\ell} = g^{\tilde{z}} \quad \text{where} \quad \tilde{z} = \sum_{\ell \in [N]} c_\ell (\mathbf{m}_\ell^\intercal \mathbf{s} + \alpha_\ell).$$

For each $\ell \in [N]$, it also computes

$$w_\ell = \prod_{\substack{i \in [N] \\ i \neq \ell}} w'_{i,i,\ell} \cdot g^{\mathbf{m}_i^\intercal ((c_i/c_\ell)) \mathbf{s}} = \prod_{\substack{i \in [N] \\ i \neq \ell}} g^{c_i/c_\ell (\mathbf{m}_i^\intercal \mathbf{s} + \alpha_i)} \tag{3.4}$$

Then, it sets $\mathsf{vk}_\varphi = (\mathcal{G}, g, u, h, B, z)$. The aggregation algorithm starts by computing

$$\sigma_{\mathrm{agg},1} = \prod_{\ell \in S} \sigma_{\ell,1}^{\omega_\ell} \quad \text{and} \quad \sigma_{\mathrm{agg},2} = \prod_{\ell \in S} \sigma_{\ell,2}^{\omega_\ell} w_\ell^{\omega_\ell} \quad \text{and} \quad \sigma_{\mathrm{agg},3} = \prod_{\ell \in S} \left(g^{1/c_\ell}\right)^{\omega_\ell}, \tag{3.5}$$

By Eq. (3.3) and bilinearity, this means

$$e(\sigma_{\mathrm{agg},1}, u^m h) = \prod_{\ell \in S} e(\sigma_{\ell,1}, u^m h)^{\omega_\ell} = \prod_{\ell \in S} \frac{e(g, \sigma_{\ell,2}^{\omega_\ell})}{e(g,g)^{\omega_\ell \alpha_\ell}} \tag{3.6}$$

Let $\tilde{\sigma}_{\mathrm{agg},3} = \sum_{\ell \in S} \omega_\ell / c_\ell$. Then $\sigma_{\mathrm{agg},3} = g^{\tilde{\sigma}_{\mathrm{agg},3}}$. Next, $\boldsymbol{\omega}^\intercal \mathbf{M} = \mathbf{e}_1^\intercal$ so

$$\sum_{\ell \in [N]} \omega_\ell \mathbf{m}_\ell^\intercal \mathbf{s} = \boldsymbol{\omega}^\intercal \mathbf{M} \mathbf{s} = \mathbf{e}_1^\intercal \mathbf{s} = s_1.$$

Combined with the fact that $\omega_\ell = 0$ for all $\ell \notin S$, we have

$$\begin{aligned}
\tilde{z} \cdot \tilde{\sigma}_{\mathrm{agg},3} &= \sum_{\ell \in [N]} \sum_{i \in [N]} c_i (\mathbf{m}_i^\intercal \mathbf{s} + \alpha_i) \cdot \frac{\omega_\ell}{c_\ell} \\
&= \sum_{\ell \in [N]} \omega_\ell \mathbf{m}_\ell^\intercal \mathbf{s} + \sum_{\ell \in S} \omega_\ell \alpha_\ell + \sum_{\ell \in S} \sum_{\substack{i \in [N] \\ i \neq \ell}} \left(\omega_\ell (\mathbf{m}_i^\intercal \mathbf{s} + \alpha_i) \cdot \frac{c_i}{c_\ell} \right) \\
&= s_1 + \sum_{\ell \in S} \omega_\ell \alpha_\ell + \sum_{\ell \in S} \sum_{\substack{i \in [N] \\ i \neq \ell}} \left(\omega_\ell (\mathbf{m}_i^\intercal \mathbf{s} + \alpha_i) \cdot \frac{c_i}{c_\ell} \right)
\end{aligned}$$

By Eq. (3.4), this means

$$e(z, \sigma_{\mathrm{agg},3}) = e(g,g)^{\tilde{z} \tilde{\sigma}_{\mathrm{agg},3}} = e(g,g)^{s_1} \cdot \prod_{\ell \in S} e(g,g)^{\omega_\ell \alpha_\ell} \cdot \prod_{\ell \in S} e(g, w_\ell)^{\omega_\ell} \tag{3.7}$$

Combining Eqs. (3.5) to (3.7), we have

$$\begin{aligned}
B \cdot e(g, \sigma_{\mathrm{agg},2}) &= e(g,g)^{s_1} \cdot \prod_{\ell \in S} e(g, \sigma_{\ell,2}^{\omega_\ell}) \cdot \prod_{\ell \in S} e(g, w_\ell)^{\omega_\ell} \\
&= e(\sigma_{\mathrm{agg},1}, u^m h) \cdot e(z, \sigma_{\mathrm{agg},3}).
\end{aligned}$$

Correspondingly, $\mathsf{Verify}(\mathsf{vk}_\varphi, m, \sigma_{\mathrm{agg}}) = 1$, as required. □

Theorem 3.7 (Static Unforgeability)**.** *Suppose the N-extended bilinear Diffie-Hellman assumption (Assumption 3.3) holds with respect to* GroupGen*. Then, Construction 3.4 satisfies static unforgeability in the registered key model for policies with N users.*

Proof. To prove security, it will be more convenient to use the following search variant of the N-extended bilinear Diffie-Hellman assumption. We state the assumption and show that it is implied by the decisional N-extended bilinear Diffie-Hellman assumption below:

Lemma 3.8. *Let λ be a security parameter. For an adversary $\mathcal{A}$, define the search N-extended bilinear Diffie-Hellman experiment as follows:*

1. *The challenger samples $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$ and a random generator $g \xleftarrow{R} \mathbb{G} \setminus \{1\}$. The challenger then samples exponents $a, b \xleftarrow{R} \mathbb{Z}_p$ and $c_1, \ldots, c_N \xleftarrow{R} \mathbb{Z}_p^*$. Define*

$$\mathsf{params} = \left(1^\lambda, \mathcal{G}, g, g^a, g^b, \{g^{c_i}, g^{1/c_i}, g^{abc_i}\}_{i \in [N]}, \{g^{c_i/c_j}, g^{abc_i/c_j}\}_{i \neq j \in [N]}\right).$$

2. *On input* params, *algorithm* $\mathcal{A}$ *outputs* $\tau_0, \tau_1, \ldots, \tau_N \in \mathbb{G}$.

3. *The output of the experiment is* $b = 1$ *if*

$$e(g,g)^{ab} = e(g,\tau_0) \cdot \prod_{i\in[N]} e(g^{c_i}, \tau_i).$$

Otherwise, the output is 0.

If the decisional N*-extended bilinear Diffie-Hellman assumption holds with respect to* GroupGen*, then for all efficient adversaries* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $\Pr[b = 1] = \mathsf{negl}(\lambda)$ *in the search* N*-extended bilinear Diffie-Hellman experiment.*

Proof. Suppose there exists an efficient adversary $\mathcal{A}$ that can solve the search N-extended bilinear Diffie-Hellman problem with advantage ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for the decisional N-extended bilinear Diffie-Hellman problem:

- On input (params, T) where

$$\mathsf{params} = \left(1^\lambda, \mathcal{G}, g, g^a, g^b, g^t, \{g^{c_i}, g^{1/c_i}, g^{abc_i}, g^{tc_i}\}_{i\in[N]}, \{g^{c_i/c_j}, g^{abc_i/c_j}\}_{i\neq j\in[N]}\right),$$

  algorithm $\mathcal{B}$ runs algorithm $\mathcal{A}$ on the following input

$$\mathsf{params}' = \left(1^\lambda, \mathcal{G}, g, g^a, g^b, \{g^{c_i}, g^{1/c_i}, g^{abc_i}\}_{i\in[N]}, \{g^{c_i/c_j}, g^{abc_i/c_j}\}_{i\neq j\in[N]}\right).$$

- Algorithm $\mathcal{B}$ outputs $\tau_0, \tau_1, \ldots, \tau_N \in \mathbb{G}$. Algorithm $\mathcal{B}$ first checks if

$$e(g^a, g^b) = e(g,\tau_0) \cdot \prod_{i\in[N]} e(g^{c_i}, \tau_i). \tag{3.8}$$

  If this does not hold, algorithm $\mathcal{B}$ outputs 0. Otherwise, algorithm $\mathcal{B}$ outputs 1 if and only if

$$T = e(g^t, \tau_0) \cdot \prod_{i\in[N]} e(g^{tc_i}, \tau_i). \tag{3.9}$$

By design, algorithm $\mathcal{B}$ perfectly simulates the parameters for the search N-extended bilinear Diffie-Hellman problem. Thus, with probability ε, it will output $\tau_0, \tau_1 \ldots, \tau_N$ that satisfy Eq. (3.8). We now compute the advantage of $\mathcal{A}$.

- Suppose $T = e(g,g)^{abt}$. If Eq. (3.8) holds, then Eq. (3.9) also holds. In this case, algorithm $\mathcal{A}$ outputs 1 whenever Eq. (3.8) holds, which occurs with probability ε.
- Suppose $T \xleftarrow{R} \mathbb{G}_T$. Since the view of $\mathcal{A}$ is independent of T, and moreover, the challenger samples $t, c_1, \ldots, c_N$ independently of T, this means Eq. (3.9) holds with probability exactly $1/p$. Thus, in this case, algorithm $\mathcal{B}$ outputs 1 with probability at most $1/p$.

We conclude that algorithm $\mathcal{B}$ distinguishes with advantage at least $\varepsilon - 1/p \geq \varepsilon - 2^{-\lambda}$ since $p > 2^\lambda$. The claim holds. □

Proof of Theorem 3.7. Suppose there exists an efficient adversary $\mathcal{A}$ that wins the static unforgeability game for policies with N users with non-negligible advantage ε. We use $\mathcal{A}$ to construct a new adversary $\mathcal{B}$ that breaks the search N-extended bilinear Diffie-Hellman assumption from Lemma 3.8. In the following, we will use a tilde (e.g., $\tilde{u}, \tilde{h}$) to denote exponents sampled by the reduction algorithm $\mathcal{B}$. Algorithm $\mathcal{B}$ works as follows:

1. On input the challenge

$$\left(1^\lambda, \mathcal{G}, g, g^a, g^b, \{g^{c_i}, g^{1/c_i}, g^{abc_i}\}_{i\in[N]}, \{g^{c_i/c_j}, g^{abc_i/c_j}\}_{i\neq j\in[N]}\right),$$

   algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ on 1^λ. Algorithm $\mathcal{A}$ outputs a policy family parameter 1^κ and a policy $\mathbf{M} \in \mathbb{Z}_p^{N\times W}$. In addition, algorithm $\mathcal{B}$ specifies a set $C \subseteq [N]$ of corrupted indices and a challenge message $m^* \in \mathbb{Z}_p$.

2. Algorithm $\mathcal{B}$ first checks if C satisfies the policy $\mathbf{M}$. If so, then it halts with output 0.

3. Otherwise, algorithm $\mathcal{B}$ constructs the common reference string crs as follows. First, it samples $\tilde{u}, \tilde{h} \xleftarrow{R} \mathbb{Z}_p$ and sets

$$u = (g^b) \cdot g^{\tilde{u}} \quad \text{and} \quad h = g^{\tilde{h}}/(g^b)^{m^*}.$$

Since the set $C \subseteq [N]$ does not satisfy the policy $\mathbf{M}$, there exists a vector $\tilde{\mathbf{w}} \in \mathbb{Z}_p^N$ such that for all indices $i \in C$, $\mathbf{m}_i^\top \tilde{\mathbf{w}} = 0$, where $\mathbf{m}_i^\top$ denotes the i^{th} row of $\mathbf{M}$, and $\mathbf{e}_1^\top \tilde{\mathbf{w}} = 1$. Algorithm $\mathcal{B}$ samples a vector $\tilde{\mathbf{s}} \xleftarrow{R} \mathbb{Z}_p^W$. Algorithm $\mathcal{B}$ now implicitly sets $\mathbf{s} = \tilde{\mathbf{s}} + ab \cdot \tilde{\mathbf{w}}$. In particular, algorithm $\mathcal{B}$ constructs the $\mathbf{s}$-dependent terms in the CRS as follows:

$$\begin{aligned} B &= e(g,g)^{\tilde{s}_1} \cdot e(g^a, g^b)^{\tilde{w}_1} = e(g,g)^{s_1} \\ g^{c_i \mathbf{s}} &= (g^{c_i})^{\tilde{\mathbf{s}}} \cdot (g^{abc_i})^{\tilde{\mathbf{w}}} \\ g^{(c_i/c_j)\mathbf{s}} &= (g^{c_i/c_j})^{\tilde{\mathbf{s}}} \cdot (g^{abc_i/c_j})^{\tilde{\mathbf{w}}}. \end{aligned}$$

Algorithm $\mathcal{B}$ sets the common reference string to be

$$\mathsf{crs} = \left(\mathcal{G}, g, B, u, h, \{g^{c_i}, g^{1/c_i}, g^{c_i \mathbf{s}}\}_{i \in [N]}, \{g^{c_i/c_j}, g^{(c_i/c_j)\mathbf{s}}\}_{i \neq j \in [N]}\right).$$

4. Next, to simulate the honest verification keys, algorithm $\mathcal{B}$ starts by sampling $\tilde{\alpha}_\ell \xleftarrow{R} \mathbb{Z}_p$ for each $\ell \in [N] \setminus C$. Then, algorithm $\mathcal{B}$ implicitly sets the secret key for user ℓ to be $\alpha_\ell = \tilde{\alpha}_\ell - ab \cdot \mathbf{m}_\ell^\top \tilde{\mathbf{w}}$. Specifically, algorithm $\mathcal{B}$ constructs the components of the verification key and the aggregation hint as follows:

$$\begin{aligned} A_\ell = e(g,g)^{\alpha_\ell} &= e(g,g)^{\tilde{\alpha}_\ell} \cdot e(g^a, g^b)^{-\mathbf{m}_\ell^\top \tilde{\mathbf{w}}} \\ v_{\ell,i} &= (g^{c_i})^{\tilde{\alpha}_\ell} \cdot (g^{abc_i})^{-\mathbf{m}_\ell^\top \tilde{\mathbf{w}}} \\ w'_{\ell,i,j} &= (g^{c_i/c_j})^{\tilde{\alpha}_\ell} \cdot (g^{abc_i/c_j})^{-\mathbf{m}_\ell^\top \tilde{\mathbf{w}}}. \end{aligned}$$

Then, algorithm $\mathcal{B}$ sets

$$\mathsf{vk}_\ell = (\mathcal{G}, g, u, h, A_\ell) \quad \text{and} \quad \mathsf{ht}_\ell = \left(\{v_{\ell,i}\}_{i \in [N]}, \{w'_{\ell,i,j}\}_{i \neq j}\right).$$

Algorithm $\mathcal{B}$ gives crs and $\{(\mathsf{vk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [N] \setminus C}$ to $\mathcal{A}$.

5. Whenever algorithm $\mathcal{A}$ makes a signing query on an index $\ell \in [N] \setminus C$ and a message $m \neq m^*$, algorithm $\mathcal{B}$ samples $\tilde{r} \xleftarrow{R} \mathbb{Z}_p$ and implicitly sets $r = \tilde{r} + a(m - m^*)^{-1} \mathbf{m}_\ell^\top \tilde{\mathbf{w}}$. Then, it computes

$$\begin{aligned} \sigma_1 &= g^{\tilde{r}} \cdot (g^a)^{(m-m^*)^{-1} \mathbf{m}_\ell^\top \tilde{\mathbf{w}}} \\ \sigma_2 &= g^{\tilde{\alpha}_\ell} \cdot (g^b)^{\tilde{r}(m-m^*)} \cdot \sigma_1^{\tilde{u}m + \tilde{h}} \end{aligned}$$

and responds to $\mathcal{A}$ with the signature $\sigma = (\sigma_1, \sigma_2)$.

6. After $\mathcal{A}$ is finished making signing queries, it outputs the key-generation randomness $\rho_\ell \in \{0,1\}^*$ used to generate the keys for the corrupted users $\ell \in C$. In addition, algorithm $\mathcal{A}$ outputs a signature $\sigma_{\text{agg}} = (\sigma_{\text{agg},1}, \sigma_{\text{agg},2}, \sigma_{\text{agg},3})$.

7. For each $\ell \in C$, algorithm $\mathcal{B}$ computes $(\mathsf{vk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell) \leftarrow \mathsf{KeyGen}(\mathsf{crs}; \rho_\ell)$. Algorithm $\mathcal{B}$ parses sk_ℓ as $\mathsf{sk}_\ell = (\mathsf{vk}_\ell, \tilde{\alpha}_\ell)$ for each $\ell \in C$. Finally, algorithm $\mathcal{B}$ outputs $(\tau_0, \tau, \ldots, \tau_N)$ where

$$\tau_0 = g^{-\tilde{s}_1} \cdot \sigma_{\text{agg},1}^{\tilde{u}m^* + \tilde{h}} \cdot \sigma_{\text{agg},2}^{-1} \quad \text{and} \quad \forall i \in [N] : \tau_i = \sigma_{\text{agg},3}^{\mathbf{m}_i^\top \tilde{\mathbf{s}} + \tilde{\alpha}_i}.$$

First, we argue that algorithm $\mathcal{B}$ correctly simulates the common reference string, the honest verification keys, and the signatures. Consider first the components of the common reference string:

- Algorithm $\mathcal{B}$ samples $\tilde{u}, \tilde{h} \xleftarrow{R} \mathbb{Z}_p$ so the distributions of u, h are also uniform over $\mathbb{G}$ (and independent of all other components in crs), exactly as in the real scheme.
- Algorithm $\mathcal{B}$ implicitly sets $\mathbf{s} = \tilde{\mathbf{s}} + ab \cdot \tilde{\mathbf{w}}$, where $\tilde{\mathbf{s}} \xleftarrow{R} \mathbb{Z}_p^N$. Thus, the distribution of $\mathbf{s}$ also coincides with its distribution in the real scheme.
- Finally, the challenger samples $c_1, \ldots, c_N \xleftarrow{R} \mathbb{Z}_p^*$, which matches the distribution in the real scheme.

We conclude that algorithm $\mathcal{B}$ constructs crs according to the same distribution as $\mathsf{Setup}(1^\lambda, 1^\kappa)$. We now consider the distribution of the honest verification keys and signatures:

- **Verification keys:** Consider the honest verification keys vk_ℓ for $\ell \in [N] \setminus C$. By construction, the verification keys vk_ℓ and hint components ht_ℓ sampled by algorithm $\mathcal{B}$ coincide with those that would be output by $\mathsf{KeyGen}(\mathsf{crs})$ with $\alpha_\ell = \tilde{\alpha}_\ell - ab \cdot \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}$. Since algorithm $\mathcal{B}$ samples $\tilde{\alpha}_\ell \xleftarrow{R} \mathbb{Z}_p$ for all $\ell \in [N] \setminus C$, the verification keys are also distributed exactly as in the real scheme.
- **Signatures:** Finally, consider the signing queries. Let $\ell \in [N] \setminus C$ be the index and $m \neq m^*$ be the message. We claim that $\sigma = (\sigma_1, \sigma_2)$ is a signature with respect to signing key α_ℓ and randomness $r = \tilde{r} + a(m - m^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}$:
  - By construction, algorithm $\mathcal{B}$ sets $\sigma_1 = g^{\tilde{r}} \cdot (g^a)^{(m-m^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} = g^r$.
  - In the real scheme, $\sigma_2 = g^{\alpha_\ell}(u^m h)^r$. Substituting the expressions for α_ℓ, u, h, r, we have

$$\begin{aligned} g^{\alpha_\ell}(u^m h)^r &= g^{\tilde{\alpha}_\ell - ab\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} (g^{bm + \tilde{u}m} g^{\tilde{h} - am^*})^{\tilde{r} + a(m-m^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} \\ &= g^{\tilde{\alpha}_\ell} (g^{\tilde{u}m + \tilde{h}})^{\tilde{r} + a(m-m^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} g^{-ab\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} g^{b(m-m^*)(\tilde{r} + a(m-m^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}})} \\ &= g^{\tilde{\alpha}_\ell} \sigma_1^{\tilde{u}m + \tilde{h}} g^{b(m-m^*)\tilde{r}}. \end{aligned}$$

    This is precisely how algorithm $\mathcal{B}$ constructs the signatures.

  Since algorithm $\mathcal{B}$ samples $\tilde{r} \xleftarrow{R} \mathbb{Z}_p$, the distribution of r is also uniform and the signature is correctly constructed.

Thus, with probability ε, algorithm $\mathcal{A}$ outputs a signature $\sigma_{\mathsf{agg}} = (\sigma_{\mathsf{agg},1}, \sigma_{\mathsf{agg},2}, \sigma_{\mathsf{agg},3})$ such that

$$\mathsf{Verify}(\mathsf{vk}_\varphi, m^*, \sigma_{\mathsf{agg}}) = 1,$$

where $\mathsf{vk}_\varphi = (\mathcal{G}, g, u, h, e(g,g)^{s_1}, z)$ and

$$z = \prod_{\ell \in [N]} g^{c_\ell \mathbf{m}_\ell^\intercal \mathbf{s}} v_{\ell,\ell}.$$

We consider each component in this product separately:

- Suppose $\ell \in C$. In this case, $v_{\ell,\ell}$ is output by $\mathsf{KeyGen}(\mathsf{crs}; \rho_\ell)$. Since $\mathsf{sk}_\ell = (\mathsf{vk}_\ell, \tilde{\alpha}_\ell)$, this means $v_{\ell,\ell} = g^{c_\ell \tilde{\alpha}_\ell}$. Since $\ell \in C$, $\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}} = 0$. This means

$$c_\ell(\mathbf{m}_\ell^\intercal \mathbf{s} + \tilde{\alpha}_\ell) = c_\ell(\mathbf{m}_\ell^\intercal(\tilde{\mathbf{s}} + ab\tilde{\mathbf{w}}) + \tilde{\alpha}_\ell) = c_\ell(\mathbf{m}_\ell^\intercal \tilde{\mathbf{s}} + \tilde{\alpha}_\ell).$$

- Suppose $\ell \in [N] \setminus C$. In this case, $v_{\ell,\ell} = g^{c_\ell \alpha_\ell} = g^{c_\ell(\tilde{\alpha}_\ell - ab\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}})}$. This means

$$c_\ell(\mathbf{m}_\ell^\intercal \mathbf{s} + \alpha_\ell) = c_\ell(\mathbf{m}_\ell^\intercal(\tilde{\mathbf{s}} + ab\tilde{\mathbf{w}}) + \tilde{\alpha}_\ell - ab\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}) = c_\ell(\mathbf{m}_\ell^\intercal \tilde{\mathbf{s}} + \tilde{\alpha}_\ell).$$

We conclude that

$$z = \prod_{\ell \in [N]} g^{c_\ell \mathbf{m}_\ell^\intercal \mathbf{s}} v_{\ell,\ell} = \prod_{\ell \in [N]} g^{c_\ell(\mathbf{m}_\ell^\intercal \tilde{\mathbf{s}} + \tilde{\alpha}_\ell)}. \tag{3.10}$$

Now, since verification passes, this means Eq. (3.2) holds so

$$e(g,g)^{s_1} \cdot e(g, \sigma_{\mathsf{agg},2}) = e(\sigma_{\mathsf{agg},1}, u^{m^*} h) \cdot e(z, \sigma_{\mathsf{agg},3}). \tag{3.11}$$

Recall that $\mathbf{e}_1^\top \tilde{\mathbf{w}} = 1$. This means $s_1 = \mathbf{e}_1^\top \mathbf{s} = \mathbf{e}_1^\top \tilde{\mathbf{s}} + ab\mathbf{e}_1^\top \tilde{\mathbf{w}} = \tilde{s}_1 + ab$. In addition, $u^{m^*} h = g^{bm^* + \tilde{u}m^*} \cdot g^{\tilde{h}} / g^{bm^*} = g^{\tilde{u}m^* + \tilde{h}}$. Combining with Eqs. (3.10) and (3.11), we thus have

$$e(g,g)^{\tilde{s}_1 + ab} \cdot e(g, \sigma_{\mathsf{agg},2}) = e(\sigma_{\mathsf{agg},1}, g^{\tilde{u}m^* + \tilde{h}}) \cdot \prod_{\ell \in [N]} e(g^{c_\ell (\mathbf{m}_\ell^\top \tilde{\mathbf{s}} + \tilde{\alpha}_\ell)}, \sigma_{\mathsf{agg},3}).$$

Rearranging and using bilinearity, we have

$$\begin{aligned} e(g,g)^{ab} &= e\big(g, g^{-\tilde{s}_1} \cdot \sigma_{\mathsf{agg},1}^{\tilde{u}m^* + \tilde{h}} \cdot \sigma_{\mathsf{agg},2}^{-1}\big) \cdot \prod_{\ell \in [N]} e(g^{c_\ell}, \sigma_{\mathsf{agg},3}^{\mathbf{m}_\ell^\top \tilde{\mathbf{s}} + \tilde{\alpha}_\ell}) \\ &= e(g, \tau_0) \cdot \prod_{\ell \in [N]} e(g^{c_\ell}, \tau_\ell). \end{aligned}$$

We conclude that $\mathcal{B}$ breaks the search N-extended bilinear Diffie-Hellman with the same advantage ε. □

Remark 3.9 (Proving Well-Formedness of the Public Key)**.** As noted in Remark 3.2, we can lift a signature scheme with semi-malicious security to one with full security by having users attach a NIZK proof of knowledge of the secret key associated with the verification key vk and aggregation hint ht. In the context of Construction 3.4, this boils down to proving knowledge of discrete log: namely, that the user knows α such that $A = e(g,g)^\alpha$, where A is the group element in the verification key. Note that well-formedness of the elements in the hint can be efficiently certified using A together with the components in the common reference string. For instance, a hint $\mathsf{ht} = \big(\{v_i\}_{i \in [N]}, \{w'_{i,j}\}_{i \neq j \in [N]}\big)$ associated with the verification key $\mathsf{vk} = (\mathcal{G}, g, u, h, A)$ is well-formed if and only if

$$\forall i \in [N] : e(v_i, g^{1/c_i}) = A \quad \text{and} \quad \forall i \neq j \in [N] : e(w'_{i,j}, g^{c_j/c_i}) = A.$$

It is straightforward to prove knowledge of discrete log in the random oracle model (e.g., via a Schnorr proof [Sch89]) or if we work in the generic group model [Sho97].[9] In the plain model, we could use any general-purpose pairing-based NIZK (e.g., [CHK03, GOS06, LPWW20]).

Remark 3.10 (Supporting Multi-Use Policies)**.** Construction 3.4 gives a distributed monotone-policy signature scheme for any policy family that can be expressed as a *one-use* linear secret sharing scheme. A straightforward way to extend the scheme to support policies where each party is associated with multiple shares is to rely on virtualization [LW11]. Specifically, consider an LSSS policy $\mathbf{M}$ where each user is associated with at most K rows. We can construct a distributed monotone-policy signature scheme for K-use LSSS policies at a cost of a factor of K blowup in the size of the common reference string, the size of the user's verification key (and hint), the size of the partial signatures, and the size of the aggregation key. The approach is for each user to generate K verification keys, one associated with each of the K possible shares that may be assigned to it. A partial signature now consists of K partial signatures for the underlying scheme, one for each of the user's K verification keys. Effectively, each user in the scheme functions as K independent "virtual" users in the underlying distributed monotone-policy signature scheme for one-use policies. While this approach increases the size of the common reference string, the user public keys, the partial signatures, and the aggregation key by a factor of K, the size of the aggregated verification key and the size of the aggregate signature is *unchanged*.

Corollary 3.11 (Distributed Monotone-Policy Signatures from Pairings)**.** *Let λ be a security parameter. Suppose the N-extended bilinear Diffie-Hellman assumption (Assumption 3.3) holds with respect to* GroupGen *for all polynomials $N = N(\lambda)$. Let $\Phi = \{\Phi_\kappa\}_{\kappa \in \mathbb{N}}$ be the class of policies that can be described by a K-use linear-secret sharing scheme with $N = N(\kappa)$ users and width $W = W(\kappa)$, where N, W are polynomially-bounded. Then, there exists a (statically-secure) distributed monotone-policy signature scheme for Φ in the registered-key model with the following efficiency properties:*

- **CRS size:** *The CRS contains $O(K^2 N^2 W)$ group elements.*
- **User key size:** *A user's verification key* vk *contains $3 + K$ group elements and the aggregation hint* ht *contains $O(K^2 N^2)$ group elements. The first 3 group elements in each user's verification key are fixed (i.e., group elements from the common reference string). The number of user-specific group elements in each verification key is K.*

[9] In fact, in the generic group model, the reduction algorithm can directly extract the discrete log associated with the user's verification key, and there is no need to include a NIZK proof of knowledge at all.

- **Partial signature size:** *A user's partial signature σ contains $2K$ group elements.*
- **Aggregation key size:** *The size of an aggregation key ak_φ for a policy $\varphi \in \Phi$ contains $2KN + 1$ group elements.*
- **Aggregate verification key size:** *An aggregated verification key vk_φ for a policy $\varphi \in \Phi$ contains 5 group elements.*
- **Aggregate signature size:** *An aggregate signature contains 3 group elements. The aggregate verification algorithm requires performing 3 pairings (and $O(1)$ group operations).*

3.2 Reducing the CRS Size Using Powers

To support monotone policies described by a (one-use) linear secret sharing scheme with up to N rows and W columns, Construction 3.4 requires a structured reference string containing $O(N^2W)$ group elements. The N^2 factor comes from the "cross terms" c_i/c_j in the CRS. A standard approach in pairing-based cryptography to reduce the size of the CRS is to set $c_i = c^i$ for some secret value $c \in \mathbb{Z}_p$. In this case $c_i/c_j = c^{i-j}$ and multiple pairs of indices (i, j) can share the same cross term c^{i-j}. Using powers, we obtain a construction where the CRS contains $O(NW)$ group elements. We now describe the powers-variant of Assumption 3.3 we use for our construction:

Assumption 3.12 (N-Extended Bilinear Diffie-Hellman Exponent)**.** Let GroupGen be a prime-order bilinear group generator. For a security parameter λ and a bit $b \in \{0, 1\}$, we define the distribution $\mathcal{D}_{b,\lambda}$ as follows:

- Sample $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$ and sample generators $g, \hat{g} \xleftarrow{R} \mathbb{G} \setminus \{1\}$. Sample exponents $a, b, t \xleftarrow{R} \mathbb{Z}_p$ and $c \xleftarrow{R} \mathbb{Z}_p^*$ and define
$$\mathsf{params} = \left(1^\lambda, \mathcal{G}, g, \hat{g}, g^a, \hat{g}^a, \hat{g}^b, g^t, \hat{g}^t, \{g^{c^i}, \hat{g}^{c^i}, \hat{g}^{abc^i}\}_{i\in[-N,N]\setminus\{0\}}, \{\hat{g}^{tc^i}\}_{i\in[N]}\right).$$
- If $b = 0$, let $T = e(g, \hat{g})^{abt}$. If $b = 1$, sample $T \xleftarrow{R} \mathbb{G}_T$. Output (params, T).

We say the decisional N-extended bilinear Diffie-Hellman exponent assumption holds with respect to GroupGen if distributions $\mathcal{D}_0 = \{\mathcal{D}_{0,\lambda}\}_{\lambda\in\mathbb{N}}$ and $\mathcal{D}_1 = \{\mathcal{D}_{1,\lambda}\}_{\lambda\in\mathbb{N}}$ are computationally indistinguishable.

Remark 3.13 (Comparison with Assumption 3.3)**.** Assumption 3.12 is essential the N-extended bilinear Diffie-Hellman assumption (Assumption 3.3) obtained by setting the exponents c_i to be powers $c_i = c^i$. However, the basic version obtained in this way is insecure since the adversary can choose three distinct indices i, j, k where $i-j+k = 0$ and compute
$$e(g^{abc^{i-j}}, g^{tc^k}) = e(g, g)^{abtc^{i-j+k}} = e(g, g)^{abt}.$$
This yields a trivial distinguisher for the assumption. To prevent this type of check, we introduce a second generator $\hat{g}$ and ask the adversary to distinguish between $e(g, \hat{g})^{abt}$ and a random target group element. The important distinction is the assumption now gives out $\hat{g}^{abc^i}$ and $\hat{g}^{tc^j}$, which are both encoded with respect to the generator $\hat{g}$. The adversarial strategy describe above would now yield $e(\hat{g}, \hat{g})^{abt}$, which is independent of the challenge $e(g, \hat{g})^{abt}$. We note that our approach is analogous to working over an *asymmetric* pairing group where g is a generator of $\mathbb{G}_1$ and $\hat{g}$ is a generator of $\mathbb{G}_2$ (or vice versa) and the pairing satisfies $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. In asymmetric groups, the adversary is not able to pair $\hat{g}^{abc^i}$ with $\hat{g}^{tc^j}$ (since they would both be elements of $\mathbb{G}_2$). We show that this version of the assumption also holds in the generic bilinear group model in Appendix A.2 (Theorem A.4).

Remark 3.14 (On the Use of Negative Powers)**.** Assumption 3.12 (and correspondingly, Construction 3.15) gives out g^{c^i} for $i \in [-N, N]\setminus\{0\}$, which in particular includes *negative* powers of c in the exponent. It is perhaps more customary to consider an assumption that only gives out *positive* powers. For instance, the bilinear Diffie-Hellman exponent (BDHE) assumption [BBG05] gives out g^{c^i} where $i \in [2N+1]\setminus\{N+1\}$. When the generator g is random, the version of the assumption that includes negative powers and the version that only includes positive powers are equivalent via a change of basis. In fact, when arguing security in the generic bilinear group model (see Appendix A.2), we first apply a change of basis to obtain an equivalent assumption that only includes positive powers (and then appeal to standard master theorems for hardness in the generic bilinear group model). We believe that the use of negative powers better highlights the cancellation structure of our construction and opt for the more intuitive presentation. However, it is straightforward (albeit notationally cumbersome) to describe our assumption and scheme with only positive powers in the exponent.

Construction 3.15 (Distributed Monotone-Policy Signatures with Shorter CRS)**.** Let λ be a security parameter and κ be a policy-family parameter. Let GroupGen be a prime-order bilinear group generator and let $p = p(\lambda)$ be the order of the group output by GroupGen. Let $\Phi = \{\Phi_\kappa\}_{\kappa\in\mathbb{N}}$ be a family of monotone policies over $N = N(\kappa)$ users and which may be described by a one-use linear secret-sharing scheme with width $W = W(\kappa)$. We represent each policy $\varphi \in \Phi$ with a matrix $\mathbf{M} \in \mathbb{Z}_p^{N\times W}$ and write $\mathbf{m}_i^\intercal$ to denote the i^{th} row of $\mathbf{M}$. We construct a distributed monotone-policy signature scheme $\Pi_{\mathsf{DMPS}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PartialVerify}, \mathsf{Preprocess}, \mathsf{Aggregate}, \mathsf{Verify})$ for the policy family Φ and message space $\mathcal{M} = \{\mathbb{Z}_{p(\lambda)}\}_{\lambda\in\mathbb{N}}$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^\kappa)$: On input the security parameter λ and the policy-family parameter κ, the setup algorithm samples $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$ and $g, \hat{g} \xleftarrow{\text{R}} \mathbb{G} \setminus \{1\}$. Let $N = N(\kappa)$ and $W = W(\kappa)$. The setup algorithm samples $\hat{u}, \hat{h} \xleftarrow{\text{R}} \mathbb{G}$, $c \xleftarrow{\text{R}} \mathbb{Z}_p^*$, and $\mathbf{s} \xleftarrow{\text{R}} \mathbb{Z}_p^W$. It sets $B = e(g, \hat{g})^{s_1}$. It outputs the common reference string

$$\mathsf{crs} = \big(\mathcal{G}, g, \hat{g}, B, \hat{u}, \hat{h}, \{g^{c^i}, \hat{g}^{c^i\mathbf{s}}\}_{i\in[-N,N]\setminus\{0\}}\big). \tag{3.12}$$

- $\mathsf{KeyGen}(\mathsf{crs})$: On input the common reference string crs (with components parsed according to Eq. (3.12)), the key-generation algorithm samples $\alpha \xleftarrow{\text{R}} \mathbb{Z}_p$ and computes $\hat{v}_i' = (\hat{g}^{c^i})^\alpha$ for all $i \in [-N, N] \setminus \{0\}$. The algorithm sets $A = e(g, \hat{g})^\alpha$ and

$$\mathsf{vk} = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, A) \quad \text{and} \quad \mathsf{ht} = \{\hat{v}_i'\}_{i\in[-N,N]\setminus\{0\}}$$

  The algorithm outputs the verification key vk, the aggregation hint ht and the signing key $\mathsf{sk} = (\mathsf{vk}, \alpha)$.

- $\mathsf{Sign}(\mathsf{sk}, m)$: On input the signing key $\mathsf{sk} = (\mathsf{vk}, \alpha)$ where $\mathsf{vk} = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, A)$ and a message $m \in \mathbb{Z}_p$, the signing algorithm samples $r \xleftarrow{\text{R}} \mathbb{Z}_p$ and outputs the signature $\sigma = (g^r, \hat{g}^\alpha(\hat{u}^m\hat{h})^r)$.

- $\mathsf{PartialVerify}(\mathsf{vk}, m, \sigma)$: On input a verification key $\mathsf{vk} = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, A)$, a message $m \in \mathbb{Z}_p$, and a signature $\sigma = (\sigma_1, \hat{\sigma}_2)$, the partial verification algorithm outputs 1 if

$$A \cdot e(\sigma_1, \hat{u}^m\hat{h}) = e(g, \hat{\sigma}_2).$$

- $\mathsf{Preprocess}(\mathsf{crs}, \varphi, \{(\mathsf{vk}_\ell, \mathsf{ht}_\ell)\}_{\ell\in[N]})$: On input the common reference string crs (with components parsed according to Eq. (3.12)), a policy $\varphi = \mathbf{M} \in \mathbb{Z}_p^{N\times W}$, and a collection of verification keys vk_ℓ and hints $\mathsf{ht}_\ell = \{v_{\ell,i}\}_{i\in[-N,N]\setminus\{0\}}$, the preprocessing algorithm computes

$$\hat{z} = \prod_{\ell\in[N]} \hat{g}^{\mathbf{m}_\ell^\intercal(c^\ell\mathbf{s})}\hat{v}_{\ell,\ell}.$$

  Then for each $\ell \in [N]$, it computes

$$\hat{v}_\ell = \prod_{\substack{i\in[N]\\ i\neq\ell}} \hat{v}'_{i,i-\ell} \cdot \hat{g}^{\mathbf{m}_i^\intercal(c^{i-\ell}\mathbf{s})}.$$

  Both $\hat{z}$ and $\hat{v}_\ell$ are computed using the policy $\mathbf{m}_\ell^\intercal$ and the elements $\hat{g}^{c^\ell\mathbf{s}}$ from the crs. Then, it sets

$$\mathsf{vk}_\varphi = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, B, \hat{z}) \quad \text{and} \quad \mathsf{ak}_\varphi = (\mathbf{M}, \hat{z}, \{g^{c^{-\ell}}, \hat{v}_\ell\}_{\ell\in[N]}).$$

  Finally, the preprocessing algorithm outputs the verification key vk_φ and the aggregation key ak_φ.

- $\mathsf{Aggregate}(\mathsf{ak}_\varphi, \{\sigma_\ell\}_{\ell\in S})$: On input the the aggregation key $\mathsf{ak}_\varphi = (\mathbf{M}, \hat{z}, \{g^{c^{-\ell}}, \hat{v}_\ell\}_{\ell\in[N]})$, and a collection of signatures $\sigma_\ell = (\sigma_{\ell,1}, \hat{\sigma}_{\ell,2})$ for $\ell \in S \subseteq [N]$, the aggregation algorithm proceeds as follows:

  - First, it checks if S satisfies the policy $\mathbf{M}$. If not, the aggregation algorithm outputs $\perp$. Otherwise, let $\boldsymbol{\omega} \in \mathbb{Z}_p^N$ be a reconstruction vector where $\boldsymbol{\omega}^\intercal\mathbf{M} = \mathbf{e}_1^\intercal$, and moreover, $\omega_\ell = 0$ for all $\ell \in [N] \setminus S$.
  - Compute

$$\sigma_{\mathsf{agg},1} = \prod_{\ell\in S} \sigma_{\ell,1}^{\omega_\ell} \quad \text{and} \quad \hat{\sigma}_{\mathsf{agg},2} = \prod_{\ell\in S} \hat{\sigma}_{\ell,2}^{\omega_\ell}\hat{v}_\ell^{\omega_\ell} \quad \text{and} \quad \sigma_{\mathsf{agg},3} = \prod_{\ell\in S} \big(g^{c^{-\ell}}\big)^{\omega_\ell}.$$

    Output the signature $\sigma_{\mathsf{agg}} = (\sigma_{\mathsf{agg},1}, \hat{\sigma}_{\mathsf{agg},2}, \sigma_{\mathsf{agg},3})$.

- $\mathsf{Verify}(\mathsf{vk}_\varphi, m, \sigma_{\mathsf{agg}})$: On input the verification key $\mathsf{vk}_\varphi = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, B, \hat{z})$, a message $m \in \mathbb{Z}_p$, and a signature $\sigma_{\mathsf{agg}} = (\sigma_{\mathsf{agg},1}, \hat{\sigma}_{\mathsf{agg},2}, \sigma_{\mathsf{agg},3})$, the verification algorithm outputs 1 if

$$B \cdot e(g, \hat{\sigma}_{\mathsf{agg},2}) = e(\sigma_{\mathsf{agg},1}, \hat{u}^m \hat{h}) \cdot e(\sigma_{\mathsf{agg},3}, \hat{z}). \tag{3.13}$$

  Otherwise, the verification algorithm outputs 0.

Correctness and security analysis. The correctness and security analysis for Construction 3.15 follow a very similar structure as that for Construction 3.4. We state the relevant theorems below, and defer their formal proofs to Appendix B.

Theorem 3.16 (Signing Correctness)**.** *Construction 3.15 satisfies signing correctness.*

Theorem 3.17 (Aggregation Correctness)**.** *Construction 3.15 satisfies aggregation correctness.*

Theorem 3.18 (Static Unforgeability)**.** *Suppose the N-extended bilinear Diffie-Hellman exponent assumption (Assumption 3.12) holds with respect to* GroupGen*. Then, Construction 3.15 satisfies static unforgeability in the registered-key model for policies with N users.*

Extensions. Much like Construction 3.4, proving well-formedness of the verification key and the aggregation hint in Construction 3.15 reduces to a NIZK proof of knowledge of the secret exponent α such that $A = e(g, \hat{g})^\alpha$ where A is the component of the verification key. This can be done using the same techniques as described in Remark 3.9. As was the case with Construction 3.4, the components in the aggregation hint can be efficiently validated using the pairing (and the components of the CRS).

Similarly, using share virtualization (Remark 3.10), we can extend Construction 3.15 to support policies with a K-use linear secret sharing scheme. This increases the CRS size, the user verification key size, the partial signature size, and the aggregation key size by a factor of K, but does *not* affect the size of the aggregated verification key or aggregated signature. We summarize the overall instantiation and its efficiency properties in the following corollary:

Corollary 3.19 (Distributed Monotone-Policy Signatures from Pairings)**.** *Let λ be a security parameter. Suppose the N-extended bilinear Diffie-Hellman exponent assumption (Assumption 3.12) holds with respect to* GroupGen *for all polynomials $N = N(\lambda)$. Let $\Phi = \{\Phi_\kappa\}_{\kappa \in \mathbb{N}}$ be the class of policies that can be described by a K-use linear-secret sharing scheme with $N = N(\kappa)$ users and width $W = W(\kappa)$, where N, W are polynomially-bounded. Then, there exists a (statically-secure) distributed monotone-policy signature scheme for Φ in the registered-key model with the following efficiency properties:*

- **CRS size:** *The CRS contains $O(KNW)$ group elements.*
- **User key size:** *A user's verification key* vk *contains $4 + K$ group elements and the aggregation hint* ht *contains $K(2KN - 1)$ group elements. The first 4 group elements in each user's verification key are fixed (i.e., group elements from the common reference string). The number of user-specific group elements in each verification key is K.*
- **Partial signature size:** *A user's partial signature σ contains $2K$ group elements.*
- **Aggregation key size:** *The size of an aggregation key ak_φ for a policy $\varphi \in \Phi$ contains $2KN + 1$ group elements.*
- **Aggregate verification key size:** *An aggregated verification key vk_φ for a policy $\varphi \in \Phi$ contains 6 group elements.*
- **Aggregate signature size:** *An aggregate signature contains 3 group elements. The aggregate verification algorithm requires performing 3 pairings (and $O(1)$ group operations).*

4 Distributed Monotone-Policy Encryption

Distributed monotone-policy encryption [GKPW24, ADM+24, DJWW25] is the analog of distributed monotone-policy signatures in the encryption setting. Much like the signature case, users in this model choose their own individual public and private keys and publish their public keys to a public-key directory. Thereafter, anyone can encrypt a message to any subset of public keys and with respect to an arbitrary access policy. Any quorum of users that satisfies the access policy can come together and decrypt (using their secret keys). In this section, we show that our distributed monotone-policy signature scheme from Section 3 directly extends to yield a distributed monotone-policy encryption scheme for the same underlying policy family. We start by introducing the formal definition below. Then, in Remark 4.3, we compare our notion to related notions from prior work.

As in Section 3, we again formulate our correctness and security notions in the semi-malicious setting (i.e., the registered key model) where we only require the properties to hold with respect to public keys that are in the support of the honest key-generation algorithm. Using the same approach from Remark 3.2, we can compile a semi-malicious scheme to one that is correct and secure against adversarially-chosen keys using (simulation-sound) non-interactive zero-knowledge proofs. Thus, for ease of exposition, we exclusively focus on semi-malicious security in this paper.

Definition 4.1 (Distributed Monotone-Policy Encryption [GKPW24, DJWW25, adapted])**.** Let λ be a security parameter and κ be a policy-family parameter. Let $\Phi = \{\Phi_\kappa\}_{\kappa\in\mathbb{N}}$ be a family of monotone policies. A tag-based distributed monotone-policy encryption scheme for policy space Φ consists of a tuple of efficient algorithms $\Pi_{\mathsf{DMPE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Preprocess}, \mathsf{Encrypt}, \mathsf{PartialDec}, \mathsf{PartialVerify}, \mathsf{Decrypt})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^\kappa) \to \mathsf{crs}$: On input the security parameter λ and the policy-family parameter κ, the setup algorithm outputs the common reference string crs. We assume the common reference string implicitly includes a description of the message space $\mathcal{M}$ and the tag space $\mathcal{T}$ for the encryption scheme.
- $\mathsf{KeyGen}(\mathsf{crs}) \to (\mathsf{pk}, \mathsf{ht}, \mathsf{sk})$: On input the common reference string crs, the key-generation algorithm outputs a public key pk, an aggregation hint ht, and a secret key sk.
- $\mathsf{Preprocess}(\mathsf{crs}, \varphi, \{(\mathsf{pk}_\ell, \mathsf{ht}_\ell)\}_{\ell\in[N]}) \to (\mathsf{ek}_\varphi, \mathsf{ak}_\varphi)$: On input the common reference string crs, a policy $\varphi\colon 2^{[N]} \to \{0,1\}$, and N public keys pk_ℓ together with their aggregation hints ht_ℓ, the preprocessing algorithm outputs an encryption key ek_φ and an aggregation key ak_φ. We assume that the encryption key ek_φ includes a description of the message space $\mathcal{M}$ and the tag space $\mathcal{T}$ (from crs). This algorithm is deterministic.
- $\mathsf{Encrypt}(\mathsf{ek}_\varphi, \tau, m) \to \mathsf{ct}$: On input an encryption key ek_φ, a tag $\tau \in \mathcal{T}$, and a message $m \in \mathcal{M}$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{PartialDec}(\mathsf{sk}, \tau, \mathsf{ct}) \to \sigma$: On input a secret key sk, a tag τ, and a ciphertext ct, the partial decryption algorithm outputs a partial decryption σ.
- $\mathsf{PartialVerify}(\mathsf{pk}, \tau, \mathsf{ct}, \sigma) \to b$: On input a public key pk, a tag τ, a ciphertext ct, and a partial decryption σ, the partial-verification algorithm outputs a bit $b \in \{0,1\}$. This algorithm is deterministic.
- $\mathsf{Decrypt}(\mathsf{ak}_\varphi, \mathsf{ct}, \{\sigma_\ell\}_{\ell\in S}) \to m$: On input the aggregation key ak_φ, the ciphertext ct, and a set of partial decryptions σ_ℓ, the decryption algorithm outputs a message $m \in \mathcal{M}$ (or a special symbol $\perp$ to indicate decryption failure). This algorithm is deterministic.

We require Π_{DMPE} to satisfy the following properties:

- **Partial decryption correctness:** For all $\lambda, \kappa \in \mathbb{N}$, all crs in the support of $\mathsf{Setup}(1^\lambda, 1^\kappa)$, all tags $\tau \in \mathcal{T}$, all messages $m \in \mathcal{M}$ (where $\mathcal{T}, \mathcal{M}$ denotes the tag space and message space defined by crs), all $\{(\mathsf{pk}_\ell, \mathsf{sk}_\ell, \mathsf{ht}_\ell)\}_{\ell\in[N]}$ where $(\mathsf{pk}_\ell, \mathsf{sk}_\ell, \mathsf{ht}_\ell)$ is in the support of $\mathsf{KeyGen}(\mathsf{crs})$, all policies $\varphi \in \Phi_\kappa$ where $\varphi\colon 2^{[N]} \to \{0,1\}$, and all indices $i^* \in [N]$, it holds that

$$\Pr\left[\mathsf{PartialVerify}(\mathsf{pk}_{i^*}, \tau, \mathsf{ct}, \sigma_{i^*}) = 1 : \begin{array}{c}(\mathsf{ek}_\varphi, \mathsf{ak}_\varphi) = \mathsf{Preprocess}(\mathsf{crs}, \varphi, \{(\mathsf{pk}_\ell, \mathsf{ht}_\ell)\}_{\ell\in[N]}) \\ \mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{ek}_\varphi, \tau, m) \\ \sigma_{i^*} \leftarrow \mathsf{PartialDec}(\mathsf{sk}_{i^*}, \tau, \mathsf{ct})\end{array}\right] = 1.$$

- **Aggregation correctness:** For all $\lambda, \kappa \in \mathbb{N}$, all crs in the support of $\mathsf{Setup}(1^\lambda, 1^\kappa)$, all tags $\tau \in \mathcal{T}$, all messages $m \in \mathcal{M}$ (where $\mathcal{T}, \mathcal{M}$ denotes the tag space and message space defined by crs), all $\{(\mathsf{pk}_\ell, \mathsf{sk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [N]}$ where $(\mathsf{pk}_\ell, \mathsf{sk}_\ell, \mathsf{ht}_\ell)$ is in the support of $\mathsf{KeyGen}(\mathsf{crs})$, all policies $\varphi \in \Phi_\kappa$, where $\varphi \colon 2^{[N]} \to \{0,1\}$, all sets $S \subseteq [N]$ where $\varphi(S) = 1$, and setting $(\mathsf{ek}_\varphi, \mathsf{ak}_\varphi) = \mathsf{Preprocess}(\mathsf{crs}, \varphi, \{(\mathsf{pk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [N]})$, and for all ciphertexts ct in the support of $\mathsf{Encrypt}(\mathsf{ek}_\varphi, \tau, m)$, all partial decryptions $\{\sigma_\ell\}_{\ell \in S}$ where for all $\ell \in S$, $\mathsf{PartialVerify}(\mathsf{pk}_\ell, \tau, \mathsf{ct}, \sigma_\ell) = 1$, it holds that
$$\mathsf{Decrypt}(\mathsf{ak}_\varphi, \mathsf{ct}, \{\sigma_\ell\}_{\ell \in S}) = m.$$

- **Static tag-based CCA-security:** For a security parameter λ and an adversary $\mathcal{A}$, we define the static tag-based CCA-security game as follows:

  1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the policy parameter 1^κ together with a policy $\varphi \in \Phi_\kappa$. In addition, algorithm $\mathcal{A}$ commits to a set of corrupted users $C \subseteq [N]$.
  2. The challenger checks that $\varphi(C) = 0$. If not the challenger halts with output $b' = 0$.
  3. The challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa)$. Then, for each index $\ell \in [N] \setminus C$, the challenger samples a key $(\mathsf{pk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell) \leftarrow \mathsf{KeyGen}(\mathsf{crs})$. It gives crs together with $\{(\mathsf{pk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [N] \setminus C}$ to $\mathcal{A}$.
  4. Algorithm $\mathcal{A}$ now specifies the key-generation randomness $\rho_\ell \in \{0,1\}^*$ used to generate the keys for each of the corrupted users $\ell \in C$.
  5. For each $\ell \in C$, the challenger computes $(\mathsf{pk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell) \leftarrow \mathsf{KeyGen}(\mathsf{crs}; \rho_\ell)$. The challenger computes and gives $(\mathsf{ek}_\varphi, \mathsf{ak}_\varphi) = \mathsf{Preprocess}(\mathsf{crs}, \varphi, \{(\mathsf{ek}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [N]})$ to the adversary $\mathcal{A}$.
  6. The adversary can now issue partial decryption queries by specifying an index $\ell \in [N] \setminus C$, a tag $\tau \in \mathcal{T}$, and a ciphertext ct. The challenger responds with $\mathsf{PartialDec}(\mathsf{sk}_\ell, \tau, \mathsf{ct})$.
  7. After $\mathcal{A}$ is finished making partial decryption queries, it outputs a pair of messages $m_0, m_1 \in \mathcal{M}$. The challenger samples a random tag $\tau^* \xleftarrow{R} \mathcal{T}$ and responds with the challenge tag τ^* together with the challenge ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Encrypt}(\mathsf{ek}_\varphi, \tau^*, m_b)$.
  8. Algorithm $\mathcal{A}$ can continue making partial decryption queries, except it is *not* allowed to query for a decryption with tag $\tau = \tau^*$. The challenger responds to each query as before. At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that Π_{DMPE} satisfies static tag-based CCA-security in the registered-key model if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,
$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda)$$
  in the tag-based CCA-security game defined above. We say that Π_{DMPE} satisfies static tag-based CCA-security in the registered-key model for policies with $N = N(\lambda)$ users if the above holds for all efficient adversaries $\mathcal{A}$ that outputs policies φ on exactly N users.

- **Succinctness:** There exists a universal polynomial $\mathsf{poly}(\cdot, \cdot)$ such that for all $\lambda, \kappa \in \mathbb{N}$, all crs in the support of $\mathsf{Setup}(1^\lambda, 1^\kappa)$, all policies $\varphi \in \Phi_\kappa$ over $N = N(\kappa)$ parties, all $(\mathsf{pk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell)$ in the support of $\mathsf{KeyGen}(\mathsf{crs})$, and setting $(\mathsf{ek}_\varphi, \mathsf{ak}_\varphi) = \mathsf{Preprocess}(\mathsf{crs}, \varphi, \{(\mathsf{pk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [N]})$, the following hold:
  - The size of the encryption key ek_φ and the size of the ciphertext output by $\mathsf{Encrypt}(\mathsf{ek}_\varphi, \cdot, \cdot)$ is $\mathsf{poly}(\lambda, \log N)$.
  - The size of the partial decryptions output by $\mathsf{PartialDec}(\mathsf{sk}, \cdot, \cdot)$ is $\mathsf{poly}(\lambda, \log N)$.

Remark 4.2 (Tag-Based CCA-Security)**.** Definition 4.1 defines a tag-based monotone-policy encryption scheme where each ciphertext is associated with a tag τ. In the security game, the adversary is allowed to make decryption queries on any ciphertext whose tag is distinct from the tag associated with the challenge ciphertext. This tag-based notion allows us to easily interpolate between the CPA-security and CCA-security notions considered in prior works [GKPW24, DJWW25]:

- **CPA-security:** The default notion of security considered in [GKPW24, DJWW25] is CPA-security where the adversary in the security game does not have the ability to request partial decryptions from honest users. A scheme that satisfies tag-based CCA-security trivially implies one that is CPA-secure by simply taking the tag to be a fixed value. Since the tag is a fixed value, we no longer need to explicitly include it as an input to the underlying algorithms.
- **CCA-security:** Both [GKPW24, DJWW25] also consider a stronger CCA-security notion where the adversary is allowed to request partial decryptions on any ciphertext other than the challenge ciphertext. It is straightforward to upgrade a scheme that satisfies tag-based CCA-security into a scheme with full CCA-security using classic transformations [CHK04, Kil06] (e.g., take the tag to be the verification key for a one-time signature scheme and include a signature on the ciphertext under the tag). Both [GKPW24, DJWW25] rely on this general strategy (c.f., [GKPW24, §6.2] and [DJWW25, §B]) to obtain CCA-secure distributed monotone-policy encryption schemes.

In this work, we focus on the intermediate tag-based notion because (1) our schemes naturally support this; and (2) it is sufficient for full CCA-security via standard techniques. At the same time, if CPA-security suffices for an application, then the tag can be set to a fixed string and dropped from the construction altogether. In this setting, the tag-based syntax does not introduce unnecessary overhead to the construction.

Remark 4.3 (Comparison with Related Notions)**.** Our definition of distributed monotone-policy encryption (Definition 4.1) combines features from threshold encryption with silent setup [GKPW24] and its adaptation to general policies from [DJWW25]. Here, we provide a comparison of the key differences between our definition and existing notions:

- **Policy-specific preprocessing:** Our definition generalizes threshold encryption with silent setup [GKPW24] to general policies, but where the policy is fixed at preprocessing time. The work of [GKPW24] shows how to support *dynamic* policies where the preprocessing algorithm takes as input the group of users, but the threshold can be chosen dynamically at encryption time. For similar reasons as discussed in Remark 3.2, it is only possible to support dynamic policies *and* fast encryption if we either allow preprocessing the policy in advance or if the policy has a short description (as is the case for threshold policies). In Section 5.2, we show that our techniques easily supports dynamic thresholds if we restrict our attention to threshold policies.

  Alternatively, we can also consider a setting *without* preprocesisng where both the group of users *and* the policy are specified at encryption time (and also provided to the decryption algorithm). In this setting, the running time of encryption and decryption both scale with the number of users and the description length of the policy, but the ciphertext size is still small. This setting essentially absorbs the Preprocess algorithm into the Encrypt and Decrypt algorithms. This more lightweight syntax is proposed in [DJWW25] and can be viewed as the direct generalization of distributed broadcast encryption [WQZD10, BZ14] to more general policies. Distributed broadcast encryption is a special case of distributed monotone-policy encryption where there is a single policy (a threshold policy with threshold set to 1).
- **Semi-malicious security:** Similar to the signature case, we work in the semi-malicious model. By the same argument as in Remark 3.2, we can lift a scheme to the fully malicious model using simulation-sound NIZK proofs of knowledge.
- **Static vs. adaptive security:** Similar to our distributed monotone-policy signature construction, we use a partitioning strategy to prove security (from q-type assumptions in the plain model). This restricts us to static security where the adversary must declare the set of corrupted users at the beginning of the security game. The work of [GKPW24] (for threshold policies) consider adaptive corruptions, but their security analysis relies on the generic group model.

Construction 4.4 (Distributed Monotone-Policy Encryption)**.** Let λ be a security parameter and κ be a policy-family parameter. Let GroupGen be a prime-order bilinear group generator and let $p = p(\lambda)$ be the order of the group output by GroupGen. Let $\Phi = \{\Phi_\kappa\}_{\kappa\in\mathbb{N}}$ be a family of monotone policies over $N = N(\kappa)$ users and which may be described by a one-use linear secret-sharing scheme with width $W = W(\kappa)$. We represent each policy $\varphi \in \Phi$ with a matrix $\mathbf{M} \in \mathbb{Z}_p^{N\times W}$, and write $\mathbf{m}_i^\intercal$ to denote the i^{th} row of $\mathbf{M}$. We construct a distributed monotone-policy encryption scheme $\Pi_{\mathsf{DMPE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Preprocess}, \mathsf{Encrypt}, \mathsf{PartialDec}, \mathsf{PartialVerify}, \mathsf{Decrypt})$ for the policy family Φ as follows:

- $\mathsf{Setup}(1^\lambda, 1^\kappa)$: On input the security parameter λ and the policy-family parameter κ, the setup algorithm proceeds exactly as in Construction 3.15. Namely, it samples $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$ and $g, \hat{g} \xleftarrow{R} \mathbb{G} \setminus \{1\}$. Let $N = N(\kappa)$ and $W = W(\kappa)$. The setup algorithm samples $\hat{u}, \hat{h} \xleftarrow{R} \mathbb{G}$, $c \xleftarrow{R} \mathbb{Z}_p^*$, and $\mathbf{s} \xleftarrow{R} \mathbb{Z}_p^W$. It sets $B = e(g, \hat{g})^{s_1}$. It outputs the common reference string

$$\mathsf{crs} = \left(\mathcal{G}, g, \hat{g}, B, \hat{u}, \hat{h}, \{g^{c^i}, \hat{g}^{c^i \mathbf{s}}\}_{i \in [-N,N] \setminus \{0\}}\right). \tag{4.1}$$

  The message space for the encryption scheme is $\mathcal{M} = \mathbb{G}_T$ and the tag space is $\mathcal{T} = \mathbb{Z}_p$.

- $\mathsf{KeyGen}(\mathsf{crs})$: On input the common reference string crs (with components parsed according to Eq. (4.1)), the key-generation algorithm samples $\alpha \xleftarrow{R} \mathbb{Z}_p$ and computes $\hat{v}'_i = (\hat{g}^{c^i})^\alpha$ for all $i \in [-N, N] \setminus \{0\}$. The algorithm sets $A = e(g, \hat{g})^\alpha$ and

$$\mathsf{pk} = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, A) \quad \text{and} \quad \mathsf{ht} = \{\hat{v}'_i\}_{i \in [-N,N] \setminus \{0\}}.$$

  The algorithm outputs the public key pk, the aggregation hint ht, and the secret key $\mathsf{sk} = (\mathsf{pk}, \alpha)$.

- $\mathsf{Preprocess}(\mathsf{crs}, \varphi, \{(\mathsf{pk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [N]})$: On input the common reference string crs (with components parsed according to Eq. (4.1)), a policy $\varphi = \mathbf{M} \in \mathbb{Z}_p^{N \times W}$, and a collection of public keys pk_ℓ and hints $\mathsf{ht}_\ell = \{v'_{\ell,i}\}_{i \in [-N,N] \setminus \{0\}}$, the preprocessing algorithm computes

$$\hat{z} = \prod_{\ell \in [N]} \hat{g}^{\mathbf{m}_\ell^\top (c^\ell \mathbf{s})} \hat{v}'_{\ell,\ell},$$

  Then for each $\ell \in [N]$, it computes

$$\hat{v}_\ell = \prod_{\substack{i \in [N] \\ i \neq \ell}} \hat{v}'_{i,i-\ell} \cdot \hat{g}^{\mathbf{m}_i^\top (c^{i-\ell} \mathbf{s})}.$$

  Both $\hat{z}$ and $\hat{v}_\ell$ are computed using the policy $\mathbf{m}_\ell^\top$ and the elements $\hat{g}^{c^\ell \mathbf{s}}$ from the crs. Then, it sets

$$\mathsf{ek}_\varphi = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, B, \hat{z}) \quad \text{and} \quad \mathsf{ak}_\varphi = (\mathbf{M}, \hat{z}, \{g^{c^{-\ell}}, \hat{v}_\ell\}_{\ell \in [N]}).$$

  Finally, the preprocessing algorithm outputs the encryption key ek_φ and the aggregation key ak_φ.

- $\mathsf{Encrypt}(\mathsf{ek}_\varphi, \tau, m)$: On input an encryption key $\mathsf{ek}_\varphi = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, B, \hat{z})$, a tag $\tau \in \mathbb{Z}_p$, and a message $m \in \mathbb{G}_T$, the encryption algorithm samples a random exponent $t \xleftarrow{R} \mathbb{Z}_p$. It outputs the ciphertext

$$\mathsf{ct} = (\tau, C_1, c_2, \hat{c}_3, \hat{c}_4) = \left(\tau,\ B^t \cdot m,\ g^t,\ (\hat{u}^\tau \hat{h})^t,\ \hat{z}^t\right).$$

- $\mathsf{PartialDec}(\mathsf{sk}, \tau, \mathsf{ct})$: On input a secret key $\mathsf{sk} = (\mathsf{pk}, \alpha)$ where $\mathsf{pk} = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, A)$, a tag $\tau \in \mathbb{Z}_p$, and a ciphertext ct, the partial decryption algorithm samples $r \xleftarrow{R} \mathbb{Z}_p$ and outputs $\sigma = (g^r,\ \hat{g}^\alpha (\hat{u}^\tau \hat{h})^r)$.

- $\mathsf{PartialVerify}(\mathsf{pk}, \tau, \mathsf{ct}, \sigma)$: On input a key $\mathsf{pk} = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, A)$, a tag $\tau \in \mathbb{Z}_p$, a ciphertext $\mathsf{ct} = (\tau', C_1, c_2, \hat{c}_3, \hat{c}_4)$, and a partial decryption $\sigma = (\sigma_1, \hat{\sigma}_2)$, the partial verification algorithm outputs 0 if $\tau \neq \tau'$. Otherwise, it outputs 1 if $A \cdot e(\sigma_1, \hat{u}^\tau \hat{h}) = e(g, \hat{\sigma}_2)$.

- $\mathsf{Decrypt}(\mathsf{ak}_\varphi, \tau, \mathsf{ct}, \{\sigma_\ell\}_{\ell \in S})$: On input the aggregation key $\mathsf{ak}_\varphi = (\mathbf{M}, \hat{z}, \{\hat{v}_\ell\}_{\ell \in [N]})$, a tag $\tau \in \mathbb{Z}_p$, a ciphertext $\mathsf{ct} = (\tau', C_1, c_2, \hat{c}_3, \hat{c}_4)$, and a collection of partial decryptions $\sigma_\ell = (\sigma_{\ell,1}, \hat{\sigma}_{\ell,2})$ for $\ell \in S \subseteq [N]$, the decryption algorithm proceeds as follows:

  1. First, it checks if S satisfies the policy $\mathbf{M}$. If not, the decryption algorithm outputs $\perp$. Otherwise, let $\boldsymbol{\omega} \in \mathbb{Z}_p^N$ be a reconstruction vector where $\boldsymbol{\omega}^\top \mathbf{M} = \mathbf{e}_1^\top$, and moreover, $\omega_\ell = 0$ for all $\ell \in [N] \setminus S$.
  2. Compute the aggregated decryption components:

$$\sigma_{\mathrm{agg},1} = \prod_{\ell \in S} \sigma_{\ell,1}^{\omega_\ell} \quad \text{and} \quad \hat{\sigma}_{\mathrm{agg},2} = \prod_{\ell \in S} \hat{\sigma}_{\ell,2}^{\omega_\ell} \hat{v}_\ell^{\omega_\ell} \quad \text{and} \quad \sigma_{\mathrm{agg},3} = \prod_{\ell \in S} \left(g^{c^{-\ell}}\right)^{\omega_\ell}.$$

  3. Output $C_1 \cdot e(\sigma_{\mathrm{agg},1}, \hat{c}_3)^{-1} \cdot e(c_2, \hat{\sigma}_{\mathrm{agg},2}) \cdot e(\sigma_{\mathrm{agg},3}, \hat{c}_4)^{-1}$.

Correctness and security analysis. The correctness and security analysis for Construction 4.4 follow a very similar structure as that for Construction 3.4, just adapted to the setting of encryption. We state the relevant theorems below, and defer their formal proofs to Appendix C.

Theorem 4.5 (Partial Decryption Correctness)**.** *Construction 4.4 satisfies partial decryption correctness.*

Theorem 4.6 (Aggregation Correctness)**.** *Construction 4.4 satisfies aggregation correctness.*

Theorem 4.7 (Static Tag-Based CCA-Security)**.** *Suppose the decisional N-extended bilinear Diffie-Hellman exponent assumption (Assumption 3.12) holds with respect to* GroupGen*. Then, Construction 4.4 satisfies static tag-based CCA-security in the registered-key model for policies with N users.*

Remark 4.8 (Signature-Based Witness Encryption)**.** We can also view Construction 4.4 as a signature-based witness encryption scheme [DHMW23, ADM+24] for the Boneh-Boyen signature scheme. In a signature-based witness encryption scheme, users encrypt to a tag τ, a collection of verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_N$ and a policy φ. Decryption is possible if one has a collection of valid signatures $\{\sigma_i\}_{i \in S}$ (with respect to $\{\mathsf{vk}_i\}_{i \in S}$) where $\varphi(S) = 1$. Previous works [DHMW23, ADM+24] considered the setting for threshold policies. The work of [DHMW23] gives a construction where the ciphertext size scales *linearly* with the number of users N associated with the policy while the subsequent works [GKPW24, ADM+24] give constructions with succinct ciphertexts for threshold policies either in the generic group model [GKPW24] or using indistinguishability obfuscation [ADM+24]. Among these schemes, the obfuscation-based construction of [ADM+24] also has short public keys. Our work gives a construction with succinct ciphertexts from q-type assumptions in the plain model. However, the size of each user's public key scales linearly with N (similar to [GKPW24]). We refer to Table 2 for a more detailed comparison.

Extensions. As was the case for our signature schemes from Section 3, we can lift our scheme from the semi-malicious model to the fully malicious model using NIZKs proof of knowledge. As in Remark 3.9, it suffices to use a NIZK proof of knowledge of discrete log (i.e., prove knowledge of α such that $A = e(g, \hat{g})^{\alpha}$, where A is the component in the public key). The well-formedness of the aggregation hint can then be checked using the pairing and the components of the CRS.

Similarly, using share virtualization (Remark 3.10), we can extend Construction 4.4 to support policies with a K-use linear secret sharing scheme. This increases the size of the common reference string, the user public key, the aggregation key, and the partial decryptions by a factor of K, but does not affect the size of the aggregated encryption key or the ciphertexts. We summarize the overall instantiation and its efficiency properties in the following corollary:

Corollary 4.9 (Distributed Monotone-Policy Encryption from Pairings)**.** *Let λ be a security parameter. Suppose the N-extended bilinear Diffie-Hellman exponent assumption (Assumption 3.12) holds with respect to* GroupGen *for all polynomials $N = N(\lambda)$. Let $\Phi = \{\Phi_\kappa\}_{\kappa \in \mathbb{N}}$ be the class of policies that can be described by a K-use linear-secret sharing scheme with $N = N(\kappa)$ users and width $W = W(\kappa)$, where N, W are polynomially-bounded. Then, there exists a (statically-secure) distributed monotone-policy encryption scheme for Φ in the registered-key model with the following efficiency properties:*

- **CRS size:** *The CRS contains $O(KNW)$ group elements.*
- **User key size:** *A user's public key* pk *contains $4 + K$ group elements and the aggregation hint* ht *contains $K(2KN - 1)$ group elements. The first 4 group elements in each user's verification key are fixed (i.e., group elements from the common reference string). The number of user-specific group elements in each verification key is K.*
- **Aggregation key size:** *The size of an aggregation key ak_φ for a policy $\varphi \in \Phi$ contains $2KN + 1$ group elements.*
- **Aggregate encryption key size:** *An aggregated encryption key ek_φ for a policy $\varphi \in \Phi$ contains 6 group elements.*
- **Ciphertext size:** *A ciphertext contains 4 group elements (excluding the tag).*
- **Partial decryption size:** *A user's partial decryption σ contains $2K$ group elements.*

5 Supporting Threshold Policies with a Linear-Size CRS

Our construction of distributed monotone-policy signatures (Constructions 3.4 and 3.15) and encryption (Construction 4.4) capture threshold policies as a special case (using the fact that threshold policies have a linear secret sharing scheme). However, if we restrict ourselves to the particular case of threshold policies, we can obtain constructions that have the following stronger functionality and efficiency properties:

- **Quasi-linear-size CRS:** Instantiating Constructions 3.15 and 4.4 for the special case of threshold policies leads to a construction whose CRS scales *quadratically* with the number of users associated with a policy. The quadratic blow-up arises from the fact that the scheme can support *any* policy with a linear secret sharing scheme over N parties (and width at most N). As we show in this section (see Section 1.2 for an overview), to support threshold policies, it suffices to consider a scheme that only supports a *single* fixed policy (which is hard-coded into the CRS). This allows us to derive schemes where the size of the CRS scales *quasi-linearly* with the maximum number of parties N in with the policy.
- **Dynamic thresholds:** The constructions in Sections 3 and 4 considered a setting where the preprocessing algorithm fixes a single policy φ. For the special case of threshold policies, we can consider a more general notion where the threshold T is determined *dynamically* either at verification time (in the setting of signatures) or at encryption time (in the setting of encryption). In this section, we show how to extend our scheme to support dynamic thresholds with only $O(\log N)$ overhead in the size of the verification or encryption key, and no overhead in the size of the aggregate signature or the ciphertext.

Taken together, our resulting construction matches the syntax and efficiency requirements of threshold signatures and threshold encryption with silent setup.

5.1 Threshold Signatures with Silent Setup

In this section, we show how to specialize Construction 3.15 to the setting of threshold policies and obtain a threshold signature scheme with silent setup [DCX+23, GJM+24]. Like [DCX+23, GJM+24], our scheme supports dynamic thresholds. As in Section 3, we focus on a simpler definition by working in the registered key model (where correctness and security hold against semi-malicious adversaries) and consider a static notion of security where the adversary has to declare the set of corrupted users upfront. We refer to Remark 3.2 for more discussion on these differences.

Definition 5.1 (Threshold Signatures with Silent Setup [GJM+24, adapted])**.** Let λ be a security parameter and $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ be the message space. A threshold signature scheme with silent setup over message space $\mathcal{M}$ consists of a tuple of efficient algorithms $\Pi_{\mathsf{STS}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PartialVerify}, \mathsf{Preprocess}, \mathsf{Aggregate}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^N) \to \mathsf{crs}$: On input the security parameter λ, and a bound on the number of users in a quorum, the setup algorithm outputs the common reference string crs.
- $\mathsf{KeyGen}(\mathsf{crs}) \to (\mathsf{vk}, \mathsf{ht}, \mathsf{sk})$: On input the common reference string crs, the key-generation algorithm outputs the verification key vk, an aggregation hint ht, and a signing key sk.
- $\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$: On input a signing key sk and a message $m \in \mathcal{M}$, the signing algorithm outputs a signature σ.
- $\mathsf{PartialVerify}(\mathsf{vk}, m, \sigma) \to b$: On input a verification key vk, a message $m \in \mathcal{M}$, and a signature σ, the partial-verification algorithm outputs a bit $b \in \{0, 1\}$. This algorithm is deterministic.
- $\mathsf{Preprocess}(\mathsf{crs}, \{(\mathsf{vk}_\ell, \mathsf{ht}_\ell)\}_{\ell\in[L]}) \to (\mathsf{vk}_{\mathsf{agg}}, \mathsf{ak})$: On input the common reference string crs, a collection of L verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_L$ together with their aggregation hints $\mathsf{ht}_1, \ldots, \mathsf{ht}_L$,[10] the preprocessing algorithm outputs an aggregated verification key $\mathsf{vk}_{\mathsf{agg}}$ and an aggregation key ak. This algorithm is deterministic.
- $\mathsf{Aggregate}(\mathsf{ak}, \{\sigma_\ell\}_{\ell\in S}) \to \sigma_{\mathsf{agg}}$: On input the aggregation key ak and a set of signatures σ_ℓ, the aggregation algorithm outputs an aggregate signature σ_{agg}. This algorithm is deterministic.

[10] Recall that $\{(\mathsf{vk}_\ell, \mathsf{ht}_\ell)\}_{\ell\in[L]}$ denotes an *ordered* set of verification keys and aggregation hints.

- $\mathsf{Verify}(\mathsf{vk}_{\mathsf{agg}}, m, \sigma_{\mathsf{agg}}, T) \to b$: On input an aggregate verification key $\mathsf{vk}_{\mathsf{agg}}$, a message $m \in \mathcal{M}$, an aggregate signature σ_{agg}, and a threshold T, the verification algorithm outputs a bit $b \in \{0,1\}$. This algorithm is deterministic.

We require Π_{STS} satisfy the following properties:

- **Signing correctness:** For all $\lambda \in \mathbb{N}$, $N \leq 2^\lambda$, all messages $m \in \mathcal{M}_\lambda$,

$$\Pr\left[\mathsf{PartialVerify}(\mathsf{crs}, m, \sigma) = 1 : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^N) \\ (\mathsf{vk}, \mathsf{ht}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{crs}) \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m) \end{array}\right] = 1.$$

- **Aggregation correctness:** For all $\lambda \in \mathbb{N}$, $N \leq 2^\lambda$, all messages $m \in \mathcal{M}_\lambda$, all $L \leq N$, all crs in the support of $\mathsf{Setup}(1^\lambda, 1^N)$, all $\{(\mathsf{vk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell)\}_{\ell \in [L]}$ where $(\mathsf{vk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell)$ is in the support of $\mathsf{KeyGen}(\mathsf{crs})$ for all $\ell \in L$, all sets of signatures $\{\sigma_\ell\}_{\ell \in S}$ where $\mathsf{PartialVerify}(\mathsf{crs}, m, \sigma_\ell) = 1$ for all $\ell \in S$, and all thresholds $T \leq |S|$,

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}_{\mathsf{agg}}, m, \sigma_{\mathsf{agg}}, T) = 1 : \begin{array}{c} (\mathsf{vk}_{\mathsf{agg}}, \mathsf{ak}) = \mathsf{Preprocess}(\mathsf{crs}, \{(\mathsf{vk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [L]}) \\ \sigma_{\mathsf{agg}} \leftarrow \mathsf{Aggregate}(\mathsf{ak}, \{\sigma_\ell\}_{\ell \in S}) \end{array}\right] = 1.$$

- **Static unforgeability:** For a security parameter λ and an adversary $\mathcal{A}$, we define the static unforgeability game as follows:

  1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the bound 1^N on the size of the quorum (where $N \leq 2^\lambda$), the challenge quorum size $L \leq N$, the indices of the corrupted users $C \subseteq [L]$, and a challenge message $m^* \in \mathbb{Z}_p$.
  2. The challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$. Then, for each $\ell \in [N] \setminus C$, the challenger samples a key $(\mathsf{vk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell) \leftarrow \mathsf{KeyGen}(\mathsf{crs})$. It gives crs and $\{(\mathsf{vk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [N] \setminus C}$ to $\mathcal{A}$.
  3. Algorithm $\mathcal{A}$ can now make signing queries by specifying an index $\ell \in [N] \setminus C$ and a message $m \neq m^*$. The challenger replies to each query with $\mathsf{Sign}(\mathsf{sk}_\ell, m)$.
  4. Once $\mathcal{A}$ is finished making signing queries, it specifies the key-generation randomness $\rho_\ell \in \{0,1\}^*$ used to generate the keys for each of the corrupted users $\ell \in C$, the forgery σ_{agg}, and the threshold T.
  5. The challenger first checks that $|C| < T \leq L$ and outputs 0 if not. For each $\ell \in C$, the challenger computes $(\mathsf{vk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell) \leftarrow \mathsf{KeyGen}(\mathsf{crs}; \rho_\ell)$. Next, the challenger computes

  $$(\mathsf{vk}_{\mathsf{agg}}, \mathsf{ak}) = \mathsf{Preprocess}(\mathsf{crs}, \{(\mathsf{vk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [L]}).$$

  The output of the experiment is $b = \mathsf{Verify}(\mathsf{vk}_{\mathsf{agg}}, m^*, \sigma_{\mathsf{agg}}, T)$.

  We say Π_{STS} satisfies static unforgeability in the registered-key model if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ in the static unforgeability game defined above. We say Π_{STS} satisfies static unforgeability in the registered-key model where the bound on the quorum size is $N = N(\lambda)$ if the above holds for all efficient adversaries $\mathcal{A}$ that declares N to be the bound on the quorum size.

- **Succinctness:** There exists a universal polynomial $\mathsf{poly}(\cdot,\cdot)$ such that for all $\lambda \in \mathbb{N}$, all $N \leq 2^\lambda$, all crs in the support of $\mathsf{Setup}(1^\lambda, 1^N)$, all $L \leq N$, all $(\mathsf{vk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell)$ in the support of $\mathsf{KeyGen}(\mathsf{crs})$, and setting $(\mathsf{vk}_{\mathsf{agg}}, \mathsf{ak}) = \mathsf{Preprocess}(\mathsf{crs}, \{(\mathsf{vk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [L]})$, the following hold:
  - The size of the aggregated verification key $\mathsf{vk}_{\mathsf{agg}}$ is $\mathsf{poly}(\lambda, \log N)$.
  - The size of the aggregate signatures output by $\mathsf{Aggregate}(\mathsf{ak}, \cdot)$ is $\mathsf{poly}(\lambda, \log N)$.

  Taken together, these two properties imply that the running time of $\mathsf{Verify}(\mathsf{vk}_{\mathsf{agg}}, \cdot, \cdot)$ is also $\mathsf{poly}(\lambda, \log N)$.

Construction 5.2 (Threshold Signatures with Silent Setup)**.** Let λ be a security parameter and GroupGen be a prime-order bilinear group generator. Let $p = p(\lambda)$ be the order of the group output by GroupGen. For ease of exposition, we make the following two simplifying assumptions in the following description. Note that both assumptions are without loss of generality:

- We assume that $p(\lambda) > 2^{\lambda+1}$. We can satisfy this requirement by taking any GroupGen′ algorithm that satisfies Definition 2.4 and defining $\mathsf{GroupGen}(1^\lambda)$ to output $\mathsf{GroupGen}'(1^{\lambda+1})$. In particular, $\mathsf{GroupGen}'(1^{\lambda+1})$ always outputs a group of prime order p where $p > 2^{\lambda+1}$.
- We assume that the bound $N = 2^n$ on the quorum size is always a power-of-two. If N is not a power-of-two, we can always pad the input to the next power-of-two with at most a $2\times$ overhead. (As we discuss in Remark 5.8, we can handle non-power-of-two N with no additional overhead, but this requires extra notation and special casing.)

We construct a threshold signature scheme with silent setup $\Pi_{\mathsf{STS}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PartialVerify}, \mathsf{Preprocess}, \mathsf{Aggregate}, \mathsf{Verify})$ over message space $\mathcal{M} = \{\mathbb{Z}_{p(\lambda)}\}_{\lambda\in\mathbb{N}}$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^N)$: On input the security parameter λ and a bound on the quorum size $N = 2^n \le 2^\lambda$, the setup algorithm proceeds as follows:
  - Sample $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$ and $g, \hat{g} \xleftarrow{R} \mathbb{G} \setminus \{1\}$, $\hat{u}, \hat{h} \xleftarrow{R} \mathbb{G}$, $c \xleftarrow{R} \mathbb{Z}_p^*$, and $\mathbf{s} \xleftarrow{R} \mathbb{Z}_p^N$. Let $B = e(g, \hat{g})^{s_1}$.
  - Let $\mathbf{M} \in \mathbb{Z}_p^{(2N-1)\times N}$ be the share-generating matrix for an N-out-of-$(2N-1)$ threshold policy from Eq. (2.1). For $\ell \in [2N-1]$, let $\mathbf{m}_\ell^\intercal$ denote the ℓ^{th} row of $\mathbf{M}$. Then compute
$$\hat{z}_0 = \prod_{\ell\in[2N-1]} \hat{g}^{c^\ell \mathbf{m}_\ell^\intercal \mathbf{s}}$$
$$\forall \ell \in [2N-1] : \hat{v}_{\ell,0} = \prod_{\substack{i\in[2N-1]\\ i\neq\ell}} \hat{g}^{c^{i-\ell}\mathbf{m}_i^\intercal \mathbf{s}}.$$
  - For each $\ell \in [N+1, 2N-1]$, sample $\gamma_\ell \xleftarrow{R} \mathbb{Z}_p$. For each $j \in [n]$, define the set $X_j = [N + 2^{j-1}, N + 2^j - 1]$. Observe that $\bigcup_{j\in[n]} X_j = [N+1, 2N-1]$. Then, for each $j \in [n]$ and $i \in I \setminus X_j$, define
$$\hat{y}_j = \prod_{k\in X_j} \hat{g}^{c^k \gamma_k} \quad \text{and} \quad \hat{\tau}_{j,i} = \prod_{k\in X_j} \hat{g}^{c^{k-i}\gamma_k}.$$

  For ease of notation, we define the interval $I = [-2N+1, 2N-1]$. Finally, it outputs the common reference string
$$\mathsf{crs} = \left(\mathcal{G}, g, \hat{g}, B, \hat{u}, \hat{h}, \{\hat{g}^{c^i}\}_{i\in I\setminus\{0\}}, \hat{z}_0, \{\hat{v}_{\ell,0}\}_{\ell\in[2N-1]}, \{\hat{y}_j\}_{j\in[n]}, \{\hat{\tau}_{j,i}\}_{j\in[n], i\in I\setminus X_j}\right). \tag{5.1}$$
- $\mathsf{KeyGen}(\mathsf{crs})$: On input the common reference string crs (with components parsed according to Eq. (5.1)), the key-generation algorithm samples $\alpha \xleftarrow{R} \mathbb{Z}_p$ and computes $\hat{v}_i' = (\hat{g}^{c^i})^\alpha$ for all $i \in I \setminus \{0\}$. The algorithm sets $A = e(g, \hat{g})^\alpha$ and
$$\mathsf{vk} = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, A) \quad \text{and} \quad \mathsf{ht} = \{\hat{v}_i'\}_{i\in I\setminus\{0\}}$$
  The algorithm outputs the verification key vk, the aggregation hint ht and the signing key $\mathsf{sk} = (\mathsf{vk}, \alpha)$.
- $\mathsf{Sign}(\mathsf{sk}, m)$: On input the signing key $\mathsf{sk} = (\mathsf{vk}, \alpha)$ where $\mathsf{vk} = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, A)$ and a message $m \in \mathbb{Z}_p$, the signing algorithm samples $r \xleftarrow{R} \mathbb{Z}_p$ and outputs the signature $\sigma = (g^r, \hat{g}^\alpha(\hat{u}^m\hat{h})^r)$.
- $\mathsf{PartialVerify}(\mathsf{vk}, m, \sigma)$: On input a verification key $\mathsf{vk} = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, A)$, a message $m \in \mathbb{Z}_p$, and a signature $\sigma = (\sigma_1, \hat{\sigma}_2)$, the partial verification algorithm outputs 1 if
$$A \cdot e(\sigma_1, \hat{u}^m\hat{h}) = e(g, \hat{\sigma}_2).$$
- $\mathsf{Preprocess}(\mathsf{crs}, \{(\mathsf{vk}_\ell, \mathsf{ht}_\ell)\}_{\ell\in[L]})$: On input the common reference string crs (with components parsed according to Eq. (5.1)) and a collection of verification keys vk_ℓ and hints $\mathsf{ht}_\ell = \{\hat{v}_{\ell,i}'\}_{i\in I\setminus\{0\}}$ for $\ell \in [L]$, where $L \le N$.

Then it computes

$$\hat{z} = \hat{z}_0 \cdot \prod_{\ell \in [L]} \hat{v}'_{\ell,\ell}$$
$$\forall \ell \in [2N-1] : \hat{v}_\ell = \hat{v}_{\ell,0} \cdot \prod_{\substack{i \in [L] \\ i \neq \ell}} \hat{v}'_{i,i-\ell}. \tag{5.2}$$

Finally, the preprocessing algorithm outputs

$$\mathsf{vk}_{\mathsf{agg}} = (L, \mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, B, \hat{z}, \{\hat{y}_j\}_{j \in [n]})$$
$$\mathsf{ak} = \left(L, \{g^{c^{-\ell}}, \hat{v}_\ell\}_{\ell \in [L]}, \{\hat{\tau}_{j,i}\}_{j \in [n], i \in I \setminus X_j}\right).$$

The preprocessing algorithm outputs the verification key $\mathsf{vk}_{\mathsf{agg}}$ and the aggregation key ak.

- Aggregate$(\mathsf{crs}, \mathsf{ak}, \{\sigma_\ell\}_{\ell \in S})$: On input the aggregation key $\mathsf{ak} = \left(L, \{g^{c^{-\ell}}, \hat{v}_\ell\}_{\ell \in [L]}, \{\hat{\tau}_{j,i}\}_{j \in [n], i \in I \setminus X_j}\right)$, and a collection of signatures $\sigma_\ell = (\sigma_{\ell,1}, \hat{\sigma}_{\ell,2})$ for $\ell \in S$, the aggregation algorithm proceeds as follows:
  - Let $T = L - |S|$. Write $T = \sum_{j \in [n]} b_j \cdot 2^{j-1}$ (i.e., $b_n \cdots b_1$ is the binary representation of T).
  - Let $\mathbf{M} \in \mathbb{Z}_p^{2N \times N}$ be the share-generating matrix for an N-out-of-$(2N-1)$ threshold policy from Eq. (2.1).[11]
  - Let $S_{\mathsf{pad}} = S \cup [L+1, N] \cup \bigcup_{j : b_j = 1} X_j$, where $X_j = [N + 2^{j-1}, N + 2^j - 1]$. By construction, $|S_{\mathsf{pad}}| = N$. Let $\boldsymbol{\omega} \in \mathbb{Z}_p^{2N-1}$ be a reconstruction vector where $\boldsymbol{\omega}^\intercal \mathbf{M} = \mathbf{e}_1^\intercal$, and moreover, $\omega_\ell = 0$ for all $\ell \notin S_{\mathsf{pad}}$.
  - Compute

$$\sigma_{\mathsf{agg},1} = \prod_{\ell \in S} \sigma_{\ell,1}^{\omega_\ell}$$
$$\hat{\sigma}_{\mathsf{agg},2} = \prod_{\ell \in S} \hat{\sigma}_{\ell,2}^{\omega_\ell} \cdot \prod_{\ell \in S_{\mathsf{pad}}} \hat{v}_\ell^{\omega_\ell} \cdot \prod_{\substack{j \in [n] \\ b_j = 0}} \prod_{\ell \in S_{\mathsf{pad}}} \hat{\tau}_{j,\ell}^{\omega_\ell}$$
$$\sigma_{\mathsf{agg},3} = \prod_{\ell \in S_{\mathsf{pad}}} \left(g^{c^{-\ell}}\right)^{\omega_\ell}.$$

  - Output the signature $\sigma_{\mathsf{agg}} = (\sigma_{\mathsf{agg},1}, \hat{\sigma}_{\mathsf{agg},2}, \sigma_{\mathsf{agg},3}, |S|)$.
- Verify$(\mathsf{vk}_{\mathsf{agg}}, m, \sigma_{\mathsf{agg}}, T)$: On input the verification key

$$\mathsf{vk}_{\mathsf{agg}} = (L, \mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, B, \hat{z}, \{\hat{y}_j\}_{j \in [n]}),$$

a message $m \in \mathbb{Z}_p$, a signature $\sigma_{\mathsf{agg}} = (\sigma_{\mathsf{agg},1}, \hat{\sigma}_{\mathsf{agg},2}, \sigma_{\mathsf{agg},3}, K)$ and a threshold $T \in [L]$, the verification algorithm first checks that $K \geq T$. If so, it computes $b_1, \ldots, b_n \in \{0,1\}$ such that $\sum_{j \in [n]} b_j 2^{j-1} = L - K$. If so, it checks

$$B \cdot e(g, \hat{\sigma}_{\mathsf{agg},2}) = e(\sigma_{\mathsf{agg},1}, \hat{u}^m \hat{h}) \cdot e(\sigma_{\mathsf{agg},3}, \hat{z}) \cdot \prod_{\substack{j \in [n] \\ b_j = 0}} e(\sigma_{\mathsf{agg},3}, \hat{y}_j). \tag{5.3}$$

Otherwise, the verification algorithm outputs 0.

Correctness and security analysis. The correctness and security analysis for Construction 5.2 follow a very similar structure as that for Construction 3.15. We state the relevant theorems below, and defer their formal proofs to Appendix D.

Theorem 5.3 (Signing Correctness). *Construction 5.2 satisfies signing correctness.*

[11] For this to be well-defined, we require that $p > 2N - 1$, which holds since $N \leq 2^\lambda < p/2$.

Theorem 5.4 (Aggregation Correctness)**.** *Construction 5.2 satisfies aggregation correctness.*

Theorem 5.5 (Static Unforgeability)**.** *Take any polynomial $N = N(\lambda)$ and suppose the $(2N-1)$-extended bilinear Diffie-Hellman exponent assumption (Assumption 3.12) holds with respect to* GroupGen*. Then, Construction 5.2 satisfies static unforgeability in the registered-key model where the bound on the quorum size is N.*

Corollary 5.6 (Threshold Signature with Silent Setup)**.** *Let λ be a security parameter. Suppose the $(2N-1)$-extended bilinear Diffie-Hellman exponent assumption (Assumption 3.12) holds with respect to* GroupGen *for all polynomials $N = N(\lambda)$. Then for every polynomial $N = N(\lambda)$, there exists a threshold signature scheme with silent setup that supports a quorum size of up to N users with the following efficiency properties:*

- **CRS size:** *The CRS contains $O(N \log N)$ group elements.*
- **User key size:** *A user's verification key* vk *contains* 5 *group elements and the aggregation hint* ht *contains $4N-2$ group elements. Note that* 4 *of the group elements in each user's verification key are fixed (elements from the CRS); only the remaining group element is user-specific. The preprocessing algorithm only needs the single user-specific element (whereas verifying partial signatures requires the* 4 *fixed elements from the CRS and the user-specific element).*
- **Partial signature size:** *A user's partial signature σ contains* 2 *group elements.*
- **Aggregation key size:** *The aggregation key* ak *contains $O(N \log N)$ group elements.*
- **Aggregate verification key size:** *The aggregate verification key* $\mathsf{vk}_{\mathsf{agg}}$ *contains $6 + \lceil \log N \rceil$ group elements (and $\lceil \log N \rceil$ additional bits to encode the quorum size).*
- **Aggregate signature size:** *An aggregate signature contains* 3 *group elements and* $\log N$ *extra bits (to represent the threshold). The aggregate verification algorithm requires performing* 4 *pairings (and $O(\log N)$ group operations).*

Remark 5.7 (Instantiation over Asymmetric Groups)**.** It is straightforward to adapt Construction 5.2 (as well as Constructions 3.4 and 3.15) to work over asymmetric pairing groups (i.e., where the pairing $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is defined over two base groups $\mathbb{G}_1$ and $\mathbb{G}_2$). The approach is to assign each element of $\mathbb{G}$ in the construction to either $\mathbb{G}_1$ or $\mathbb{G}_2$, subject to the restriction that the pairing is only defined on $\mathbb{G}_1 \times \mathbb{G}_2$. In fact, Construction 5.2 already decomposes naturally along these lines, where we can assign all the variables without hats (e.g., g) to $\mathbb{G}_1$ and the variables with hats (e.g., $\hat{g}, \hat{h}, \hat{u}$) to $\mathbb{G}_2$. With this instantiation, an aggregate signature in Construction 5.2 consists of 2 $\mathbb{G}_1$ elements and 1 $\mathbb{G}_2$ element. Over a standard pairing curve at the 128-bit security level (e.g., BLS-381 [BLS02]), this yields a scheme where the aggregate signatures are 192 bytes (48 bytes per $\mathbb{G}_1$ element and 96 bytes per $\mathbb{G}_2$ element). This is 2.8–3× shorter than the scheme of [DCX+23] (536 bytes) and of [GJM+24] (592 bytes).

Remark 5.8 (Non-Power-of-Two N)**.** For ease of notation, Construction 5.2 assumes that the bound on the number of users N is a power of 2. While we can always pad N to the next power of two with at most a 2× overhead in the CRS and aggregation hint sizes, we can also avoid padding altogether. To see this, observe that correctness and security of Construction 5.2 relies on the following two properties of the sets $X_1, \ldots, X_n$:

- The sets X_i are pairwise disjoint and $\bigcup_{i \in [n]} X_i = [N+1, 2N-1]$.
- For all $K \in [N-1]$, there is a collection of bits $b_1, \ldots, b_n$ such that $|\bigcup_{i:b_i=1} X_i| = K$. The value K essentially corresponds to the number of "free" signatures the aggregator is using.

When N is a power of 2, and the sets X_i are defined as in Construction 5.2, then the bits $b_1, \ldots, b_n$ precisely correspond to the binary decomposition of K. Constructing a set system that satisfies this property is straightforward even when N is not a power of 2. In this case, let $n = \lceil \log N \rceil$. Define $X_1, \ldots, X_{n-1}$ exactly as in Construction 5.2 and let $X_n = [N + 2^{n-1}, 2N-1]$. By construction, the first property is satisfied. For the second property, take any $K \in [N-1]$, and let $b_n \cdots b_1$ be the binary representation of K.

- If $b_n = 0$, then $|\bigcup_{i:b_i=1} X_i| = K$.
- If $b_n = 1$, then define $b'_n = 1$ and let $b'_{n-1} \cdots b'_1$ be the binary representation of $K - |X_n|$. Then, by construction $|\bigcup_{i:b'_i=1} X_i| = K$.

By defining the sets X_i in this way, and deriving the bits $b_1, \ldots, b_n$ associated with a target value K in the above manner, we can adapt Construction 5.2 to support arbitrary N without any additional overhead.

Remark 5.9 (Dynamic Policies, More Generally)**.** The approach in Construction 5.2 for supporting dynamic policies is to add dummy users in the policy, and allow the aggregator (and correspondingly, the verifier) the ability to decide whether a group of dummy users are automatically included in the signing quorum (i.e., by zeroing our their signing keys in the verification key) or automatically excluded (i.e., by including their signing key as part of the verification key). This approach can also be applied to Constructions 3.4 and 3.15. Namely, when generating the verification key for a share-generating matrix $\mathbf{M}$, the preprocessing algorithm can always designate different (non-overlapping) sets of rows of $\mathbf{M}$ as rows associated with dummy users (these correspond to the sets $X_1, \ldots, X_n$ in Construction 5.2). The aggregator and the verifier in turn has the flexibility of deciding whether each preset group of dummy users are automatically included or excluded in the signing quorum during signature aggregation and verification, respectively. For each set of rows, the preprocessing algorithm would need to pre-compute the aggregated public key for the block (i.e., the analog of the $\hat{y}_j$ in Construction 5.2) together with the cross terms (i.e., the analog of the $\hat{\tau}_{j,i}$ in Construction 5.2) and include those as part of the verification key and aggregation key, respectively.

5.2 Threshold Encryption with Silent Setup

Just like our distributed monotone-policy signature scheme (Construction 3.15) immediately yields a distributed monotone-policy encryption scheme, our threshold signature scheme with silent setup (Construction 5.2) immediately implies the analogous notion of threshold encryption with silent setup [GKPW24]. The transformation preserves the functionality and efficiency properties of the underlying signature scheme (i.e., encrypting to dynamic thresholds and having a quasi-linear-size CRS). We give the formal definition of a threshold encryption scheme with silent setup and give our construction and security analysis below. As in Section 5.1, we give our definition in the registered-key model (where correctness and security hold against semi-malicious adversaries) and consider a static notion of security. As in Section 4, we consider a tag-based CCA-security notion which easily implies CPA-security (by setting the tag to a fixed string) and CCA-security (by setting the tag to be the verification key for a one-time signature scheme).

Definition 5.10 (Threshold Encryption with Silent Setup [GKPW24, adapted])**.** Let λ be a security parameter. A (tag-based) threshold encryption scheme with silent setup consists of a tuple of efficient algorithms $\Pi_{\mathsf{STE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Preprocess}, \mathsf{Encrypt}, \mathsf{PartialDec}, \mathsf{PartialVerify}, \mathsf{Decrypt})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^N) \to \mathsf{crs}$: On input the security parameter λ and a bound on the maximum number of users in a quorum, the setup algorithm outputs the common reference string crs. We assume crs implicitly includes a description of the message space $\mathcal{M}$ and the tag space $\mathcal{T}$ for the encryption scheme.
- $\mathsf{KeyGen}(\mathsf{crs}) \to (\mathsf{pk}, \mathsf{ht}, \mathsf{sk})$: On input the common reference string crs, the key-generation algorithm outputs a public key pk, an aggregation hint ht, and a secret key sk.
- $\mathsf{Preprocess}(\mathsf{crs}, \{(\mathsf{pk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [L]}) \to (\mathsf{ek}, \mathsf{ak})$: On input the common reference string crs, a collection of L public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_L$ together with their aggregation hints $\mathsf{ht}_1, \ldots, \mathsf{ht}_L$, the preprocessing algorithm outputs an encryption key ek and an aggregation key ak. This algorithm is deterministic. We assume ek also includes a description of the message space $\mathcal{M}$ and tag space $\mathcal{T}$ (from crs).
- $\mathsf{Encrypt}(\mathsf{ek}, \tau, m, T) \to \mathsf{ct}$: On input an encryption key ek, a tag $\tau \in \mathcal{T}$, a message $m \in \mathcal{M}$, and a threshold T, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{PartialDec}(\mathsf{sk}, \tau, \mathsf{ct}) \to \sigma$: On input a secret key sk, a tag τ, and a ciphertext ct, the partial decryption algorithm outputs a partial decryption σ.
- $\mathsf{PartialVerify}(\mathsf{pk}, \tau, \mathsf{ct}, \sigma) \to b$: On input a public key pk, a tag τ, a ciphertext ct, and a partial decryption σ, the partial-verification algorithm outputs a bit $b \in \{0, 1\}$. This algorithm is deterministic.
- $\mathsf{Decrypt}(\mathsf{ak}, \mathsf{ct}, \{\sigma_\ell\}_{\ell \in S}) \to m$: On input the aggregation key ak, the ciphertext ct, and a set of partial decryptions σ_ℓ, the decryption algorithm outputs a message $m \in \mathcal{M}$ (or a special symbol $\perp$ to indicate decryption failure). This algorithm is deterministic.

We require Π_{STE} satisfy the following properties:

- **Partial decryption correctness:** For all $\lambda \in \mathbb{N}$, all $N \leq 2^\lambda$, all crs in the support of $\mathsf{Setup}(1^\lambda, 1^N)$, all tags $\tau \in \mathcal{T}$, all messages $m \in \mathcal{M}$ (where $\mathcal{T}, \mathcal{M}$ denotes the tag space and message space defined by crs), all quorum sizes $L \leq N$, any threshold $T \in [L]$, all $\{(\mathsf{pk}_\ell, \mathsf{sk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [L]}$ where $(\mathsf{pk}_\ell, \mathsf{sk}_\ell, \mathsf{ht}_\ell)$ is in the support of $\mathsf{KeyGen}(\mathsf{crs})$, and all indices $i^* \in [L]$, it holds that

$$\Pr\left[\mathsf{PartialVerify}(\mathsf{pk}_{i^*}, \tau, \mathsf{ct}, \sigma_{i^*}) = 1 : \begin{array}{c}(\mathsf{ek}, \mathsf{ak}) = \mathsf{Preprocess}(\mathsf{crs}, \{(\mathsf{pk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [L]}) \\ \mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{ek}, \tau, m, T) \\ \sigma_{i^*} \leftarrow \mathsf{PartialDec}(\mathsf{sk}_{i^*}, \tau, \mathsf{ct})\end{array}\right] = 1.$$

- **Aggregation correctness:** For all $\lambda \in \mathbb{N}$, all $N \leq 2^\lambda$, all crs in the support of $\mathsf{Setup}(1^\lambda, 1^N)$, all tags $\tau \in \mathcal{T}$, all messages $m \in \mathcal{M}$ (where $\mathcal{T}, \mathcal{M}$ denotes the tag space and message space defined by crs), all quorum sizes $L \leq N$, all $\{(\mathsf{pk}_\ell, \mathsf{sk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [L]}$ where $(\mathsf{pk}_\ell, \mathsf{sk}_\ell, \mathsf{ht}_\ell)$ is in the support of $\mathsf{KeyGen}(\mathsf{crs})$, all subsets $S \subseteq [L]$, and all thresholds $1 \leq T \leq |S|$, and setting $(\mathsf{ek}, \mathsf{ak}) = \mathsf{Preprocess}(\mathsf{crs}, \{(\mathsf{pk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [L]})$, and for all ciphertexts ct in the support of $\mathsf{Encrypt}(\mathsf{ek}, \tau, m, T)$, all partial decryptions $\{\sigma_\ell\}_{\ell \in S}$ where for all $\ell \in S$, $\mathsf{PartialVerify}(\mathsf{pk}_\ell, \tau, \mathsf{ct}, \sigma_\ell) = 1$, we have

$$\mathsf{Decrypt}(\mathsf{crs}, \mathsf{ak}, \mathsf{ct}, \{\sigma_\ell\}_{\ell \in S}) = m.$$

- **Static tag-based CCA-security:** For a security parameter λ and an adversary $\mathcal{A}$, we define the static tag-based CCA-security game as follows:

  1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs a bound 1^N on the size of the quorum (where $N \leq 2^\lambda$), the challenge quorum size $L \leq N$, the indices of the corrupted users $C \subseteq [N]$, and a challenge message $m^* \in \mathbb{Z}_p$.
  2. The challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$. Then, for each index $\ell \in [L] \setminus C$, the challenger samples a key $(\mathsf{pk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell) \leftarrow \mathsf{KeyGen}(\mathsf{crs})$. It gives crs together with $\{(\mathsf{pk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [L] \setminus C}$ to $\mathcal{A}$.
  3. Algorithm $\mathcal{A}$ now specifies the key-generation randomness $\rho_\ell \in \{0,1\}^*$ used to generate the keys for each of the corrupted users $\ell \in C$.
  4. For each $\ell \in C$, the challenger computes $(\mathsf{pk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell) \leftarrow \mathsf{KeyGen}(\mathsf{crs}; \rho_\ell)$. The challenger computes and gives $(\mathsf{ek}, \mathsf{ak}) = \mathsf{Preprocess}(\mathsf{crs}, \{(\mathsf{ek}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [L]})$ to the adversary $\mathcal{A}$.
  5. The adversary can now issue partial decryption queries by specifying an index $\ell \in [L] \setminus C$, a tag $\tau \in \mathcal{T}$, and a ciphertext ct. The challenger responds with $\mathsf{PartialDec}(\mathsf{sk}_\ell, \tau, \mathsf{ct})$.
  6. After $\mathcal{A}$ is finished making partial decryption queries, it outputs a pair of messages $m_0, m_1 \in \mathcal{M}$ along with a threshold T.
  7. The challenger first checks that $|C| < T \leq L$. If not, the challenger outputs 0. Otherwise, it samples a random tag $\tau^* \xleftarrow{R} \mathcal{T}$ and responds with the challenge tag τ^* together with the challenge ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Encrypt}(\mathsf{ek}, \tau^*, m_b, T)$.
  8. Algorithm $\mathcal{A}$ can continue making partial decryption queries, except it is *not* allowed to query for a decryption with tag $\tau = \tau^*$. The challenger responds to each query as before. At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that Π_{STE} satisfies static tag-based CCA-security if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda)$$

  in the tag-based CCA-security game defined above. We say Π_{STE} satisfies static unforgeability in the registered-key model where the bound on the quorum size is $N = N(\lambda)$ if the above holds for all efficient adversaries $\mathcal{A}$ that declares N to be the bound on the quorum size.

- **Succinctness:** There exists a universal polynomial $\mathsf{poly}(\cdot,\cdot)$ such that for all $\lambda \in \mathbb{N}$, all $N \le 2^\lambda$, all crs in the support of $\mathsf{Setup}(1^\lambda, 1^N)$, all $L \le N$, all $(\mathsf{pk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell)$ in the support of $\mathsf{KeyGen}(\mathsf{crs})$, and setting $(\mathsf{ek}, \mathsf{ak}) = \mathsf{Preprocess}(\mathsf{crs}, \{(\mathsf{pk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [N]})$, the following hold:
  - The size of the encryption key ek and the size of the ciphertext output by $\mathsf{Encrypt}(\mathsf{ek}, \cdot, \cdot, \cdot)$ is $\mathsf{poly}(\lambda, \log N)$.
  - The size of the partial decryptions output by $\mathsf{PartialDec}(\mathsf{sk}, \cdot, \cdot)$ is $\mathsf{poly}(\lambda, \log N)$.

Construction 5.11 (Threshold Encryption with Silent Setup)**.** Let λ be a security parameter and GroupGen be a prime-order bilinear group generator. Let $p = p(\lambda)$ be the order of the group output by GroupGen. As in Construction 5.2, we make the following two simplifying assumptions:

- We assume that $p(\lambda) > 2^{\lambda+1}$.
- We assume that $N = 2^n$ is always a power of two.

As argued in Construction 5.2, both assumptions are without loss of generality. We construct a (tag-based) threshold encryption scheme $\Pi_{\mathsf{STE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Preprocess}, \mathsf{Encrypt}, \mathsf{PartialDec}, \mathsf{PartialVerify}, \mathsf{Decrypt})$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^N)$: On input the security parameter λ and the bound on the quorum size $N = 2^n \le 2^\lambda$, the setup algorithm proceeds exactly as in Construction 5.2:
  - Sample $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$ and $g, \hat{g} \xleftarrow{R} \mathbb{G} \setminus \{1\}$, $\hat{u}, \hat{h} \xleftarrow{R} \mathbb{G}$, $c \xleftarrow{R} \mathbb{Z}_p^*$, and $\mathbf{s} \xleftarrow{R} \mathbb{Z}_p^N$. Let $B = e(g, \hat{g})^{s_1}$.
  - Let $\mathbf{M} \in \mathbb{Z}_p^{(2N-1)\times N}$ be the share-generating matrix for an N-out-of-$(2N-1)$ threshold policy from Eq. (2.1). For $\ell \in [2N-1]$, let $\mathbf{m}_\ell^\intercal$ denote the ℓ^{th} row of $\mathbf{M}$. Then compute
$$\hat{z}_0 = \prod_{\ell \in [2N-1]} \hat{g}^{c^\ell \mathbf{m}_\ell^\intercal \mathbf{s}}$$
$$\forall \ell \in [2N-1] : \hat{v}_{\ell,0} = \prod_{\substack{i \in [2N-1] \\ i \neq \ell}} \hat{g}^{c^{i-\ell} \mathbf{m}_i^\intercal \mathbf{s}}.$$
  - For each $\ell \in [N+1, 2N-1]$, sample $\gamma_\ell \xleftarrow{R} \mathbb{Z}_p$. For each $j \in [n]$, define the set $X_j = [N + 2^{j-1}, N + 2^j - 1]$. Then, for each $j \in [n]$ and $i \in [-2N+1, 2N-1] \setminus X_j$, define
$$\hat{y}_j = \prod_{k \in X_j} \hat{g}^{c^k \gamma_k} \quad \text{and} \quad \hat{\tau}_{j,i} = \prod_{k \in X_j} \hat{g}^{c^{k-i} \gamma_k}.$$

  For ease of notation, we define the interval $I = [-2N+1, 2N-1]$. Finally, it outputs the common reference string
$$\mathsf{crs} = \left(\mathcal{G}, g, \hat{g}, B, \hat{u}, \hat{h}, \{\hat{g}^{c^i}\}_{i \in I \setminus \{0\}}, \hat{z}_0, \{\hat{v}_{\ell,0}\}_{\ell \in [2N-1]}, \{\hat{y}_j\}_{j \in [n]}, \{\hat{\tau}_{j,i}\}_{j \in [n], i \in I \setminus X_j}\right). \tag{5.4}$$
  The message space for the encryption scheme is $\mathcal{M} = \mathbb{G}_T$ and the tag space is $\mathcal{T} = \mathbb{Z}_p$.
- $\mathsf{KeyGen}(\mathsf{crs})$: On input the common reference string crs (with components parsed according to Eq. (5.4)), the key-generation algorithm proceeds as in Construction 5.2. Namely, it samples $\alpha \xleftarrow{R} \mathbb{Z}_p$ and computes $\hat{v}_i' = (\hat{g}^{c^i})^\alpha$ for all $i \in I \setminus \{0\}$. The algorithm sets $A = e(g, \hat{g})^\alpha$ and
$$\mathsf{pk} = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, A) \quad \text{and} \quad \mathsf{ht} = \{\hat{v}_i'\}_{i \in I \setminus \{0\}}.$$
  The algorithm outputs the public key pk, the aggregation hint ht, and the secret key $\mathsf{sk} = (\mathsf{pk}, \alpha)$.

- Preprocess$(\mathsf{crs}, \{(\mathsf{pk}_\ell, \mathsf{ht}_\ell)\}_{\ell\in[L]})$: On input the common reference string crs (with components parsed according to Eq. (5.4)) and a collection of public keys pk_ℓ and hints $\mathsf{ht}_\ell = \{v'_{\ell,i}\}_{i\in I\setminus\{0\}}$, the preprocessing algorithm proceeds as in Construction 5.2. Namely, it computes

$$\begin{aligned} \hat{z} &= \hat{z}_0 \cdot \prod_{\ell\in[L]} \hat{v}'_{\ell,\ell} \\ \forall \ell \in [2N-1] : \hat{v}_\ell &= \hat{v}_{\ell,0} \cdot \prod_{\substack{i\in[L]\\ i\neq\ell}} \hat{v}'_{i,i-\ell}. \end{aligned} \tag{5.5}$$

  Then, it sets

$$\begin{aligned} \mathsf{ek} &= (L, \mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, B, \hat{z}, \{\hat{y}_j\}_{j\in[n]}) \\ \mathsf{ak} &= \left(L, \{g^{c^{-\ell}}, \hat{v}_\ell\}_{\ell\in[L]}, \{\hat{\tau}_{j,i}\}_{j\in[n], i\in I\setminus X_j}\right). \end{aligned}$$

  The preprocessing algorithm outputs the encryption key ek and the aggregation key ak.

- Encrypt$(\mathsf{ek}, \tau, m, T)$: On input an encryption key $\mathsf{ek} = (L, \mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, B, \hat{z}, \{\hat{y}_j\}_{j\in[n]})$, a tag $\tau \in \mathbb{Z}_p$, a message $m \in \mathbb{G}_T$, and a threshold $T \in [L]$ the encryption algorithm proceeds as follows:
  - Compute the binary decomposition $b_n \cdots b_1 \in \{0,1\}^n$ of $L - T$: namely, $L - T = \sum_{j\in[n]} b_j 2^{j-1}$.
  - Sample a random exponent $t \xleftarrow{R} \mathbb{Z}_p$.

  Output the ciphertext

$$\mathsf{ct} = (\tau, T, C_1, c_2, \hat{c}_3, \hat{c}_4) = \left(\tau\,, T\,,\ B^t \cdot m\,,\ g^t\,,\ (\hat{u}^\tau \hat{h})^t\,,\ \hat{z}^t \textstyle\prod_{j\in[n]:b_j=0} \hat{y}_j^t\right).$$

- PartialDec$(\mathsf{sk}, \tau, \mathsf{ct})$: On input a secret key $\mathsf{sk} = (\mathsf{pk}, \alpha)$ where $\mathsf{pk} = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, A)$, a tag $\tau \in \mathbb{Z}_p$, and a ciphertext ct, the partial decryption algorithm samples $r \xleftarrow{R} \mathbb{Z}_p$ and outputs $\sigma = (g^r,\ \hat{g}^\alpha(\hat{u}^\tau\hat{h})^r)$.

- PartialVerify$(\mathsf{pk}, \tau, \mathsf{ct}, \sigma)$: On input a key $\mathsf{pk} = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, A)$, a tag $\tau \in \mathbb{Z}_p$, a ciphertext $\mathsf{ct} = (\tau', T, C_1, c_2, \hat{c}_3, \hat{c}_4)$, and a partial decryption $\sigma = (\sigma_1, \hat{\sigma}_2)$, the partial verification algorithm outputs 0 if $\tau \neq \tau'$. Otherwise, it outputs 1 if $A \cdot e(\sigma_1, \hat{u}^\tau\hat{h}) = e(g, \hat{\sigma}_2)$.

- Decrypt$(\mathsf{ak}, \tau, \mathsf{ct}, \{\sigma_\ell\}_{\ell\in S})$: On input the aggregation key $\mathsf{ak} = \left(L, \{g^{c^{-\ell}}, \hat{v}_\ell\}_{\ell\in[L]}, \{\hat{\tau}_{j,i}\}_{j\in[n], i\in I\setminus X_j}\right)$, a tag $\tau \in \mathbb{Z}_p$, a ciphertext $\mathsf{ct} = (\tau', T, C_1, c_2, \hat{c}_3, \hat{c}_4)$, and a collection of partial decryptions $\sigma_\ell = (\sigma_{\ell,1}, \hat{\sigma}_{\ell,2})$ for $\ell \in S \subseteq [N]$, the decryption algorithm proceeds as follows:
  - If $|S| \leq T$, output $\perp$. Otherwise, assume that $|S|$ contains *exactly* T elements (if S has more than T partial decryptions, consider the restriction of S to the first T partial decryption in lexicographic order).
  - Write $L - T = \sum_{j\in[n]} b_j \cdot 2^{j-1}$ (i.e., $b_n \cdots b_1$ is the binary representation of $L - T$).
  - Let $\mathbf{M} \in \mathbb{Z}_p^{(2N-1)\times N}$ be the share-generating matrix for the N-out-of-$(2N-1)$ threshold policy from Eq. (2.1).
  - Let $S_{\mathsf{pad}} = S \cup [L+1, N] \cup \bigcup_{j\in[n]:b_j=1} X_j$, where $X_j = [N + 2^{j-1}, N + 2^j - 1]$. By construction,

$$|S_{\mathsf{pad}}| = |S| + (N - L) + \sum_{j\in[n]:b_j=1} 2^{j-1} = T + (N - L) + (L - T) = N.$$

    Let $\boldsymbol{\omega} \in \mathbb{Z}_p^{2N-1}$ be a reconstruction vector where $\boldsymbol{\omega}^\intercal \mathbf{M} = \mathbf{e}_1^\intercal$, and moreover, $\omega_\ell = 0$ for all $\ell \notin S_{\mathsf{pad}}$.

– Compute the aggregated decryption components:

$$\sigma_{\mathsf{agg},1} = \prod_{\ell \in S} \sigma_{\ell,1}^{\omega_\ell}$$
$$\hat{\sigma}_{\mathsf{agg},2} = \prod_{\ell \in S} \hat{\sigma}_{\ell,2}^{\omega_\ell} \cdot \prod_{\ell \in S_{\mathsf{pad}}} \hat{v}_\ell^{\omega_\ell} \cdot \prod_{\substack{j \in [n] \\ b_j = 0}} \prod_{\ell \in S_{\mathsf{pad}}} \hat{\tau}_{j,\ell}^{\omega_\ell}$$
$$\sigma_{\mathsf{agg},3} = \prod_{\ell \in S_{\mathsf{pad}}} \left(g^{c^{-\ell}}\right)^{\omega_\ell}.$$

– Output $C_1 \cdot e(\sigma_{\mathsf{agg},1}, \hat{c}_3)^{-1} \cdot e(c_2, \hat{\sigma}_{\mathsf{agg},2}) \cdot e(\sigma_{\mathsf{agg},3}, \hat{c}_4)^{-1}$.

Correctness and security analysis. The correctness and security analysis for Construction 5.11 follow a very similar structure as that for Construction 5.2 adapted to the setting of encryption. We state the relevant theorems below, and defer their formal proofs to Appendix E.

Theorem 5.12 (Partial Decryption Correctness)**.** *Construction 5.11 satisfies partial decryption correctness.*

Theorem 5.13 (Aggregation Correctness)**.** *Construction 5.11 satisfies aggregation correctness.*

Theorem 5.14 (Static Tag-Based CCA-Security)**.** *Suppose the decisional N-extended bilinear Diffie-Hellman exponent assumption (Assumption 3.12) holds with respect to* GroupGen*. Then, Construction 5.11 satisfies static tag-based CCA-security in the registered-key model where the bound on the quorum size is N.*

Corollary 5.15 (Threshold Encryption with Silent Setup)**.** *Let λ be a security parameter. Suppose the $(2N-1)$-extended bilinear Diffie-Hellman exponent assumption (Assumption 3.12) holds with respect to* GroupGen *for all polynomials $N = N(\lambda)$. Then for every polynomial $N = N(\lambda)$, there exists a threshold signature scheme with silent setup that supports a quorum size of up to N users with the following efficiency properties:*

- **CRS size:** *The CRS contains $O(N \log N)$ group elements.*
- **User key size:** *A user's public key* pk *contains* 5 *group elements and the aggregation hint* ht *contains $4N-2$ group elements. Note that* 4 *of the group elements in each user's public key are fixed (elements from the CRS); only the remaining group element is user-specific. The preprocessing algorithm only needs the single user-specific element (whereas verifying partial decryptions requires the* 4 *fixed elements from the CRS and the user-specific element).*
- **Aggregation key size:** *The aggregation key* ak *contains $O(N \log N)$ group elements.*
- **Aggregate encryption key size:** *The aggregate encryption key* ek *contains $6 + \lceil \log N \rceil$ group elements.*
- **Ciphertext size:** *The ciphertext contains* 4 *group elements and $\log N + \log p$ additional bits (to represent the threshold and the tag).*
- **Partial decryption size:** *A user's partial decryption σ contains* 2 *group elements.*

Extensions. As discussed in Remark 4.2, we can generically lift Construction 5.11 to achieve full CCA-security by setting the tag to be a verification key for a one-time signature scheme and signing the ciphertext (and only decrypt ciphertexts with valid signatures). Similarly, if we only require CPA-security, then we can set the tag to a fixed value, in which case, it no longer needs to be included as part of the ciphertext.

Using the technique from Remark 5.8, we can also modify Construction 5.11 to directly support quorum sizes N that is not a power of 2 without padding. Similarly, the use of dummy users to support dynamic threshold policies can also be applied to the distributed monotone-policy encryption scheme (Construction 4.4) in the same manner as described in Remark 5.9. Namely, when encrypting to a policy $\mathbf{M}$, we can allow the encrypter (and decrypter) to decide whether each preset of dummy users are automatically included or excluded from the decryption quorum.

Remark 5.16 (Distributed Threshold IBE)**.** We note that we can also interpret Construction 5.11 as a *distributed* threshold identity-based encryption (IBE) scheme. Recall that in a standard IBE scheme [Sha84, BF01], one can encrypt a message with respect to an identity id and only a user who holds a secret key $\mathsf{sk}_{\mathsf{id}}$ associated with id can decrypt. In traditional IBE, a trusted key issuer generates the secret identity keys for each user. Similarly, in a threshold IBE scheme [BBH06], a trusted dealer gives out shares of the IBE master secret key to different parties. Construction 5.11 essentially gives a distributed threshold IBE scheme. In this setting, each user can become a key-issuing authority for an IBE scheme by sampling and publishing their public key. Then, one can encrypt a message respect to any collection of IBE authorities, an identity id, and a threshold T. In the context of Construction 5.11, the tag in the ciphertext would be set to the identity id. To decrypt, one would need to collect at least T secret keys for the identity id from the collection of IBE authorities. Previously, the work of [BBH06] describe a threshold IBE scheme based on the Boneh-Boyen IBE scheme. Construction 5.11 can be viewed as a *distributed* version of the [BBH06] scheme. More generally, we can view Construction 4.4 from Section 4 as a distributed monotone-policy IBE scheme.

Remark 5.17 (Flexible Broadcast Encryption)**.** Flexible broadcast encryption [FWW23] is a special case of threshold encryption with silent setup where the threshold is always fixed to 1. Flexible broadcast encryption generalizes distributed broadcast encryption [WQZD10, BZ14] in that it allows the encrypter to encrypt a message to an arbitrary set of recipients (identified by their public keys); the notion of distributed broadcast encryption imposes an additional restriction where users' public keys are tied to a slot, and one can only encrypt to a collection of public keys that occupy *distinct* slots. Previously, the work of [GLWW23] show how to generically lift any distributed broadcast encryption scheme into a flexible scheme with $\omega(\log \lambda)$ blowup in the size of user public keys. Construction 5.11 gives a direct construction of flexible broadcast encryption. Since the policy in distributed broadcast encryption is *fixed* and there is no need to support dynamic thresholds, we can achieve this notion with a slimmed down version of Construction 5.11 where there are no dummy users and we take $\mathbf{M}$ to be the share-generating matrix for a 1-out-of-N threshold policy. This yields a construction where the size of each user's public key contains $2N$ group elements (excluding the elements from the CRS) and the size of the ciphertext contains 4 group elements (excluding the tag). In the current best pairing-based distributed broadcast encryption scheme [KMW23], user public keys contain $2N - 2$ group elements while ciphertexts contain 3 group elements, and security relies on a q-type assumption in the plain model. Our work thus achieves the flexible notion where the public keys are larger by 2 group elements and the ciphertext size is larger by a single group element. In both schemes, the size of the CRS contain $4N + O(1)$ group elements.

Acknowledgments

We thank Hoeteck Wee for numerous insightful discussions. This work was done in part while the authors were visiting the Simons Institute for the Theory of Computing. Brent Waters is supported by NSF CNS-2318701 and a Simons Investigator Award. David J. Wu is supported by NSF CNS-2140975, CNS-2318701, a Sloan Fellowship, a Microsoft Research Faculty Fellowship, a Google Research Scholar Award, and an Amazon Research Award.

References

[ADM+24] Gennaro Avitabile, Nico Döttling, Bernardo Magri, Christos Sakkas, and Stella Wohnig. Signature-based witness encryption with compact ciphertext. In *ASIACRYPT*, 2024.

[AT24] Nuttapong Attrapadung and Junichi Tomida. A modular approach to registered ABE for unbounded predicates. In *CRYPTO*, 2024.

[BB04a] Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity-based encryption without random oracles. In *EUROCRYPT*, 2004.

[BB04b] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In *EUROCRYPT*, 2004.

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *EUROCRYPT*, 2005.

[BBH06] Dan Boneh, Xavier Boyen, and Shai Halevi. Chosen ciphertext secure public key threshold encryption without random oracles. In *CT-RSA*, 2006.

[BBK+23] Zvika Brakerski, Maya Farber Brodsky, Yael Tauman Kalai, Alex Lombardi, and Omer Paneth. SNARGs for monotone policy batch NP. In *CRYPTO*, 2023.

[BCF+25] Jan Bormet, Arka Rai Choudhuri, Sebastian Faust, Sanjam Garg, Hussien Othman, Guru-Vamsi Policharla, Ziyan Qu, and Mingyuan Wang. BEAST-MEV: Batched threshold encryption with silent setup for MEV prevention. *IACR Cryptol. ePrint Arch.*, 2025.

[BCJP24] Maya Farber Brodsky, Arka Rai Choudhuri, Abhishek Jain, and Omer Paneth. Monotone-policy aggregate signatures. In *EUROCRYPT*, 2024.

[BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *ASIACRYPT*, 2018.

[Bei96] Amos Beimel. *Secure Schemes for Secret Sharing and Key Distribution.* PhD thesis, Technion, 1996.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In *CRYPTO*, 2001.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *STOC*, 1988.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, 2001.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT*, 2003.

[BGW05] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *CRYPTO*, 2005.

[BHK17] Zvika Brakerski, Justin Holmgren, and Yael Tauman Kalai. Non-interactive delegation and batch NP verification from standard computational assumptions. In *STOC*, 2017.

[BL88] Josh Cohen Benaloh and Jerry Leichter. Generalized secret sharing and monotone functions. In *CRYPTO*, 1988.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In *ASIACRYPT*, 2001.

[BLS02] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In *SCN*, 2002.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *CRYPTO*, 2014.

[CES21] Kelong Cong, Karim Eldefrawy, and Nigel P. Smart. Optimizing registration based encryption. In *Cryptography and Coding*, 2021.

[CF13] Dario Catalano and Dario Fiore. Vector commitments and their applications. In *PKC*, 2013.

[CHK03] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT*, 2003.

[CHK04] Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In *EUROCRYPT*, 2004.

[CHW25] Jeffrey Champion, Yao-Ching Hsieh, and David J. Wu. Registered ABE and adaptively-secure broadcast encryption from succinct LWE. In *CRYPTO*, 2025.

[CW24] Jeffrey Champion and David J. Wu. Distributed broadcast encryption from lattices. In *TCC*, 2024.

[DCX+23] Sourav Das, Philippe Camacho, Zhuolun Xiang, Javier Nieto, Benedikt Bünz, and Ling Ren. Threshold signatures from inner product argument: Succinct, weighted, and multi-threshold. In *ACM CCS*, 2023.

[DDFY94] Alfredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. How to share a function securely. In *STOC*, 1994.

[Des87] Yvo Desmedt. Society and group oriented cryptography: A new concept. In *CRYPTO*, 1987.

[DF89] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In *CRYPTO*, 1989.

[DHMW23] Nico Döttling, Lucjan Hanzlik, Bernardo Magri, and Stella Wohnig. McFly: Verifiable encryption to the future made practical. In *Financial Cryptography and Data Security*, 2023.

[DJWW25] Lalita Devadas, Abhishek Jain, Brent Waters, and David J. Wu. Succinct witness encryption for batch languages and applications. In *ASIACRYPT*, 2025.

[DKL+23] Nico Döttling, Dimitris Kolonelos, Russell W. F. Lai, Chuanwei Lin, Giulio Malavolta, and Ahmadreza Rahimi. Efficient laconic cryptography from learning with errors. In *EUROCRYPT*, 2023.

[DPY24] Pratish Datta, Tapas Pal, and Shota Yamada. Registered FE beyond predicates: (attribute-based) linear functions and more. In *ASIACRYPT*, 2024.

[FFM+23] Danilo Francati, Daniele Friolo, Monosij Maitra, Giulio Malavolta, Ahmadreza Rahimi, and Daniele Venturi. Registered (inner-product) functional encryption. In *ASIACRYPT*, 2023.

[FKdP23] Dario Fiore, Dimitris Kolonelos, and Paola de Perthuis. Cuckoo commitments: Registration-based encryption and key-value map commitments for large spaces. In *ASIACRYPT*, 2023.

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In *CRYPTO*, 2018.

[Fra89] Yair Frankel. A practical protocol for large group oriented networks. In *EUROCRYPT*, 1989.

[FWW23] Cody Freitag, Brent Waters, and David J. Wu. How to use (plain) witness encryption: Registered abe, flexible broadcast, and more. In *CRYPTO*, 2023.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *STOC*, 2013.

[GHK+25] Sanjam Garg, Mohammad Hajiabadi, Dimitris Kolonelos, Abhiram Kothapalli, and Guru-Vamsi Policharla. A framework for witness encryption from linearly verifiable SNARKs and applications. In *CRYPTO*, 2025.

[GHM+19] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, Ahmadreza Rahimi, and Sruthi Sekar. Registration-based encryption from standard assumptions. In *PKC*, 2019.

[GHMR18] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, and Ahmadreza Rahimi. Registration-based encryption: Removing private-key generator from IBE. In *TCC*, 2018.

[GJM+24] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. hinTS: Threshold signatures with silent setup. In *IEEE S&P*, 2024.

[GKMR23] Noemi Glaeser, Dimitris Kolonelos, Giulio Malavolta, and Ahmadreza Rahimi. Efficient registration-based encryption. In *ACM CCS*, 2023.

[GKPW24] Sanjam Garg, Dimitris Kolonelos, Guru-Vamsi Policharla, and Mingyuan Wang. Threshold encryption with silent setup. In *CRYPTO*, 2024.

[GLWW23] Rachit Garg, George Lu, Brent Waters, and David J. Wu. Realizing flexible broadcast encryption: How to broadcast to a public-key directory. In *ACM CCS*, 2023.

[GLWW24] Rachit Garg, George Lu, Brent Waters, and David J. Wu. Reducing the CRS size in registered ABE systems. In *CRYPTO*, 2024.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *STOC*, 1985.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In *EUROCRYPT*, 2006.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*, 2016.

[GV20] Rishab Goyal and Satyanarayana Vusirikala. Verifiable registration-based encryption. In *CRYPTO*, 2020.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC*, 2011.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *IACR Cryptol. ePrint Arch.*, 2019.

[HLWW23] Susan Hohenberger, George Lu, Brent Waters, and David J. Wu. Registered attribute-based encryption. In *EUROCRYPT*, 2023.

[HW15] Pavel Hubácek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In *ITCS*, 2015.

[Jou00] Antoine Joux. A one round protocol for tripartite diffie-hellman. In *ANTS*, 2000.

[Kil06] Eike Kiltz. Chosen-ciphertext security from tag-based encryption. In *TCC*, 2006.

[KLVW23] Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and RAM delegation. In *STOC*, pages 1545–1552, 2023.

[KMW23] Dimitris Kolonelos, Giulio Malavolta, and Hoeteck Wee. Distributed broadcast encryption from bilinear groups. In *ASIACRYPT*, 2023.

[KPY19] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In *STOC*, 2019.

[LOS+06] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In *EUROCRYPT*, 2006.

[LPWW20] Benoît Libert, Alain Passelègue, Hoeteck Wee, and David J. Wu. New constructions of statistical NIZKs: Dual-mode DV-NIZKs and more. In *EUROCRYPT*, 2020.

[LW11] Allison B. Lewko and Brent Waters. Decentralizing attribute-based encryption. In *EUROCRYPT*, 2011.

[LWW25] George Lu, Brent Waters, and David J. Wu. Multi-authority registered attribute-based encryption. In *EUROCRYPT*, 2025.

[MRV+21] Silvio Micali, Leonid Reyzin, Georgios Vlachos, Riad S. Wahby, and Nickolai Zeldovich. Compact certificates of collective knowledge. In *IEEE S&P*, 2021.

[NWW24] Shafik Nassar, Brent Waters, and David J. Wu. Monotone policy BARGs from BARGs and additively homomorphic encryption. In *TCC*, 2024.

[NWW25] Shafik Nassar, Brent Waters, and David J. Wu. Monotone-policy BARGs and more from BARGs and quadratic residuosity. In *PKC*, 2025.

[RSY21] Leonid Reyzin, Adam D. Smith, and Sophia Yakoubov. Turning HATE into LOVE: compact homomorphic ad hoc threshold encryption for scalable MPC. In *Cyber Security Cryptography and Machine Learning*, 2021.

[RW22] Doreen Riepel and Hoeteck Wee. FABEO: fast attribute-based encryption with optimal security. In *ACM CCS*, 2022.

[RY07] Thomas Ristenpart and Scott Yilek. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In *EUROCRYPT*, 2007.

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO*, 1989.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11), 1979.

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, 1984.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, 1997.

[TCZ+20] Alin Tomescu, Robert Chen, Yiming Zheng, Ittai Abraham, Benny Pinkas, Guy Golan-Gueta, and Srinivas Devadas. Towards scalable threshold cryptosystems. In *IEEE S&P*, 2020.

[Wat11] Brent Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *PKC*, 2011.

[WQZD10] Qianhong Wu, Bo Qin, Lei Zhang, and Josep Domingo-Ferrer. Ad hoc broadcast encryption. In *ACM CCS*, 2010.

[WW22] Brent Waters and David J. Wu. Batch arguments for NP and more from standard bilinear group assumptions. In *CRYPTO*, 2022.

[WW25] Hoeteck Wee and David J. Wu. Unbounded distributed broadcast encryption and registered ABE from succinct LWE. In *CRYPTO*, 2025.

[ZZC+25] Ziqi Zhu, Kai Zhang, Zhili Chen, Junqing Gong, and Haifeng Qian. Black-box registered ABE from lattices. *IACR Cryptol. ePrint Arch.*, 2025.

[ZZGQ23] Ziqi Zhu, Kai Zhang, Junqing Gong, and Haifeng Qian. Registered ABE via predicate encodings. In *ASIACRYPT*, 2023.

A Generic Hardness of Extended Bilinear Diffie-Hellman Assumptions

In this section, we show that the extended bilinear Diffie-Hellman assumptions (Assumptions 3.3 and 3.12) hold in the generic bilinear group model. In the generic bilinear group model [Sho97, BBG05], we model a generic symmetric bilinear group of order p with label space $\mathcal{L}$ as two random injective functions $\sigma, \sigma_T : \mathbb{Z}_p \to \mathcal{L}$. In addition, algorithms in the generic bilinear group model have access to the following three oracles:

- **Base group evaluation:** On input two labels $\ell_1, \ell_2 \in \mathcal{L}$, the base group evaluation oracle first checks that ℓ_1 and ℓ_2 are in the image of σ. If so, it returns $\sigma(\sigma^{-1}(\ell_1) + \sigma^{-1}(\ell_2))$. Otherwise, it outputs $\perp$.
- **Target group evaluation:** On input two labels $\ell_1, \ell_2 \in \mathcal{L}$, the target group evaluation oracle first checks that ℓ_1 and ℓ_2 are in the image of σ_T. If so, it returns $\sigma_T(\sigma_T^{-1}(\ell_1) + \sigma_T^{-1}(\ell_2))$. Otherwise, it outputs $\perp$.
- **Pairing oracle:** On input two labels $\ell_1, \ell_2 \in \mathcal{L}$, the pairing oracle first checks that ℓ_1 and ℓ_2 are in the image of σ. If so, it returns $\sigma_T(\sigma^{-1}(\ell_1) \cdot \sigma^{-1}(\ell_2))$. Otherwise, it outputs $\perp$.

The work of Boneh, Boyen, and Goh [BBG05] provide a set of sufficient conditions under which a cryptographic assumption holds unconditionally in the generic bilinear group model. We recall a simplified version that suffices for our analysis below. We first define the notion of independence for polynomials and then give the general hardness statement.

Definition A.1 (Independence of Polynomials)**.** Let $\mathcal{P} = \{P_i\}_{i\in[k]}$ be a collection of n-variate polynomials $P_i \in \mathbb{Z}_p[X_1, \ldots, X_n]$. We say that a polynomial $f \in \mathbb{Z}_p[X_1, \ldots, X_n]$ is dependent on $\mathcal{P}$ if there exist coefficients $\alpha_0, \alpha_1, \ldots, \alpha_k \in \mathbb{Z}_p$ such that

$$f = \alpha_0 + \sum_{i\in[k]} \alpha_k P_k.$$

We say f is independent of $\mathcal{P}$ if f is not dependent on $\mathcal{P}$.

Theorem A.2 (Generic Hardness in Prime-Order Groups [BBG05, Theorem A.2, adapted])**.** *Let p be a prime, and $\mathcal{P} = \{P_i\}_{i\in[k]}$ be a collection of n-variate polynomials $P_i \in \mathbb{Z}_p[X_1, \ldots, X_n]$. Let $T \in \mathbb{Z}_p[X_1, \ldots, X_n]$. For an adversary $\mathcal{A}$ and a bit $b \in \{0, 1\}$, define the following distinguishing experiment in the generic bilinear group model of order p:*

- *At the beginning of the game, the challenger samples $x_1, \ldots, x_n \xleftarrow{R} \mathbb{Z}_p$. For each $i \in [k]$, it computes $\ell_i = \sigma(P_i(x_1, \ldots, x_n))$.*
- *If $b = 0$, the challenger computes $\tau = \sigma_T(T(x_1, \ldots, x_n))$. If $b = 1$, the challenger samples $r \xleftarrow{R} \mathbb{Z}_p$ and sets $\tau = \sigma_T(r)$.*
- *The challenger gives $(\ell_1, \ldots, \ell_k, \tau)$ to $\mathcal{A}$. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, which is the output of the experiment.*

Let $\mathcal{P}^2 = \{P_iP_j : i, j \in [k]\}$, and let d be a bound on the total degree of the polynomials in $\mathcal{P}^2 \cup \{T\}$. If T is independent of $\mathcal{P}^2$, then for all adversaries $\mathcal{A}$ that make at most q queries to the generic bilinear group oracles, it holds that

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \leq \frac{(q + k + 1)^2 d}{p}$$

in the above distinguishing experiment.

A.1 Generic Hardness of Assumption 3.3

In this section, we show that for all $N = \mathrm{poly}(\lambda)$, the N-extended decisional bilinear Diffie-Hellman assumption (Assumption 3.3) holds unconditionally in the generic bilinear group model.

Theorem A.3 (Generic Hardness of Assumption 3.3)**.** *Take $N \in \mathbb{N}$. Let $\mathcal{A}$ be any generic adversary for the N-extended decisional bilinear Diffie-Hellman assumption that makes at most q generic group oracle queries. Then, the advantage of $\mathcal{A}$ is at most $O(q^2N^5)/p$ in the generic bilinear group model. In particular, the advantage of $\mathcal{A}$ is negligible for all polynomials $q, N = \mathrm{poly}(\lambda)$ and $p > 2^{\omega(\log\lambda)}$.*

Proof. Fix a prime p and a finite label space $\mathcal{L} \subset \{0, 1\}^*$ of size at least p. We define the following distributions:

- $\mathcal{D}_0$: Sample random injective functions $\sigma, \sigma_T : \mathbb{Z}_p \to \mathcal{L}$. Sample $a, b, s \xleftarrow{R} \mathbb{Z}_p$ and $r, c_i \xleftarrow{R} \mathbb{Z}_p^*$ for all $i \in [N]$. In the following, we will use implicit notation to denote group elements; specifically, we will write $[x]_{\mathbb{G}}$ to denote $\sigma(x)$ and $[x]_{\mathbb{G}_T}$ to denote $\sigma_T(x)$. Let

$$\mathsf{params} = \begin{pmatrix} [r]_{\mathbb{G}}, [ra]_{\mathbb{G}}, [rb]_{\mathbb{G}}, [rs]_{\mathbb{G}}, \\ \left\{[rc_i]_{\mathbb{G}}, [r/c_i]_{\mathbb{G}}, [rabc_i]_{\mathbb{G}}, [rsc_i]_{\mathbb{G}}\right\}_{i\in[N]} \\ \left\{[rc_i/c_j]_{\mathbb{G}}, [rabc_i/c_j]_{\mathbb{G}}\right\}_{i\neq j\in[N]} \end{pmatrix}. \tag{A.1}$$

  Let $T = [r^2abs]_{\mathbb{G}_T}$. Output $\mathcal{A}(\mathsf{params}, T)$. This corresponds to the pseudorandom distribution in Assumption 3.3 where $[r]_{\mathbb{G}} = \sigma(r)$ represents the random generator g.

- $\mathcal{D}_1$: Same as $\mathcal{D}_0$, except the challenger samples $\tilde{r} \xleftarrow{R} \mathbb{Z}_p^*$ and sets $r = \tilde{r}\prod_{\ell\in[N]} c_\ell$. Moreover, the challenger constructs params as

$$\mathsf{params} = \begin{pmatrix} [r]_{\mathbb{G}}, [ra]_{\mathbb{G}}, [rb]_{\mathbb{G}}, [rs]_{\mathbb{G}}, \\ \left\{[rc_i]_{\mathbb{G}}, [\tilde{r}\prod_{\ell\neq i} c_i]_{\mathbb{G}}, [rabc_i]_{\mathbb{G}}, [rsc_i]_{\mathbb{G}}\right\}_{i\in[N]} \\ \left\{[c_i\tilde{r}\prod_{\ell\neq j} c_\ell]_{\mathbb{G}}, [abc_i\tilde{r}\prod_{\ell\neq j} c_\ell]_{\mathbb{G}}\right\}_{i\neq j\in[N]} \end{pmatrix}. \tag{A.2}$$

  The challenger sets $T = [rabs]_{\mathbb{G}_T}$ and the output is $\mathcal{A}(\mathsf{params}, T)$.

- $\mathcal{D}_2$: Same as $\mathcal{D}_1$, except the challenger samples $\tilde{r}, c_1, \ldots, c_N \xleftarrow{R} \mathbb{Z}_p$.
- $\mathcal{D}_3$: Same as $\mathcal{D}_2$, except the challenger samples $\tau \xleftarrow{R} \mathbb{Z}_p$ and sets $T = [\tau]_{\mathbb{G}_T}$.
- $\mathcal{D}_4$: Same as $\mathcal{D}_3$, except the challenger samples $\tilde{r}, c_1, \ldots, c_N \xleftarrow{R} \mathbb{Z}_p^*$.
- $\mathcal{D}_5$: Same as $\mathcal{D}_4$, except the challenger samples $r \xleftarrow{R} \mathbb{Z}_p^*$ and sets params according to Eq. (A.1). This corresponds to the random distribution in Assumption 3.3.

We show that each pair of adjacent distributions are statistically indistinguishable in the generic bilinear group model:

- First, distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ are identical distributions. Specifically, in distribution $\mathcal{D}_1$, the challenger samples $\tilde{r} \xleftarrow{R} \mathbb{Z}_p^*$ and sets $r = \tilde{r} \cdot \prod_{\ell \in [N]} c_\ell$. Since $c_\ell \in \mathbb{Z}_p^*$ is invertible, the distribution of r remains uniform over $\mathbb{Z}_p^*$. Moreover, the components in params are constructed exactly as in $\mathcal{D}_0$.
- The only difference between $\mathcal{D}_1$ and $\mathcal{D}_2$ is $r, c_i \xleftarrow{R} \mathbb{Z}_p$ in $\mathcal{D}_1$ and $r, c_i \xleftarrow{R} \mathbb{Z}_p^*$ in $\mathcal{D}_2$. Since the statistical distance between the uniform distribution over $\mathbb{Z}_p$ and the uniform distribution over $\mathbb{Z}_p^*$ is $1/p$, the statistical distance between $\mathcal{D}_1$ and $\mathcal{D}_2$ is $(N+1)/p$.
- Distributions $\mathcal{D}_2$ and $\mathcal{D}_3$ are statistically indistinguishable by Theorem A.2. To argue this, we first define a set of polynomials $\mathcal{P} = \{P_i\}_{i \in [k]}$ over the formal variables $\tilde{r}, a, b, s, c_1, \ldots, c_N$, where the polynomials P_i correspond to the components in Eq. (A.2) and $k = 4 + 4N + N(N-1)$. By inspection, the polynomials in $\mathcal{P}$ have degree at most $N+4$. The challenge polynomial $T = r^2 abs = \tilde{r}^2 abs \prod_{\ell \in [N]} c_\ell^2$ has degree $2N+5$.

  To appeal to Theorem A.2, we need to argue that the polynomial T is independent of the polynomials $\mathcal{P}^2$. Since the elements of $\mathcal{P}^2$ and the polynomial T all consist of a single monomial, it suffices to argue that $T \notin \mathcal{P}^2$. Take any $P_i, P_j \in \mathcal{P}$. We show that $P_i P_j \neq T$.

  - Suppose $P_i \in \{r, ra, rb, rs\}$. Then $P_i P_j = T$ only if $P_j \in \{rabs, rbs, ras, rab\}$. None of these options are contained in $\mathcal{P}$ so $P_i P_j \neq T$.
  - Suppose $P_i = \{rc_i, rabc_i, rsc_i\}$ for some $i \in [N]$. Then $P_i P_j = T$ only if
  $$P_j \in \left\{abs\tilde{r} \textstyle\prod_{\ell \neq i} c_\ell \ , \ s\tilde{r} \prod_{\ell \neq i} c_\ell \ , \ ab\tilde{r} \prod_{\ell \neq i} c_\ell\right\}.$$
  None of these options are contained in $\mathcal{P}$, so $P_i P_j \neq T$.
  - Suppose $P_i = \tilde{r} \prod_{\ell \neq i} c_\ell$ for some $i \in [N]$. Then $P_i P_j = T$ only if $P_j = rabsc_i \notin \mathcal{P}$.
  - Suppose $P_i = c_i \tilde{r} \prod_{\ell \neq j} c_\ell$ for some $i \neq j \in [N]$. Then, $P_i P_j = T$ only if $P_j = absc_j \tilde{r} \prod_{\ell \neq i} c_i \notin \mathcal{P}$.
  - Suppose $P_i = abc_i \tilde{r} \prod_{\ell \neq j} c_\ell$ for some $i \neq j \in [N]$. Then, $P_i P_j = T$ only if $P_j = sc_j \tilde{r} \prod_{\ell \neq i} c_i \notin \mathcal{P}$.

  We conclude that the target polynomial T is independent of $\mathcal{P}^2$. By Theorem A.2, this means the statistical distance between distributions $\mathcal{D}_2$ and $\mathcal{D}_3$ against a generic adversary $\mathcal{A}$ making at most q generic group queries is bounded by
  $$\frac{(q+k+1)^2(2(N+4))}{p} = \frac{O(q^2N^5)}{p}.$$
- The statistical distance between $\mathcal{D}_3$ and $\mathcal{D}_4$ is $(N+1)/p$ by the same argument used to compute the statistical distance between $\mathcal{D}_1$ and $\mathcal{D}_2$.
- Finally, distributions $\mathcal{D}_4$ and $\mathcal{D}_5$ are identical experiments by the same argument used to analyze distributions $\mathcal{D}_0$ and $\mathcal{D}_1$.

By a hybrid argument, we conclude that the statistical distance between $\mathcal{D}_0$ and $\mathcal{D}_5$ is bounded by $O(q^2N^5)/p$, as desired. □

A.2 Generic Hardness of Assumption 3.12

In this section, we show that for all $N = \mathsf{poly}(\lambda)$, the N-extended decisional bilinear Diffie-Hellman exponent assumption (Assumption 3.12) also holds unconditionally in the generic bilinear group model.

Theorem A.4 (Generic Hardness of Assumption 3.12)**.** *Take $N \in \mathbb{N}$. Let $\mathcal{A}$ be any generic adversary for the N-extended decisional bilinear Diffie-Hellman assumption that makes at most q generic group oracle queries. Then, the advantage of $\mathcal{A}$ is at most $O(q^2N^3)/p$ in the generic bilinear group model. In particular, the advantage of $\mathcal{A}$ is negligible for all polynomials $q, N = \mathsf{poly}(\lambda)$ and $p > 2^{\omega(\log \lambda)}$.*

Proof. Similar to the proof of Theorem A.3, we proceed via a hybrid argument. Specifically, fix a prime p and a finite label space $\mathcal{L} \subset \{0,1\}^*$ of size at least p. We now define the following distributions:

- $\mathcal{D}_0$: Sample random injective functions $\sigma, \sigma_T : \mathbb{Z}_p \to \mathcal{L}$. Sample $a, b, t \xleftarrow{R} \mathbb{Z}_p$ and $r, \hat{r}, c \xleftarrow{R} \mathbb{Z}_p^*$ for all $i \in [N]$. In the following, we will use implicit notation to denote group elements; specifically, we will write $[x]_{\mathbb{G}}$ to denote $\sigma(x)$ and $[x]_{\mathbb{G}_T}$ to denote $\sigma_T(x)$. Let

$$\mathsf{params} = \left(\begin{array}{c} [r]_{\mathbb{G}}, [\hat{r}]_{\mathbb{G}}, [ra]_{\mathbb{G}}, [\hat{r}a]_{\mathbb{G}}, [\hat{r}b]_{\mathbb{G}}, [rt]_{\mathbb{G}}, [\hat{r}t]_{\mathbb{G}} \\ \{[rc^i]_{\mathbb{G}}, [\hat{r}c^i]_{\mathbb{G}}, [\hat{r}abc^i]_{\mathbb{G}}\}_{i \in [-N,N] \setminus \{0\}}, \{[\hat{r}tc^i]_{\mathbb{G}}\}_{i \in [N]} \end{array} \right). \tag{A.3}$$

  Let $T = [r\hat{r}abt]_{\mathbb{G}_T}$. Output $\mathcal{A}(\mathsf{params}, T)$. This corresponds to the pseudorandom distribution in Assumption 3.12 where $[r]_{\mathbb{G}} = \sigma(r)$ and $[\hat{r}]_{\mathbb{G}} = \sigma(\hat{r})$ represent the random generators g and $\hat{g}$, respectively.

- $\mathcal{D}_1$: Same as $\mathcal{D}_0$, except the challenger samples $s, \hat{s} \xleftarrow{R} \mathbb{Z}_p^*$ and sets $r = sc^N$ and $\hat{r} = \hat{s}c^N$. Moreover, the challenger constructs params as

$$\mathsf{params} = \left(\begin{array}{c} [sc^N]_{\mathbb{G}}, [\hat{s}c^N]_{\mathbb{G}}, [sac^N]_{\mathbb{G}}, [\hat{s}ac^N]_{\mathbb{G}}, [\hat{s}bc^N]_{\mathbb{G}}, [stc^N]_{\mathbb{G}}, [\hat{s}tc^N]_{\mathbb{G}} \\ \{[sc^i]_{\mathbb{G}}, [\hat{s}c^i]_{\mathbb{G}}, [\hat{s}abc^i]_{\mathbb{G}}\}_{i \in [0,2N] \setminus \{N\}}, \{[\hat{s}tc^i]_{\mathbb{G}}\}_{i \in [N+1,2N]} \end{array} \right). \tag{A.4}$$

  The challenger sets $T = [s\hat{s}abtc^{2N}]_{\mathbb{G}_T}$ and the output is $\mathcal{A}(\mathsf{params}, T)$.

- $\mathcal{D}_2$: Same as $\mathcal{D}_1$, except the challenger samples $s, \hat{s}, c \xleftarrow{R} \mathbb{Z}_p$.

- $\mathcal{D}_3$: Same as $\mathcal{D}_2$, except the challenger samples $\tau \xleftarrow{R} \mathbb{Z}_p$ and sets $T = [\tau]_{\mathbb{G}_T}$.

- $\mathcal{D}_4$: Same as $\mathcal{D}_3$, except the challenger samples $s, \hat{s}, c \xleftarrow{R} \mathbb{Z}_p^*$.

- $\mathcal{D}_5$: Same as $\mathcal{D}_4$, except the challenger samples $r, \hat{r} \xleftarrow{R} \mathbb{Z}_p^*$ and sets params according to Eq. (A.3). This corresponds to the random distribution in Assumption 3.12.

We now show that each pair of adjacent distributions are statistical indistinguishable in the generic bilinear group model:

- Distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ are identical. Specifically, in distribution $\mathcal{D}_1$, the challenger samples $s, \hat{s} \xleftarrow{R} \mathbb{Z}_p$ and sets $r = sc^N$ and $\hat{r} = \hat{s}c^N$. Since $c \in \mathbb{Z}_p^*$ is invertible, the distribution of $r, \hat{r}$ in $\mathcal{D}_1$ remain uniform. The components in params are then constructed exactly as prescribed in $\mathcal{D}_0$.

- The only difference between $\mathcal{D}_1$ and $\mathcal{D}_2$ is the distributions of s, $\hat{s}$, and c. In $\mathcal{D}_1$, these are uniform over $\mathbb{Z}_p$ whereas in $\mathcal{D}_2$, they are uniform over $\mathbb{Z}_p^*$. The statistical distance between the uniform distribution over $\mathbb{Z}_p$ and that over $\mathbb{Z}_p^*$ is $1/p$, so the statistical distance between $\mathcal{D}_1$ and $\mathcal{D}_2$ is in turn $3/p$.

- Distributions $\mathcal{D}_2$ and $\mathcal{D}_3$ are statistically indistinguishable by Theorem A.2. To argue this, we first define a set of polynomials $\mathcal{P} = \{P_i\}_{i \in [k]}$ over the formal variables $s, \hat{s}, c, a, b, t$, where the polynomials P_i correspond to the components in Eq. (A.4) and $k = 7N + 7$. By inspection, the polynomials in $\mathcal{P}$ have degree at most $2N + 3$. The challenge polynomial $T = s\hat{s}abtc^{2N}$ has degree $2N + 5$.

  To appeal to Theorem A.2, we need to argue that the polynomial T is independent of the polynomials in $\mathcal{P}^2$. Since the elements of $\mathcal{P}^2$ and the polynomial T all consist of a single monomial, it suffices to argue that $T \notin \mathcal{P}^2$. Take any $P_i, P_j \in \mathcal{P}$. We show that $P_iP_j \neq T$.

– Suppose $P_i \in \{sc^N, \hat{s}c^N, sac^N, \hat{s}ac^N, \hat{s}bc^N, stc^N, \hat{s}tc^N\}$. Then $P_iP_j = s\hat{s}abtc^{2N}$ only if

$$P_j \in \{\hat{s}abtc^N, sabtc^N, \hat{s}btc^N, sbtc^N, satc^N, \hat{s}abc^N, sabc^N\}.$$

By inspection, this means $P_j \notin \mathcal{P}$, as required.

– Suppose $P_i = \{sc^i, \hat{s}c^i\}$ for some $i \in [0, 2N]\backslash\{N\}$. Then $P_iP_j = s\hat{s}abtc^{2N}$ only if $P_j \in \{\hat{s}abtc^{2N-i}, sabtc^{2N-i}\}$. This means $P_j \notin \mathcal{P}$.

– Suppose $P_i = \hat{s}abc^i$ for some $i \in [0, 2N] \setminus \{N\}$. Then $P_iP_j = s\hat{s}abtc^{2N}$ only if $P_j = stc^{2N-i} \notin \mathcal{P}$.

– Suppose $P_i = \hat{s}tc^i$ for some $i \in [N+1, 2N]$. Then $P_iP_j = s\hat{s}abtc^{2N}$ only if $P_j = sabc^{2N-i} \notin \mathcal{P}$.

We conclude that the target polynomial T is independent of $\mathcal{P}^2$. By Theorem A.2, this means the statistical distance between $\mathcal{D}_2$ and $\mathcal{D}_3$ against a generic adversary $\mathcal{A}$ making at most q generic group queries is bounded by

$$\frac{(q+k+1)^2(2N+5)}{p} = \frac{O(q^2N^3)}{p}.$$

- The statistical distance between $\mathcal{D}_3$ and $\mathcal{D}_4$ is $3/p$ by the same argument used to analyze the statistical distance between $\mathcal{D}_1$ and $\mathcal{D}_2$.
- Distributions $\mathcal{D}_4$ and $\mathcal{D}_5$ are identical by the same argument used to analyze the distributions $\mathcal{D}_0$ and $\mathcal{D}_1$.

The theorem now follows by a hybrid argument. □

B Analysis of Construction 3.15 (Monotone-Policy Signature)

In this section, we give the correctness and security analysis of Construction 3.15. The proofs and analysis follow a similar structure as the analogous properties for Construction 3.4.

B.1 Proof of Theorem 3.16 (Signing Correctness)

Take any $\lambda, \kappa \in \mathbb{N}$ and message $m \in \mathbb{Z}_p$. Let $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa)$, $(\mathsf{vk}, \mathsf{ht}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{crs})$ and $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$. By construction, we can write

$$\begin{aligned}
\mathsf{crs} &= \left(\mathcal{G}, g, \hat{g}, B, \hat{u}, \hat{h}, \{g^{c^i}, \hat{g}^{c^i s}\}_{i\in[-N,N]\backslash\{0\}}\right)\\
\mathsf{vk} &= (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, A)\\
\sigma &= (\sigma_1, \hat{\sigma}_2) = (g^r, \hat{g}^\alpha(\hat{u}^m\hat{h})^r),
\end{aligned}$$

where $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, e)$ and $A = e(g, \hat{g})^\alpha$. By bilinearity, we now have

$$e(g, \hat{\sigma}_2) = e(g, \hat{g}^\alpha(\hat{u}^m\hat{h})^r) = e(g, \hat{g})^\alpha \cdot e(g^r, \hat{u}^m\hat{h}) = A \cdot e(\sigma_1, \hat{u}^m\hat{h}).$$

We conclude that $\mathsf{PartialVerify}(\mathsf{vk}, m, \sigma)$ outputs 1, as required. □

B.2 Proof of Theorem 3.17 (Aggregation Correctness)

Take any $\lambda, \kappa \in \mathbb{N}$, message $m \in \mathbb{Z}_p$, policy $\varphi \in \Phi_\kappa$ (with associated matrix $\mathbf{M} \in \mathbb{Z}_p^{N\times W}$), any set $S \subseteq [N]$ where $\varphi(S) = 1$, any crs in the support of $\mathsf{Setup}(1^\lambda, 1^\kappa)$, any $\{(\mathsf{vk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell)\}_{\ell\in[N]}$ where $(\mathsf{vk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell)$ is in the support of $\mathsf{KeyGen}(\mathsf{crs})$ for all $\ell \in [N]$, and any collection of signatures $\{\sigma_\ell\}_{\ell\in S}$ where $\mathsf{PartialVerify}(\mathsf{crs}, m, \sigma_\ell) = 1$ for all $\ell \in S$. By construction, we can now write

$$\begin{aligned}
\mathsf{crs} &= \left(\mathcal{G}, g, \hat{g}, B, \hat{u}, \hat{h}, \{g^{c^i}, \hat{g}^{c^i s}\}_{i\in[-N,N]\backslash\{0\}}\right)\\
\mathsf{vk}_\ell &= (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, A_\ell)\\
\mathsf{ht}_\ell &= \{\hat{v}'_{\ell,i}\}_{i\in[-N,N]\backslash\{0\}}\\
\sigma_\ell &= (\sigma_{\ell,1}, \hat{\sigma}_{\ell,2}),
\end{aligned}$$

where $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, e)$, $B = e(g,\hat{g})^{s_1}$, and $A_\ell = e(g,\hat{g})^{\alpha_\ell}$. By construction of KeyGen, $\hat{v}'_{\ell,i} = \hat{g}^{\alpha_\ell c^i}$ for all $i \in [-N, N] \setminus \{0\}$. Since PartialVerify$(\mathsf{crs}, m, \sigma_\ell) = 1$ for all $\ell \in S$, this means

$$\forall \ell \in S : e(g, \hat{\sigma}_{\ell,2}) = A_\ell \cdot e(\sigma_{\ell,1}, \hat{u}^m \hat{h}) = e(g,\hat{g})^{\alpha_\ell} \cdot e(\sigma_{\ell,1}, \hat{u}^m \hat{h}). \tag{B.1}$$

Let

$$(\mathsf{vk}_\varphi, \mathsf{ak}_\varphi) = \mathsf{Preprocess}(\mathsf{crs}, \mathbf{M}, \{(\mathsf{vk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [N]})$$
$$\sigma_{\mathrm{agg}} = (\sigma_{\mathrm{agg},1}, \hat{\sigma}_{\mathrm{agg},2}, \sigma_{\mathrm{agg},3}) = \mathsf{Aggregate}(\mathsf{ak}_\varphi, \{\sigma_\ell\}_{\ell \in S}).$$

By construction, the preprocessing algorithm first computes

$$\hat{z} = \prod_{\ell \in [N]} \hat{g}^{\mathbf{m}_\ell^\top (c^\ell \mathbf{s})} \hat{v}_{\ell,\ell} = \hat{g}^{\tilde{z}} \quad \text{where} \quad \tilde{z} = \sum_{\ell \in [N]} c^\ell (\mathbf{m}_\ell^\top \mathbf{s} + \alpha_\ell).$$

For each $\ell \in [N]$, it also computes

$$\hat{v}_\ell = \prod_{\substack{i \in [N] \\ i \neq \ell}} \hat{v}'_{i,i-\ell} \cdot \hat{g}^{c^{i-\ell} \mathbf{m}_i^\top \mathbf{s}} = \prod_{\substack{i \in [N] \\ i \neq \ell}} \hat{g}^{c^{i-\ell}(\mathbf{m}_i^\top \mathbf{s} + \alpha_i)}. \tag{B.2}$$

Then, it sets $\mathsf{vk}_\varphi = (\mathcal{G}, g, u, h, B, \hat{z})$. The aggregation algorithm starts by computing:

$$\sigma_{\mathrm{agg},1} = \prod_{\ell \in S} \sigma_{\ell,1}^{\omega_\ell} \quad \text{and} \quad \hat{\sigma}_{\mathrm{agg},2} = \prod_{\ell \in S} \hat{\sigma}_{\ell,2}^{\omega_\ell} \hat{v}_\ell^{\omega_\ell} \quad \text{and} \quad \sigma_{\mathrm{agg},3} = \prod_{\ell \in S} \left(g^{c^{-\ell}}\right)^{\omega_\ell}. \tag{B.3}$$

By Eq. (B.1) and bilinearity, this means

$$e(\sigma_{\mathrm{agg},1}, \hat{u}^m \hat{h}) = \prod_{\ell \in S} e(\sigma_{\ell,1}, \hat{u}^m \hat{h})^{\omega_\ell} = \prod_{\ell \in S} \frac{e(g, \hat{\sigma}_{\ell,2}^{\omega_\ell})}{e(g,\hat{g})^{\omega_\ell \alpha_\ell}} \tag{B.4}$$

Let $\tilde{\sigma}_{\mathrm{agg},3} = \sum_{\ell \in S} \omega_\ell c^{-\ell}$. Then $\sigma_{\mathrm{agg},3} = g^{\tilde{\sigma}_{\mathrm{agg},3}}$. Next, $\boldsymbol{\omega}^\top \mathbf{M} = \mathbf{e}_1^\top$ so

$$\sum_{\ell \in [N]} \omega_\ell \mathbf{m}_\ell^\top \mathbf{s} = \boldsymbol{\omega}^\top \mathbf{M} \mathbf{s} = \mathbf{e}_1^\top \mathbf{s} = s_1.$$

Combined with the fact that $\omega_\ell = 0$ for all $\ell \notin S$, we have

$$\begin{aligned}
\tilde{z} \cdot \tilde{\sigma}_{\mathrm{agg},3} &= \sum_{\ell \in [N]} \sum_{i \in [N]} c^i (\mathbf{m}_i^\top \mathbf{s} + \alpha_i) \cdot \omega_\ell c^{-\ell} \\
&= \sum_{\ell \in [N]} \omega_\ell \mathbf{m}_\ell^\top \mathbf{s} + \sum_{\ell \in S} \omega_\ell \alpha_\ell + \sum_{\ell \in S} \sum_{\substack{i \in [N] \\ i \neq \ell}} \left(\omega_\ell (\mathbf{m}_i^\top \mathbf{s} + \alpha_i) \cdot c^{i-\ell}\right) \\
&= s_1 + \sum_{\ell \in S} \omega_\ell \alpha_\ell + \sum_{\ell \in S} \sum_{\substack{i \in [N] \\ i \neq \ell}} \left(\omega_\ell (\mathbf{m}_i^\top \mathbf{s} + \alpha_i) c^{i-\ell}\right)
\end{aligned}$$

By Eq. (B.2), this means

$$e(\sigma_{\mathrm{agg},3}, \hat{z}) = e(g,\hat{g})^{\tilde{z}\tilde{\sigma}_{\mathrm{agg},3}} = e(g,\hat{g})^{s_1} \cdot \prod_{\ell \in S} e(g,\hat{g})^{\omega_\ell \alpha_\ell} \cdot \prod_{\ell \in S} e(g, \hat{v}_\ell)^{\omega_\ell}. \tag{B.5}$$

Combining Eqs. (B.3) to (B.5), we have

$$\begin{aligned}
e(g,\hat{g})^{s_1} \cdot e(g, \hat{\sigma}_{\mathrm{agg},2}) &= e(g,\hat{g})^{s_1} \cdot \prod_{\ell \in S} e(g, \hat{\sigma}_{\ell,2}^{\omega_\ell}) \cdot \prod_{\ell \in S} e(g, \hat{v}_\ell)^{\omega_\ell}. \\
&= e(\sigma_{\mathrm{agg},1}, \hat{u}^m \hat{h}) \cdot e(\sigma_{\mathrm{agg},3}, \hat{z}).
\end{aligned}$$

Correspondingly, Verify$(\mathsf{vk}_\varphi, m, \sigma_{\mathrm{agg}}) = 1$, as required. □

B.3 Proof of Theorem 3.18 (Static Unforgeability)

Similar to the proof of Theorem 3.7, it will be more convenient to use the following search variant of the N-extended bilinear Diffie-Hellman exponent assumption. We state the assumption and show that it is implied by the decisional N-extended bilinear Diffie-Hellman exponent assumption below:

Lemma B.1. *Let λ be a security parameter. For an adversary $\mathcal{A}$, define the search N-extended bilinear Diffie-Hellman exponent experiment as follows:*

1. *The challenger samples $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$, $g, \hat{g} \xleftarrow{R} \mathbb{G} \setminus \{1\}$, $a, b, t \xleftarrow{R} \mathbb{Z}_p$, and $c \xleftarrow{R} \mathbb{Z}_p^*$. Define*

$$\mathsf{params} = \left(1^\lambda, \mathcal{G}, g, \hat{g}, g^a, \hat{g}^a, \hat{g}^b, \{g^{c^i}, \hat{g}^{c^i}, \hat{g}^{abc^i}\}_{i \in [-N,N] \setminus \{0\}}\right).$$

2. *On input* params*, algorithm $\mathcal{A}$ outputs $\tau_0, \hat{\tau}_0, \tau_1, \ldots, \tau_N \in \mathbb{G}$.*

3. *The output of the experiment is $b = 1$ if*

$$e(g, \hat{g})^{ab} = e(\tau_0, \hat{g}) \cdot e(g, \hat{\tau}_0) \cdot \prod_{i \in [N]} e(\tau_i, \hat{g}^{c^i}).$$

   Otherwise, the output is 0*.*

If the decisional N-extended bilinear Diffie-Hellman exponent assumption holds with respect to GroupGen*, then for all efficient adversaries $\mathcal{A}$, there exists a negligible function* negl$(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,* $\Pr[b = 1] = \mathsf{negl}(\lambda)$ *in the search N-extended bilinear Diffie-Hellman exponent experiment.*

Proof. Suppose there exists an efficient adversary $\mathcal{A}$ that can solve the search N-extended bilinear Diffie-Hellman exponent problem with advantage ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for the decisional N-extended bilinear Diffie-Hellman exponent problem.

- On input (params, T) where

$$\mathsf{params} = \left(1^\lambda, \mathcal{G}, g, \hat{g}, g^a, \hat{g}^a, \hat{g}^b, g^t, \hat{g}^t, \{g^{c^i}, \hat{g}^{c^i}, \hat{g}^{abc^i}\}_{i \in [-N,N] \setminus \{0\}}, \{\hat{g}^{tc^i}\}_{i \in [N]}\right).$$

  algorithm $\mathcal{B}$ runs algorithm $\mathcal{A}$ on the following input

$$\mathsf{params}' = \left(1^\lambda, \mathcal{G}, g, \hat{g}, g^a, \hat{g}^a, \hat{g}^b, \{g^{c^i}, \hat{g}^{c^i}, \hat{g}^{abc^i}\}_{i \in [-N,N] \setminus \{0\}}\right).$$

- Algorithm $\mathcal{B}$ outputs $\tau_0, \hat{\tau}_0, \tau_1, \ldots, \tau_N \in \mathbb{G}$. Algorithm $\mathcal{B}$ first checks if

$$e(g^a, \hat{g}^b) = e(\tau_0, \hat{g}) \cdot e(g, \hat{\tau}_0) \cdot \prod_{i \in [N]} e(\tau_i, \hat{g}^{c^i}). \tag{B.6}$$

  If this does not hold, algorithm $\mathcal{B}$ outputs 0. Otherwise, algorithm $\mathcal{B}$ outputs 1 if and only if

$$T = e(\tau_0, \hat{g}^t) \cdot e(g^t, \hat{\tau}_0) \cdot \prod_{i \in [N]} e(\tau_i, \hat{g}^{tc^i}). \tag{B.7}$$

Algorithm $\mathcal{B}$ perfectly simulates the parameters for the search N-extended bilinear Diffie-Hellman exponent problem. Thus, with probability ε, it will output $\tau_0, \hat{\tau}_0, \tau_1 \ldots, \tau_N$ that satisfy Eq. (B.6). We now compute the advantage of $\mathcal{A}$:

- Suppose $T = e(g, \hat{g})^{abt}$. If Eq. (B.6) holds, then Eq. (B.7) also holds. In this case, algorithm $\mathcal{A}$ outputs 1 whenever Eq. (B.6) holds, which occurs with probability ε.
- Suppose $T \xleftarrow{R} \mathbb{G}_T$. Since the view of $\mathcal{A}$ is independent of T, and moreover, the exponents t, c are sampled independently of T, this means Eq. (B.7) holds with probability exactly $1/p$. Thus, in this case, algorithm $\mathcal{B}$ outputs 1 with probability at most $1/p$.

We conclude that algorithm $\mathcal{B}$ distinguishes with advantage at least $\varepsilon - 1/p \geq \varepsilon - 2^{-\lambda}$ since $p > 2^\lambda$. The claim holds. □

Proof of Theorem 3.18. Suppose there exists an efficient adversary $\mathcal{A}$ that wins the static unforgeability game for policies with N users with non-negligible advantage ε. We use $\mathcal{A}$ to construct a new adversary $\mathcal{B}$ that breaks the search N-extended bilinear Diffie-Hellman exponent assumption from Lemma B.1. Our proof follows the same structure as the proof of Theorem 3.7. In the following, we will use a tilde (e.g., $\tilde{u}, \tilde{h}$) to denote exponents sampled by the reduction algorithm $\mathcal{B}$. Algorithm $\mathcal{B}$ works as follows:

1. On input the challenge
$$\big(1^\lambda, \mathcal{G}, g, \hat{g}, g^a, \hat{g}^a, \hat{g}^b, \{g^{c^i}, \hat{g}^{c^i}, \hat{g}^{abc^i}\}_{i\in[-N,N]\setminus\{0\}}\big),$$
algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ on 1^λ. Algorithm $\mathcal{A}$ outputs a policy family parameter 1^κ and a policy $\mathbf{M} \in \mathbb{Z}_p^{N\times W}$. In addition, algorithm $\mathcal{B}$ specifies a set $C \subseteq [N]$ of corrupted indices and a challenge message $m^* \in \mathbb{Z}_p$.

2. Algorithm $\mathcal{B}$ first checks if C satisfies the policy $\mathbf{M}$. If so, then it halts with output 0.

3. Otherwise, algorithm $\mathcal{B}$ constructs the common reference string crs as follows. First, it samples $\tilde{u}, \tilde{h} \xleftarrow{R} \mathbb{Z}_p$ and sets
$$\hat{u} = (\hat{g}^b) \cdot \hat{g}^{\tilde{u}} \quad \text{and} \quad \hat{h} = \hat{g}^{\tilde{h}}/(\hat{g}^b)^{m^*}.$$
Since the set $C \subseteq [N]$ does not satisfy the policy $\mathbf{M}$, there exists a vector $\tilde{\mathbf{w}} \in \mathbb{Z}_p^N$ such that for all indices $i \in C$, $\mathbf{m}_i^\top \tilde{\mathbf{w}} = 0$, where $\mathbf{m}_i^\top$ denotes the i^{th} row of $\mathbf{M}$, and $\mathbf{e}_1^\top \tilde{\mathbf{w}} = 1$. Algorithm $\mathcal{B}$ samples a vector $\tilde{\mathbf{s}} \xleftarrow{R} \mathbb{Z}_p^W$. Algorithm $\mathcal{B}$ now implicitly sets $\mathbf{s} = \tilde{\mathbf{s}} + ab\cdot\tilde{\mathbf{w}}$. In particular, algorithm $\mathcal{B}$ constructs the $\mathbf{s}$-dependent terms in the CRS as follows:
$$B = e(g,\hat{g})^{\tilde{s}_1} \cdot e(g^a, \hat{g}^b)^{\tilde{w}_1} = e(g,\hat{g})^{s_1}$$
$$\hat{g}^{c^i \mathbf{s}} = (\hat{g}^{c^i})^{\tilde{\mathbf{s}}} \cdot (\hat{g}^{abc^i})^{\tilde{\mathbf{w}}}$$
Algorithm $\mathcal{B}$ sets the common reference string to be
$$\mathsf{crs} = \big(\mathcal{G}, g, \hat{g}, B, \hat{u}, \hat{h}, \{g^{c^i}, \hat{g}^{c^i \mathbf{s}}\}_{i\in[-N,N]\setminus\{0\}}\big).$$

4. Next, to simulate the honest verification keys, algorithm $\mathcal{B}$ starts by sampling $\tilde{\alpha}_\ell \xleftarrow{R} \mathbb{Z}_p$ for each $\ell \in [N] \setminus C$. Then, algorithm $\mathcal{B}$ implicitly sets the secret key for user ℓ to be $\alpha_\ell = \tilde{\alpha}_\ell - ab \cdot \mathbf{m}_\ell^\top \tilde{\mathbf{w}}$. Specifically, algorithm $\mathcal{B}$ constructs the components of the verification key and the aggregation hint as follows:
$$A_\ell = e(g,\hat{g})^{\tilde{\alpha}_\ell} \cdot e(g^a, \hat{g}^b)^{-\mathbf{m}_\ell^\top \tilde{\mathbf{w}}} = e(g,\hat{g})^{\alpha_\ell}$$
$$\hat{v}'_{\ell,i} = (\hat{g}^{c^i})^{\tilde{\alpha}_\ell} \cdot (\hat{g}^{abc^i})^{-\mathbf{m}_\ell^\top \tilde{\mathbf{w}}}$$
Then, algorithm $\mathcal{B}$ sets
$$\mathsf{vk}_\ell = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, A_\ell) \quad \text{and} \quad \mathsf{ht}_\ell = \{\hat{v}'_{\ell,i}\}_{i\in[-N,N]\setminus\{0\}}.$$
Algorithm $\mathcal{B}$ gives crs and $\{(\mathsf{vk}_\ell, \mathsf{ht}_\ell)\}_{\ell\in[N]\setminus C}$ to $\mathcal{A}$.

5. Whenever algorithm $\mathcal{A}$ makes a signing query on an index $\ell \in [N] \setminus C$ and a message $m \neq m^*$, algorithm $\mathcal{B}$ samples $\tilde{r} \xleftarrow{R} \mathbb{Z}_p$ and implicitly sets $r = \tilde{r} + a(m-m^*)^{-1}\mathbf{m}_\ell^\top \tilde{\mathbf{w}}$. Then, it computes
$$\sigma_1 = g^{\tilde{r}} \cdot (g^a)^{(m-m^*)^{-1}\mathbf{m}_\ell^\top \tilde{\mathbf{w}}}$$
$$\hat{\sigma}_1 = \hat{g}^{\tilde{r}} \cdot (\hat{g}^a)^{(m-m^*)^{-1}\mathbf{m}_\ell^\top \tilde{\mathbf{w}}}$$
$$\hat{\sigma}_2 = \hat{g}^{\tilde{\alpha}_\ell} \cdot (\hat{g}^b)^{\tilde{r}(m-m^*)} \cdot \hat{\sigma}_1^{\tilde{u}m+\tilde{h}}.$$
and responds to $\mathcal{A}$ with the signature $\sigma = (\sigma_1, \hat{\sigma}_2)$.

6. After $\mathcal{A}$ is finished making signing queries, it outputs the key-generation randomness $\rho_\ell \in \{0,1\}^*$ used to generate the keys for the corrupted users $\ell \in C$. Algorithm $\mathcal{A}$ also outputs a signature $\sigma_{\text{agg}} = (\sigma_{\text{agg},1}, \hat{\sigma}_{\text{agg},2}, \sigma_{\text{agg},3})$.

7. For each $\ell \in C$, algorithm $\mathcal{B}$ computes $(\mathsf{vk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell) \leftarrow \mathsf{KeyGen}(\mathsf{crs}; \rho_\ell)$. Algorithm $\mathcal{B}$ parses $\mathsf{sk}_\ell = (\mathsf{vk}_\ell, \tilde{\alpha}_\ell)$ for each $\ell \in C$. Finally, algorithm $\mathcal{B}$ outputs $(\tau_0, \hat{\tau}_0, \tau, \ldots, \tau_N)$ where

$$\tau_0 = g^{-\tilde{s}_1} \cdot \sigma_{\mathsf{agg},1}^{\tilde{u}m^*+\tilde{h}} \quad \text{and} \quad \hat{\tau}_0 = \hat{\sigma}_{\mathsf{agg},2}^{-1} \quad \text{and} \quad \forall i \in [N] : \tau_i = \sigma_{\mathsf{agg},3}^{\mathbf{m}_i^\intercal \tilde{\mathbf{s}}+\tilde{\alpha}_i}.$$

First, we argue that algorithm $\mathcal{B}$ correctly simulates the common reference string, the honest verification keys, and the signatures. Consider first the components of the common reference string:

- Algorithm $\mathcal{B}$ samples $\tilde{u}, \tilde{h} \xleftarrow{R} \mathbb{Z}_p$ so the distributions of $\hat{u}, \hat{h}$ are also uniform over $\mathbb{G}$ (and independent of all other components in crs), exactly as in the real scheme.
- Algorithm $\mathcal{B}$ implicitly sets $\mathbf{s} = \tilde{\mathbf{s}} + ab \cdot \tilde{\mathbf{w}}$, where $\tilde{\mathbf{s}} \xleftarrow{R} \mathbb{Z}_p^N$. Thus, the distribution of $\mathbf{s}$ also coincides with its distribution in the real scheme.
- Finally, the challenger samples $c \xleftarrow{R} \mathbb{Z}_p^*$, which matches the distribution in the real scheme.

We conclude that algorithm $\mathcal{B}$ constructs crs according to the same distribution as $\mathsf{Setup}(1^\lambda, 1^\kappa)$. We now consider the honest verification keys and the signatures:

- **Verification keys:** Consider the honest verification keys vk_ℓ for $\ell \in [N] \setminus C$. By construction, the verification keys vk_ℓ and hint components ht_ℓ sampled by algorithm $\mathcal{B}$ coincide with those that would be output by $\mathsf{KeyGen}(\mathsf{crs})$ with $\alpha_\ell = \tilde{\alpha}_\ell - ab \cdot \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}$. Since algorithm $\mathcal{B}$ samples $\tilde{\alpha}_\ell \xleftarrow{R} \mathbb{Z}_p$ for all $\ell \in [N] \setminus C$, the verification keys are also distributed exactly as in the real scheme.
- **Signatures:** Finally, consider the signing queries. Let $\ell \in [N] \setminus C$ be the index and $m \neq m^*$ be the message. We claim that $\sigma = (\sigma_1, \hat{\sigma}_2)$ is a signature with respect to signing key α_ℓ and randomness $r = \tilde{r} + a(m - m^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}$:
  - By construction, algorithm $\mathcal{B}$ sets

    $$\sigma_1 = g^{\tilde{r}} \cdot (g^a)^{(m-m^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} = g^{\tilde{r} + a(m-m^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} = g^r.$$

    By the same calculation, this means $\hat{\sigma}_1 = \hat{g}^r$.
  - In the real scheme, $\hat{\sigma}_2 = \hat{g}^{\alpha_\ell} (\hat{u}^m \hat{h})^r$. Substituting the expressions for α_ℓ, $\hat{u}$, $\hat{h}$, and r, we have

    $$\begin{aligned}
    \hat{g}^{\alpha_\ell} (\hat{u}^m \hat{h})^r &= \hat{g}^{(\tilde{\alpha}_\ell - ab \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}})} (\hat{g}^{bm + \tilde{u}m} \hat{g}^{\tilde{h} - bm^*})^{\tilde{r} + a(m-m^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} \\
    &= \hat{g}^{\tilde{\alpha}_\ell} (\hat{g}^{\tilde{u}m + \tilde{h}})^{\tilde{r} + a(m-m^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} \hat{g}^{-ab \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} \hat{g}^{b(m-m^*)(\tilde{r} + a(m-m^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}})} \\
    &= \hat{g}^{\tilde{\alpha}_\ell} \hat{\sigma}_1^{\tilde{u}m + \tilde{h}} \hat{g}^{b(m-m^*)\tilde{r}}.
    \end{aligned}$$

    This is precisely how algorithm $\mathcal{B}$ constructs the signatures.

  Since algorithm $\mathcal{B}$ samples $\tilde{r} \xleftarrow{R} \mathbb{Z}_p$, the distribution of r is also uniform and the signature is correctly constructed.

Thus, with probability ε, algorithm $\mathcal{A}$ outputs a signature $\sigma_{\mathsf{agg}} = (\sigma_{\mathsf{agg},1}, \hat{\sigma}_{\mathsf{agg},2}, \sigma_{\mathsf{agg},3})$ such that $\mathsf{Verify}(\mathsf{vk}_\varphi, m^*, \sigma_{\mathsf{agg}}) = 1$, where $\mathsf{vk}_\varphi = (\mathcal{G}, g, u, h, B, \hat{z})$ and

$$\hat{z} = \prod_{\ell \in [N]} \hat{g}^{c^\ell \mathbf{m}_\ell^\intercal \mathbf{s}} \hat{v}_{\ell,\ell}.$$

We consider each component in this product separately:

- Suppose $\ell \in C$. In this case, $\hat{v}_{\ell,\ell}$ is output by $\mathsf{KeyGen}(\mathsf{crs}; \rho_\ell)$. Since $\mathsf{sk}_\ell = (\mathsf{vk}_\ell, \tilde{\alpha}_\ell)$, this means $\hat{v}_{\ell,\ell} = \hat{g}^{c^\ell \tilde{\alpha}_\ell}$. Since $\ell \in C$, $\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}} = 0$. This means

$$c^\ell (\mathbf{m}_\ell^\intercal \mathbf{s} + \tilde{\alpha}_\ell) = c^\ell (\mathbf{m}_\ell^\intercal (\tilde{\mathbf{s}} + ab\tilde{\mathbf{w}}) + \tilde{\alpha}_\ell) = c^\ell (\mathbf{m}_\ell^\intercal \tilde{\mathbf{s}} + \tilde{\alpha}_\ell).$$

- Suppose $\ell \in [N] \setminus C$. In this case, $\hat{v}_{\ell,\ell} = \hat{g}^{c^\ell \alpha_\ell} = \hat{g}^{c^\ell(\tilde{\alpha}_\ell - ab\mathbf{m}_\ell^\top \tilde{\mathbf{w}})}$. This means

$$c^\ell(\mathbf{m}_\ell^\top \mathbf{s} + \alpha_\ell) = c^\ell(\mathbf{m}_\ell^\top(\tilde{\mathbf{s}} + ab\tilde{\mathbf{w}}) + \tilde{\alpha}_\ell - ab\mathbf{m}_\ell^\top \tilde{\mathbf{w}}) = c^\ell(\mathbf{m}_\ell^\top \tilde{\mathbf{s}} + \tilde{\alpha}_\ell).$$

We conclude that

$$\hat{z} = \prod_{\ell \in [N]} \hat{g}^{c^\ell \mathbf{m}_\ell^\top \mathbf{s}} \hat{v}_{\ell,\ell} = \prod_{\ell \in [N]} \hat{g}^{c^\ell(\mathbf{m}_\ell^\top \tilde{\mathbf{s}} + \tilde{\alpha}_\ell)}. \tag{B.8}$$

Now, since verification passes, this means Eq. (3.13) holds so

$$e(g, \hat{g})^{s_1} \cdot e(g, \hat{\sigma}_{\mathrm{agg},2}) = e(\sigma_{\mathrm{agg},1}, \hat{u}^m \hat{h}) \cdot e(\sigma_{\mathrm{agg},3}, \hat{z}). \tag{B.9}$$

Recall that $\mathbf{e}_1^\top \tilde{\mathbf{w}} = 1$. This means $s_1 = \mathbf{e}_1^\top \mathbf{s} = \mathbf{e}_1^\top \tilde{\mathbf{s}} + ab\mathbf{e}_1^\top \tilde{\mathbf{w}} = \tilde{s}_1 + ab$. In addition, $\hat{u}^{m^*} \hat{h} = \hat{g}^{am^* + \tilde{u}m^*} \cdot \hat{g}^{\tilde{h}} / \hat{g}^{am^*} = \hat{g}^{\tilde{u}m^* + \tilde{h}}$. Combining with Eqs. (B.8) and (B.9), we thus have

$$e(g, \hat{g})^{\tilde{s}_1 + ab} \cdot e(g, \hat{\sigma}_{\mathrm{agg},2}) = e(\sigma_{\mathrm{agg},1}, \hat{g}^{\tilde{u}m^* + \tilde{h}}) \cdot \prod_{\ell \in [N]} e(\sigma_{\mathrm{agg},3}, \hat{g}^{c^\ell(\mathbf{m}_\ell^\top \tilde{\mathbf{s}} + \tilde{\alpha}_\ell)}).$$

Rearranging and using bilinearity, we have

$$\begin{aligned} e(g, \hat{g})^{ab} &= e\big(g^{-\tilde{s}_1} \cdot \sigma_{\mathrm{agg},1}^{\tilde{u}m^* + \tilde{h}}, \hat{g}\big) \cdot e\big(g, \hat{\sigma}_{\mathrm{agg},2}^{-1}\big) \cdot \prod_{\ell \in [N]} e(\sigma_{\mathrm{agg},3}^{\mathbf{m}_\ell^\top \tilde{\mathbf{s}} + \tilde{\alpha}_\ell}, \hat{g}^{c^\ell}) \\ &= e(\tau_0, \hat{g}) \cdot e(g, \hat{\tau}_0) \cdot \prod_{\ell \in [N]} e(\tau_\ell, \hat{g}^{c^\ell}). \end{aligned}$$

We conclude that $\mathcal{B}$ breaks the search N-extended bilinear Diffie-Hellman exponent assumption with the same advantage ε. □

C Analysis of Construction 4.4 (Monotone-Policy Encryption)

In this section, we give the correctness and security analysis of Construction 4.4. The proofs and analysis follow a similar structure as the corresponding analysis for Construction 3.4.

C.1 Proof of Theorem 4.5 (Partial Decryption Correctness)

Take any $\lambda, \kappa \in \mathbb{N}$ and any policy $\varphi \in \Phi_\kappa$ (with associated policy $\mathbf{M} \in \mathbb{Z}_p^{N \times W}$). Take any crs in the support of $\mathsf{Setup}(1^\lambda, 1^\kappa)$. Then we can write

$$\mathsf{crs} = \big(\mathcal{G}, g, \hat{g}, B, \hat{u}, \hat{h}, \{g^{c^i}, \hat{g}^{c^i \mathbf{s}}\}_{i \in [-N,N] \setminus \{0\}}\big).$$

Take any tag $\tau \in \mathbb{Z}_p$ and any message $m \in \mathbb{G}_T$. Take any collection $\{(\mathsf{pk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell)\}_{\ell \in [N]}$ where $(\mathsf{pk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell)$ is in the support of $\mathsf{KeyGen}(\mathsf{crs})$ for all $\ell \in [N]$. This means

$$\mathsf{pk}_\ell = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, A_\ell) = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, e(g, \hat{g})^{\alpha_\ell})$$

Let $(\mathsf{ek}_\varphi, \mathsf{ak}_\varphi) = \mathsf{Preprocess}(\mathsf{crs}, \varphi, \{(\mathsf{pk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [N]})$ and $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{ek}_\varphi, \tau, m)$. Take any index $i^* \in [N]$ and let $\sigma_{i^*} \leftarrow \mathsf{PartialDec}(\mathsf{sk}_{i^*}, \tau, \mathsf{ct})$. Then

$$\sigma_{i^*} = (\sigma_{i^*,1}, \hat{\sigma}_{i^*,2}) = (g^r, \hat{g}^{\alpha_{i^*}} (\hat{u}^\tau \hat{h})^r).$$

Consider now the relation checked by PartialVerify. By construction,

$$e(g, \hat{\sigma}_{i^*,2}) = e(g, \hat{g}^{\alpha_{i^*}} (\hat{u}^\tau \hat{h})^r) = e(g, \hat{g})^{\alpha_{i^*}} e(g^r, \hat{u}^\tau \hat{h}) = A_{i^*} \cdot e(\sigma_{i^*,1}, \hat{u}^\tau \hat{h}).$$

In this case, $\mathsf{PartialVerify}(\mathsf{pk}_{i^*}, \tau, \mathsf{ct}, \sigma_{i^*}) = 1$. □

C.2 Proof of Theorem 4.6 (Aggregation Correctness)

Take any $\lambda, \kappa \in \mathbb{N}$, any policy $\varphi \in \Phi_\kappa$ (with associated policy $\mathbf{M} \in \mathbb{Z}_p^{N \times W}$), and any set $S \subseteq [N]$ where $\varphi(S) = 1$. Take any crs in the support of $\mathsf{Setup}(1^\lambda, 1^\kappa)$. Then we can write

$$\mathsf{crs} = \left(\mathcal{G}, g, \hat{g}, B, \hat{u}, \hat{h}, \{g^{c^i}, \hat{g}^{c^i \mathbf{s}}\}_{i \in [-N,N] \setminus \{0\}}\right).$$

Take any tag $\tau \in \mathbb{Z}_p$ and any message $m \in \mathbb{G}_T$. Take any collection $\{(\mathsf{pk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell)\}_{\ell \in [N]}$ where $(\mathsf{pk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell)$ is in the support of $\mathsf{KeyGen}(\mathsf{crs})$ for all $\ell \in [N]$. This means

$$\begin{aligned}\mathsf{pk}_\ell &= (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, A_\ell) = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, e(g,\hat{g})^{\alpha_\ell}) \\ \mathsf{ht}_\ell &= \{\hat{v}'_{\ell,i}\}_{i \in [-N,N] \setminus \{0\}}.\end{aligned}$$

By construction of KeyGen, $\hat{v}'_{\ell,i} = \hat{g}^{\alpha_\ell c^i}$ for all $i \in [-N, N] \setminus \{0\}$. Let $(\mathsf{ek}_\varphi, \mathsf{ak}_\varphi) = \mathsf{Preprocess}(\mathsf{crs}, \varphi, \{(\mathsf{pk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [N]})$. Specifically, the preprocessing algorithm computes

$$\hat{z} = \prod_{\ell \in [N]} \hat{g}^{\mathbf{m}_\ell^\top (c^\ell \mathbf{s})} \hat{v}_{\ell,\ell} = \hat{g}^{\tilde{z}} \quad \text{where} \quad \tilde{z} = \sum_{\ell \in [N]} c^\ell (\mathbf{m}_\ell^\top \mathbf{s} + \alpha_\ell).$$

Then, for all $\ell \in [N]$, it computes

$$\hat{v}_\ell = \prod_{\substack{i \in [N] \\ i \neq \ell}} \hat{v}'_{i,i-\ell} \cdot \hat{g}^{c^{i-\ell} \mathbf{m}_i^\top \mathbf{s}} = \prod_{\substack{i \in [N] \\ i \neq \ell}} \hat{g}^{c^{i-\ell}(\mathbf{m}_i^\top \mathbf{s} + \alpha_i)}. \tag{C.1}$$

Finally, it sets $\mathsf{ek}_\varphi = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, B, \hat{z})$ and $\mathsf{ak}_\varphi = (\mathbf{M}, \hat{z}, \{g^{c^{-\ell}}, \hat{v}_\ell\}_{\ell \in [N]})$. Let $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{ek}_\varphi, \tau, m)$. Then

$$\mathsf{ct} = (\tau, C_1, c_2, \hat{c}_3, \hat{c}_4) = (\tau, B^t \cdot m, g^t, (\hat{u}^\tau \hat{h})^t, \hat{z}^t).$$

Take any collection of partial decryptions $\{\sigma_\ell\}_{\ell \in S}$ where for all $\ell \in S$, $\mathsf{PartialVerify}(\mathsf{pk}_\ell, \tau, \mathsf{ct}, \sigma_\ell) = 1$. By construction, if we parse $\sigma_\ell = (\sigma_{\ell,1}, \hat{\sigma}_{\ell,2})$, this means

$$\forall \ell \in S : e(g, \hat{\sigma}_{\ell,2}) = A_\ell \cdot e(\sigma_{\ell,1}, \hat{u}^\tau \hat{h}) = e(g,\hat{g})^{\alpha_\ell} \cdot e(\sigma_{\ell,1}, \hat{u}^\tau \hat{h}). \tag{C.2}$$

Consider now $\mathsf{Decrypt}(\mathsf{crs}, \mathsf{ak}_\varphi, \mathsf{ct}, \{\sigma_\ell\}_{\ell \in S})$. First, the decryption algorithm computes the aggregated decryption components:

$$\sigma_{\mathsf{agg},1} = \prod_{\ell \in S} \sigma_{\ell,1}^{\omega_\ell} \quad \text{and} \quad \hat{\sigma}_{\mathsf{agg},2} = \prod_{\ell \in S} \hat{\sigma}_{\ell,2}^{\omega_\ell} \hat{v}_\ell^{\omega_\ell} \quad \text{and} \quad \sigma_{\mathsf{agg},3} = \prod_{\ell \in S} \left(g^{c^{-\ell}}\right)^{\omega_\ell}. \tag{C.3}$$

By Eq. (C.2) and bilinearity, this means

$$e(\sigma_{\mathsf{agg},1}, \hat{c}_3) = \prod_{\ell \in S} e(\sigma_{\ell,1}, \hat{u}^\tau \hat{h})^{\omega_\ell t} = \prod_{\ell \in S} \frac{e(g, \hat{\sigma}_{\ell,2}^{\omega_\ell t})}{e(g,\hat{g})^{\omega_\ell \alpha_\ell t}} \tag{C.4}$$

Let $\tilde{\sigma}_{\mathsf{agg},3} = \sum_{\ell \in S} \omega_\ell c^{-\ell}$. Then $\sigma_{\mathsf{agg},3} = g^{\tilde{\sigma}_{\mathsf{agg},3}}$. Next, $\boldsymbol{\omega}^\top \mathbf{M} = \mathbf{e}_1^\top$ so $\sum_{\ell \in [N]} \omega_\ell \mathbf{m}_\ell^\top \mathbf{s} = \boldsymbol{\omega}^\top \mathbf{M} \mathbf{s} = \mathbf{e}_1^\top \mathbf{s} = s_1$. Combined with the fact that $\omega_\ell = 0$ for all $\ell \notin S$, we have

$$\begin{aligned}\tilde{z} \cdot \tilde{\sigma}_{\mathsf{agg},3} &= \sum_{\ell \in [N]} \sum_{i \in [N]} c^i (\mathbf{m}_i^\top \mathbf{s} + \alpha_i) \cdot \omega_\ell c^{-\ell} \\ &= \sum_{\ell \in [N]} \omega_\ell \mathbf{m}_\ell^\top \mathbf{s} + \sum_{\ell \in S} \omega_\ell \alpha_\ell + \sum_{\ell \in S} \sum_{\substack{i \in [N] \\ i \neq \ell}} \left(\omega_\ell (\mathbf{m}_i^\top \mathbf{s} + \alpha_i) \cdot c^{i-\ell}\right) \\ &= s_1 + \sum_{\ell \in S} \omega_\ell \alpha_\ell + \sum_{\ell \in S} \sum_{\substack{i \in [N] \\ i \neq \ell}} \left(\omega_\ell (\mathbf{m}_i^\top \mathbf{s} + \alpha_i) c^{i-\ell}\right)\end{aligned}$$

Combined with Eq. (C.1), we thus have

$$\begin{aligned} e(\sigma_{\mathrm{agg},3}, \hat{c}_4) = e(\sigma_{\mathrm{agg},3}, \hat{z}^t) &= e(g,\hat{g})^{\tilde{z}\tilde{\sigma}_{\mathrm{agg},3}t} \\ &= e(g,\hat{g})^{s_1 t} \cdot \prod_{\ell \in S} e(g,\hat{g})^{\omega_\ell \alpha_\ell t} \cdot \prod_{\ell \in S} e(g,\hat{v}_\ell)^{\omega_\ell t} \end{aligned} \tag{C.5}$$

Combining Eqs. (C.3) to (C.5), we have

$$\begin{aligned} e(g,\hat{g})^{s_1 t} \cdot e(c_2, \hat{\sigma}_{\mathrm{agg},2}) &= e(g,\hat{g})^{s_1 t} \cdot e(g^t, \hat{\sigma}_{\mathrm{agg},2}) \\ &= e(g,\hat{g})^{s_1 t} \cdot \prod_{\ell \in S} e(g, \hat{\sigma}_{\ell,2}^{\omega_\ell t}) \cdot \prod_{\ell \in S} e(g, \hat{v}_\ell^{\omega_\ell t}) \\ &= e(\sigma_{\mathrm{agg},1}, \hat{c}_3) \cdot e(\sigma_{\mathrm{agg},3}, \hat{c}_4). \end{aligned}$$

Finally, since $B = e(g,\hat{g})^{s_1}$, we have

$$C_1 \cdot e(\sigma_{\mathrm{agg},1}, \hat{c}_3)^{-1} \cdot e(c_2, \hat{\sigma}_{\mathrm{agg},2}) \cdot e(\sigma_{\mathrm{agg},3}, \hat{c}_4)^{-1} = m \cdot \frac{e(g,\hat{g})^{s_1 t} \cdot e(c_2, \hat{\sigma}_{\mathrm{agg},2})}{e(\sigma_{\mathrm{agg},1}, \hat{c}_3) \cdot e(\sigma_{\mathrm{agg},3}, \hat{c}_4)} = m,$$

and decryption succeeds. □

C.3 Proof of Theorem 4.7 (Static Tag-Based CCA-Security)

We begin by defining a sequence of hybrid experiments. Each experiment is parameterized by a bit $b \in \{0,1\}$, and implicitly, by an adversary $\mathcal{A}$ (outputting policies with N users) and a security parameter λ.

- $\mathsf{Hyb}_0^{(b)}$: This is real tag-based CCA-security experiment with bit $b \in \{0,1\}$. For completeness, we recall the game below:

  1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the policy parameter 1^κ together with a policy $\varphi \in \Phi_\kappa$ on N users. In addition, algorithm $\mathcal{A}$ commits to a set of corrupted users $C \subseteq [N]$.
  2. The challenger checks that $\varphi(C) = 0$. If not the challenger halts with output $b' = 0$.
  3. The challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa)$. Namely, it samples $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$, generators $g, \hat{g} \xleftarrow{R} \mathbb{G} \setminus \{1\}$, and $\hat{u}, \hat{h} \xleftarrow{R} \mathbb{G}$, $c \xleftarrow{R} \mathbb{Z}_p^*$, and $\mathbf{s} \xleftarrow{R} \mathbb{Z}_p^W$. It sets $B = e(g,\hat{g})^{s_1}$ and

     $$\mathsf{crs} = \big(\mathcal{G}, g, \hat{g}, B, \hat{u}, \hat{h}, \{g^{c^i}, \hat{g}^{c^i \mathbf{s}}\}_{i \in [-N,N]\setminus\{0\}}\big).$$

     Then, for each index $\ell \in [N] \setminus C$, the challenger samples a key $(\mathsf{pk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell) \leftarrow \mathsf{KeyGen}(\mathsf{crs})$. Specifically, it samples $\alpha_\ell \xleftarrow{R} \mathbb{Z}_p$ and computes $\hat{v}'_{\ell,i} = (\hat{g}^{c^i})^{\alpha_\ell}$ for all $i \in [-N,N] \setminus \{0\}$. The algorithm sets $A_\ell = e(g,g)^{\alpha_\ell}$ and

     $$\mathsf{pk}_\ell = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, A_\ell) \quad \text{and} \quad \mathsf{ht}_\ell = \{\hat{v}'_{\ell,i}\}_{i \in [-N,N]\setminus\{0\}}.$$

     The challenger gives crs together with $\{(\mathsf{pk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [N]\setminus C}$ to $\mathcal{A}$.
  4. Algorithm $\mathcal{A}$ now specifies the key-generation randomness $\rho_\ell \in \{0,1\}^*$ used to generate the keys for each of the corrupted users $\ell \in C$.
  5. For each $\ell \in C$, the challenger computes $(\mathsf{pk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell) \leftarrow \mathsf{KeyGen}(\mathsf{crs}; \rho_\ell)$. The challenger responds with $(\mathsf{ek}_\varphi, \mathsf{ak}_\varphi) = \mathsf{Preprocess}(\mathsf{crs}, \varphi, \{(\mathsf{pk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [N]})$.
  6. Whenever $\mathcal{A}$ makes a partial decryption query on an index $\ell \in [N] \setminus C$, a tag $\tau \in \mathbb{Z}_p$, and a ciphertext ct, the challenger responds with $\mathsf{PartialDec}(\mathsf{sk}_\ell, \tau, \mathsf{ct})$. Specifically, the challenger sample $r \xleftarrow{R} \mathbb{Z}_p$ and responds with $\sigma = (g^r, \hat{g}^{\alpha_\ell}(\hat{u}^\tau \hat{h})^r)$.

7. After $\mathcal{A}$ is finished making partial decryption queries, it outputs a pair of messages $m_0, m_1 \in \mathbb{G}_T$. The challenger samples a random tag $\tau^* \xleftarrow{R} \mathbb{Z}_p$ and responds with $\mathsf{ct}^* \leftarrow \mathsf{Encrypt}(\mathsf{ek}_\varphi, \tau^*, m_b)$. Specifically, the challenger samples $t \xleftarrow{R} \mathbb{Z}_p$ and responds with the tag τ^* and the challenge ciphertext

$$\mathsf{ct}^* = (\tau^*, C_1^*, c_2^*, \hat{c}_3^*, \hat{c}_4^*) = \left(\tau^*,\ B^t \cdot m_b,\ g^t,\ (\hat{u}^{\tau^*}\hat{h})^t,\ \hat{z}^t\right),$$

where

$$\hat{z} = \prod_{\ell \in [N]} \hat{g}^{\mathbf{m}_\ell^\intercal (c^\ell \mathbf{s})} \hat{v}'_{\ell,\ell}$$

and $\mathsf{ht}_\ell = \{\hat{v}'_{\ell,i}\}_{i \in [-N,N]\setminus\{0\}}$.

8. Algorithm $\mathcal{A}$ can continue making partial decryption queries, except it is *not* allowed to make a decryption query with tag $\tau = \tau^*$. The challenger responds to the queries exactly as before. At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

- $\mathsf{Hyb}_1^{(b)}$: Same as $\mathsf{Hyb}_0^{(b)}$ except the challenger samples the challenge tag $\tau^* \xleftarrow{R} \mathbb{Z}_p$ at the beginning of the experiment. Then, if the adversary makes a partial decryption query on τ^* *before* it chooses the challenge messages m_0, m_1, the challenger outputs 0.
- $\mathsf{Hyb}_2^{(b)}$: Same as $\mathsf{Hyb}_1^{(b)}$ except when constructing the challenge ciphertext, the challenger samples $C_1^* \xleftarrow{R} \mathbb{G}_T$. Notably, in this experiment, the adversary's view is independent of the bit $b \in \{0,1\}$.

For an adversary $\mathcal{A}$, we write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the output distribution of an execution of experiment Hyb_i with adversary $\mathcal{A}$ (and the implicit security parameter λ). We now argue that each adjacent pair of experiments are indistinguishable.

Lemma C.1. *For all adversaries $\mathcal{A}$ making at most $Q = Q(\lambda)$ partial decryption queries and all bits $b \in \{0,1\}$, we have that*

$$\left|\Pr[\mathsf{Hyb}_0^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1]\right| \le Q/p.$$

Proof. Take $b \in \{0,1\}$. By definition, the only difference between the two experiments is if in $\mathsf{Hyb}_1^{(b)}$, algorithm $\mathcal{A}$ makes a partial decryption query on the challenge tag τ^* *before* choosing the challenge messages. However, the view of adversary $\mathcal{A}$ in $\mathsf{Hyb}_1^{(b)}$ is independent of τ^* prior to the challenge phase. Since the challenger samples $\tau^* \xleftarrow{R} \mathbb{Z}_p$ and the adversary makes at most Q partial decryption queries, the probability that $\mathcal{A}$ queries on τ^* is at most Q/p. □

Lemma C.2. *Suppose the decisional N-extended bilinear Diffie-Hellman exponent assumption (Assumption 3.12) holds with respect to* GroupGen. *Then, for all efficient adversaries $\mathcal{A}$ and all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. Suppose there exists $b \in \{0,1\}$ and an efficient adversary $\mathcal{A}$ where

$$\left|\Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1]\right| \ge \varepsilon$$

for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for the N-extended bilinear Diffie-Hellman exponent assumption. The proof follows a nearly identical structure as the proof of Theorem 3.18, just adapted to the setting of monotone-policy encryption. As in previous proofs, we will use a tilde (e.g., $\tilde{u}, \tilde{h}$) to denote exponents sampled by the reduction algorithm $\mathcal{B}$. Algorithm $\mathcal{B}$ works as follows:

1. On input the challenge

$$\left(1^\lambda, \mathcal{G}, g, \hat{g}, g^a, \hat{g}^a, \hat{g}^b, g^t, \hat{g}^t, \{g^{c^i}, \hat{g}^{c^i}, \hat{g}^{abc^i}\}_{i \in [-N,N]\setminus\{0\}}, \{\hat{g}^{tc^i}\}_{i \in [N]}, T\right),$$

where $T = e(g,\hat{g})^{abt}$ or $T \xleftarrow{R} \mathbb{G}_T$, algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ on input 1^λ. Algorithm $\mathcal{A}$ outputs the policy parameter 1^κ together with a policy $\varphi \in \Phi_\kappa$ (over N users). In addition, algorithm $\mathcal{A}$ commits to a set of corrupted users $C \subseteq [N]$.

2. Algorithm $\mathcal{B}$ checks that $\varphi(C) = 0$. If not the challenger halts with output $b' = 0$.

3. Algorithm $\mathcal{B}$ now samples $\tau^* \xleftarrow{R} \mathbb{Z}_p$. It constructs the components of the common reference string as follows. First, it samples $\tilde{u}, \tilde{h} \xleftarrow{R} \mathbb{Z}_p$ and sets

$$\hat{u} = (\hat{g}^b) \cdot \hat{g}^{\tilde{u}} \quad \text{and} \quad \hat{h} = \hat{g}^{\tilde{h}} / (\hat{g}^b)^{\tau^*}.$$

Since the set $C \subseteq [N]$ does not satisfy the policy $\mathbf{M}$, there exists a vector $\tilde{\mathbf{w}} \in \mathbb{Z}_p^N$ such that for all indices $i \in C$, $\mathbf{m}_i^\top \tilde{\mathbf{w}} = 0$, where $\mathbf{m}_i^\top$ denotes the i^{th} row of $\mathbf{M}$, and $\mathbf{e}_1^\top \tilde{\mathbf{w}} = 1$. Algorithm $\mathcal{B}$ samples a vector $\tilde{\mathbf{s}} \xleftarrow{R} \mathbb{Z}_p^W$. Algorithm $\mathcal{B}$ now implicitly sets $\mathbf{s} = \tilde{\mathbf{s}} + ab \cdot \tilde{\mathbf{w}}$. In particular, algorithm $\mathcal{B}$ constructs the $\mathbf{s}$-dependent terms in the CRS as follows:

$$B = e(g, \hat{g})^{\tilde{s}_1} \cdot e(g^a, \hat{g}^b)^{\tilde{w}_1} = e(g, \hat{g})^{s_1}$$
$$\hat{g}^{c^i \mathbf{s}} = (\hat{g}^{c^i})^{\tilde{\mathbf{s}}} \cdot (\hat{g}^{abc^i})^{\tilde{\mathbf{w}}}$$

Algorithm $\mathcal{B}$ sets the common reference string to be

$$\mathsf{crs} = \left(\mathcal{G}, g, \hat{g}, B, \hat{u}, \hat{h}, \{g^{c^i}, \hat{g}^{c^i \mathbf{s}}\}_{i \in [-N,N] \setminus \{0\}}\right).$$

4. Next, to simulate the honest public keys, algorithm $\mathcal{B}$ starts by sampling $\tilde{\alpha}_\ell \xleftarrow{R} \mathbb{Z}_p$ for each $\ell \in [N] \setminus C$. Then, algorithm $\mathcal{B}$ implicitly sets the secret key for user ℓ to be $\alpha_\ell = \tilde{\alpha}_\ell - ab \cdot \mathbf{m}_\ell^\top \tilde{\mathbf{w}}$. Specifically, algorithm $\mathcal{B}$ constructs the components of the public key and the aggregation hint as follows:

$$A_\ell = e(g, \hat{g})^{\tilde{\alpha}_\ell} \cdot e(g^a, \hat{g}^b)^{-\mathbf{m}_\ell^\top \tilde{\mathbf{w}}} = e(g, \hat{g})^{\alpha_\ell}$$
$$\hat{v}'_{\ell,i} = (\hat{g}^{c^i})^{\tilde{\alpha}_\ell} \cdot (\hat{g}^{abc^i})^{-\mathbf{m}_\ell^\top \tilde{\mathbf{w}}}$$

Then, algorithm $\mathcal{B}$ sets

$$\mathsf{pk}_\ell = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, A_\ell) \quad \text{and} \quad \mathsf{ht}_\ell = \{\hat{v}'_{\ell,i}\}_{i \in [-N,N] \setminus \{0\}}.$$

Algorithm $\mathcal{B}$ gives crs and $\{(\mathsf{pk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [N] \setminus C}$ to $\mathcal{A}$.

5. Algorithm $\mathcal{A}$ now specifies the key-generation randomness $\rho_\ell \in \{0,1\}^*$ used to generate the keys for each of the corrupted users $\ell \in C$.

6. For each $\ell \in C$, algorithm $\mathcal{B}$ computes $(\mathsf{pk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell) \leftarrow \mathsf{KeyGen}(\mathsf{crs}; \rho_\ell)$. It parses $\mathsf{sk}_\ell = (\mathsf{pk}_\ell, \tilde{\alpha}_\ell)$ for some $\tilde{\alpha}_\ell \in \mathbb{Z}_p$. Then, algorithm $\mathcal{B}$ responds with $(\mathsf{ek}_\varphi, \mathsf{ak}_\varphi) = \mathsf{Preprocess}(\mathsf{crs}, \varphi, \{(\mathsf{pk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [N]})$.

7. Whenever algorithm $\mathcal{A}$ makes a partial decryption query on an index $\ell \in [N] \setminus C$, a tag τ, and a ciphertext ct, algorithm $\mathcal{B}$ first checks if $\tau = \tau^*$. If so, then algorithm $\mathcal{B}$ halts with output 0. Otherwise, algorithm $\mathcal{B}$ samples $\tilde{r} \xleftarrow{R} \mathbb{Z}_p$ and implicitly sets $r = \tilde{r} + a(\tau - \tau^*)^{-1} \mathbf{m}_\ell^\top \tilde{\mathbf{w}}$. Then, it computes

$$\sigma_1 = g^{\tilde{r}} \cdot (g^a)^{(\tau - \tau^*)^{-1} \mathbf{m}_\ell^\top \tilde{\mathbf{w}}}$$
$$\hat{\sigma}_1 = \hat{g}^{\tilde{r}} \cdot (\hat{g}^a)^{(\tau - \tau^*)^{-1} \mathbf{m}_\ell^\top \tilde{\mathbf{w}}}$$
$$\hat{\sigma}_2 = \hat{g}^{\tilde{\alpha}_\ell} \cdot (\hat{g}^b)^{\tilde{r}(\tau - \tau^*)} \cdot \hat{\sigma}_1^{\tilde{u}\tau + \tilde{h}}.$$

and responds to $\mathcal{A}$ with the partial decryption $\sigma = (\sigma_1, \hat{\sigma}_2)$.

8. After $\mathcal{A}$ is finished making partial decryption queries, it outputs a pair of messages $m_0, m_1 \in \mathbb{G}_T$. Algorithm $\mathcal{B}$ replies to $\mathcal{A}$ with the tag τ^* and the challenge ciphertext

$$\mathsf{ct}^* = (\tau^*, C_1^*, c_2^*, \hat{c}_3^*, \hat{c}_4^*) = \left(\tau^*,\ T \cdot e(g, \hat{g}^t)^{\tilde{s}_1} \cdot m_b,\ g^t,\ (\hat{g}^t)^{\tilde{u}\tau^* + \tilde{h}},\ \prod_{\ell \in [N]} (\hat{g}^{tc^\ell})^{\mathbf{m}_\ell^\top \tilde{\mathbf{s}} + \tilde{\alpha}_\ell}\right).$$

9. Algorithm $\mathcal{A}$ can continue to make partial decryption queries on tags $\tau \neq \tau^*$. Algorithm $\mathcal{B}$ responds exactly as described above.

10. At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b \in \{0,1\}$, which algorithm $\mathcal{B}$ also outputs.

We first argue that algorithm $\mathcal{B}$ correctly simulates the common reference string, the honest public keys, and the partial decryption queries for $\mathcal{A}$. The analysis is nearly identical to that in the proof of Theorem 3.18. Consider the components of the common reference string:

- Algorithm $\mathcal{B}$ samples $\tilde{u}, \tilde{h} \xleftarrow{R} \mathbb{Z}_p$ so the distributions of $\hat{u}, \hat{h}$ are also uniform over $\mathbb{G}$ (and independent of all other components in crs), exactly as in $\mathsf{Hyb}_1^{(b)}$ and $\mathsf{Hyb}_2^{(b)}$.
- Algorithm $\mathcal{B}$ implicitly sets $\mathbf{s} = \tilde{\mathbf{s}} + ab \cdot \tilde{\mathbf{w}}$, where $\tilde{\mathbf{s}} \xleftarrow{R} \mathbb{Z}_p^N$. Thus, the distribution of $\mathbf{s}$ also coincides with its distribution in the real scheme.
- Finally, the challenger samples $c \xleftarrow{R} \mathbb{Z}_p^*$, which matches the distribution in the real scheme.

We conclude that algorithm $\mathcal{B}$ constructs crs according to the distribution in $\mathsf{Hyb}_1^{(b)}$ and $\mathsf{Hyb}_2^{(b)}$. We now consider the public keys and the partial decryption queries.

- **Public keys:** Consider the honest public keys pk_ℓ for $\ell \in [N] \setminus C$. By construction, the public keys pk_ℓ and hints ht_ℓ sampled by algorithm $\mathcal{B}$ coincide with those that would be output by $\mathsf{KeyGen}(\mathsf{crs})$ with $\alpha_\ell = \tilde{\alpha}_\ell - ab \cdot \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}$. Since algorithm $\mathcal{B}$ samples $\tilde{\alpha}_\ell \xleftarrow{R} \mathbb{Z}_p$ for all $\ell \in [N] \setminus C$, the public keys are also distributed exactly as in the real scheme.
- **Partial decryption queries:** Next, consider the partial decryption queries on tags $\tau \neq \tau^*$. Note that if $\mathcal{A}$ ever makes a partial decryption query on tag $\tau = \tau^*$, then algorithm $\mathcal{B}$ outputs 0, which matches the behavior in $\mathsf{Hyb}_1^{(b)}$ and $\mathsf{Hyb}_2^{(b)}$. Let $\ell \in [N] \setminus C$ be the index and $\tau \in \mathbb{Z}_p$ be the tag associated with the partial decryption query. We claim that $\sigma = (\sigma_1, \hat{\sigma}_2)$ is a valid partial decryption with respect to signing key α_ℓ and randomness $r = \tilde{r} + a(\tau - \tau^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}$:
  - By construction, algorithm $\mathcal{B}$ sets
  $$\sigma_1 = g^{\tilde{r}} \cdot (g^a)^{(\tau-\tau^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} = g^{\tilde{r} + a(\tau-\tau^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} = g^r.$$
  By the same calculation, this means $\hat{\sigma}_1 = \hat{g}^r$.
  - In $\mathsf{Hyb}_1^{(b)}$ and $\mathsf{Hyb}_2^{(b)}$, the challenger would normally set $\hat{\sigma}_2 = \hat{g}^{\alpha_\ell} (\hat{u}^\tau \hat{h})^r$. Substituting the expressions for α_ℓ, $\hat{u}$, $\hat{h}$, and r, we have
  $$\begin{aligned}
  \hat{g}^{\alpha_\ell} (\hat{u}^\tau \hat{h})^r &= \hat{g}^{(\tilde{\alpha}_\ell - ab \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}})} (\hat{g}^{b\tau + \tilde{u}\tau} \hat{g}^{\tilde{h} - b\tau^*})^{\tilde{r} + a(\tau-\tau^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} \\
  &= \hat{g}^{\tilde{\alpha}_\ell} (\hat{g}^{\tilde{u}\tau + \tilde{h}})^{\tilde{r} + a(\tau-\tau^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} \hat{g}^{-ab \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} \hat{g}^{b(\tau-\tau^*)(\tilde{r} + a(\tau-\tau^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}})} \\
  &= \hat{g}^{\tilde{\alpha}_\ell} \hat{\sigma}_1^{\tilde{u}\tau + \tilde{h}} \hat{g}^{b(\tau-\tau^*)\tilde{r}}.
  \end{aligned}$$
  This is precisely how algorithm $\mathcal{B}$ constructs the partial decryptions.

  Since algorithm $\mathcal{B}$ samples $\tilde{r} \xleftarrow{R} \mathbb{Z}_p$, the distribution of r is also uniform and the partial decryption is correctly constructed.

Algorithm $\mathcal{B}$ constructs $(\mathsf{ek}_\varphi, \mathsf{ak}_\varphi)$ using the same procedure as in $\mathsf{Hyb}_1^{(b)}$ and $\mathsf{Hyb}_2^{(b)}$. It suffices to consider the components of the challenge ciphertext. We claim that ct^* is an encryption of m_b with randomness t according to either the specification of $\mathsf{Hyb}_1^{(b)}$ or $\mathsf{Hyb}_2^{(b)}$. We consider each component individually:

- Consider C_1^*. Algorithm $\mathcal{B}$ sets $C_1^* = T \cdot e(g, \hat{g}^t)^{\tilde{s}_1}$. Recall that $\mathbf{e}_1^\intercal \tilde{\mathbf{w}} = 1$. This means
  $$s_1 = \mathbf{e}_1^\intercal \mathbf{s} = \mathbf{e}_1^\intercal \tilde{\mathbf{s}} + ab \mathbf{e}_1^\intercal \tilde{\mathbf{w}} = \tilde{s}_1 + ab.$$
  We now consider the two possibilities for the challenge element T:

– Suppose $T = e(g,\hat{g})^{abt}$. Then

$$C_1^* = T \cdot e(g,\hat{g}^t)^{\tilde{s}_1} \cdot m_b = e(g,\hat{g})^{t(ab+\tilde{s}_1)} \cdot m_b = e(g,\hat{g})^{s_1 t} \cdot m_b = B^t \cdot m_b.$$

This is the distribution of C_1^* in $\mathsf{Hyb}_1^{(b)}$.

– Suppose $T \xleftarrow{R} \mathbb{G}_T$. Then C_1 is also distributed uniformly over $\mathbb{G}_T$. This is the distribution of C_1^* in $\mathsf{Hyb}_2^{(b)}$.

- Consider c_2^*. Algorithm $\mathcal{B}$ sets $c_2^* = g^t$, which matches the behavior in $\mathsf{Hyb}_1^{(b)}$ and $\mathsf{Hyb}_2^{(b)}$.
- Consider $\hat{c}_3^*$. Algorithm $\mathcal{B}$ sets $c_3^* = (\hat{g}^t)^{\tilde{u}\tau^*+\tilde{h}} = (\hat{u}^{\tau^*}\hat{h})^t$ since $\hat{u}^{\tau^*}\hat{h} = \hat{g}^{b\tau^*+\tilde{u}\tau^*+\tilde{h}-b\tau^*} = \hat{g}^{\tilde{u}\tau^*+\tilde{h}}$. This matches the distribution in $\mathsf{Hyb}_1^{(b)}$ and $\mathsf{Hyb}_2^{(b)}$.
- Consider $\hat{c}_4^*$. In the reduction, algorithm $\mathcal{B}$ sets

$$\hat{c}_4^* = \prod_{\ell\in[N]} (\hat{g}^{tc^\ell})^{\mathbf{m}_\ell^\top \tilde{\mathbf{s}}+\tilde{\alpha}_\ell}.$$

We show that $\hat{c}_4^* = \hat{z}^t$ where

$$\hat{z} = \prod_{\ell\in[N]} \hat{g}^{c^\ell \mathbf{m}_\ell^\top \mathbf{s}} \hat{v}'_{\ell,\ell}.$$

Then this would coincide with the distribution of $\hat{c}_4^*$ in $\mathsf{Hyb}_1^{(b)}$ and $\mathsf{Hyb}_2^{(b)}$. We consider each component in $\hat{z}$ separately:

– Suppose $\ell \in C$. In this case, $\hat{v}_{\ell,\ell}$ is output by $\mathsf{KeyGen}(\mathsf{crs};\rho_\ell)$. Since $\mathsf{sk}_\ell = (\mathsf{pk}_\ell, \tilde{\alpha}_\ell)$, this means $\hat{v}_{\ell,\ell} = \hat{g}^{c^\ell\tilde{\alpha}_\ell}$. Since $\ell \in C$, $\mathbf{m}_\ell^\top \tilde{\mathbf{w}} = 0$. This means

$$c^\ell(\mathbf{m}_\ell^\top \mathbf{s} + \tilde{\alpha}_\ell) = c^\ell(\mathbf{m}_\ell^\top(\tilde{\mathbf{s}} + ab\tilde{\mathbf{w}}) + \tilde{\alpha}_\ell) = c^\ell(\mathbf{m}_\ell^\top \tilde{\mathbf{s}} + \tilde{\alpha}_\ell).$$

– Suppose $\ell \in [N] \setminus C$. In this case, $\hat{v}_{\ell,\ell} = \hat{g}^{c^\ell\alpha_\ell} = \hat{g}^{c^\ell(\tilde{\alpha}_\ell - ab\mathbf{m}_\ell^\top\tilde{\mathbf{w}})}$. This means

$$c^\ell(\mathbf{m}_\ell^\top \mathbf{s} + \alpha_\ell) = c^\ell(\mathbf{m}_\ell^\top(\tilde{\mathbf{s}} + ab\tilde{\mathbf{w}}) + \tilde{\alpha}_\ell - ab\mathbf{m}_\ell^\top\tilde{\mathbf{w}}) = c^\ell(\mathbf{m}_\ell^\top\tilde{\mathbf{s}} + \tilde{\alpha}_\ell).$$

Thus, we conclude that

$$\hat{z} = \prod_{\ell\in[N]} \hat{g}^{c^\ell \mathbf{m}_\ell^\top \mathbf{s}} \hat{v}'_{\ell,\ell} = \prod_{\ell\in[N]} \hat{g}^{c^\ell(\mathbf{m}_\ell^\top\tilde{\mathbf{s}}+\tilde{\alpha}_\ell)},$$

and correspondingly, $\hat{c}_4^* = \hat{z}^t$, as required.

We conclude that if $T = e(g,\hat{g})^{abt}$, then algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_1^{(b)}$ for $\mathcal{A}$. If $T \xleftarrow{R} \mathbb{G}_T$, then algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_2^{(b)}$ for $\mathcal{A}$. We conclude that $\mathcal{B}$ breaks the decisional N-extended bilinear Diffie-Hellman exponent assumption with the same advantage ε. □

Lemma C.3. *For all adversaries $\mathcal{A}$, $\Pr[\mathsf{Hyb}_2^{(0)}(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_2^{(1)}(\mathcal{A}) = 1]$.*

Proof. These are identical distributions by definition so the claim holds. □

Since $\mathcal{A}$ is computationally-bounded, it makes at most $Q = \mathsf{poly}(\lambda)$ partial decryption queries in the security game. Since $p > 2^\lambda$, this means $Q/p = \mathsf{negl}(\lambda)$. Theorem 4.7 now follows by combining Lemmas C.1 to C.3 and a standard hybrid argument. □

D Analysis of Construction 5.2 (Threshold Signature)

In this section, we give the correctness and security analysis of Construction 5.2. The proofs and analysis follow a similar structure as the corresponding analysis for Construction 3.15 in Appendix B.

D.1 Proof of Theorem 5.3 (Signing Correctness)

The structure of the individual signatures in Construction 5.2 is identical to that of Construction 3.15. Signing correctness follows by an analogous argument as that in the proof of Theorem 3.16. □

D.2 Proof of Theorem 5.4 (Aggregation Correctness)

This proof follows the same structure as the proof of Theorem 3.17. Specifically, take any $\lambda \in \mathbb{N}$ and $N = 2^n \leq 2^\lambda < 2^{\lambda+1} \leq p/2$, any number $L \leq N$, any message $m \in \mathbb{Z}_p$, any crs in the support of $\mathsf{Setup}(1^\lambda, 1^\kappa)$, any $\{(\mathsf{vk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell)\}_{\ell\in[L]}$ where $(\mathsf{vk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell)$ is in the support of $\mathsf{KeyGen}(\mathsf{crs})$ for all $\ell \in [L]$, any non-empty collection of signatures $\{\sigma_\ell\}_{\ell\in S}$ where $\mathsf{PartialVerify}(\mathsf{crs}, m, \sigma_\ell) = 1$ for all $\ell \in S \subseteq [L]$, and any threshold $T \leq |S|$. For each $j \in [n]$, let $X_j = [N + 2^{j-1}, N + 2^j - 1]$. By construction, we can now write

$$\begin{aligned}
\mathsf{crs} &= \left(\mathcal{G}, g, \hat{g}, B, \hat{u}, \hat{h}, \{g^{c^i}\}_{i\in I\setminus\{0\}}, \hat{z}_0, \{\hat{v}_{\ell,0}\}_{\ell\in[2N-1]}, \{\hat{y}_j\}_{j\in[n]}, \{\hat{\tau}_{j,i}\}_{j\in[n],i\in I\setminus X_j}\right)\\
\mathsf{vk}_\ell &= (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, A_\ell)\\
\mathsf{ht}_\ell &= \{\hat{v}'_{\ell,i}\}_{i\in I\setminus\{0\}}\\
\sigma_\ell &= (\sigma_{\ell,1}, \hat{\sigma}_{\ell,2}),
\end{aligned}$$

where $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, e)$,

$$B = e(g,\hat{g})^{s_1}\,, \quad \hat{z}_0 = \prod_{\ell\in[2N-1]} \hat{g}^{c^\ell \mathbf{m}_\ell^\intercal \mathbf{s}}\,, \quad \hat{v}_{\ell,0} = \prod_{\substack{i\in[2N-1]\\ i\neq\ell}} \hat{g}^{c^{\ell-i}\mathbf{m}_i^\intercal \mathbf{s}}\,, \quad \hat{y}_j = \prod_{k\in X_j} \hat{g}^{c^k\gamma_k}\,, \quad \hat{\tau}_{j,i} = \prod_{k\in X_j} \hat{g}^{c^{k-i}\gamma_k}\,,$$

and $\mathbf{M} \in \mathbb{Z}_q^{2N\times N}$ is the share-generation matrix for an N-out-of-$(2N-1)$ threshold policy (Eq. (2.1)). Next, by construction of KeyGen,

$$A_\ell = e(g,\hat{g})^{\alpha_\ell} \quad \text{and} \quad \hat{v}'_{\ell,i} = \hat{g}^{\alpha_\ell c^i}.$$

Since $\mathsf{PartialVerify}(\mathsf{crs}, m, \sigma_\ell) = 1$ for all $\ell \in S$, this means

$$\forall \ell \in S : e(g, \hat{\sigma}_{\ell,2}) = A_\ell \cdot e(\sigma_{\ell,1}, \hat{u}^m \hat{h}) = e(g,\hat{g})^{\alpha_\ell} \cdot e(\sigma_{\ell,1}, \hat{u}^m\hat{h}). \tag{D.1}$$

Let

$$\begin{aligned}
(\mathsf{vk}_{\mathsf{agg}}, \mathsf{ak}) &= \mathsf{Preprocess}(\mathsf{crs}, \mathbf{M}, \{(\mathsf{vk}_\ell, \mathsf{ht}_\ell)\}_{\ell\in[N]})\\
\sigma_{\mathsf{agg}} = (\sigma_{\mathsf{agg},1}, \hat{\sigma}_{\mathsf{agg},2}, \sigma_{\mathsf{agg},3}, |S|) &= \mathsf{Aggregate}(\mathsf{ak}, \{\sigma_\ell\}_{\ell\in S}).
\end{aligned}$$

By construction, the preprocessing algorithm computes

$$\hat{z} = \hat{z}_0 \cdot \prod_{\ell\in[L]} \hat{v}'_{\ell,\ell} = \prod_{\ell\in[2N-1]} \hat{g}^{c^\ell \mathbf{m}_\ell^\intercal \mathbf{s}} \cdot \prod_{\ell\in[L]} \hat{g}^{c^\ell\alpha_\ell} = \hat{g}^{\tilde{z}} \quad \text{where} \quad \tilde{z} = \sum_{\ell\in[2N-1]} c^\ell \mathbf{m}_\ell^\intercal \mathbf{s} + \sum_{\ell\in[L]} c^\ell\alpha_\ell.$$

Next, the preprocessing algorithm computes $\hat{v}_\ell$ according to Eq. (5.2). This means

$$\hat{v}_\ell = \hat{v}_{\ell,0} \cdot \prod_{\substack{i\in[L]\\ i\neq\ell}} \hat{v}'_{i,i-\ell} = \prod_{\substack{i\in[2N-1]\\ i\neq\ell}} \hat{g}^{c^{i-\ell}\mathbf{m}_i^\intercal \mathbf{s}} \prod_{\substack{i\in[L]\\ i\neq\ell}} \hat{g}^{c^{i-\ell}\alpha_i}. \tag{D.2}$$

Then, it sets

$$\begin{aligned}
\mathsf{vk}_{\mathsf{agg}} &= (L, \mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, B, \hat{z}, \{\hat{y}_j\}_{j\in[n]})\\
\mathsf{ak} &= \left(L, \{g^{c^{-\ell}}, \hat{v}_\ell\}_{\ell\in[L]}, \{\hat{\tau}_{j,i}\}_{j\in[n],i\in I\setminus X_j}\right).
\end{aligned}$$

Let $b_n \cdots b_1$ be the binary representation of $L - |S|$. The aggregation algorithm defines $S_{\text{pad}} = S \cup [L+1, N] \cup \bigcup_{j:b_j=1} X_j$ and computes

$$\begin{aligned}
\sigma_{\text{agg},1} &= \prod_{\ell \in S} \sigma_{\ell,1}^{\omega_\ell} \\
\hat{\sigma}_{\text{agg},2} &= \prod_{\ell \in S} \hat{\sigma}_{\ell,2}^{\omega_\ell} \cdot \prod_{\ell \in S_{\text{pad}}} \hat{v}_\ell^{\omega_\ell} \cdot \prod_{\substack{j \in [n] \\ b_j = 0}} \prod_{\ell \in S_{\text{pad}}} \hat{\tau}_{j,\ell}^{\omega_\ell} \\
\sigma_{\text{agg},3} &= \prod_{\ell \in S_{\text{pad}}} \left(g^{c^{-\ell}}\right)^{\omega_\ell}.
\end{aligned} \tag{D.3}$$

As in the proof of Theorem 3.17, by Eq. (D.1) and bilinearity, this means

$$e(\sigma_{\text{agg},1}, \hat{u}^m \hat{h}) = \prod_{\ell \in S} e(\sigma_{\ell,1}, \hat{u}^m \hat{h})^{\omega_\ell} = \prod_{\ell \in S} \frac{e(g, \hat{\sigma}_{\ell,2}^{\omega_\ell})}{e(g, \hat{g})^{\omega_\ell \alpha_\ell}} \tag{D.4}$$

Let $\tilde{\sigma}_{\text{agg},3} = \sum_{\ell \in S_{\text{pad}}} \omega_\ell c^{-\ell}$. Then $\sigma_{\text{agg},3} = g^{\tilde{\sigma}_{\text{agg},3}}$. Next, $\boldsymbol{\omega}^\intercal \mathbf{M} = \mathbf{e}_1^\intercal$ so

$$\sum_{\ell \in [2N-1]} \omega_\ell \mathbf{m}_\ell^\intercal \mathbf{s} = \boldsymbol{\omega}^\intercal \mathbf{M} \mathbf{s} = \mathbf{e}_1^\intercal \mathbf{s} = s_1.$$

Combined with the fact that $\omega_\ell = 0$ for all $\ell \notin S_{\text{pad}}$, we have

$$\begin{aligned}
\tilde{z} \cdot \tilde{\sigma}_{\text{agg},3} &= \sum_{\ell \in S_{\text{pad}}} \sum_{i \in [2N-1]} c^i \mathbf{m}_i^\intercal \mathbf{s} \cdot \omega_\ell c^{-\ell} + \sum_{\ell \in S_{\text{pad}}} \sum_{i \in [L]} c^i \alpha_i \cdot \omega_\ell c^{-\ell} \\
&= \sum_{\ell \in S_{\text{pad}}} \omega_\ell \mathbf{m}_\ell^\intercal \mathbf{s} + \sum_{\ell \in S_{\text{pad}} \cap [L]} \omega_\ell \alpha_\ell + \sum_{\ell \in S_{\text{pad}}} \sum_{\substack{i \in [2N-1] \\ i \neq \ell}} \omega_\ell c^{i-\ell} \mathbf{m}_i^\intercal \mathbf{s} + \sum_{\ell \in S_{\text{pad}}} \sum_{\substack{i \in [L] \\ i \neq \ell}} \omega_\ell \alpha_i c^{i-\ell} \\
&= s_1 + \sum_{\ell \in S} \omega_\ell \alpha_\ell + \sum_{\ell \in S_{\text{pad}}} \omega_\ell \left(\sum_{\substack{i \in [2N-1] \\ i \neq \ell}} c^{i-\ell} \mathbf{m}_i^\intercal \mathbf{s} + \sum_{\substack{i \in [L] \\ i \neq \ell}} c^{i-\ell} \alpha_i \right)
\end{aligned}$$

Using the definition of $\hat{v}_\ell$ from Eq. (D.2), we have

$$\begin{aligned}
e(\sigma_{\text{agg},3}, \hat{z}) &= e(g, \hat{g})^{\tilde{\sigma}_{\text{agg},3} \tilde{z}} \\
&= e(g, \hat{g})^{s_1} \cdot \prod_{\ell \in S} e(g, \hat{g})^{\omega_\ell \alpha_\ell} \cdot \prod_{\ell \in S_{\text{pad}}} \left(\prod_{\substack{i \in [2N-1] \\ i \neq \ell}} e(g, \hat{g}^{c^{i-\ell} \mathbf{m}_i^\intercal \mathbf{s}}) \prod_{\substack{i \in [L] \\ i \neq \ell}} e(g, \hat{g}^{c^{i-\ell} \alpha_i}) \right)^{\omega_\ell} \\
&= e(g, \hat{g})^{s_1} \cdot \prod_{\ell \in S} e(g, \hat{g})^{\omega_\ell \alpha_\ell} \cdot \prod_{\ell \in S_{\text{pad}}} e(g, \hat{v}_\ell^{\omega_\ell}).
\end{aligned} \tag{D.5}$$

Next, let $\tilde{y}_j = \sum_{k \in X_j} c^k \gamma_k$. In particular, $\hat{y}_j = \hat{g}^{\tilde{y}_j}$. Then,

$$\tilde{y}_j \cdot \tilde{\sigma}_{\text{agg},3} = \sum_{\ell \in S_{\text{pad}}} \sum_{k \in X_j} c^k \gamma_k \cdot \omega_\ell c^{-\ell} = \sum_{\ell \in S_{\text{pad}}} \sum_{k \in X_j} \omega_\ell \gamma_k c^{k-\ell}.$$

By construction, S_{pad} contains X_j where $b_j = 1$. Moreover, the sets $X_1, \ldots, X_n$ are pairwise disjoint, so for all $j \in [n]$ where $b_j = 0$, $S_{\text{pad}} \cap X_j = \varnothing$. Thus,

$$\forall j \in [n] \text{ where } b_j = 0: \quad e(\sigma_{\text{agg},3}, \hat{y}_j) = e(g, \hat{g})^{\tilde{y}_j \tilde{\sigma}_{\text{agg},3}} = \prod_{\ell \in S_{\text{pad}}} e(g, \hat{\tau}_{j,\ell}^{\omega_\ell}). \tag{D.6}$$

Combining Eqs. (D.3) to (D.6), we have

$$\begin{aligned}
&e(g,\hat{g})^{s_1} \cdot e(g,\hat{\sigma}_{\mathsf{agg},2}) \\
&\quad = e(g,\hat{g})^{s_1} \cdot \prod_{\ell\in S} e(g,\hat{\sigma}_{\ell,2}^{\omega_\ell}) \cdot \prod_{\ell\in S_{\mathsf{pad}}} e(g,\hat{v}_\ell^{\omega_\ell}) \cdot \prod_{\substack{j\in[n]\\ b_j=0}} \prod_{\ell\in S_{\mathsf{pad}}} e(g,\hat{\tau}_{j,\ell}^{\omega_\ell}) \\
&\quad = e(\sigma_{\mathsf{agg},3},\hat{z}) \cdot \prod_{\substack{j\in[n]\\ b_j=0}} e(\sigma_{\mathsf{agg},3},\hat{y}_j) \cdot \prod_{\ell\in S} \frac{e(g,\hat{\sigma}_{\ell,2}^{\omega_\ell})}{e(g,\hat{g})^{\omega_\ell\alpha_\ell}} \\
&\quad = e(\sigma_{\mathsf{agg},1},\hat{u}^m\hat{h}) \cdot e(\sigma_{\mathsf{agg},3},\hat{z}) \cdot \prod_{\substack{j\in[n]\\ b_j=0}} e(\sigma_{\mathsf{agg},3},\hat{y}_j).
\end{aligned}$$

By assumption, $|S| \geq T$, so we conclude that $\mathsf{Verify}(\mathsf{vk}_{\mathsf{agg}}, m, \sigma_{\mathsf{agg}}, T) = 1$, as required. □

D.3 Proof of Theorem 5.5 (Static Unforgeability)

We begin by defining two hybrid experiments. Each experiment is implicitly parameterized by an adversary $\mathcal{A}$ and a security parameter λ.

- Hyb_0: This is the static unforgeability security game for the threshold signature scheme (from Definition 5.1).
- Hyb_1: Same as Hyb_0, except at the beginning of the security game, after the adversary commits to the quorum size L, the challenger samples a random threshold $T^* \xleftarrow{R} [|C|+1, L]$. At the end of the game, after the adversary outputs its forgery $\sigma_{\mathsf{agg}} = (\sigma_{\mathsf{agg},1}, \hat{\sigma}_{\mathsf{agg},2}, \sigma_{\mathsf{agg},3}, K)$, the challenger outputs 0 if $K \neq T^*$.

For an adversary $\mathcal{A}$, we write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the output of an execution of Hyb_i with adversary $\mathcal{A}$. We now analyze the output distributions of the two hybrids.

Lemma D.1. *For all efficient adversaries $\mathcal{A}$, there exists a polynomial $Q = Q(\lambda)$ such that* $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] = \frac{1}{Q} \cdot \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]$.

Proof. By construction, the adversary's view of the two experiments is identically distributed. Suppose $\mathsf{Hyb}_0(\mathcal{A})$ outputs 1. This means that the adversary $\mathcal{A}$ outputs a threshold K where $K \in [|C|+1, L]$. The challenger's output in Hyb_1 is 1 if and only if $\mathsf{Hyb}_0(\mathcal{A}) = 1$ and $T^* = K$. Since the challenger samples $T^* \xleftarrow{R} [|C|+1, L]$ and T^* is independent of the view of the adversary, the event $T^* = K$ occurs with probability $1/(L - |C|)$. Thus, $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]/(L - |C|)$. Since $\mathcal{A}$ is efficient, both L and $|C|$ are polynomially-bounded, and the claim follows. □

Lemma D.2. *Suppose the $(2N-1)$-extended bilinear Diffie-Hellman exponent assumption (Assumption 3.12) holds with respect to* GroupGen. *Then, for all efficient adversaries $\mathcal{A}$ (outputting quorums of size N), there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$, we have* $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] = \mathsf{negl}(\lambda)$.

Proof. Similar to the proof of Theorem 3.18, we will use the search $(2N-1)$-extended bilinear Diffie-Hellman exponent assumption from Lemma B.1. Suppose there exists an efficient adversary $\mathcal{A}$ where $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] \geq \varepsilon$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for the search $(2N-1)$-extended bilinear Diffie-Hellman assumption from Lemma B.1. As in the proof of Theorem 3.18, we use a tilde (e.g., $\tilde{u}, \tilde{h}$) to denote exponents sampled by the reduction algorithm $\mathcal{B}$. Algorithm $\mathcal{B}$ works as follows:

1. Let $I = [-2N+1, 2N-1]$. On input the challenge

$$\left(1^\lambda, \mathcal{G}, g, \hat{g}, g^a, \hat{g}^a, \hat{g}^b, \{g^{c^i}, \hat{g}^{c^i}, \hat{g}^{abc^i}\}_{i\in I\setminus\{0\}}\right),$$

   algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ on 1^λ. Algorithm $\mathcal{A}$ outputs the quorum size $L \leq N$, the indices of the corrupted users $C \subseteq [N]$ and a challenge message $m^* \in \mathbb{Z}_p$.

2. Algorithm $\mathcal{B}$ starts by sampling the target threshold $T^* \xleftarrow{R} [|C|+1, L]$. Let $b_n^* \cdots b_1^*$ be the binary representation of $L - T^*$ (namely, $L - T^* = \sum_{j\in[n]} b_j^* 2^{j-1}$). Let $S^*_{\mathsf{pad}} = C \cup [L+1, N] \cup \bigcup_{j:b_j^*=1} X_j$ where $X_j = [N+2^{j-1}, N+2^j-1]$. By construction,

$$\left|S^*_{\mathsf{pad}}\right| = |C| + (N-L) + \sum_{j\in[n]:b_j^*=1} 2^{j-1} = |C| + (N-L) + (L-T^*) = N + (|C| - T^*) < N,$$

since $T^* > |C|$. Let $\mathbf{M} \in \mathbb{Z}_p^{(2N-1)\times N}$ be the share-generating matrix for the N-out-of-$(2N-1)$ threshold policy from Eq. (2.1). Since $|S^*_{\mathsf{pad}}| < N$, the set S^*_{pad} does not satisfy the threshold policy defined by $\mathbf{M}$. Thus, there exists a vector $\tilde{\mathbf{w}} \in \mathbb{Z}_p^N$ such that for all indices $i \in S^*_{\mathsf{pad}}$, $\mathbf{m}_i^\intercal \tilde{\mathbf{w}} = 0$, where $\mathbf{m}_i^\intercal$ denotes the i^{th} row of $\mathbf{M}$, and $\mathbf{e}_1^\intercal \tilde{\mathbf{w}} = 1$.

3. Algorithm $\mathcal{B}$ constructs the common reference string crs as follows. First, it samples $\tilde{u}, \tilde{h} \xleftarrow{R} \mathbb{Z}_p$ and sets

$$\hat{u} = (\hat{g}^b) \cdot \hat{g}^{\tilde{u}} \quad \text{and} \quad \hat{h} = \hat{g}^{\tilde{h}} / (\hat{g}^b)^{m^*}.$$

Algorithm $\mathcal{B}$ samples a vector $\tilde{\mathbf{s}} \xleftarrow{R} \mathbb{Z}_p^N$. Algorithm $\mathcal{B}$ implicitly sets $\mathbf{s} = \tilde{\mathbf{s}} + ab \cdot \tilde{\mathbf{w}}$ and computes

$$B = e(g,\hat{g})^{\tilde{s}_1} \cdot e(g^a, \hat{g}^b)^{\tilde{w}_1} = e(g,\hat{g})^{s_1}.$$

Next, algorithm $\mathcal{B}$ computes

$$\hat{z}_0 = \prod_{\ell\in[2N-1]} \left((\hat{g}^{c^\ell})^{\mathbf{m}_\ell^\intercal \tilde{\mathbf{s}}} \cdot (\hat{g}^{c^\ell ab})^{\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} \right) = \prod_{\ell\in[2N-1]} \hat{g}^{c^\ell \mathbf{m}_\ell^\intercal \mathbf{s}}$$
$$\forall \ell \in [2N-1] : \hat{v}_{\ell,0} = \prod_{\substack{i\in[2N-1] \\ i\neq\ell}} \left((\hat{g}^{c^{i-\ell}})^{\mathbf{m}_i^\intercal \tilde{\mathbf{s}}} \cdot (\hat{g}^{c^{i-\ell} ab})^{\mathbf{m}_i^\intercal \tilde{\mathbf{w}}} \right) = \prod_{\substack{i\in[2N-1] \\ i\neq\ell}} \hat{g}^{c^{i-\ell} \mathbf{m}_i^\intercal \mathbf{s}}.$$

Finally, for each $\ell \in [N+1, 2N-1]$, algorithm $\mathcal{B}$ samples $\tilde{\gamma}_\ell \xleftarrow{R} \mathbb{Z}_p$ and implicitly sets

$$\gamma_\ell = \begin{cases} \tilde{\gamma}_\ell & \ell \in X_j \text{ for some } j \in [n] \text{ where } b_j^* = 1 \\ \tilde{\gamma}_\ell - ab \cdot \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}} & \ell \in X_j \text{ for some } j \in [n] \text{ where } b_j^* = 0. \end{cases}$$

Specifically, algorithm $\mathcal{B}$ constructs $\hat{y}_j$ and $\hat{\tau}_{j,i}$ for $j \in [n]$ and $i \in I \setminus X_j$ as follows:

- If $b_j^* = 1$, then algorithm $\mathcal{B}$ sets

$$\hat{y}_j = \prod_{k\in X_j} (\hat{g}^{c^k})^{\tilde{\gamma}_k} = \prod_{k\in X_j} \hat{g}^{c^k \gamma_k} \quad \text{and} \quad \hat{\tau}_{j,i} = \prod_{k\in X_j} (\hat{g}^{c^{k-i}})^{\tilde{\gamma}_k} = \prod_{k\in X_j} \hat{g}^{c^{k-i}\gamma_k}.$$

- If $b_j^* = 0$, then algorithm $\mathcal{B}$ sets

$$\hat{y}_j = \prod_{k\in X_j} \left((\hat{g}^{c^k})^{\tilde{\gamma}_k} \cdot (\hat{g}^{c^k ab})^{-\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} \right) = \prod_{k\in X_j} \hat{g}^{c^k \gamma_k}$$
$$\hat{\tau}_{j,i} = \prod_{k\in X_j} \left((\hat{g}^{c^{k-i}})^{\tilde{\gamma}_k} \cdot (\hat{g}^{c^{k-i} ab})^{-\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} \right) = \prod_{k\in X_j} \hat{g}^{c^{k-i}\gamma_k}.$$

Algorithm $\mathcal{B}$ sets the common reference string to be

$$\mathsf{crs} = \left(\mathcal{G}, g, \hat{g}, B, \hat{u}, \hat{h}, \{g^{c^i}\}_{i\in I\setminus\{0\}}, \hat{z}_0, \{\hat{v}_{\ell,0}\}_{\ell\in[2N-1]}, \{\hat{y}_j\}_{j\in[n]}, \{\hat{\tau}_{j,i}\}_{j\in[n], i\in I\setminus X_j} \right).$$

4. Next, to simulate the honest verification keys, algorithm $\mathcal{B}$ starts by sampling $\tilde{\alpha}_\ell \xleftarrow{R} \mathbb{Z}_p$ for each $\ell \in [L] \setminus C$. Then, algorithm $\mathcal{B}$ implicitly sets the secret key for user ℓ to be $\alpha_\ell = \tilde{\alpha}_\ell - ab \cdot \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}$. Specifically, algorithm $\mathcal{B}$ constructs the components of the verification key and the aggregation hint as follows:

$$A_\ell = e(g, \hat{g})^{\tilde{\alpha}_\ell} \cdot e(g^a, \hat{g}^b)^{-\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} = e(g, \hat{g})^{\alpha_\ell}$$
$$\hat{v}'_{\ell,i} = (\hat{g}^{c^i})^{\tilde{\alpha}_\ell} \cdot (\hat{g}^{abc^i})^{-\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} = (\hat{g}^{c^i \alpha_\ell}).$$

Then, algorithm $\mathcal{B}$ sets

$$\mathsf{vk}_\ell = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, A_\ell) \quad \text{and} \quad \mathsf{ht}_\ell = \{\hat{v}'_{\ell,i}\}_{i \in I \setminus \{0\}}.$$

Algorithm $\mathcal{B}$ gives crs and $\{(\mathsf{vk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [L] \setminus C}$ to $\mathcal{A}$.

5. Whenever algorithm $\mathcal{A}$ makes a signing query on an index $\ell \in [L] \setminus C$ and a message $m \neq m^*$, algorithm $\mathcal{B}$ samples $\tilde{r} \xleftarrow{R} \mathbb{Z}_p$ and implicitly sets $r = \tilde{r} + a(m - m^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}$. Then, it computes

$$\sigma_1 = g^{\tilde{r}} \cdot (g^a)^{(m-m^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}}$$
$$\hat{\sigma}_1 = \hat{g}^{\tilde{r}} \cdot (\hat{g}^a)^{(m-m^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}}$$
$$\hat{\sigma}_2 = \hat{g}^{\tilde{\alpha}_\ell} \cdot (\hat{g}^b)^{\tilde{r}(m-m^*)} \cdot \hat{\sigma}_1^{\tilde{u}m + \tilde{h}}.$$

and responds to $\mathcal{A}$ with the signature $\sigma = (\sigma_1, \hat{\sigma}_2)$.

6. After $\mathcal{A}$ is finished making signing queries, it outputs the key-generation randomness $\rho_\ell \in \{0,1\}^*$ used to generate the keys for each of the corrupted users $\ell \in C$. Algorithm $\mathcal{A}$ also outputs a signature $\sigma_{\mathsf{agg}} = (\sigma_{\mathsf{agg},1}, \hat{\sigma}_{\mathsf{agg},2}, \sigma_{\mathsf{agg},3}, K)$.

7. Algorithm $\mathcal{B}$ checks if $K = T^*$. If not, algorithm $\mathcal{B}$ halts with output 0.

8. Otherwise, for each $\ell \in C$, algorithm $\mathcal{B}$ computes $(\mathsf{vk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell) \leftarrow \mathsf{KeyGen}(\mathsf{crs}; \rho_\ell)$. Algorithm $\mathcal{B}$ parses $\mathsf{sk}_\ell = (\mathsf{vk}_\ell, \tilde{\alpha}_\ell)$ for some $\tilde{\alpha}_\ell \in \mathbb{Z}_p$. Finally, for all $\ell \in [2N-1]$, algorithm $\mathcal{B}$ defines the exponent

$$\tilde{\zeta}_\ell = \begin{cases} \tilde{\alpha}_\ell & \ell \in [L] \\ 0 & \ell \in [L+1, N] \cup \bigcup_{j \in [n] : b_j^* = 1} X_j \\ \tilde{\gamma}_\ell & \ell \in \bigcup_{j \in [n] : b_j^* = 0} X_j. \end{cases} \tag{D.7}$$

Then, algorithm $\mathcal{B}$ outputs $(\tau_0, \hat{\tau}_0, \tau_1, \ldots, \tau_{2N-1})$ where

$$\tau_0 = g^{-\tilde{s}_1} \cdot \sigma_{\mathsf{agg},1}^{\tilde{u}m^* + \tilde{h}} \quad \text{and} \quad \hat{\tau}_0 = \hat{\sigma}_{\mathsf{agg},2}^{-1} \quad \text{and} \quad \forall \ell \in [2N-1] : \tau_\ell = \sigma_{\mathsf{agg},3}^{\mathbf{m}_\ell^\intercal \tilde{\mathbf{s}} + \tilde{\zeta}_\ell}.$$

Similar to the proof of Theorem 3.18, we first argue that algorithm $\mathcal{B}$ correctly simulates the common reference string, the honest verification keys, and the signatures according to the specification of the real scheme (which coincides with the distribution in Hyb_1). Consider first the components of the common reference string:

- Algorithm $\mathcal{B}$ samples $\tilde{u}, \tilde{h} \xleftarrow{R} \mathbb{Z}_p$ so the distributions of $\hat{u}, \hat{h}$ are also uniform over $\mathbb{G}$ (and independent of all other components in crs), exactly as in the real scheme.
- Algorithm $\mathcal{B}$ implicitly sets $\mathbf{s} = \tilde{\mathbf{s}} + ab \cdot \tilde{\mathbf{w}}$, where $\tilde{\mathbf{s}} \xleftarrow{R} \mathbb{Z}_p^N$. Thus, the distribution of $\mathbf{s}$ also coincides with its distribution in the real scheme. Correspondingly, the elements B, $\hat{z}_0$, and $\hat{v}_{\ell,0}$ are distributed exactly as in the real scheme.
- For $\ell \in [N+1, 2N-1]$, algorithm $\mathcal{B}$ either sets $\gamma_\ell = \tilde{\gamma}_\ell$ or $\gamma_\ell = \tilde{\gamma}_\ell - ab\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}$, where $\tilde{\gamma}_\ell \xleftarrow{R} \mathbb{Z}_p$. This coincides with the distribution of γ_ℓ in the real scheme. Hence, we conclude that the elements $\hat{y}_j$ and $\hat{\tau}_{j,i}$ are distributed exactly as in the real scheme.
- Finally, the challenger samples $c \xleftarrow{R} \mathbb{Z}_p^*$, which matches the distribution in the real scheme.

We conclude that algorithm $\mathcal{B}$ constructs crs according to the same distribution as $\mathsf{Setup}(1^\lambda, 1^\kappa)$. We now consider the honest verification keys and the signatures:

- **Verification keys:** Consider the honest verification keys vk_ℓ for $\ell \in [L] \setminus C$. By construction, the verification keys vk_ℓ and hint components ht_ℓ sampled by algorithm $\mathcal{B}$ coincide with those that would be output by $\mathsf{KeyGen}(\mathsf{crs})$ with $\alpha_\ell = \tilde{\alpha}_\ell - ab \cdot \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}$. Since algorithm $\mathcal{B}$ samples $\tilde{\alpha}_\ell \xleftarrow{R} \mathbb{Z}_p$ for all $\ell \in [L] \setminus C$, the verification keys are also distributed exactly as in the real scheme.
- **Signatures:** Next, consider the signing queries. Let $\ell \in [L] \setminus C$ be the index and $m \neq m^*$ be the message. We claim that $\sigma = (\sigma_1, \hat{\sigma}_2)$ is a signature with respect to signing key α_ℓ and randomness $r = \tilde{r} + a(m - m^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}$:
  - By construction, algorithm $\mathcal{B}$ sets
  $$\sigma_1 = g^{\tilde{r}} \cdot (g^a)^{(m-m^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} = g^{\tilde{r} + a(m-m^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} = g^r.$$
  By the same calculation, this means $\hat{\sigma}_1 = \hat{g}^r$.
  - In the real scheme, $\hat{\sigma}_2 = \hat{g}^{\alpha_\ell} (\hat{u}^m \hat{h})^r$. Substituting the expressions for α_ℓ, $\hat{u}$, $\hat{h}$, and r, we have
  $$\begin{aligned}
  \hat{g}^{\alpha_\ell} (\hat{u}^m \hat{h})^r &= \hat{g}^{(\tilde{\alpha}_\ell - ab \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}})} (\hat{g}^{bm + \tilde{u}m} \hat{g}^{\tilde{h} - bm^*})^{\tilde{r} + a(m-m^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} \\
  &= \hat{g}^{\tilde{\alpha}_\ell} (\hat{g}^{\tilde{u}m + \tilde{h}})^{\tilde{r} + a(m-m^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} \hat{g}^{-ab \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} \hat{g}^{b(m-m^*)(\tilde{r} + a(m-m^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}})} \\
  &= \hat{g}^{\tilde{\alpha}_\ell} \hat{\sigma}_1^{\tilde{u}m + \tilde{h}} \hat{g}^{b(m-m^*)\tilde{r}}.
  \end{aligned}$$
  This is precisely how algorithm $\mathcal{B}$ constructs the signatures.

  Since algorithm $\mathcal{B}$ samples $\tilde{r} \xleftarrow{R} \mathbb{Z}_p$, the distribution of r is also uniform and the signature is correctly constructed.

Thus, with probability ε, algorithm $\mathcal{A}$ outputs a forgery $\sigma_{\mathsf{agg}} = (\sigma_{\mathsf{agg},1}, \hat{\sigma}_{\mathsf{agg},2}, \sigma_{\mathsf{agg},3}, K)$ and a threshold $T > |C|$ such that $T = K$ and $\mathsf{Verify}(\mathsf{vk}_{\mathsf{agg}}, m^*, \sigma_{\mathsf{agg}}, T) = 1$, where $\mathsf{vk}_{\mathsf{agg}} = (L, \mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, B, \hat{z}, \{\hat{y}_j\}_{j \in [n]})$ and

$$\hat{z} = \hat{z}_0 \cdot \prod_{\ell \in [L]} \hat{v}'_{\ell,\ell} = \prod_{\ell \in [2N-1]} \hat{g}^{c^\ell \mathbf{m}_\ell^\intercal \mathbf{s}} \cdot \prod_{\ell \in [L]} \hat{v}'_{\ell,\ell}.$$

This means

$$\hat{z} \cdot \prod_{\substack{j \in [n] \\ b_j^* = 0}} \hat{y}_j = \prod_{\ell \in [2N-1]} \hat{g}^{c^\ell \mathbf{m}_\ell^\intercal \mathbf{s}} \cdot \prod_{\ell \in [L]} \hat{v}'_{\ell,\ell} \cdot \prod_{\substack{j \in [n] \\ b_j^* = 0}} \prod_{k \in X_j} \hat{g}^{c^k \gamma_k}. \tag{D.8}$$

We group the components of this product based on the terms that depend on c^ℓ for all $\ell \in [2N-1]$.

- Suppose $\ell \in C$. In this case, $\hat{v}'_{\ell,\ell}$ is output by $\mathsf{KeyGen}(\mathsf{crs}; \rho_\ell)$. Since $\mathsf{sk}_\ell = (\mathsf{vk}_\ell, \tilde{\alpha}_\ell)$, this means $\hat{v}'_{\ell,\ell} = \hat{g}^{c^\ell \tilde{\alpha}_\ell}$. Since $\ell \in C \subseteq S^*_{\mathsf{pad}}$, we have $\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}} = 0$. This means
$$c^\ell (\mathbf{m}_\ell^\intercal \mathbf{s} + \tilde{\alpha}_\ell) = c^\ell (\mathbf{m}_\ell^\intercal (\tilde{\mathbf{s}} + ab\tilde{\mathbf{w}}) + \tilde{\alpha}_\ell) = c^\ell (\mathbf{m}_\ell^\intercal \tilde{\mathbf{s}} + \tilde{\alpha}_\ell).$$
- Suppose $\ell \in [L] \setminus C$. In this case, $\hat{v}'_{\ell,\ell} = \hat{g}^{c^\ell \alpha_\ell} = \hat{g}^{c^\ell (\tilde{\alpha}_\ell - ab \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}})}$. This means
$$c^\ell (\mathbf{m}_\ell^\intercal \mathbf{s} + \alpha_\ell) = c^\ell (\mathbf{m}_\ell^\intercal (\tilde{\mathbf{s}} + ab\tilde{\mathbf{w}}) + \tilde{\alpha}_\ell - ab \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}) = c^\ell (\mathbf{m}_\ell^\intercal \tilde{\mathbf{s}} + \tilde{\alpha}_\ell).$$
- Suppose $\ell \in [L+1, N]$. Then, $\ell \in S^*_{\mathsf{pad}}$, so as in the first case, $\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}} = \mathbf{0}$. This means
$$c^\ell \mathbf{m}_\ell^\intercal \mathbf{s} = c^\ell (\mathbf{m}_\ell^\intercal (\tilde{\mathbf{s}} + ab\tilde{\mathbf{w}})) = c^\ell \mathbf{m}_\ell^\intercal \tilde{\mathbf{s}}.$$
- Suppose $\ell \in X_j$ for some $j \in [n]$ where $b_j^* = 1$. Then, $\ell \in S^*_{\mathsf{pad}}$, so as in the previous case, $\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}} = \mathbf{0}$. This means
$$c^\ell \mathbf{m}_\ell^\intercal \mathbf{s} = c^\ell (\mathbf{m}_\ell^\intercal (\tilde{\mathbf{s}} + ab\tilde{\mathbf{w}})) = c^\ell \mathbf{m}_\ell^\intercal \tilde{\mathbf{s}}.$$

- Suppose $\ell \in X_j$ for some $j \in [n]$ where $b_j^* = 0$. In this case $\gamma_\ell = \tilde{\gamma}_\ell - ab \cdot \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}$. This means

$$c^\ell(\mathbf{m}_\ell^\intercal \mathbf{s} + \gamma_\ell) = c^\ell(\mathbf{m}_\ell^\intercal(\tilde{\mathbf{s}} + ab\tilde{\mathbf{w}}) + \tilde{\gamma}_\ell - ab\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}) = c^\ell(\mathbf{m}_\ell^\intercal \tilde{\mathbf{s}} + \tilde{\gamma}_\ell).$$

Taken altogether, we have that

$$\hat{z} \cdot \prod_{\substack{j \in [n] \\ b_j^* = 0}} \hat{y}_j = \prod_{\ell \in [2N-1]} \hat{g}^{c^\ell(\mathbf{m}_\ell^\intercal \tilde{\mathbf{s}} + \tilde{\zeta}_\ell)}, \tag{D.9}$$

where the exponents $\tilde{\zeta}_\ell$ are defined in Eq. (D.7). Now, since verification passes, this means Eq. (5.3) holds so

$$e(g, \hat{g})^{s_1} \cdot e(g, \hat{\sigma}_{\mathsf{agg},2}) = e(\sigma_{\mathsf{agg},1}, \hat{u}^{m^*} \hat{h}) \cdot e(\sigma_{\mathsf{agg},3}, \hat{z}) \cdot \prod_{\substack{j \in [n] \\ b_j^* = 0}} e(\sigma_{\mathsf{agg},3}, \hat{y}_j). \tag{D.10}$$

Recall that $\mathbf{e}_1^\intercal \tilde{\mathbf{w}} = 1$. This means $s_1 = \mathbf{e}_1^\intercal \mathbf{s} = \mathbf{e}_1^\intercal \tilde{\mathbf{s}} + ab\mathbf{e}_1^\intercal \tilde{\mathbf{w}} = \tilde{s}_1 + ab$. In addition, $\hat{u}^{m^*} \hat{h} = \hat{g}^{am^* + \tilde{u}m^*} \cdot \hat{g}^{\tilde{h}} / \hat{g}^{am^*} = \hat{g}^{\tilde{u}m^* + \tilde{h}}$. Combining with Eqs. (D.9) and (D.10), we have

$$e(g, \hat{g})^{\tilde{s}_1 + ab} \cdot e(g, \hat{\sigma}_{\mathsf{agg},2}) = e(\sigma_{\mathsf{agg},1}, \hat{g}^{\tilde{u}m^* + \tilde{h}}) \cdot \prod_{\ell \in [2N-1]} e(\sigma_{\mathsf{agg},3}, \hat{g}^{c^\ell(\mathbf{m}_\ell^\intercal \tilde{\mathbf{s}} + \tilde{\zeta}_\ell)})$$

Rearranging and using bilinearity, we have

$$\begin{aligned} e(g, \hat{g})^{ab} &= e\big(g^{-\tilde{s}_1} \cdot \sigma_{\mathsf{agg},1}^{\tilde{u}m^* + \tilde{h}}, \hat{g}\big) \cdot e\big(g, \hat{\sigma}_{\mathsf{agg},2}^{-1}\big) \cdot \prod_{\ell \in [2N-1]} e(\sigma_{\mathsf{agg},3}^{\mathbf{m}_\ell^\intercal \tilde{\mathbf{s}} + \tilde{\zeta}_\ell}, \hat{g}^{c^\ell}) \\ &= e(\tau_0, \hat{g}) \cdot e(g, \hat{\tau}_0) \cdot \prod_{\ell \in [2N-1]} e(\tau_\ell, \hat{g}^{c^\ell}). \end{aligned}$$

We conclude that $\mathcal{B}$ breaks the search $(2N-1)$-extended bilinear Diffie-Hellman exponent assumption with the same advantage ε and the claim holds. □

Combining Lemmas D.1 and D.2, we have that for all efficient adversaries $\mathcal{A}$,

$$\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] \leq Q(\lambda) \cdot \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] = \mathsf{negl}(\lambda),$$

since $Q = Q(\lambda)$ is polynomially-bounded. Theorem 5.5 follows. □

E Analysis of Construction 5.11 (Threshold Encryption)

In this section, we give the correctness and security analysis of Construction 5.11. The proofs and analysis follow a similar structure as the corresponding analysis for Construction 5.2 in Appendix D.

E.1 Proof of Theorem 5.12 (Partial Decryption Correctness)

This proof proceeds similarly to the proof of Theorem 4.5. Take any $\lambda \in \mathbb{N}$, any $N \leq 2^\lambda$, and any crs in the support of $\mathsf{Setup}(1^\lambda, 1^N)$. Then we can write

$$\mathsf{crs} = \big(\mathcal{G}, g, \hat{g}, B, \hat{u}, \hat{h}, \{\hat{g}^{c^i}\}_{i \in I \setminus \{0\}}, \hat{z}_0, \{\hat{\sigma}_{\ell,0}\}_{\ell \in [2N-1]}, \{\hat{y}_j\}_{j \in [n]}, \{\hat{\tau}_{j,i}\}_{j \in [n], i \in I \setminus X_j}\big).$$

Take any tag $\tau \in \mathbb{Z}_p$ and any message $m \in \mathbb{G}_T$. Take any quorum size $L \leq N$, any threshold $T \in [L]$, and any collection $\{(\mathsf{pk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell)\}_{\ell \in [L]}$ where $(\mathsf{pk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell)$ is in the support of $\mathsf{KeyGen}(\mathsf{crs})$ for all $\ell \in [L]$. This means

$$\mathsf{pk}_\ell = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, A_\ell) = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, e(g, \hat{g})^{\alpha_\ell})$$

Let $(\mathsf{ek}, \mathsf{ak}) = \mathsf{Preprocess}(\mathsf{crs}, \{(\mathsf{pk}_\ell, \mathsf{ht}_\ell)\}_{\ell\in[L]})$ and $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{ek}, \tau, m, T)$. Take any index $i^* \in [L]$ and let $\sigma_{i^*} \leftarrow \mathsf{PartialDec}(\mathsf{sk}_{i^*}, \tau, \mathsf{ct})$. Then

$$\sigma_{i^*} = (\sigma_{i^*,1}, \hat{\sigma}_{i^*,2}) = (g^r, \hat{g}^{\alpha_{i^*}}(\hat{u}^\tau\hat{h})^r).$$

Consider now the relation checked by PartialVerify. By construction,

$$e(g, \hat{\sigma}_{i^*,2}) = e(g, \hat{g}^{\alpha_{i^*}}(\hat{u}^\tau\hat{h})^r) = e(g,\hat{g})^{\alpha_{i^*}} e(g^r, \hat{u}^\tau\hat{h}) = A_{i^*} \cdot e(\sigma_{i^*,1}, \hat{u}^\tau\hat{h}).$$

In this case, $\mathsf{PartialVerify}(\mathsf{pk}_{i^*}, \tau, \mathsf{ct}, \sigma_{i^*}) = 1$. □

E.2 Proof of Theorem 5.13 (Aggregation Correctness)

This proof is a combination of the proofs of Theorems 4.6 and 5.4. Take any $\lambda \in \mathbb{N}$, any $N \le 2^\lambda$, and any crs in the support of $\mathsf{Setup}(1^\lambda, 1^N)$. Then we can write

$$\mathsf{crs} = \big(\mathcal{G}, g, \hat{g}, B, \hat{u}, \hat{h}, \{\hat{g}^{c^i}\}_{i\in I\setminus\{0\}}, \hat{z}_0, \{\hat{v}_{\ell,0}\}_{\ell\in[2N-1]}, \{\hat{y}_j\}_{j\in[n]}, \{\hat{\tau}_{j,i}\}_{j\in[n], i\in I\setminus X_j}\big),$$

where

$$B = e(g,\hat{g})^{s_1}, \quad \hat{z}_0 = \prod_{\ell\in[2N-1]} \hat{g}^{c^\ell \mathbf{m}_\ell^\intercal \mathbf{s}}, \quad \hat{v}_{\ell,0} = \prod_{\substack{i\in[2N-1]\\ i\neq\ell}} \hat{g}^{c^{\ell-i}\mathbf{m}_i^\intercal\mathbf{s}}, \quad \hat{y}_j = \prod_{k\in X_j} \hat{g}^{c^k\gamma_k}, \quad \hat{\tau}_{j,i} = \prod_{k\in X_j} \hat{g}^{c^{k-i}\gamma_k}.$$

Take any tag $\tau \in \mathbb{Z}_p$ and any message $m \in \mathbb{G}_T$. Take any quorum size $L \le N$, any threshold $T \in [L]$, and any collection $\{(\mathsf{pk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell)\}_{\ell\in[L]}$ where $(\mathsf{pk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell)$ is in the support of $\mathsf{KeyGen}(\mathsf{crs})$ for all $\ell \in [L]$. This means

$$\begin{aligned}\mathsf{pk}_\ell &= (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, A_\ell) = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, e(g,\hat{g})^{\alpha_\ell})\\ \mathsf{ht}_\ell &= \{\hat{v}'_{\ell,i}\}_{i\in I\setminus\{0\}},\end{aligned}$$

where $\hat{v}'_{\ell,i} = g^{\alpha_\ell c^i}$. Let $(\mathsf{ek}, \mathsf{ak}) = \mathsf{Preprocess}(\mathsf{crs}, \{(\mathsf{pk}_\ell, \mathsf{ht}_\ell)\}_{\ell\in[L]})$. By definition, the preprocessing algorithm computes

$$\hat{z} = \hat{z}_0 \cdot \prod_{\ell\in[L]} \hat{v}'_{\ell,\ell} = \prod_{\ell\in[2N-1]} \hat{g}^{c^\ell \mathbf{m}_\ell^\intercal\mathbf{s}} \cdot \prod_{\ell\in[L]} \hat{g}^{c^\ell\alpha_\ell} = \hat{g}^{\tilde{z}} \quad \text{where} \quad \tilde{z} = \sum_{\ell\in[2N-1]} c^\ell \mathbf{m}_\ell^\intercal\mathbf{s} + \sum_{\ell\in[L]} c^\ell\alpha_\ell$$

Next, the preprocessing algorithm computes $\hat{v}_\ell$ according to Eq. (5.5). This means

$$\hat{v}_\ell = \hat{v}_{\ell,0} \cdot \prod_{\substack{i\in[L]\\ i\neq\ell}} \hat{v}'_{i,i-\ell} = \prod_{\substack{i\in[2N-1]\\ i\neq\ell}} \hat{g}^{c^{i-\ell}\mathbf{m}_i^\intercal\mathbf{s}} \prod_{\substack{i\in[L]\\ i\neq\ell}} \hat{g}^{c^{i-\ell}\alpha_i} \tag{E.1}$$

Then, it sets

$$\begin{aligned}\mathsf{ek} &= (L, \mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, B, \hat{z}, \{\hat{y}_j\}_{j\in[n]})\\ \mathsf{ak} &= \big(L, \{g^{c^{-\ell}}, \hat{v}_\ell\}_{\ell\in[L]}, \{\hat{\tau}_{j,i}\}_{j\in[n], i\in I\setminus X_j}\big).\end{aligned}$$

Take any ct in the support of $\mathsf{Encrypt}(\mathsf{ek}, \tau, m, T)$. Then, we can write

$$\mathsf{ct} = (\tau, T, C_1, c_2, \hat{c}_3, \hat{c}_4) = \Big(\tau\,,\, T\,,\, B^t \cdot m\,,\, g^t\,,\, (\hat{u}^\tau\hat{h})^t\,,\, \hat{z}^t \textstyle\prod_{j\in[n]:b_j=0} \hat{y}_j^t\Big), \tag{E.2}$$

where $L - T = \sum_{j\in[n]} b_j 2^{j-1}$. Take any set of partial decryptions $\{\sigma_i\}_{i\in S}$ where $|S| \ge T$ where for all $i \in S$, $\mathsf{PartialVerify}(\mathsf{pk}_i, \tau, \mathsf{ct}, \sigma_i) = 1$. By construction, if we parse $\sigma_\ell = (\sigma_{\ell,1}, \hat{\sigma}_{\ell,2})$, this means

$$\forall \ell \in S : e(g, \hat{\sigma}_{\ell,2}) = A_\ell \cdot e(\sigma_{\ell,1}, \hat{u}^\tau\hat{h}) = e(g,\hat{g})^{\alpha_\ell} \cdot e(\sigma_{\ell,1}, \hat{u}^\tau\hat{h}). \tag{E.3}$$

Consider now $\mathsf{Decrypt}(\mathsf{crs}, \mathsf{ak}, \mathsf{ct}, \{\sigma_\ell\}_{\ell \in S})$. Without loss of generality, let $|S| = T$ (this is without loss of generality since the behavior of Decrypt only depends on the first T partial decryptions in S). First, the decryption algorithm sets

$$S_{\mathsf{pad}} = S \cup [L+1, N] \cup \bigcup_{j \in [n] : b_j = 1} X_j.$$

Critically, these are the same bits $b_1, \ldots, b_n$ as computed by the Encrypt algorithm (specifically, $b_n \cdots b_1$ is the binary representation of $L - T$). Then it computes the aggregated decryption components:

$$\begin{aligned}
\sigma_{\mathsf{agg},1} &= \prod_{\ell \in S} \sigma_{\ell,1}^{\omega_\ell} \\
\hat{\sigma}_{\mathsf{agg},2} &= \prod_{\ell \in S} \hat{\sigma}_{\ell,2}^{\omega_\ell} \cdot \prod_{\ell \in S_{\mathsf{pad}}} \hat{v}_\ell^{\omega_\ell} \cdot \prod_{\substack{j \in [n] \\ b_j = 0}} \prod_{\ell \in S_{\mathsf{pad}}} \hat{\tau}_{j,\ell}^{\omega_\ell} \\
\sigma_{\mathsf{agg},3} &= \prod_{\ell \in S_{\mathsf{pad}}} \left(g^{c^{-\ell}}\right)^{\omega_\ell}.
\end{aligned} \tag{E.4}$$

The rest of this proof follows almost verbatim as the proof of Theorem 5.4. We include it here to provide a self-contained exposition. First, by bilinearity, we have

$$e(\sigma_{\mathsf{agg},1}, \hat{u}^\tau \hat{h}) = \prod_{\ell \in S} e(\sigma_{\ell,1}, \hat{u}^\tau \hat{h})^{\omega_\ell} = \prod_{\ell \in S} \frac{e(g, \hat{\sigma}_{\ell,2}^{\omega_\ell})}{e(g, \hat{g})^{\omega_\ell \alpha_\ell}} \tag{E.5}$$

Let $\tilde{\sigma}_{\mathsf{agg},3} = \sum_{\ell \in S_{\mathsf{pad}}} \omega_\ell c^{-\ell}$. Then $\sigma_{\mathsf{agg},3} = g^{\tilde{\sigma}_{\mathsf{agg},3}}$. Next, $\boldsymbol{\omega}^\top \mathbf{M} = \mathbf{e}_1^\top$ so

$$\sum_{\ell \in [2N-1]} \omega_\ell \mathbf{m}_\ell^\top \mathbf{s} = \boldsymbol{\omega}^\top \mathbf{M} \mathbf{s} = \mathbf{e}_1^\top \mathbf{s} = s_1.$$

Combined with the fact that $\omega_\ell = 0$ for all $\ell \notin S_{\mathsf{pad}}$, we have

$$\begin{aligned}
\tilde{z} \cdot \tilde{\sigma}_{\mathsf{agg},3} &= \sum_{\ell \in S_{\mathsf{pad}}} \sum_{i \in [2N-1]} c^i \mathbf{m}_i^\top \mathbf{s} \cdot \omega_\ell c^{-\ell} + \sum_{\ell \in S_{\mathsf{pad}}} \sum_{i \in [L]} c^i \alpha_i \cdot \omega_\ell c^{-\ell} \\
&= \sum_{\ell \in S_{\mathsf{pad}}} \omega_\ell \mathbf{m}_\ell^\top \mathbf{s} + \sum_{\ell \in S_{\mathsf{pad}} \cap [L]} \omega_\ell \alpha_\ell + \sum_{\ell \in S_{\mathsf{pad}}} \sum_{\substack{i \in [2N-1] \\ i \neq \ell}} \omega_\ell c^{i-\ell} \mathbf{m}_i^\top \mathbf{s} + \sum_{\ell \in S_{\mathsf{pad}}} \sum_{\substack{i \in [L] \\ i \neq \ell}} \omega_\ell \alpha_i c^{i-\ell} \\
&= s_1 + \sum_{\ell \in S} \omega_\ell \alpha_\ell + \sum_{\ell \in S_{\mathsf{pad}}} \omega_\ell \left(\sum_{\substack{i \in [2N-1] \\ i \neq \ell}} c^{i-\ell} \mathbf{m}_i^\top \mathbf{s} + \sum_{\substack{i \in [L] \\ i \neq \ell}} c^{i-\ell} \alpha_i \right)
\end{aligned}$$

Using the definition of $\hat{v}_\ell$ from Eq. (E.1), we have

$$\begin{aligned}
e(\sigma_{\mathsf{agg},3}, \hat{z}) &= e(g, \hat{g})^{\tilde{\sigma}_{\mathsf{agg},3} \tilde{z}} \\
&= e(g, \hat{g})^{s_1} \cdot \prod_{\ell \in S} e(g, \hat{g})^{\omega_\ell \alpha_\ell} \cdot \prod_{\ell \in S_{\mathsf{pad}}} \left(\prod_{\substack{i \in [2N-1] \\ i \neq \ell}} e(g, \hat{g}^{c^{i-\ell} \mathbf{m}_i^\top \mathbf{s}}) \prod_{\substack{i \in [L] \\ i \neq \ell}} e(g, \hat{g}^{c^{i-\ell} \alpha_i}) \right)^{\omega_\ell} \\
&= e(g, \hat{g})^{s_1} \cdot \prod_{\ell \in S} e(g, \hat{g})^{\omega_\ell \alpha_\ell} \cdot \prod_{\ell \in S_{\mathsf{pad}}} e(g, \hat{v}_\ell^{\omega_\ell}).
\end{aligned} \tag{E.6}$$

Next, let $\tilde{y}_j = \sum_{k \in X_j} c^k \gamma_k$. In particular, $\hat{y}_j = \hat{g}^{\tilde{y}_j}$. Then,

$$\tilde{y}_j \cdot \tilde{\sigma}_{\mathsf{agg},3} = \sum_{\ell \in S_{\mathsf{pad}}} \sum_{k \in X_j} c^k \gamma_k \cdot \omega_\ell c^{-\ell} = \sum_{\ell \in S_{\mathsf{pad}}} \sum_{k \in X_j} \omega_\ell \gamma_k c^{k-\ell}.$$

By construction, S_{pad} contains X_j where $b_j = 1$. Moreover, the sets $X_1, \ldots, X_n$ are pairwise disjoint, so for all $j \in [n]$ where $b_j = 0$, $S_{\text{pad}} \cap X_j = \emptyset$. Thus,

$$\forall j \in [n] \text{ where } b_j = 0: \quad e(\sigma_{\text{agg},3}, \hat{y}_j) = e(g, \hat{g})^{\tilde{y}_j \tilde{\sigma}_{\text{agg},3}} = \prod_{\ell \in S_{\text{pad}}} e(g, \hat{\tau}_{j,\ell}^{\omega_\ell}). \tag{E.7}$$

Combining Eqs. (E.4) to (E.7), we have

$$\begin{aligned}
& e(g, \hat{g})^{s_1} \cdot e(g, \hat{\sigma}_{\text{agg},2}) \\
&= e(g, \hat{g})^{s_1} \cdot \prod_{\ell \in S} e(g, \hat{\sigma}_{\ell,2}^{\omega_\ell}) \cdot \prod_{\ell \in S_{\text{pad}}} e(g, \hat{v}_\ell^{\omega_\ell}) \cdot \prod_{\substack{j \in [n] \\ b_j = 0}} \prod_{\ell \in S_{\text{pad}}} e(g, \hat{\tau}_{j,\ell}^{\omega_\ell}) \\
&= e(\sigma_{\text{agg},3}, \hat{z}) \cdot \prod_{\substack{j \in [n] \\ b_j = 0}} e(\sigma_{\text{agg},3}, \hat{y}_j) \cdot \prod_{\ell \in S} \frac{e(g, \hat{\sigma}_{\ell,2}^{\omega_\ell})}{e(g, \hat{g})^{\omega_\ell \alpha_\ell}} \\
&= e(\sigma_{\text{agg},1}, \hat{u}^\tau \hat{h}) \cdot e(\sigma_{\text{agg},3}, \hat{z}) \cdot \prod_{\substack{j \in [n] \\ b_j = 0}} e(\sigma_{\text{agg},3}, \hat{y}_j).
\end{aligned}$$

Combined with Eq. (E.2), this means

$$\begin{aligned}
C_1 \cdot e(c_2, \hat{\sigma}_{\text{agg},2}) &= B^t \cdot m \cdot e(g^t, \hat{\sigma}_{\text{agg},2}) \\
&= m \cdot \left(e(g, \hat{g})^{s_1} \cdot e(g, \hat{\sigma}_{\text{agg},2}) \right)^t \\
&= m \cdot \left(e(\sigma_{\text{agg},1}, \hat{u}^\tau \hat{h}) \cdot e(\sigma_{\text{agg},3}, \hat{z}) \cdot \prod_{\substack{j \in [n] \\ b_j = 0}} e(\sigma_{\text{agg},3}, \hat{y}_j) \right)^t \\
&= m \cdot e(\sigma_{\text{agg},1}, \hat{c}_3) \cdot e(\sigma_{\text{agg},3}, \hat{c}_4).
\end{aligned}$$

We conclude then that

$$\text{Decrypt}(\text{crs}, \text{ak}, \text{ct}, \{\sigma_\ell\}_{\ell \in S}) = C_1 \cdot e(\sigma_{\text{agg},1}, \hat{c}_3)^{-1} \cdot e(c_2, \hat{\sigma}_{\text{agg},2}) \cdot e(\sigma_{\text{agg},3}, \hat{c}_4)^{-1} = m.$$

Aggregation correctness follows. □

E.3 Proof of Theorem 5.14 (Static Tag-Based CCA-Security)

This proof is a combination of the proofs of Theorems 4.7 and 5.5. As in the proof of Theorem 4.7, we begin by defining a sequence of hybrid experiments. Each experiment is parameterized by a bit $b \in \{0, 1\}$, and implicitly, by an adversary $\mathcal{A}$ and a security parameter λ.

- $\text{Hyb}_0^{(b)}$: This is the real tag-based CCA security experiment with bit $b \in \{0, 1\}$.
- $\text{Hyb}_1^{(b)}$: Same as $\text{Hyb}_0^{(b)}$, except at the beginning of the security game, after the adversary commits to the quorum size L, the challenger samples a random threshold $T^* \xleftarrow{R} [|C| + 1, L]$. After algorithm $\mathcal{A}$ outputs its desired threshold T, the challenger outputs 0 if $T \neq T^*$.
- $\text{Hyb}_2^{(b)}$: Same as $\text{Hyb}_1^{(b)}$, except the challenger sample the challenge tag $\tau^* \xleftarrow{R} \mathbb{Z}_p$ at the beginning of the experiment. Then, if the adversary makes a partial decryption query on τ^* *before* it chooses the challenge messages m_0, m_1, then the challenge outputs 0.
- $\text{Hyb}_3^{(b)}$: Same as $\text{Hyb}_2^{(b)}$, except when constructing the ciphertext, the challenger samples $C_1^* \xleftarrow{R} \mathbb{G}_T$. Notably, in this experiment, the adversary's view is independent of the bit $b \in \{0, 1\}$.

For an adversary $\mathcal{A}$, we write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the output distribution of an execution of experiment Hyb_i with adversary $\mathcal{A}$ (and the implicit security parameter λ). We now argue that each adjacent pair of experiments are indistinguishable.

Lemma E.1. *For all efficient adversaries $\mathcal{A}$, there exists a fixed polynomial $Q_1 = Q_1(\lambda)$ such that for all $b \in \{0,1\}$, $\Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1] = \frac{1}{Q_1} \cdot \Pr[\mathsf{Hyb}_0^{(b)}(\mathcal{A}) = 1]$.*

Proof. This follows by the same argument as in the proof of Lemma D.1. Namely, the output in $\mathsf{Hyb}_1^{(b)}$ is 1 if and only if the output in $\mathsf{Hyb}_0^{(b)}$ is 1 and $T^* = T$, where T is the threshold chosen by $\mathcal{A}$. By construction, the output in $\mathsf{Hyb}_0^{(b)}$ is 1 only if $|C| < T < L$. The challenger in $\mathsf{Hyb}_1^{(b)}$ samples $T^* \xleftarrow{R} [|C|+1, L]$, and moreover, the view of adversary $\mathcal{A}$ is independent of T^*. Correspondingly, we conclude that

$$\Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1] = \frac{1}{L - |C|} \cdot \Pr[\mathsf{Hyb}_0^{(b)}(\mathcal{A}) = 1].$$

Since the adversary $\mathcal{A}$ is efficient, both $|C|, L$ are polynomially-bounded, and the claim follows. Note that the value of $Q_1 \coloneqq L - |C|$ is a function of the adversary $\mathcal{A}$ only, and importantly, does not depend on the bit $b \in \{0,1\}$. □

Lemma E.2. *For all adversaries $\mathcal{A}$ making at most $Q_2 = Q_2(\lambda)$ partial decryption queries and all bits $b \in \{0,1\}$, we have that*

$$\left|\Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1]\right| \leq Q_2/p.$$

Proof. This follows by the same argument as in the proof of Lemma C.2. □

Lemma E.3. *Suppose the decisional N-extended bilinear Diffie-Hellman exponent assumption (Assumption 3.12) holds with respect to* GroupGen. *Then, for all efficient adversaries $\mathcal{A}$ and all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3^{(b)}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. This proof is a combination of the proofs of Lemmas C.2 and D.2. For completeness, we include the full argument here. Suppose there exists $b \in \{0,1\}$ and an efficient adversary $\mathcal{A}$ where

$$\left|\Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3^{(b)}(\mathcal{A}) = 1]\right| \geq \varepsilon$$

for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for the N-extended bilinear Diffie-Hellman exponent assumption. As in previous proofs, we will use a tilde (e.g., $\tilde{u}, \tilde{h}$) to denote exponents sampled by the reduction algorithm $\mathcal{B}$. Algorithm $\mathcal{B}$ works as follows:

1. Let $I = [-2N+1, 2N-1]$. On input the challenge

$$\left(1^\lambda, \mathcal{G}, g, \hat{g}, g^a, \hat{g}^a, \hat{g}^b, g^t, \hat{g}^t, \{g^{c^i}, \hat{g}^{c^i}, \hat{g}^{abc^i}\}_{i \in I \setminus \{0\}}, \{\hat{g}^{tc^i}\}_{i \in [2N-1]}, X\right),$$

   where $X = e(g,\hat{g})^{abt}$ or $X \xleftarrow{R} \mathbb{G}_T$, algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ on 1^λ. Algorithm $\mathcal{A}$ outputs the quorum size $L \leq N$ and the indices of the corrupted users $C \subseteq [L]$.

2. Algorithm $\mathcal{B}$ starts by sampling the target threshold $T^* \xleftarrow{R} [|C|+1, L]$. Let $b_n^* \cdots b_1^*$ be the binary representation of $L - T^*$ (namely, $L - T^* = \sum_{j \in [n]} b_j^* 2^{j-1}$). Let

$$S_{\mathsf{pad}}^* = C \cup [L+1, N] \cup \bigcup_{j \in [n] : b_j^* = 1} X_j$$

   where $X_j = [N + 2^{j-1}, N + 2^j - 1]$. By construction,

$$\left|S_{\mathsf{pad}}^*\right| = |C| + (N - L) + \sum_{j \in [n] : b_j^* = 1} 2^{j-1} = |C| + (N-L) + (L - T^*) = N + (|C| - T^*) < N,$$

since $T^* > |C|$. Let $\mathbf{M} \in \mathbb{Z}_p^{(2N-1)\times N}$ be the share-generating matrix for the N-out-of-$(2N-1)$ threshold policy from Eq. (2.1). Since $|S^*_{\mathsf{pad}}| < N$, the set S^*_{pad} does not satisfy the threshold policy defined by $\mathbf{M}$. Thus, there exists a vector $\tilde{\mathbf{w}} \in \mathbb{Z}_p^N$ such that for all indices $i \in S^*_{\mathsf{pad}}$, $\mathbf{m}_i^\intercal \tilde{\mathbf{w}} = 0$, where $\mathbf{m}_i^\intercal$ denotes the i^{th} row of $\mathbf{M}$, and $\mathbf{e}_1^\intercal \tilde{\mathbf{w}} = 1$.

3. Algorithm $\mathcal{B}$ constructs the common reference string crs as follows. First, it samples $\tilde{u}, \tilde{h} \xleftarrow{R} \mathbb{Z}_p$ and sets

$$\hat{u} = (\hat{g}^b) \cdot \hat{g}^{\tilde{u}} \quad \text{and} \quad \hat{h} = \hat{g}^{\tilde{h}} / (\hat{g}^b)^{m^*}.$$

Algorithm $\mathcal{B}$ samples a vector $\tilde{\mathbf{s}} \xleftarrow{R} \mathbb{Z}_p^N$. Algorithm $\mathcal{B}$ implicitly sets $\mathbf{s} = \tilde{\mathbf{s}} + ab \cdot \tilde{\mathbf{w}}$ and computes

$$B = e(g, \hat{g})^{\tilde{s}_1} \cdot e(g^a, \hat{g}^b)^{\tilde{w}_1} = e(g, \hat{g})^{s_1}.$$

Next, algorithm $\mathcal{B}$ computes

$$\hat{z}_0 = \prod_{\ell \in [2N-1]} \left((\hat{g}^{c^\ell})^{\mathbf{m}_\ell^\intercal \tilde{\mathbf{s}}} \cdot (\hat{g}^{c^\ell ab})^{\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} \right) = \prod_{\ell \in [2N-1]} \hat{g}^{c^\ell \mathbf{m}_\ell^\intercal \mathbf{s}}$$

$$\forall \ell \in [2N-1] : \hat{v}_{\ell,0} = \prod_{\substack{i \in [2N-1] \\ i \neq \ell}} \left((\hat{g}^{c^{i-\ell}})^{\mathbf{m}_i^\intercal \tilde{\mathbf{s}}} \cdot (\hat{g}^{c^{i-\ell} ab})^{\mathbf{m}_i^\intercal \tilde{\mathbf{w}}} \right) = \prod_{\substack{i \in [2N-1] \\ i \neq \ell}} \hat{g}^{c^{i-\ell} \mathbf{m}_i^\intercal \mathbf{s}}$$

Finally, for each $\ell \in [N+1, 2N-1]$, algorithm $\mathcal{B}$ samples $\tilde{\gamma}_\ell \xleftarrow{R} \mathbb{Z}_p$ and implicitly sets

$$\gamma_\ell = \begin{cases} \tilde{\gamma}_\ell & \ell \in X_j \text{ for some } j \in [n] \text{ where } b_j^* = 1 \\ \tilde{\gamma}_\ell - ab \cdot \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}} & \ell \in X_j \text{ for some } j \in [n] \text{ where } b_j^* = 0. \end{cases}$$

Specifically, algorithm $\mathcal{B}$ constructs $\hat{y}_j$ and $\hat{\tau}_{j,i}$ for $j \in [n]$ and $i \in I \setminus X_j$ as follows:

- If $b_j^* = 1$, then algorithm $\mathcal{B}$ sets

$$\hat{y}_j = \prod_{k \in X_j} (\hat{g}^{c^k})^{\tilde{\gamma}_k} = \prod_{k \in X_j} \hat{g}^{c^k \gamma_k} \quad \text{and} \quad \hat{\tau}_{j,i} = \prod_{k \in X_j} (\hat{g}^{c^{k-i}})^{\tilde{\gamma}_k} = \prod_{k \in X_j} \hat{g}^{c^{k-i} \gamma_k}.$$

- If $b_j^* = 0$, then algorithm $\mathcal{B}$ sets

$$\hat{y}_j = \prod_{k \in X_j} \left((\hat{g}^{c^k})^{\tilde{\gamma}_k} \cdot (\hat{g}^{c^k ab})^{-\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} \right) = \prod_{k \in X_j} \hat{g}^{c^k \gamma_k}$$

$$\hat{\tau}_{j,i} = \prod_{k \in X_j} \left((\hat{g}^{c^{k-i}})^{\tilde{\gamma}_k} \cdot (\hat{g}^{c^{k-i} ab})^{-\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} \right) = \prod_{k \in X_j} \hat{g}^{c^{k-i} \gamma_k}.$$

Algorithm $\mathcal{B}$ sets the common reference string to be

$$\mathsf{crs} = \left(\mathcal{G}, g, \hat{g}, B, \hat{u}, \hat{h}, \{g^{c^i}\}_{i \in I \setminus \{0\}}, \hat{z}_0, \{\hat{v}_{\ell,0}\}_{\ell \in [2N-1]}, \{\hat{y}_j\}_{j \in [n]}, \{\hat{\tau}_{j,i}\}_{j \in [n], i \in I \setminus X_j} \right).$$

4. Next, to simulate the honest public keys, algorithm $\mathcal{B}$ starts by sampling $\tilde{\alpha}_\ell \xleftarrow{R} \mathbb{Z}_p$ for each $\ell \in [L] \setminus C$. Then, algorithm $\mathcal{B}$ implicitly sets the secret key for user ℓ to be $\alpha_\ell = \tilde{\alpha}_\ell - ab \cdot \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}$. Specifically, algorithm $\mathcal{B}$ constructs the components of the public key and the aggregation hint as follows:

$$A_\ell = e(g, \hat{g})^{\tilde{\alpha}_\ell} \cdot e(g^a, \hat{g}^b)^{-\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} = e(g, \hat{g})^{\alpha_\ell}$$

$$\hat{v}'_{\ell,i} = (\hat{g}^{c^i})^{\tilde{\alpha}_\ell} \cdot (\hat{g}^{abc^i})^{-\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} = (\hat{g}^{c^i \alpha_\ell}).$$

Then, algorithm $\mathcal{B}$ sets

$$\mathsf{vk}_\ell = (\mathcal{G}, g, \hat{g}, \hat{u}, \hat{h}, A_\ell) \quad \text{and} \quad \mathsf{ht}_\ell = \{\hat{v}'_{\ell,i}\}_{i \in I \setminus \{0\}}.$$

Algorithm $\mathcal{B}$ gives crs and $\{(\mathsf{pk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [L] \setminus C}$ to $\mathcal{A}$.

5. Algorithm $\mathcal{A}$ now specifies the key-generation randomness $\rho_\ell \in \{0,1\}^*$ used to generate the keys for each of the corrupted users $\ell \in C$.

6. For each $\ell \in C$, algorithm $\mathcal{B}$ computes $(\mathsf{vk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell) \leftarrow \mathsf{KeyGen}(\mathsf{crs}; \rho_\ell)$. Algorithm $\mathcal{B}$ parses $\mathsf{sk}_\ell = (\mathsf{vk}_\ell, \tilde{\alpha}_\ell)$ for some $\tilde{\alpha}_\ell \in \mathbb{Z}_p$. Then it computes and gives $(\mathsf{ek}, \mathsf{ak}) = \mathsf{Preprocess}(\mathsf{crs}, \{(\mathsf{ek}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [L]})$ to $\mathcal{A}$.

7. Whenever algorithm $\mathcal{A}$ makes a partial decryption query on an index $\ell \in [L] \setminus C$ a tag $\tau \in \mathbb{Z}_p$, and a ciphertext ct, algorithm $\mathcal{B}$ first checks if $\tau = \tau^*$. If so, then algorithm $\mathcal{B}$ halts with output 0. Otherwise, algorithm $\mathcal{B}$ samples $\tilde{r} \xleftarrow{R} \mathbb{Z}_p$ and implicitly sets $r = \tilde{r} + a(\tau - \tau^*)^{-1}\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}$. Then, it computes

$$\sigma_1 = g^{\tilde{r}} \cdot (g^a)^{(\tau-\tau^*)^{-1}\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}}$$
$$\hat{\sigma}_1 = \hat{g}^{\tilde{r}} \cdot (\hat{g}^a)^{(\tau-\tau^*)^{-1}\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}}$$
$$\hat{\sigma}_2 = \hat{g}^{\tilde{\alpha}_\ell} \cdot (\hat{g}^b)^{\tilde{r}(\tau-\tau^*)} \cdot \hat{\sigma}_1^{\tilde{u}\tau+\tilde{h}}.$$

   and responds to $\mathcal{A}$ with the partial decryption $\sigma = (\sigma_1, \hat{\sigma}_2)$.

8. After $\mathcal{A}$ is finished making partial decryption queries, it outputs a pair of messages $m_0, m_1 \in \mathbb{G}_T$ and a threshold T. Algorithm $\mathcal{B}$ outputs 0 if $T \neq T^*$. Otherwise, algorithm $\mathcal{B}$ defines the exponent $\tilde{\zeta}_\ell \in \mathbb{Z}_p$ for each $\ell \in [N]$ as

$$\tilde{\zeta}_\ell = \begin{cases} \tilde{\alpha}_\ell & \ell \in [L] \\ 0 & \ell \in [L+1, N] \cup \bigcup_{j \in [n]: b_j^* = 1} X_j \\ \tilde{\gamma}_\ell & \ell \in \bigcup_{j \in [n]: b_j^* = 0} X_j. \end{cases} \tag{E.8}$$

   Algorithm $\mathcal{B}$ replies to $\mathcal{A}$ with the tag τ^* and the challenge ciphertext

$$\mathsf{ct}^* = (\tau^*, C_1^*, c_2^*, \hat{c}_3^*, \hat{c}_4^*) = \left(\tau^* \ , \ T^* \ , \ X \cdot e(g, \hat{g}^t)^{\tilde{s}_1} \cdot m_b \ , \ g^t \ , \ (\hat{g}^t)^{\tilde{u}\tau^*+\tilde{h}} \ , \ \prod_{\ell \in [2N-1]} (\hat{g}^{tc^\ell})^{\mathbf{m}_\ell^\intercal \tilde{\mathbf{s}} + \tilde{\zeta}_\ell} \right).$$

9. Algorithm $\mathcal{A}$ can continue to make partial decryption queries on tags $\tau \neq \tau^*$. Algorithm $\mathcal{B}$ responds exactly as described above.

10. At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b \in \{0,1\}$, which algorithm $\mathcal{B}$ also outputs.

Similar to the proofs of Lemmas C.2 and D.2, we first argue that algorithm $\mathcal{B}$ correctly simulates the common reference string, the honest public keys, and the partial decryption queries of $\mathcal{A}$ according to the specification of the real scheme (which coincides with the distribution in $\mathsf{Hyb}_2^{(b)}$ and $\mathsf{Hyb}_3^{(b)}$). Consider first the components of the common reference string:

- Algorithm $\mathcal{B}$ samples $\tilde{u}, \tilde{h} \xleftarrow{R} \mathbb{Z}_p$ so the distributions of $\hat{u}, \hat{h}$ are also uniform over $\mathbb{G}$ (and independent of all other components in crs), exactly as in the real scheme.
- Algorithm $\mathcal{B}$ implicitly sets $\mathbf{s} = \tilde{\mathbf{s}} + ab \cdot \tilde{\mathbf{w}}$, where $\tilde{\mathbf{s}} \xleftarrow{R} \mathbb{Z}_p^N$. Thus, the distribution of $\mathbf{s}$ also coincides with its distribution in the real scheme. Correspondingly, the elements B, $\hat{z}_0$, and $\hat{v}_{\ell,0}$ are distributed exactly as in the real scheme.
- For $\ell \in [N+1, 2N-1]$, algorithm $\mathcal{B}$ either sets $\gamma_\ell = \tilde{\gamma}_\ell$ or $\gamma_\ell = \tilde{\gamma}_\ell - ab\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}$, where $\tilde{\gamma}_\ell \xleftarrow{R} \mathbb{Z}_p$. This coincides with the distribution of γ_ℓ in the real scheme. Hence, we conclude that the elements $\hat{y}_j$ and $\hat{\tau}_{j,i}$ are distributed exactly as in the real scheme.
- Finally, the challenger samples $c \xleftarrow{R} \mathbb{Z}_p^*$, which matches the distribution in the real scheme.

We conclude that algorithm $\mathcal{B}$ constructs crs exactly according to the distribution in $\mathsf{Hyb}_2^{(b)}$ and $\mathsf{Hyb}_3^{(b)}$. We now consider the public keys and the partial decryption queries:

- **Public keys:** Consider the honest public keys pk_ℓ for $\ell \in [L] \setminus C$. By construction, the public keys pk_ℓ and hint components ht_ℓ sampled by algorithm $\mathcal{B}$ coincide with those that would be output by $\mathsf{KeyGen}(\mathsf{crs})$ with $\alpha_\ell = \tilde{\alpha}_\ell - ab \cdot \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}$. Since algorithm $\mathcal{B}$ samples $\tilde{\alpha}_\ell \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ for all $\ell \in [L] \setminus C$, the public keys are also distributed exactly as in the real scheme.

- **Partial decryption queries:** Next, consider the partial decryption queries on tags $\tau \neq \tau^*$. Note that if algorithm $\mathcal{A}$ ever makes a partial decryption query on $\tau = \tau^*$, algorithm $\mathcal{B}$ outputs 0, which matches the behavior in $\mathsf{Hyb}_2^{(b)}$ and $\mathsf{Hyb}_3^{(b)}$. Let $\ell \in [L] \setminus C$ be the index and $\tau \in \mathbb{Z}_p$ be the tag associated with the partial decryption query. We claim that $\sigma = (\sigma_1, \hat{\sigma}_2)$ is a partial decryption with respect to the secret key α_ℓ and randomness $r = \tilde{r} + a(\tau - \tau^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}$:

  - By construction, algorithm $\mathcal{B}$ sets

    $$\sigma_1 = g^{\tilde{r}} \cdot (g^a)^{(\tau-\tau^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} = g^{\tilde{r} + a(\tau-\tau^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} = g^r.$$

    By the same calculation, this means $\hat{\sigma}_1 = \hat{g}^r$.
  - In the real scheme, $\hat{\sigma}_2 = \hat{g}^{\alpha_\ell} (\hat{u}^\tau \hat{h})^r$. Substituting the expressions for α_ℓ, $\hat{u}$, $\hat{h}$, and r, we have

    $$\begin{aligned}
    \hat{g}^{\alpha_\ell} (\hat{u}^\tau \hat{h})^r &= \hat{g}^{(\tilde{\alpha}_\ell - ab \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}})} (\hat{g}^{b\tau + \tilde{u}\tau} \hat{g}^{\tilde{h} - b\tau^*})^{\tilde{r} + a(\tau-\tau^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} \\
    &= \hat{g}^{\tilde{\alpha}_\ell} (\hat{g}^{\tilde{u}\tau + \tilde{h}})^{\tilde{r} + a(\tau-\tau^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} \hat{g}^{-ab \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}} \hat{g}^{b(\tau-\tau^*)(\tilde{r} + a(\tau-\tau^*)^{-1} \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}})} \\
    &= \hat{g}^{\tilde{\alpha}_\ell} \hat{\sigma}_1^{\tilde{u}\tau + \tilde{h}} \hat{g}^{b(\tau-\tau^*)\tilde{r}}.
    \end{aligned}$$

    This is precisely how algorithm $\mathcal{B}$ constructs the partial decryptions.

  Since algorithm $\mathcal{B}$ samples $\tilde{r} \xleftarrow{\mathrm{R}} \mathbb{Z}_p$, the distribution of r is also uniform and the partial decryption is correctly constructed.

Finally, algorithm $\mathcal{B}$ constructs $(\mathsf{ek}, \mathsf{ak}) = \mathsf{Preprocess}(\mathsf{crs}, \{(\mathsf{ek}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [L]})$. This is the same procedure as in $\mathsf{Hyb}_2^{(b)}$ and $\mathsf{Hyb}_3^{(b)}$. It suffices to reason about the distribution of the challenge ciphertext. We claim that ct^* is an encryption of m_b with randomness $t \in \mathbb{Z}_p$ according to either the specification of $\mathsf{Hyb}_2^{(b)}$ or $\mathsf{Hyb}_3^{(b)}$. We consider each component of the challenge ciphertext:

- Consider C_1^*. In the reduction, algorithm $\mathcal{B}$ sets

  $$C_1^* = X \cdot e(g, \hat{g}^t)^{\tilde{s}_1} \cdot m_b.$$

  When $X = e(g,g)^{abt}$, then

  $$C_1^* = e(g, \hat{g})^{(\tilde{s}_1 + ab)t} \cdot m_b = e(g, \hat{g})^{s_1 t} \cdot m_b = B^t \cdot m_b,$$

  since algorithm $\mathcal{B}$ implicitly sets $\mathbf{s} = \tilde{\mathbf{s}} + ab \cdot \tilde{\mathbf{w}}$, and $\mathbf{e}_1^\intercal \tilde{\mathbf{w}} = 1$. This corresponds to the distribution of C_1^* in $\mathsf{Hyb}_2^{(b)}$. Suppose instead that $X \xleftarrow{\mathrm{R}} \mathbb{G}_T$. This is the distribution of C_1^* in $\mathsf{Hyb}_3^{(b)}$.

- Consider c_2^*. In the reduction, algorithm $\mathcal{B}$ sets $c_2^* = g^t$, which matches the behavior in $\mathsf{Hyb}_2^{(b)}$ and $\mathsf{Hyb}_3^{(b)}$.

- Consider $\hat{c}_3^*$. In the reduction, algorithm $\mathcal{B}$ sets $c_3^* = (\hat{g}^t)^{\tilde{u}\tau^* + \tilde{h}} = (\hat{u}^{\tau^*} \hat{h})^t$ since $\hat{u}^{\tau^*} \hat{h} = \hat{g}^{b\tau^* + \tilde{u}\tau^* + \tilde{h} - b\tau^*} = \hat{g}^{\tilde{u}\tau^* + \tilde{h}}$. This matches the distribution in $\mathsf{Hyb}_2^{(b)}$ and $\mathsf{Hyb}_3^{(b)}$.

- Consider $\hat{c}_4^*$. In the reduction, algorithm $\mathcal{B}$ sets

  $$\hat{c}_4^* = \prod_{\ell \in [2N-1]} (\hat{g}^{t c^\ell})^{\mathbf{m}_\ell^\intercal \tilde{\mathbf{s}} + \tilde{\zeta}_\ell},$$

where the exponents $\tilde{\zeta}_\ell \in \mathbb{Z}_p$ are as defined in Eq. (E.8). We claim that $\hat{c}_4^* = \hat{z}^t \prod_{j\in[n]:b_j^*=0} \hat{y}_j^t$, where

$$\hat{z} = \hat{z}_0 \cdot \prod_{\ell\in[L]} \hat{v}'_{\ell,\ell} = \prod_{\ell\in[2N-1]} \hat{g}^{c^\ell \mathbf{m}_\ell^\intercal \mathbf{s}} \cdot \prod_{\ell\in[L]} \hat{v}'_{\ell,\ell}$$

is as defined by the Preprocess. This means

$$\hat{z} \cdot \prod_{\substack{j\in[n]\\ b_j^*=0}} \hat{y}_j = \prod_{\ell\in[2N-1]} \hat{g}^{c^\ell \mathbf{m}_\ell^\intercal \mathbf{s}} \cdot \prod_{\ell\in[L]} \hat{v}'_{\ell,\ell} \cdot \prod_{\substack{j\in[n]\\ b_j^*=0}} \prod_{k\in X_j} \hat{g}^{c^k \gamma_k}. \tag{E.9}$$

As in the proof of Lemma D.2, we consider the terms in this product grouped by the powers c^ℓ. Then, we obtain the same case analysis as before.

- If $\ell \in C$, then $v'_{\ell,\ell}$ is output $\mathsf{KeyGen}(\mathsf{crs}; \rho_\ell)$. Since $\mathsf{sk}_\ell = (\mathsf{vk}_\ell, \tilde{\alpha}_\ell)$, this means $\hat{v}'_{\ell,\ell} = \hat{g}^{c^\ell \tilde{\alpha}_\ell}$. In addition, since $\ell \in C \subseteq S^*_{\mathsf{pad}}$, we have $\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}} = 0$. This means

$$c^\ell(\mathbf{m}_\ell^\intercal \mathbf{s} + \tilde{\alpha}_\ell) = c^\ell(\mathbf{m}_\ell^\intercal(\tilde{\mathbf{s}} + ab\tilde{\mathbf{w}}) + \tilde{\alpha}_\ell) = c^\ell(\mathbf{m}_\ell^\intercal \tilde{\mathbf{s}} + \tilde{\alpha}_\ell) = c^\ell(\mathbf{m}_\ell^\intercal \tilde{\mathbf{s}} + \tilde{\zeta}_\ell).$$

- If $\ell \in [L] \setminus C$, then $\hat{v}'_{\ell,\ell} = \hat{g}^{c^\ell \alpha_\ell} = \hat{g}^{c^\ell(\tilde{\alpha}_\ell - ab\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}})}$. This means

$$c^\ell(\mathbf{m}_\ell^\intercal \mathbf{s} + \alpha_\ell) = c^\ell(\mathbf{m}_\ell^\intercal(\tilde{\mathbf{s}} + ab\tilde{\mathbf{w}}) + \tilde{\alpha}_\ell - ab\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}) = c^\ell(\mathbf{m}_\ell^\intercal \tilde{\mathbf{s}} + \tilde{\alpha}_\ell) = c^\ell(\mathbf{m}_\ell^\intercal \tilde{\mathbf{s}} + \tilde{\zeta}_\ell).$$

- Suppose $\ell \in [L+1, N]$. Then, $\ell \in S^*_{\mathsf{pad}}$, so as in the first case, $\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}} = \mathbf{0}$. This means

$$c^\ell \mathbf{m}_\ell^\intercal \mathbf{s} = c^\ell(\mathbf{m}_\ell^\intercal(\tilde{\mathbf{s}} + ab\tilde{\mathbf{w}})) = c^\ell \mathbf{m}_\ell^\intercal \tilde{\mathbf{s}} = c^\ell(\mathbf{m}_\ell^\intercal \tilde{\mathbf{s}} + \tilde{\zeta}_\ell).$$

- Suppose $\ell \in X_j$ for some $j \in [n]$ where $b_j^* = 1$. Then, $\ell \in S^*_{\mathsf{pad}}$, so as in the previous case, $\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}} = \mathbf{0}$. This means

$$c^\ell \mathbf{m}_\ell^\intercal \mathbf{s} = c^\ell(\mathbf{m}_\ell^\intercal(\tilde{\mathbf{s}} + ab\tilde{\mathbf{w}})) = c^\ell \mathbf{m}_\ell^\intercal \tilde{\mathbf{s}} = c^\ell(\mathbf{m}_\ell^\intercal \tilde{\mathbf{s}} + \tilde{\zeta}_\ell).$$

- Suppose $\ell \in X_j$ for some $j \in [n]$ where $b_j^* = 0$. In this case $\gamma_\ell = \tilde{\gamma}_\ell - ab \cdot \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}$. This means

$$c^\ell(\mathbf{m}_\ell^\intercal \mathbf{s} + \gamma_\ell) = c^\ell(\mathbf{m}_\ell^\intercal(\tilde{\mathbf{s}} + ab\tilde{\mathbf{w}}) + \tilde{\gamma}_\ell - ab\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}) = c^\ell(\mathbf{m}_\ell^\intercal \tilde{\mathbf{s}} + \tilde{\gamma}_\ell) = c^\ell(\mathbf{m}_\ell^\intercal \tilde{\mathbf{s}} + \tilde{\zeta}_\ell).$$

In combination with Eq. (E.9), we now have

$$\hat{z}^t \cdot \prod_{\substack{j\in[n]\\ b_j^*=0}} \hat{y}_j^t = \left(\prod_{\ell\in[2N-1]} \hat{g}^{c^\ell \mathbf{m}_\ell^\intercal \mathbf{s}} \cdot \prod_{\ell\in[L]} \hat{v}'_{\ell,\ell} \cdot \prod_{\substack{j\in[n]\\ b_j^*=0}} \prod_{k\in X_j} \hat{g}^{c^k \gamma_k} \right)^t = \prod_{\ell\in[2N-1]} \hat{g}^{t c^\ell (\mathbf{m}_\ell^\intercal \tilde{\mathbf{s}} + \tilde{\zeta}_\ell)} = \hat{c}_4^*.$$

When $T = T^*$ (in which case, $b_n^* \cdots b_1^*$ is the binary representation of $L - T$), this coincides with the distribution of $\hat{c}_4^*$ in $\mathsf{Hyb}_2^{(b)}$ and $\mathsf{Hyb}_3^{(b)}$.

We conclude that if $X = e(g,\hat{g})^{abt}$, then algorithm $\mathcal{B}$ simulates ct^* according to the distribution in $\mathsf{Hyb}_2^{(b)}$, whereas if $X \xleftarrow{R} \mathbb{G}_T$, then algorithm $\mathcal{B}$ simulates ct^* according to the distribution in $\mathsf{Hyb}_3^{(b)}$. Correspondingly, algorithm $\mathcal{B}$ breaks the N-extended bilinear Diffie-Hellman exponent assumption with the same advantage ε and the claim holds. □

Lemma E.4. *For all adversaries $\mathcal{A}$, $\Pr[\mathsf{Hyb}_3^{(0)}(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_3^{(1)}(\mathcal{A}) = 1]$.*

Proof. These are identical distributions by definition so the claim holds. □

Since $\mathcal{A}$ is computationally-bounded, it makes at most $Q_2 = \mathsf{poly}(\lambda)$ partial decryption queries in the security game. Since $p > 2^\lambda$, this means $Q_2/p = \mathsf{negl}(\lambda)$. By Lemmas E.2 to E.4 and a hybrid argument, this means for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \in \mathbb{N}$,

$$\left|\Pr[\mathsf{Hyb}_1^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(1)}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

By Lemma E.1, this means

$$\left|\Pr[\mathsf{Hyb}_0^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_0^{(1)}(\mathcal{A}) = 1]\right| = Q_1(\lambda) \cdot \mathsf{negl}(\lambda).$$

Since Q_1 is polynomially-bounded, we conclude that $\mathsf{Hyb}_0^{(0)}(\mathcal{A})$ and $\mathsf{Hyb}_1^{(1)}(\mathcal{A})$ are computationally indistinguishable and tag-based CCA-security follows. □