Full Domain Hash from (Leveled) Multilinear Maps and Identity-Based Aggregate Signatures

Susan Hohenberger * Johns Hopkins University

Amit Sahai † UCLA

Brent Waters ‡ University of Texas at Austin

July 30, 2013

Abstract

In this work, we explore building constructions with full domain hash structure, but with standard model proofs that do not employ the random oracle heuristic. The launching point for our results will be the utilization of a "leveled" *multilinear map* setting for which Garg, Gentry, and Halevi (GGH) recently gave an approximate candidate. Our first step is the creation of a standard model signature scheme that exhibits the structure of the Boneh, Lynn and Shacham signatures. In particular, this gives us a signature that admits unrestricted aggregation.

We build on this result to offer the first *identity-based* aggregate signature scheme that admits unrestricted aggregation. In our construction, an arbitrary-sized set of signatures on identity/message pairs can be aggregated into a single group element, which authenticates the entire set. The identity-based setting has important advantages over regular aggregate signatures in that it eliminates the considerable burden of having to store, retrieve or verify a set of verification keys, and minimizes the total cryptographic overhead that must be attached to a set of signer/message pairs. While identity-based signatures are trivial to achieve, their aggregate counterparts are not. To the best of our knowledge, no prior candidate for realizing unrestricted identity-based aggregate signatures exists in either the standard or random oracle models.

A key technical idea underlying these results is the realization of a hash function with a Naor-Reingold-type structure that is publicly computable using repeated application of the multilinear map. We present our results in a generic "leveled" multilinear map setting and then show how they can be translated to the GGH graded algebras analogue of multilinear maps.

*Supported by the National Science Foundation (NSF) CNS-1154035, CNS-1228443; the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211, DARPA N11AP20006, the Office of Naval Research under contract N00014-11-1-0470, and a Microsoft Faculty Fellowship.

†Research supported in part from a DARPA/ONR PROCEED award, NSF grants 1228984, 1136174, 1118096, 1065276, 0916574 and 0830803, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014-11-1-0389. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

‡Supported by NSF CNS-0915361, CNS-0952692, CNS-1228599; DARPA through the U.S. Office of Naval Research under Contract N00014-11-1-0382, DARPA N11AP20006, a Google Faculty Research Award, an Alfred P. Sloan Fellowship, a Microsoft Faculty Fellowship, and a Packard Foundation Fellowship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Department of Defense or the U.S. Government.

1 Introduction

Applying a full domain hash is a technique introduced by Bellare and Rogaway [4, 5] where a hash function, modeled as a random oracle, is used to hash a string into the full domain of a set. Originally, the concept referred to a signature scheme where one hashed into the range of a trapdoor permutation [4]. Subsequently, full domain hash has been treated as a more general concept and applied in bilinear map cryptography where typically a hash function $H : \{0,1\}^* \rightarrow \mathbb{G}$ is used to hash a string into a bilinear group. (We note that multiple early works [12, 14, 13] employ this terminology.) Pairing-based applications of Full Domain Hash include: the original Boneh-Franklin [12], short and aggregate signatures [14, 13], Hierarchical Identity-Based Encryption [26], and decentralized Attribute-Based Encryption [31]. Typically, proofs of such schemes will use the random oracle heuristic to "program" the output of the hash function in a certain way for which there is no known standard model equivalent (see [29]).

Given that there are well-known issues with random oracle instantiability in general [17] and problems with Full Domain Hash in particular [21, 20], there has been a push to find standard model realizations of these applications. These endeavors have been successful in several applications such as signatures [10, 41] and (Hierarchical) Identity-Based Encryption [18, 8, 9, 41, 24, 42]. Despite this progress, the current state is not entirely satisfactory on two fronts. First, each of the standard model examples given above created new cryptographic constructions with fundamentally different structure than the original Full Domain Hash construction. While creating a new structure is a completely valid and novel approach, that path does not necessarily lend insight or further understanding of the original constructions.

Second, there are important applications of the Full Domain Hash method where implementing such a hash using a random oracle introduces significant limitations in the applicability of the Full Domain Hash method. A prominent example concerns aggregate signature schemes and their identity-based counterparts:

An aggregate signature system is one in which a signature σ' on verification key/message pair (VK', M') can be combined with a signature $\tilde{\sigma}$ on $(\tilde{\mathrm{VK}}, \tilde{M})$ producing a new signature σ on the set $S = \{(\mathrm{VK}', M'), (\tilde{\mathrm{VK}}, \tilde{M})\}$. This process can be repeated indefinitely to aggregrate an arbitrary number of signatures together. Crucially, the size of σ should be independent of the number of signatures aggregated, although the description of the set S will grow. The ultimate goal, however, is to minimize the entire transmission size [36].

The need for a public-key infrastructure for verification keys is a major drawback of traditional public-key cryptography, and for this reason identity-based cryptography has flourished [40, 12]: In an *identity-based* aggregate signature scheme, verification keys like VK would be replaced with simple identity strings like $\mathcal{I}$ ="harrypotter@hogwarts.edu". This offers a very meaningful savings for protocols such as BGPsec, which require routers to store, retrieve and verify certificates for over 36,000 public keys [19, 16]. We note that while identity-based signatures follow trivially from standard signatures, identity-based aggregate signatures are nontrivial (more on this below).

A decade ago, the Boneh, Gentry, Lynn and Shacham (BGLS) [13] aggregate signature scheme was built using the Full Domain Hash methodology. In the original vision of BGLS, aggregation could be performed by any third party on any number of signatures. The authors showed how the Boneh, Lynn and Shacham (BLS) [14] signatures (which are in turn comprised of Boneh-Franklin [12] private IBE keys) can be aggregated in this manner. The BLS construction uses a full domain hash and its security proof is in the random oracle model. However, even though the BGLS scheme was built upon the key mechanism for Boneh-Franklin Identity-Based Encryption, BGLS

does *not* support identity-based aggregation. The Full Domain Hash in BGLS is realized using a random oracle, which destroys the structure that would be needed for identity-based aggregate signatures. To the best of our knowledge, no prior solution to identity-based aggregate signatures in either the standard or random oracle models exists. Prior work considered ID-based aggregates restricted to a common nonce [25] (e.g., where signatures can only be aggregated if they were created with the same nonce or time period) or sequential additions [7] (e.g., where a group of signers sequentially form an aggregate by each adding their own signature to the aggregate-so-far).

Our results in a nutshell. In this work, we give a new method for implementing the Full Domain Hash method using leveled multilinear maps, including the ones recently proposed by Garg, Gentry, and Halevi (GGH) [22]. We show how to use this method to implement aggregate signatures in the standard model in a way that naturally extends to give the first full solution to the problem of identity-based aggregate signatures (also in the standard model).

Prior work on standard model aggregate signatures. All previous work on achieving standard model aggregate signatures did so by departing fundamentally from the Full Domain Hash methodology.

Subsequently to BGLS [13], different standard model solutions were proposed, but with different restrictions on aggregation. These include: constructions [32] where the signatures must be sequentially added in by the signers, multisignatures [32] where aggregation can occur only for the same message M, or where aggregation is limited to signatures associated with the same nonce or time period [1].[1] These restrictions limit their practical applicability.

In 2009, Rückert and Schröder [39] gave an intriguing vision on how multilinear maps might enable standard model constructions of aggregate signatures, also departing from the Full Domain Hash methodology. They did not discuss or achieve identity-based aggregate signatures. Their proposal came before the Garg, Gentry and Halevi [22] candidate and used the earlier Boneh-Silverberg [15] view of multilinear maps, where a k-linear map would allow the *simultaneous* multiplication of k source group elements into one target group element. The GGH candidate in contrast allows for encodings to exist on multiple levels and a pairing between an encoding on level i and one on level j gives an encoding on level $i + j$ as long as $i + j$ is less than or equal to some k. One drawback of the Rückert and Schröder construction is that the security proof requires access to an interactive (or oracle-type) assumption in order to answer the signature queries where the structure of the oracle output is essentially identical to the signatures required. This property seems to be tightly coupled with the modeling of a multilinear map as a one time multiplication. In contrast, we will exploit the leveling of the GGH abstraction to actually replace the hash function in a BLS-type structure and obtain proofs from non-interactive assumptions.

1.1 Overview of our Aggregate Signature Constructions

We now overview the constructions and their security claims. To simplify the description of the main ideas, we describe the constructions here in terms of leveled multilinear maps. Later on, we explicitly give translations of both constructions to the GGH framework.

[1] We remark that these restrictions were considered in other works such as [38, 37, 34, 6, 33] prior to the standard model constructions cited above.

The Base Construction. A trusted setup algorithm will take as input security parameter λ and message bit-length ℓ and run a group generator $\mathcal{G}(1^\lambda, k = \ell + 1)$ and outputs a sequence of groups $\vec{\mathbb{G}} = (\mathbb{G}_1, \ldots, \mathbb{G}_k)$ of prime order p.[2] The group sequence will have canonical generators $g = g_1, g_2, \ldots, g_k$ along with a pairing operation that computes $e(g_i^a, g_j^b) = g_{i+j}^{ab}$ for any $a, b \in \mathbb{Z}_p$ and $i + j \leq k$. The setup algorithm will also choose $\vec{A} = (A_{1,0} = g^{a_{1,0}}, A_{1,1} = g^{a_{1,1}}), \ldots, (A_{\ell,0} = g^{a_{\ell,0}}, A_{\ell,1} = g^{a_{\ell,1}}) \in \mathbb{G}_1^2$. We define $H : \{0,1\}^\ell \to \mathbb{G}_{k-1}$ as $H(M) = g_{k-1}^{\prod_{i \in [1,\ell]} a_{i,m_i}}$, where m_i are the bits of message M. The hash function hashes a message into the group $\mathbb{G}_{k-1}$. It exhibits a Naor-Reingold [35]-type structure and is publicly computable using repeated application of a multilinear map. Since a group element in $\mathbb{G}_{k-1}$ has one pairing left, it intuitively reflects the bilinear map setting. In our scheme a private key contains a random exponent $\alpha \in \mathbb{Z}_p$ and the corresponding verification key VK contains g^α. A signature on a message M is computed as $\sigma = H(M)^\alpha$ and verified by testing $e(\sigma, g) \stackrel{?}{=} e(H(M), g^\alpha)$.

Stepping back, the structure of our scheme very closely resembles BLS signatures. For this reason it is possible to aggregate them in the BGLS fashion by simply multiplying two together. The size of an aggregate signature depends on the security parameter plus message length ℓ (assuming the group representation size increases with $k = \ell + 1$), but is independent of the number of times aggregation is applied. Aggregation is unrestricted and can be done by any third party.

The Rückert and Schröder construction [39] also insightfully uses a Naor-Reingold type function for aggregation. A key distinction is that in the RS method there is a *unique* NR function for each signer and it is privately computed by each signer per each message/input. In our construction the Naor-Reingold function is computed as a *public* hash using the levels of the multilinear map. A signer simply multiplies in his secret exponent after computing the hash. Thus, this mimicks the BLS structure much more closely. One advantage of our structure is that the hash function can be derived from a single common reference string and then public keys are just a single group element. In addition, we will see that our structure is amenable to proofs under non-interactive assumptions and allow us to extend to the identity-based setting. In the aggregation setting, where bandwidth is at a premium, our smaller public keys and the ability to go identity-based is important.

Proofs of Security. We view our aggregate signatures as signatures on a multiset of message/verification key pairs for full generality. We prove security in a modular way as a two step process. First, we define a weaker "distinct message" variant of security that only considers an attacker successful if the aggregate forgery no two signers sign the same message. We then show how to transform any distinct message secure scheme into one with standard security. The transformation captures the BGLS idea (formalized by Bellare, Namprempre and Neven [2]) of hashing the public key plus message together. Using the transformation we can focus on designing proofs in the distinct message game. We first prove selective security under a natural analog of the CDH assumption we call the k-Multilinear Computational Diffie-Hellman (k-MCDH) assumption. We next show full (a.k.a., adaptive) security using a subexponentially secure version of the assumption. Finally, we show full security with only polynomial factors in the reduction using a non-interactive, but parameterized assumption.

Realizing Identity-Based Aggregation. The authority will run a setup algorithm that takes the message bit-length ℓ and identity bit-length n. It runs a group generator $\mathcal{G}(1^\lambda, k = \ell + n)$

[2] In practice one will perform a CRHF of an arbitrary length message to ℓ bits.

and outputs a sequence of groups $\vec{\mathbb{G}} = (\mathbb{G}_1, \ldots, \mathbb{G}_k)$ of prime order p. It creates the parameters $\vec{A}$ as in the prior scheme and $\vec{B} = (B_{1,0} = g^{b_{1,0}}, B_{1,1} = g^{b_{1,1}}), \ldots, (B_{n,0} = g^{b_{n,0}}, B_{n,1} = g^{b_{n,1}}) \in \mathbb{G}_1^2$. We define $H : \{0,1\}^n \times \{0,1\}^\ell \to \mathbb{G}_{k-1}$ as $H(\mathcal{I}, M) = g_k^{(\prod_{i\in[1,n]} b_{i,\mathrm{id}_i})(\prod_{i\in[1,\ell]} a_{i,m_i})}$, where m_i are the bits of message M and id_i the bits of $\mathcal{I}$. The hash function is publicly computable from the multilinear map. A secret key for identity $\mathcal{I}$ is computed as $\mathrm{SK}_{\mathcal{I}} = g_{n-1}^{\prod_{i\in[1,n]} b_{i,\mathrm{id}_i}} \in G_{n-1}$. This can be used to produce a signature on message M by computing $(g_{k-1})^{(\prod_{i\in[1,n]} b_{i,\mathrm{id}_i})(\prod_{i\in[1,\ell]} a_{i,m_i})}$ using the multilinear map. Finally, a signature can be verified by checking $e(\sigma, g) \stackrel{?}{=} H(\mathcal{I}, M)$. The signatures will aggregate in the same manner by multiplying together.

The distinct message translation is not required in the identity-based setting, because there is no rogue key problem. We first prove selective security under the k-MCDH assumption, and then show full security using a subexponentially secure version of the assumption. We provide these proofs in both the generic multilinear and the GGH framework.

Further Applications. Taken altogether we show that multilinear forms provide an opportunity for revisiting cryptographic structures that were strongly associated with the random oracle heuristic. It remains to be seen how widely this direction will apply. One interesting example of an application that currently requires the full domain hash is the decentralized Attribute-Based Encryption system of Lewko and Waters [31]. There is no standard model candidate that has comparable expressiveness. Here performing an analogous transformation to our aggregate signatures hash function gives a candidate construction that we do not immediately see how to break. However, it is less easy to see how our proof techniques would extend to the variant of the Lewko-Waters [31] decentralized ABE scheme.

2 Leveled Multilinear Maps and the GGH Graded Encoding

We give a description of generic, leveled multilinear maps. The assumptions used in this setting are defined inline with their respective security proofs. More details of the GGH graded algebras analogue of mulitlinear maps are included in Appendix A, and for further details, please refer to [22].

For generic, leveled multilinear maps, we assume the existence of a group generator $\mathcal{G}$, which takes as input a security parameter 1^λ and a positive integer k to indicate the number of allowed pairing operations. $\mathcal{G}(1^\lambda, k)$ outputs a sequence of groups $\vec{G} = (\mathbb{G}_1, \ldots, \mathbb{G}_k)$ each of large prime order $p > 2^\lambda$. In addition, we let g_i be a canonical generator of $\mathbb{G}_i$ (and is known from the group's description). We let $g = g_1$.

We assume the existence of a set of bilinear maps $\{e_{i,j} : G_i \times G_j \to G_{i+j} \mid i, j \geq 1;\ i + j \leq k\}$. The map $e_{i,j}$ satisfies the following relation:

$$e_{i,j}\left(g_i^a, g_j^b\right) = g_{i+j}^{ab} \ : \ \forall a, b \in \mathbb{Z}_p$$

We observe that one consequence of this is that $e_{i,j}(g_i, g_j) = g_{i+j}$ for each valid i, j.

When the context is obvious, we will sometimes abuse notation and drop the subscripts i, j, For example, we may simply write $e\left(g_i^a, g_j^b\right) = g_{i+j}^{ab}$.

Algorithmic components of GGH encodings. While we assume familiarity with the basics of GGH encodings [22], we now review the algorithmic components of the GGH encodings that we will use in our constructions and proofs. The setup algorithm $\mathsf{InstGen}(1^\lambda, 1^k)$ takes as input a security parameter 1^λ and the level of multilinearity 1^k, and outputs the public parameters params needed for using the remaining GGH algorithms, along with a special parameter $\mathbf{p}_{zt}$ to be used for zero testing. The sampling algorithm $\mathsf{samp}(\mathsf{params})$ outputs a level-0 encoding of a randomly chosen element. The canonicalizing encoding $\mathsf{cenc}_e(\mathsf{params}, i, \alpha)$ algorithm takes as input an encoding α of some element a, and outputs a level-i encoding of a, with re-randomization parameter e. This canonicalizing encoding algorithm can re-randomize an encoding for a fixed constant number of re-randomization parameters e. Finally, the zero-testing algorithm $\mathsf{isZero}(\mathbf{p}_{zt}, \alpha)$ takes as input a level-k encoding α, and accepts iff α is an encoding of 0. A more elaborate review of these algorithms can be found in Appendix A.

3 Definitions for Aggregate and ID-based Aggregate Signatures

We now give our definitions for aggregate signatures. In our setting, each aggregate signature is associated with a *multiset* S over verification key/message pairs (or identity/message pairs in the ID-based setting). A set S is of the form $\{(\mathrm{VK}_1, M_1), \ldots, (\mathrm{VK}_{|S|}, M_{|S|})\}$. Since S is a multiset it is possible to have $(\mathrm{VK}_i, M_i) = (\mathrm{VK}_j, M_j)$ for $i \neq j$. All signatures, including those that come out of the sign algorithm, are considered to be aggregate signatures. The aggregation algorithm is general in that it can take any two aggregate signatures and combine them into a new aggregate signature.

Our definition allows for an initial trusted setup that will generate a set of common public parameters PP. This will define a bit length of all messages (and identities). In practice one could set these fixed lengths to be the output length ℓ of a collision resistant hash function and allow arbitrary-length messages/identities by first hashing them down to ℓ bits. In the ID-based setting, the authority also produces a master secret key used later to run the key generation algorithm.

We emphasize a few features of our setting. First, aggregation is very general in that it allows for the combination of any two aggregate signatures into a single one. Some prior definitions required an aggregate signature to be combined with a single message signature. This is a limitation for applications where an aggregator comes across two aggregate signatures that is wishes to combine. The aggregation operation does not require any secret keys. The multiset structure allows one to combine two aggregate signatures which both include the same message from the same signer.

We begin formally with the ID-based definition, because it is novel to this work, and then discuss its simpler counterpart.

Authority-Setup$(1^\lambda, \ell, n)$ The trusted setup algorithm takes as input the security parameter as well the bit-length ℓ of messages and bit-length n of the identities. It outputs a common set of public parameters PP and master secret key MSK.

KeyGen$(\mathrm{MSK}, \mathcal{I} \in \{0,1\}^n)$ The key generation algorithm is run by the authority. It takes as input the system master secret key and an identity $\mathcal{I}$, and outputs a secret signing key $\mathrm{SK}_\mathcal{I}$.

Sign$(\mathrm{PP}, \mathrm{SK}_\mathcal{I}, \mathcal{I} \in \{0,1\}^n, M \in \{0,1\}^\ell)$ The signing algorithm takes as input a secret signing key and corresponding identity $\mathcal{I} \in \{0,1\}^n$, the common public parameters as well as a message

$M \in \{0,1\}^\ell$. It outputs a signature σ for identity $\mathcal{I}$. *We emphasize that a single signature that is output by this algorithm is considered to also be an aggregate signature.*

Aggregate$(\mathrm{PP}, \tilde{S}, S', \tilde{\sigma}, \sigma')$**.** The aggregation algorithm takes as input two multisets $\tilde{S}$ and S' and purported signatures $\tilde{\sigma}$ and σ'. The elements of $\tilde{S}$ consist of identity/message pairs $\{(\tilde{\mathcal{I}}_1, \tilde{M}_1), \ldots, (\tilde{\mathcal{I}}_{|\tilde{S}|}, \tilde{M}_{|\tilde{S}|})\}$ and the elements of S' consist of $\{(\mathcal{I}'_1, M'_1), \ldots, (\mathcal{I}'_{|S'|}, M'_{|S'|})\}$. The process produces a signature σ on the multiset $S = \tilde{S} \cup S'$, where $\cup$ is a multiset union.

Verify(PP, S, σ)**.** The verification algorithm takes as input the public parameters, a multiset S of identity and message pairs and an aggregate signature σ. It outputs true or false to indicate whether verification succeeded.

Correctness The correctness property states that all valid aggregate signatures will pass the verification algorithm, where a valid aggregate is defined recursively as an aggregate signature derived by an application of the aggregation algorithm on two valid inputs or the signing algorithm. More formally, for all integers $\lambda, \ell, n, k \geq 1$, all $\mathrm{PP} \in \text{Authority-Setup}(1^\lambda, \ell, n)$, all $\mathcal{I}_1, \ldots, \mathcal{I}_k \in \{0,1\}^n$, all $\mathrm{SK}_{\mathcal{I}_i} \in \mathrm{KeyGen}(\mathrm{PP}, \mathcal{I}_i)$, $\mathrm{Verify}(\mathrm{PP}, S, \sigma) = 1$, if σ is a *valid* aggregate for multiset S under PP. We say that an aggregate signature σ is *valid* for multiset S if: (1) $S = \{(\mathcal{I}_i, M)\}$ for some $i \in [1,k]$, $M \in \{0,1\}^\ell$ and $\sigma \in \mathrm{Sign}(\mathrm{PP}, \mathrm{SK}_{\mathcal{I}_i}, \mathcal{I}_i, M)$; or (2) there exists multisets $S', \tilde{S}$ where $S = S' \cup \tilde{S}$ and valid aggregate signatures $\sigma', \tilde{\sigma}$ on them respectively such that $\sigma \in \mathrm{Aggregate}(\mathrm{PP}, \tilde{S}, S', \tilde{\sigma}, \sigma')$.

Security Model for Aggregate Signatures. Adapting aggregation [13, 2] to the identity-based setting takes some care in considering how keys are handled and which query requests the adversary should be allowed to make. Informally, in the unforgeability game, it should be computationally infeasible for any adversary to produce a forgery implicating an honest identity, even when the adversary can control all other identities involved in the aggregate and can mount a chosen-message attack on the honest identity. This is defined using a game between a challenger and an adversary $\mathcal{A}$ with respect to scheme $\Pi =$ (Authority-Setup, KeyGen, Sign, Aggregate, Verify).

– **ID-Unforg**$(\Pi, \mathcal{A}, \lambda, \ell, n)$:

Setup. The challenger runs $\text{Authority-Setup}(1^\lambda, \ell, n)$ to obtain PP. It sends PP to $\mathcal{A}$.
Queries. Proceeding adaptively, $\mathcal{A}$ can make three types of requests:

1. Create New Key: The challenger begins with an index $i = 1$ and an empty sequence of index/identity/private key triples T. On input an identity $\mathcal{I} \in \{0,1\}^n$, the challenger runs $\mathrm{KeyGen}(\mathrm{MSK}, \mathcal{I})$ to obtain $\mathrm{SK}_{\mathcal{I}}$. It adds the triple $(i, \mathcal{I}, \mathrm{SK}_{\mathcal{I}})$ to T and then increments i for the next call. Nothing is returned to the adversary. *We note that the adversary can query this oracle multiple times for the same identity. This will capture security for applications that might release more than one secret key per identity.*
2. Corrupt User: On input an index $i \in [1, |T|]$, the challenger returns to the adversary the triple $(i, \mathcal{I}_i, \mathrm{SK}_{\mathcal{I}_i}) \in T$. It returns an error if T is empty or i is out of range.
3. Sign: On input an index $i \in [1, |T|]$ and a message $M \in \{0,1\}^\ell$, the challenger obtains the triple $(i, \mathcal{I}_i, \mathrm{SK}_{\mathcal{I}_i}) \in T$ (returning an error if it does not exist) and returns the signature resulting from $\mathrm{Sign}(\mathrm{PP}, \mathrm{SK}_{\mathcal{I}_i}, \mathcal{I}_i, M)$ to $\mathcal{A}$.

Response. Finally, $\mathcal{A}$ outputs a multiset S^* of identity/message pairs and a purported aggregate signature σ^*.

We say the adversary "wins" or that the output of this experiment is 1 if: (1) $\text{Verify}(\text{PP}, S^*, \sigma^*) = 1$ and (2) there exists an element $(\mathcal{I}^*, M^*) \in S^*$ such that M^* was not queried for a signature by the adversary on any index corresponding to $\mathcal{I}^*$; i.e., any index i such that $(i, \mathcal{I}^*, \cdot) \in T$. Otherwise, the output is 0. Define $\textbf{ID-Forg}_{\mathcal{A}}$ as the probability that $\textbf{Unforg}(\Pi, \mathcal{A}, \lambda, \ell, n) = 1$, where the probability is over the coin tosses of the Authority-Setup, KeyGen, and Sign algorithms and of $\mathcal{A}$.

Definition 3.1 (Adaptive Unforgeability) *An ID-based aggregate signature scheme* Π *is existentially* unforgeable *with respect to adaptive chosen-message attacks if for all probabilistic polynomial-time adversaries* $\mathcal{A}$*, the function* $\textbf{ID-Forg}_A$ *is negligible in* λ*.*

Selective Security. We consider a selective variant to **ID-Unforg** (selective in both the identity and the message) where there is an Init phase before the Setup phase, wherein $\mathcal{A}$ gives to the challenger a forgery identity/message pair $(\mathcal{I}^* \in \{0,1\}^n, M^* \in \{0,1\}^\ell)$. The adversary cannot request a signing key for $\mathcal{I}^*$. (It may request that the challenger create one or more keys for this identity, but it cannot corrupt any user index i associated with $\mathcal{I}^*$.) Moreover, the adversary only "wins" causing the experiment output to be 1 if the normal checks hold (i.e., its signature verifies and it did not request that $\mathcal{I}^*$ sign M^*) and additionally $(\mathcal{I}^*, M^*)$ appears in S^*.

Non-ID-Based Aggregates and the Distinct Message Variant. We provide security definitions for the non-ID-based setting in Appendix B that follow from [13, 2]. We provide adaptive and selective variants. We also identify a weaker "distinct message" security game that is easier to work with. In Appendix C, we describe and prove secure a simple transformation from distinct message security to standard aggregate signature security. The transformation captures the idea of hashing the public key and message together [13, 2] in a modular way. Focusing on distinct message security allows one to avoid the "rogue key" attack (see Section 4.3). We do not consider distinct message security in the ID-based setting, because there are no verification keys.

4 Our Base Aggregate Signature Construction

4.1 Generic Multlinear Construction

Setup$(1^\lambda, \ell)$ The trusted setup algorithm takes as input the security parameter as well as the length ℓ of messages. It first runs $\mathcal{G}(1^\lambda, k = \ell+1)$ and outputs a sequence of groups $\vec{\mathbb{G}} = (\mathbb{G}_1, \ldots, \mathbb{G}_k)$ of prime order p, with canonical generators $g_1, \ldots, g_k$, where we let $g = g_1$.

Next, it outputs random group elements $(A_{1,0}, A_{1,1}), \ldots, (A_{\ell,0}, A_{\ell,1}) \in \mathbb{G}_1^2$. These will be used to compute a function $H(M) : \{0,1\}^\ell \to \mathbb{G}_{k-1}$, which serves as the analog of the full domain hash function of the BGLS [13] construction. Let $m_1, \ldots, m_\ell$ be the bits of message M. It is computed iteratively as $H_1(M) = A_{1,m_1}$ and for $i \in [2, \ell]$, $H_i(M) = e(H_{i-1}(M), A_{i,m_i})$. We define $H(M) = H_\ell(M)$. The public parameters, PP, consist of the group descriptions plus $(A_{1,0}, A_{1,1}), \ldots, (A_{\ell,0}, A_{\ell,1})$.

KeyGen(PP) The key generation algorithm first chooses random $\alpha \in \mathbb{Z}_p$. It outputs the public verification key as $\text{VK} = g^\alpha$. The secret key SK is $\alpha \in \mathbb{Z}_p$.

Sign$(\mathrm{PP}, \mathrm{SK}, M \in \{0,1\}^\ell)$ The signing algorithm computes the signature as $\sigma = H(M)^\alpha \in G_{k-1}$. This serves as an aggregate signature for the (single element) multiset $S = (\mathrm{VK}, M)$.

Aggregate$(\mathrm{PP}, \tilde{S}, S', \tilde{\sigma}, \sigma')$**.** The aggregation algorithm simply computes the output signature σ as $\sigma = \tilde{\sigma} \cdot \sigma'$. The serves as a signature on the multiset $S = \tilde{S} \cup S'$, where $\cup$ is a *multiset union*.

Verify(PP, S, σ)**.** The verification algorithm parses S as $\{(\mathrm{VK}_1, M_1), \ldots, (\mathrm{VK}_{|S|}, M_{|S|})\}$. It then checks that $e(\sigma, g) \stackrel{?}{=} \prod_{i=1,\ldots,|S|} e(H(M_i), \mathrm{VK}_i)$ and accepts if and only if it holds.

Correctness To see correctness, an aggregate σ on $S = \{(\mathrm{VK}_1, M_1), \ldots, (\mathrm{VK}_{|S|}, M_{|S|}\}$ will be the product of individual signatures; i.e., $\sigma = \prod_{i=1}^{|S|} H(M_i)^{\alpha_i}$ where $\mathrm{VK}_i = g^{\alpha_i}$, and thus will pass the verification equation as $e(\sigma, g) = e(\prod_{i=1}^{|S|} H(M_i)^{\alpha_i}, g) = \prod_{i=1}^{|S|} e(H(M_i)^{\alpha_i}, g) = \prod_{i=1}^{|S|} e(H(M_i), g)^{\alpha_i} = \prod_{i=1}^{|S|} e(H(M_i), g^{\alpha_i}) = \prod_{i=1}^{|S|} e(H(M_i), VK_i)$.

Efficiency and Tradeoffs An aggregate signature is one group element in $\mathbb{G}_{k-1}$ independent of the number of messages aggregated. In a multilinear setting, the space to represent a group element might grow with k (which is $\ell+1$). Indeed, this happens in the GGH [22] graded algebra translation. One way to mitigate this is to differ the message alphabet size in a tradeoff of computation versus storage. The above construction uses a binary message alphabet. If it used an alphabet of 2^d symbols, then the aggregate signature could resident in the group $G_{\ell/d}$ with $\ell/d - 1$ pairings required to compute it, at the cost of the public parameters requiring $2^d\ell$ group elements in $\mathbb{G}$.

4.2 Construction in the GGH Framework

We give a translation of the above construction to the GGH [22] framework in Appendix E.

4.3 Security Analysis

Assumption 4.1 (Multilinear Computational Diffie-Hellman: k-MCDH) *The k-Multilinear Computational Diffie-Hellman (k-MCDH) problem states the following: A challenger runs $\mathcal{G}(1^\lambda, k)$ to generate groups and generators of order p. Then it picks random $c_1, \ldots, c_k \in \mathbb{Z}_p$. The assumption then states that given $g = g_1, g^{c_1}, \ldots, g^{c_k}$ it is hard for any poly-time algorithm to compute $g_{k-1}^{\prod_{j\in[1,k]} c_j}$ with better than negligible advantage (in security parameter λ).*

We say that the k-MCDH assumption holds against subexponential advantage if there exists a universal constant $\epsilon_0 > 0$ such that no polynomial-time algorithm can succeed in the experiment above with probability greater than $2^{-\lambda^{\epsilon_0}}$. In Section 5.3, we will give a variant of the k-MCDH assumption in the approximate multilinear maps setting of GGH [22] that we will call the GGH k-MCDH assumption. We note that the best cryptanalysis available of the GGH framework [22] suggests that the GGH k-MCDH assumption holds against subexponential advantage.

In Appendix D, we show that the basic aggregate signature scheme for message length ℓ in the distinct message unforgeability game is:

- (Theorem D.1) selectively secure under the $(\ell+1)$-Multilinear Computational Diffie-Hellman (MCDH) assumption.

- (Corollary D.2) fully secure under the $(\ell+1)$-MCDH assumption against subexponential advantage.
- (Theorem D.4) fully secure under a non-interactive, parameterized assumption which depends on ℓ, the number of adversarial signing queries and the number of messages in the adversary's forgery.

By applying the simple transformation of Appendix C, the distinct message requirement can be removed. Without this transformation, there is a simple attack where the attacker sets some $\mathrm{VK}' = \mathrm{VK}^{-1}$ and submits the identity element in $\mathbb{G}_{k-1}$ as an aggregate forgery for $S = \{(\mathrm{VK}, M), (\mathrm{VK}', M)\}$ for any message M of its choosing.

5 Our ID-Based Aggregate Signature Construction

5.1 Generic Multilinear Construction

Authority-Setup$(1^\lambda, \ell, n)$ The trusted setup algorithm is run by the master authority of the ID-based system. It takes as input the security parameter as well the bit-length ℓ of messages and bit-length n of identities. It first runs $\mathcal{G}(1^\lambda, k = \ell + n)$ and outputs a sequence of groups $\vec{\mathbb{G}} = (\mathbb{G}_1, \ldots, \mathbb{G}_k)$ of prime order p, with canonical generators $g_1, \ldots, g_k$, where we let $g = g_1$.

Next, it chooses random group elements $(A_{1,0} = g^{a_{1,0}}, A_{1,1} = g^{a_{1,1}}), \ldots, (A_{\ell,0} = g^{a_{\ell,0}}, A_{\ell,1} = g^{a_{\ell,1}}) \in \mathbb{G}_1^2$. It also chooses random exponents $(b_{1,0}, b_{1,1}), \ldots, (b_{n,0}, b_{n,1}) \in {\mathbb{Z}_p}^2$ and sets $B_{i,\beta} = g^{b_{i,\beta}}$ for $i \in [1, n]$ and $\beta \in \{0, 1\}$.

These will be used to define a function $H(\mathcal{I}, M) : \{0,1\}^n \times \{0,1\}^\ell \to \mathbb{G}_k$. Let $m_1, \ldots, m_\ell$ be the bits of message M and $\mathrm{id}_1, \ldots, \mathrm{id}_n$ as the bits of $\mathcal{I}$. It is computed iteratively as

$$H_1(\mathcal{I}, M) = B_{1,\mathrm{id}_1} \quad \text{for } i \in [2, n] \quad H_i(\mathcal{I}, M) = e(H_{i-1}(\mathcal{I}, M), B_{i,\mathrm{id}_i})$$

$$\text{for } i \in [n+1, n+\ell = k] \quad H_i(\mathcal{I}, M) = e(H_{i-1}(\mathcal{I}, M), A_{i-n,m_{i-n}}).$$

We define $H(\mathcal{I}, M) = H_{k=\ell+n}(\mathcal{I}, M)$.

The public parameters, PP, consist of the group sequence description plus:

$$(A_{1,0}, A_{1,1}), \ldots, (A_{\ell,0}, A_{\ell,1}), (B_{1,0}, B_{1,1}), \ldots, (B_{n,0}, B_{n,1})$$

The master secret key MSK includes PP together with the values $(b_{1,0}, b_{1,1}), \ldots, (b_{n,0}, b_{n,1})$.

KeyGen$(\mathrm{MSK}, \mathcal{I} \in \{0,1\}^n)$ The signing key for identity $\mathcal{I}$ is $\mathrm{SK}_\mathcal{I} = g_{n-1}^{\prod_{i \in [1,n]} b_{i,\mathrm{id}_i}} \in G_{n-1}$.

Sign$(\mathrm{PP}, \mathrm{SK}_\mathcal{I}, \mathcal{I} \in \{0,1\}^n, M \in \{0,1\}^\ell)$ The signing algorithm lets temporary variable $D_0 = \mathrm{SK}_\mathcal{I}$. Then for $i = 1$ to ℓ it computes $D_i = e(D_{i-1}, A_{i,m_i}) \in G_{n-1+i}$. The output signature is

$$\sigma = D_\ell = (g_{k-1})^{(\prod_{i \in [1,n]} b_{i,\mathrm{id}_i})(\prod_{i \in [1,\ell]} a_{i,m_i})}.$$

This serves as an ID-based aggregate signature for the (single element) multiset $S = (\mathcal{I}, M)$.

Aggregate$(\mathrm{PP}, \tilde{S}, S', \tilde{\sigma}, \sigma')$**.** The aggregation algorithm simply computes the output signature σ as $\sigma = \tilde{\sigma} \cdot \sigma'$. The serves as a signature on the multiset $S = \tilde{S} \cup S'$, where $\cup$ is a multiset union.

Verify(PP, S, σ)**.** It parses S as $\{(\mathcal{I}_1, M_1), \ldots, (\mathcal{I}_{|S|}, M_{|S|})\}$. It then accepts if and only if

$$e(\sigma, g) \stackrel{?}{=} \prod_{i=1,\ldots,|S|} H(\mathcal{I}_i, M_i).$$

Correctness and Security To see correctness, an aggregate σ on $S = \{(\mathcal{I}_1, M_1), \ldots, (\mathcal{I}_{|S|}, M_{|S|})$ will be the product of individual signatures; i.e., σ_i where $e(\sigma_i, g) = H(\mathcal{I}_i, M_i)$, and thus we have $\prod_{i=1}^{|S|} e(\sigma_i, g) = e(\prod_{i=1}^{|S|} \sigma_i, g) = e(\sigma, g) = \prod_{i=1}^{|S|} H(\mathcal{I}_i, M_i)$. We show that this scheme is selectively secure under the k-MCDH assumption in Appendix F, where the proof is very similar to that of the GGH translation which we provide shortly in Section 5.3.

5.2 ID-Based Construction in the GGH Framework

We show how to modify our ID-based construction to use the GGH [22] graded algebras analogue of multilinear maps. Please note that we use the same notation developed in [22], with some minor changes: Firstly, we use the canonical encoding function cenc provided by the GGH framework more than once at each level of the encoding, but only a globally fixed constant number of times per level. This is compatible with the GGH encoding [22], and allows for a simpler exposition of our scheme and proof. Also, **for ease of notation on the reader, we suppress repeated** params **arguments that are provided to every algorithm.** Thus, for instance, we will write $\alpha \leftarrow \mathsf{samp}()$ instead of $\alpha \leftarrow \mathsf{samp}(\mathsf{params})$. Note that in our scheme, there will only ever be a single uniquely chosen value for params throughout the scheme, so there is no cause for confusion. Finally, we use the variant of the GGH framework with "strong" zero-testing, where the zero test statistically guarantees that a vector is a valid encoding of zero if it passes the zero test. For further details on the GGH framework, please refer to [22]. See also the summary of [23] as included in Appendix A.

Authority-Setup$(1^\lambda, \ell, n)$ The trusted setup algorithm is run by the master authority of the ID-based system. It takes as input the security parameter as well the bit-length ℓ of messages and bit-length n of identities. It then runs $(\mathsf{params}, \mathbf{p}_{zt}) \leftarrow \mathsf{InstGen}(1^\lambda, 1^{k=\ell+n})$. Recall that params will be implicitly given as input to all GGH-related algorithms below.

Next, it chooses random encodings $a_{i,\beta} = \mathsf{samp}()$ for $i \in [1, \ell]$ and $\beta \in \{0, 1\}$; and random encodings $b_{i,\beta} = \mathsf{samp}()$ for $i \in [1, n]$ and $\beta \in \{0, 1\}$. Then it assigns $A_{i,\beta} = \mathsf{cenc}_1(1, a_{i,\beta})$ for $i \in [1, \ell]$ and $\beta \in \{0, 1\}$; and it assigns $B_{i,\beta} = \mathsf{cenc}_1(1, b_{i,\beta})$ for $i \in [1, n]$ and $\beta \in \{0, 1\}$.

These will be used to compute a function H mapping $\ell + n$ bit strings to level $k - 1$ encodings. Let $m_1, \ldots, m_\ell$ be the bits of M and $\mathrm{id}_1, \ldots, \mathrm{id}_n$ be the bits of $\mathcal{I}$. It is computed iteratively as

$$H_1(\mathcal{I}, M) = B_{1,\mathrm{id}_1} \quad \text{for } i \in [2, n] \;\; H_i(\mathcal{I}, M) = H_{i-1}(\mathcal{I}, M) \cdot B_{i,\mathrm{id}_i}$$

$$\text{for } i \in [n+1, n+\ell = k] \;\; H_i(\mathcal{I}, M) = H_{i-1}(\mathcal{I}, M) \cdot A_{i-n,m_{i-n}}.$$

We define $H(\mathcal{I}, M) = \mathsf{cenc}_2(k, H_{k=\ell+n}(\mathcal{I}, M))$.

The public parameters, PP, consist of the $\mathsf{params}, \mathbf{p}_{zt}$ plus:

$$(A_{1,0}, A_{1,1}), \ldots, (A_{\ell,0}, A_{\ell,1}), (B_{1,0}, B_{1,1}), \ldots, (B_{n,0}, B_{n,1})$$

Note that params includes a level 1 encoding of 1, which we denote as g.

The master secret key MSK includes PP together with the encodings $(b_{1,0}, b_{1,1}), \ldots, (b_{n,0}, b_{n,1})$.

KeyGen$(\mathrm{MSK}, \mathcal{I} \in \{0,1\}^n)$ The signing key for identity $\mathcal{I}$ is $\mathrm{SK}_{\mathcal{I}} = \mathsf{cenc}_2(n-1, \prod_{i\in[1,n]} b_{i,\mathrm{id}_i})$.

Sign$(\mathrm{PP}, \mathrm{SK}_{\mathcal{I}}, \mathcal{I} \in \{0,1\}^n, M \in \{0,1\}^\ell)$ The signing algorithm lets temporary variable $D_0 = \mathrm{SK}_{\mathcal{I}}$. Then for $i = 1$ to ℓ it computes $D_i = D_{i-1} \cdot A_{i,m_i}$. The output signature is

$$\sigma = \mathsf{cenc}_3(k-1, D_\ell).$$

This serves as an ID-based aggregate signature for the (single element) multiset $S = (\mathcal{I}, M)$.

Aggregate$(\mathrm{PP}, \tilde{S}, S', \tilde{\sigma}, \sigma')$**.** The aggregation algorithm simply computes the output signature σ as $\sigma = \tilde{\sigma} + \sigma'$. The serves as a signature on the multiset $S = \tilde{S} \cup S'$, where $\cup$ is a multiset union.

Verify(PP, S, σ)**.** The verification algorithm parses S as $\{(\mathcal{I}_1, M_1), \ldots, (\mathcal{I}_{|S|}, M_{|S|})\}$. It rejects if the multiplicity of any identity/message pair is greater than 2^λ.

The algorithm then proceeds to check the signature by setting $\tau = \mathsf{cenc}_2(1, g)$, and testing: :

$$\mathsf{isZero}\left(\mathbf{p}_{zt}, \tau \cdot \sigma - \sum_{i=1,\ldots,|S|} H(\mathcal{I}_i, M_i)\right)$$

and accepts if and only if the zero testing procedure outputs true. Recall that g above is a canonical level 1 encoding of 1 that is included in params, part of the public parameters.

Correctness. Correctness follows from the same argument as for the ID-based aggregate signature scheme in the generic multilinear setting.

5.3 Proof of Security for ID-based Aggregate Signatures in the GGH framework

We now describe how to modify our proof of security for our ID-based construction to use the GGH [22] graded algebras analogue of multilinear maps. As before, for ease of notation on the reader, we suppress repeated params arguments that are provided to every algorithm. For further details on the GGH framework, please refer to Appendix A or [22].

We begin by describing the GGH analogue of the k-MCDH assumption that we will employ:

Assumption 5.1 (GGH analogue of k-MCDH: GGH k-MCDH) *The GGH k-Multilinear Computational Diffie-Hellman (GGH k-MCDH) problem states the following: A challenger runs* $\mathsf{InstGen}(1^\lambda, 1^k)$ *to obtain* $(\mathsf{params}, \mathbf{p}_{zt})$*. Note that* params *includes a level 1 encoding of 1, which we denote as g. Then it picks random $c_1, \ldots, c_k$ each equal to the result of a fresh call to* $\mathsf{samp}()$*.*

The assumption then states that given $\mathsf{params}, \mathbf{p}_{zt}, \mathsf{cenc}_1(1, c_1), \ldots, \mathsf{cenc}_1(1, c_k)$ *it is hard for any poly-time algorithm to compute an integer $t \in [1, 2^\lambda]$ and an encoding z such that*

$$zTst\left(\mathbf{p}_{zt}, \mathsf{cenc}_2(1, g) \cdot z - \mathsf{cenc}_1(k, t \cdot \prod_{j\in[1,k]} c_j)\right)$$

outputs true.

We say the GGH k-MCDH assumption holds against subexponential advantage if there exists a universal constant $\epsilon_0 > 0$ such that no polynomial-time algorithm can succeed in the experiment above with probability greater than $2^{\lambda^{\epsilon_0}}$. The best cryptanalysis available of the GGH framework [22] suggests that the GGH k-MCDH assumption holds against subexponential advantage.

We establish full security of our ID-based aggregate signature scheme conditioned on the k-MCDH assumption holding against subexponential advantage. This follows immediately from the following theorem and a standard complexity leveraging argument:

Theorem 5.2 (Selective Security of GGH ID-Based Construction) *The ID-based aggregate signature scheme for message length ℓ and identity length n in Section 5.2 is selectively secure in the unforgeability game in Section 3 under the GGH $(\ell + n)$-MCDH assumption.*

Corollary 5.3 *The ID-based aggregate signature scheme for message length ℓ in Section 5.2 is fully secure in the distinct message unforgeability game under the GGH $(\ell + n)$-MCDH assumption against subexponential advantage.*

Proof. This follows immediately from a complexity leveraging argument: the security parameter λ is chosen to ensure that $2^{\lambda^{\epsilon_0}} >> 2^{\ell}$, where $2^{-\lambda^{\epsilon_0}}$ is the maximum probability of success allowed in the k-MCDH assumption against subexponential advantage. Now, to establish full security, the simulator performs exactly as in the selective security proof, but first it simply guesses the message that will be forged (instead of expecting the adversary to produce this message). Because this guess will be correct with probability at least $2^{-\ell}$, and the security parameter λ is chosen carefully, full security with polynomial advantage (or even appropriately defined subexponential advantage) implies an attacker on the GGH k-MCDH assumption with subexponential advantage. □

Proof. **(of Theorem 5.2)** We show that if there exists a PPT adversary $\mathcal{A}$ that can break the selective security of the ID-based aggregate signature scheme in the unforgettability game with probability ϵ for message length ℓ, identity length n and security parameter λ, then there exists a PPT simulator that can break the GGH $(\ell + n)$-MCDH assumption for security parameter λ with probability ϵ. The simulator takes as input a GGH MCDH instance $\mathsf{params}, \mathbf{p}_{zt}, C_1 = \mathsf{cenc}_1(1, c_1), \ldots, C_k = \mathsf{cenc}_1(1, c_k)$ where $k = \ell + n$. Let m_i denote the ith bit of M and id_i denote the ith bit of $\mathcal{I}$. The simulator plays the role of the challenger in the game as follows.

Init. Let $\mathcal{I}^* \in \{0,1\}^n$ and $M^* \in \{0,1\}^\ell$ be the forgery identity/message pair output by $\mathcal{A}$.

Setup. The simulator chooses random $x_1, \ldots, x_\ell, y_1, \ldots, y_n$ with fresh calls to $\mathsf{samp}()$. For $i = 1$ to ℓ, let $A_{i,m_i^*} = C_{i+n}$ and $A_{i,\bar{m_i^*}} = \mathsf{cenc}_1(1, x_i)$. For $i = 1$ to n, let $B_{i,\mathrm{id}_i^*} = C_i$ and $B_{i,\bar{\mathrm{id}_i^*}} = \mathsf{cenc}_1(1, y_i)$. We remark that the parameters are distributed independently and uniformly at random as in the real scheme.

Queries. Conceptually, the simulator will be able to create keys or signatures for the adversary, because his requests will differ from the challenge identity or message in at least one bit. More specifically,

1. Create New Key: The simulator begins with an index $i = 1$ and an empty sequence of index/identity/private key triples T. On input an identity $\mathcal{I} \in \{0,1\}^n$, if $\mathcal{I} = \mathcal{I}^*$, the simulator records $(i, \mathcal{I}^*, \perp)$ in T. Otherwise, the simulator computes the secret key as follows. Let β be the first index such that $\mathrm{id}_i \neq \mathrm{id}_i^*$. Compute $s = \prod_{i=1,\ldots,n \wedge i \neq \beta} B_{i,\mathrm{id}_i}$.

Then compute $\mathrm{SK}_{\mathcal{I}} = \mathsf{cenc}_2(n-1, s \cdot y_\beta)$. Record $(i, \mathcal{I}, \mathrm{SK}_{\mathcal{I}})$ in T. Secret keys are well-formed and, due to the rerandomization in the cenc_2 algorithm, are distributed in a manner statistically exponentially close to the keys generated in the real game.

2. Corrupt User: On input an index $i \in [1, |T|]$, the simulator returns to the adversary the triple $(i, \mathcal{I}_i, \mathrm{SK}_{\mathcal{I}_i}) \in T$. It returns an error if T is empty or i is out of range. Recall that i cannot be associated with $\mathcal{I}^*$ in this game.

3. Sign: On input an index $i \in [1, |T|]$ and a message $M \in \{0,1\}^\ell$, the simulator obtains the triple $(i, \mathcal{I}_i, \mathrm{SK}_{\mathcal{I}_i}) \in T$ or returns an error if it does not exist. If $\mathcal{I}_i \neq \mathcal{I}^*$, then the simulator signs M with $\mathrm{SK}_{\mathcal{I}_i}$ in the usual way.

   If $\mathcal{I}_i = \mathcal{I}^*$, then we know $M \neq M^*$. Let β be the first index such that $m_\beta \neq m^*_\beta$. First compute $\sigma' = \prod_{i=1,\ldots,\ell \wedge i \neq \beta} A_{i,m_i}$. Next, compute $\sigma'' = \sigma' \cdot x_i$. Also compute $\gamma = \prod_{i=1,\ldots,n} B_{i,\mathrm{id}_i}$. Finally, compute $\sigma = \mathsf{cenc}_3(k-1, \gamma \cdot \sigma'')$ Return σ to $\mathcal{A}$. Signatures are are well-formed and, due to the rerandomization in the cenc_3 algorithm, are distributed in a manner statistically exponentially close to the keys generated in the real game.

Response. Eventually, $\mathcal{A}$ outputs an aggregate signature σ^* on multiset S^* where $(\mathcal{I}^*, M^*) \in S^*$. The simulator will extract from this a solution to the MCDH problem. This works by iteratively computing all the other signatures in S^* and then subtracting them out of the aggregate until only one or more signatures on $(\mathcal{I}^*, M^*)$ remain. That is, the simulator takes an aggregate for S^* and computes an aggregate signature for S' where S' has one less verification key/message pair than S at each step. These signatures will be computed as in the query phase.

Eventually, we have an aggregate σ' on $t \geq 1$ instances of $(\mathcal{I}^*, M^*)$. However recall that $H(\mathcal{I}^*, M^*)$ is a level k encoding of $(\prod_{i \in [1,n]} b_{i,\mathrm{id}^*_i})(\prod_{i \in [1,\ell]} a_{i,m^*_i}) = \prod_{i \in [k]} c_i$. Thus verification of the signature σ' implies that (t, σ') is a solution to the GGH k-MCDH problem, and so the simulator returns (t, σ') to break the GGH k-MCDH assumption.

As remarked above, the responses of the challenger are distributed statistically exponentially closely to the real unforgeability game. The simulator succeeds whenever $\mathcal{A}$ does. □

References

[1] Jae Hyun Ahn, Matthew Green, and Susan Hohenberger. Synchronized aggregate signatures: new definitions, constructions and applications. In *ACM Conference on Computer and Communications Security*, pages 473–484, 2010.

[2] Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Unrestricted aggregate signatures. In *ICALP*, pages 411–422, 2007.

[3] Mihir Bellare and Thomas Ristenpart. Simulation without the artificial abort: Simplified proof and improved concrete security for waters' ibe scheme. In *EUROCRYPT*, pages 407–424, 2009.

[4] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[5] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures - hww to sign with rsa and rabin. In *EUROCRYPT*, pages 399–416, 1996.

[6] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *Public Key Cryptography*, pages 31–46, 2003.

[7] Alexandra Boldyreva, Craig Gentry, Adam O'Neill, and Dae Hyun Yum. Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. In *ACM Conference on Computer and Communications Security*, pages 276–285, 2007.

[8] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *EUROCRYPT*, pages 223–238, 2004.

[9] Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In *CRYPTO*, pages 443–459, 2004.

[10] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In *EUROCRYPT*, pages 56–73, 2004.

[11] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *EUROCRYPT*, pages 440–456, 2005.

[12] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003. extended abstract in Crypto 2001.

[13] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT*, pages 416–432, 2003.

[14] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *ASIACRYPT*, pages 514–532, 2001.

[15] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *IACR Cryptology ePrint Archive*, 2002:80, 2002.

[16] Kyle Brogle, Sharon Goldberg, and Leonid Reyzin. Sequential aggregate signatures with lazy verification from trapdoor permutations - (extended abstract). In *ASIACRYPT*, pages 644–662, 2012.

[17] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[18] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT*, pages 255–271, 2003.

[19] Ying-Ju Chi, Ricardo Oliveira, and Lixia Zhang. Cyclops: The Internet AS-level Observatory. In *ACM SIGCOMM CCR*, 2008.

[20] Yevgeniy Dodis, Iftach Haitner, and Aris Tentes. On the instantiability of hash-and-sign rsa signatures. In *TCC*, pages 112–132, 2012.

[21] Yevgeniy Dodis, Roberto Oliveira, and Krzysztof Pietrzak. On the generic insecurity of the full domain hash. In *CRYPTO*, pages 449–466, 2005.

[22] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices and applications. In *EUROCRYPT*, 2013.

[23] Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-based encryption for circuits from multilinear maps. In *CRYPTO*, 2013.

[24] Craig Gentry. Practical identity-based encryption without random oracles. In *EUROCRYPT*, pages 445–464, 2006.

[25] Craig Gentry and Zulfikar Ramzan. Identity-based aggregate signatures. In *Public Key Cryptography*, pages 257–273, 2006.

[26] Craig Gentry and Alice Silverberg. Hierarchical id-based cryptography. In *ASIACRYPT*, pages 548–566, 2002.

[27] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. Ntru: A ring-based public key cryptosystem. In *ANTS*, pages 267–288, 1998.

[28] Dennis Hofheinz, Tibor Jager, and Edward Knapp. Waters signatures with optimal security reduction. In *Public Key Cryptography*, pages 66–83, 2012.

[29] Dennis Hofheinz and Eike Kiltz. Programmable hash functions and their applications. *J. Cryptology*, 25(3):484–527, 2012.

[30] Susan Hohenberger and Brent Waters. Constructing verifiable random functions with large input spaces. In *EUROCRYPT*, pages 656–672, 2010.

[31] Allison B. Lewko and Brent Waters. Decentralizing attribute-based encryption. In *EUROCRYPT*, pages 568–588, 2011.

[32] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In *EUROCRYPT*, pages 465–485, 2006.

[33] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In *EUROCRYPT*, pages 74–90, 2004.

[34] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures: extended abstract. In *ACM Conference on Computer and Communications Security*, pages 245–254, 2001.

[35] Moni Naor and Omer Reingold. Constructing pseudo-random permutations with a prescribed structure. *J. Cryptology*, 15(2):97–102, 2002.

[36] Gregory Neven. Efficient sequential aggregate signed data. *IEEE Transactions on Information Theory*, 57(3):1803–1815, 2011.

[37] Kazuo Ohta and Tatsuaki Okamoto. A digital multisignature scheme based on the fiat-shamir scheme. In *ASIACRYPT*, pages 139–148, 1991.

[38] Tatsuaki Okamoto. A digital multisignature schema using bijective public-key cryptosystems. *ACM Trans. Comput. Syst.*, 6(4):432–441, 1988.

[39] Markus Rückert and Dominique Schröder. Aggregate and verifiably encrypted signatures from multilinear maps without random oracles. In *ISA*, pages 750–759, 2009.

[40] Adi Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, pages 47–53, 1984.

[41] Brent Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT*, pages 114–127, 2005.

[42] Brent Waters. Dual system encryption: Realizing fully secure ibe and hibe under simple assumptions. In *CRYPTO*, pages 619–636, 2009.

A Background on GGH

In this section, we provide some background on the GGH framework. We use the GGH framework in a manner very similar to the way it was used in the recent work of Garg, Gentry, Halevi, Sahai, and Waters on constructing Attribute-Based Encryption for Circuits [23]. For consistency, the following text is taken verbatim from [23]:

A.1 Graded Encoding Systems: Definition

Garg, Gentry and Halevi (GGH) [22] defined an "approximate" version of a multilinear group family, which they call a *graded encoding system*. As a starting point, they view g_i^α in a multilinear group family as simply an *encoding* of α at "level-i". This encoding permits basic functionalities, such as equality testing (it is easy to check that two level-i encodings encode the same exponent), additive homomorphism (via the group operation in $\mathbb{G}_i$), and bounded multiplicative homomorphism (via the multilinear map e). They retain the notion of a somewhat homomorphic encoding with equality testing, but they use probabilistic encodings, and replace the multilinear group family with "less structured" sets of encodings related to lattices.

Abstractly, their n-graded encoding system for a ring R includes a system of sets $\mathcal{S} = \{S_i^{(\alpha)} \subset \{0,1\}^* : i \in [0,n], \alpha \in R\}$ such that, for every fixed $i \in [0,n]$, the sets $\{S_i^{(\alpha)} : \alpha \in R\}$ are disjoint (and thus form a partition of $S_i := \bigcup_\alpha S_i^{(\alpha)}$). The set $S_i^{(\alpha)}$ consists of the "level-i encodings of α". Moreover, the system comes equipped with efficient procedures, as follows:[3]

Instance Generation. The randomized $\mathsf{InstGen}(1^\lambda, 1^n)$ takes as input the security parameter λ and integer n. The procedure outputs $(\mathsf{params}, \mathbf{p}_{zt})$, where params is a description of an n-graded encoding system as above, and $\mathbf{p}_{zt}$ is a level-n "zero-test parameter".

Ring Sampler. The randomized $\mathsf{samp}(\mathsf{params})$ outputs a "level-zero encoding" $a \in S_0$, such that the induced distribution on α such that $a \in S_0^{(\alpha)}$ is statistically uniform.

Encoding. The (possibly randomized) $\mathsf{enc}(\mathsf{params}, i, a)$ takes $i \in [n]$ and a level-zero encoding $a \in S_0^{(\alpha)}$ for some $\alpha \in R$, and outputs a level-i encoding $u \in S_i^{(\alpha)}$ for the same α.

[3] Since GGH's realization of a graded encoding system uses "noisy" encodings over ideal lattices, the procedures incorporate information about the magnitude of the noise.

Re-Randomization. The randomized $\mathsf{reRand}(\mathsf{params}, i, u)$ re-randomizes encodings to the same level, as long as the initial encoding is under a given noise bound. Specifically, for a level $i \in [n]$ and encoding $u \in S_i^{(\alpha)}$, it outputs another encoding $u' \in S_i^{(\alpha)}$. Moreover for any two encodings $u_1, u_2 \in S_i^{(\alpha)}$ whose noise bound is at most some b, the output distributions of $\mathsf{reRand}(\mathsf{params}, i, u_1)$ and $\mathsf{reRand}(\mathsf{params}, i, u_2)$ are statistically the same.

Addition and negation. Given params and two encodings at the same level, $u_1 \in S_i^{(\alpha_1)}$ and $u_2 \in S_i^{(\alpha_2)}$, we have $\mathsf{add}(\mathsf{params}, u_1, u_2) \in S_i^{(\alpha_1+\alpha_2)}$, and $\mathsf{neg}(\mathsf{params}, u_1) \in S_i^{(-\alpha_1)}$, subject to bounds on the noise.

Multiplication. For $u_1 \in S_{i_1}^{(\alpha_1)}$, $u_2 \in S_{i_2}^{(\alpha_2)}$, we have $\mathsf{mult}(\mathsf{params}, u_1, u_2) \in S_{i_1+i_2}^{(\alpha_1 \cdot \alpha_2)}$.

Zero-test. The procedure $\mathsf{isZero}(\mathsf{params}, \mathbf{p}_{zt}, u)$ outputs 1 if $u \in S_n^{(0)}$ and 0 otherwise. Note that in conjunction with the procedure for subtracting encodings, this gives us an equality test.

Extraction. This procedure extracts a "canonical" and "random" representation of ring elements from their level-n encoding. Namely $\mathsf{ext}(\mathsf{params}, \mathbf{p}_{zt}, u)$ outputs (say) $K \in \{0,1\}^\lambda$, such that:

(a) With overwhelming probability over the choice of $\alpha \in R$, for any two $u_1, u_2 \in S_n^{(\alpha)}$, $\mathsf{ext}(\mathsf{params}, \mathbf{p}_{zt}, u_1) = \mathsf{ext}(\mathsf{params}, \mathbf{p}_{zt}, u_2)$,

(b) The distribution $\{\mathsf{ext}(\mathsf{params}, \mathbf{p}_{zt}, u) : \alpha \in R, u \in S_n^{(\alpha)}\}$ is statistically uniform over $\{0,1\}^\lambda$.

We can extend add and mult to handle more than two encodings as inputs, by applying the binary versions of add and mult iteratively. Also, it is convenient to define a canonicalized encoding algorithm $\mathsf{cenc}\ell\mathsf{params}, i, a)$ which takes as input encoding of a and generates another encoding according to a "nice" distribution. The parameter ℓ denotes that the noise introduced with ℓ successive invocations of the reRand operation. This parameter was implicit in [22] encoding scheme and we make it explicit. This parameter essentially captures the noise present in the encodings. In our scheme we will need to re-randomize at most a constant number of times and hence the maximum value ℓ takes will be a small constant.

A.2 Graded Encoding Systems: Realization

Concretely, GGH's n-graded encoding system works as follows. (This is a whirlwind overview; see [22] for details.) The system uses three rings. First, it uses the ring of integers $\mathcal{O}$ of the m-th cyclotomic field. This ring is typically represented as the ring of polynomials $\mathcal{O} = \mathbb{Z}[x]/(\Phi_m(x))$, where $\Phi_m(x)$ is the m-th cyclotomic polynomial, which has degree $N = \phi(m)$. Second, for some suitable integer modulus q, it uses the quotient ring $\mathcal{O}/(q) = \mathbb{Z}_q[x]/(\Phi_m(x))$, similar to the NTRU encryption scheme [27]. The encodings live in $\mathcal{O}/(q)$. Finally, it uses the quotient ring $R = \mathcal{O}/\mathcal{I}$, where $\mathcal{I} = \langle g \rangle$ is a principal ideal of $\mathcal{O}$ that is generated by g and where $|\mathcal{O}/\mathcal{I}|$ is a large prime. This is the ring "R" referred to above; elements of R are what is encoded.

What does a GGH encoding look like? For a fixed random $z \in \mathcal{O}/(q)$, an element of $S_i^{(\alpha)}$ – that is, a level-i encoding of $\alpha \in R$ – has the form $e/z^i \in \mathcal{O}/(q)$, where $e \in \mathcal{O}$ is a "small" representative of the coset $\alpha + \mathcal{I}$ (it has coefficients that are very small compared to q). To add encodings $e_1/z^i \in S_i^{(\alpha_1)}$ and $e_2/z^i \in S_i^{(\alpha_2)}$, just add them in $\mathcal{O}/(q)$ to obtain $(e_1+e_2)/z^i$, which is in $S_i^{(\alpha_1+\alpha_2)}$ if $e_1 + e_2$ is "small". To mult encodings $e_1/z^{i_1} \in S_{i_1}^{(\alpha_1)}$ and $e_2/z^{i_2} \in S_{i_2}^{(\alpha_2)}$, just multiply them in

$\mathcal{O}/(q)$ to obtain $e_1 \cdot e_2 / z^{i_1+i_2}$, which is in $S_{i_1+i_2}^{(\alpha_1 \cdot \alpha_2)}$ if $e_1 \cdot e_2$ is "small". This smallness condition limits the GGH encoding system to degree polynomial in the security parameter. Intuitively, dividing encodings does not "work", since the resulting denominator has a nontrivial term that is not z.

The GGH params allow everyone to generate encodings of random (known) values. The params include a level-1 encoding of 1 (from which one can generate encodings of 1 at other levels), and (for each $i \in [n]$) a sufficient number of level-i encodings of 0 to enable re-randomization. To encode (say at level-1), run $\mathsf{samp}(\mathsf{params})$ to sample a small element a from $\mathcal{O}$, e.g. according to a discrete Gaussian distribution. For a Gaussian with appropriate deviation, this will induce a statistically uniform distribution over the cosets of $\mathcal{I}$. Then, multiply a with the level-1 encoding of 1 to get a level-1 encoding u of $a \in R$. Finally, run $\mathsf{reRand}(\mathsf{params}, 1, u)$, which involves adding a random Gaussian linear combination of the level-1 encodings of 0, whose noisiness (i.e., numerator size) "drowns out" the initial encoding. The parameters for the GGH scheme can be instantiated such that the re-randomization procedure can be used for any pre-specified polynomial number of times.

To permit testing of whether a level-n encoding $u = e/z^n \in S_n$ encodes 0, GGH publishes a level-n zero-test parameter $\mathbf{p}_{zt} = hz^n/g$, where h is "somewhat small"[4] and g is the generator of $\mathcal{I}$. The procedure $\mathsf{isZero}(\mathsf{params}, \mathbf{p}_{zt}, u)$ simply computes $\mathbf{p}_{zt} \cdot u$ and tests whether its coefficients are small modulo q. If u encodes 0, then $e \in \mathcal{I}$ and equals $g \cdot c$ for some (small) c, and thus $\mathbf{p}_{zt} \cdot u = h \cdot c$ has no denominator and is small modulo q. If u encodes something nonzero, $\mathbf{p}_{zt} \cdot u$ has g in the denominator and is not small modulo q. The $\mathsf{ext}(\mathsf{params}, \mathbf{p}_{zt}, u)$ procedure works by applying a strong extractor to the most significant bits of $\mathbf{p}_{zt} \cdot u$. For any two $u_1, u_2 \in S_n^{(\alpha)}$, we have (subject to noise issues) $u_1 - u_2 \in S_n^{(0)}$, which implies $\mathbf{p}_{zt}(u_1 - u_2)$ is small, and hence $\mathbf{p}_{zt} \cdot u_1$ and $\mathbf{p}_{zt} \cdot u_2$ have the same most significant bits (for an overwhelming fraction of α's).

Garg et al. provide an extensive cryptanalysis of the encoding system, which we will not review here. We remark that the underlying assumptions are stronger, but related to, the hardness assumption underlying the NTRU encryption scheme: that it is hard to distinguish a uniformly random element from $\mathcal{O}/(q)$ from a ratio of "small" elements – i.e., an element $u/v \in \mathcal{O}/(q)$ where $u, v \in \mathcal{O}/(q)$ both have coefficients that are on the order of (say) q^ϵ for small constant ϵ.

B Definitions for Aggregate Signatures

We introduced our general definitional setting in Section 3. Now, for our regular aggregate security definition, we define adaptive and selective variants. We also identify a slightly weaker "distinct message" security game that is easier to work with. In Appendix C, we describe and prove secure a simple transformation from distinct message security to standard aggregate signature security. The transformation captures the idea of hashing the public key and message together [13, 2] in a modular way.

An aggregate signature scheme is comprised of the following algorithms:

$\mathbf{Setup}(1^\lambda, \ell)$ The trusted setup algorithm takes as input the security parameter as well the bit-length ℓ of messages. It outputs a common set of public parameters PP.

[4] Its coefficients are on the order of (say) $q^{2/3}$, while other terms – such as a numerator e or the principal ideal generator g – are much, much smaller.

KeyGen(PP**)** The key generation algorithm takes as input the system public parameters and outputs a signature verification key and secret key pair $(\mathrm{VK}, \mathrm{SK})$.

Sign($\mathrm{PP}, \mathrm{SK}, M \in \{0,1\}^\ell$**)** The signing algorithm takes as input a secret signing key, the common public parameters as well as a message $M \in \{0,1\}^\ell$. It outputs a signature σ. *We emphasize that a single signature that is output by this algorithm is considered to also be an aggregate signature.*

Aggregate($\mathrm{PP}, \tilde{S}, S', \tilde{\sigma}, \sigma'$**).** The aggregation algorithm takes as input two multisets $\tilde{S}$ and S', two purported signatures on these multisets and the public parameters. The elements of $\tilde{S}$ consist of verification key/message pairs $\{(\tilde{\mathrm{VK}}_1, \tilde{M}_1), \ldots, (\tilde{\mathrm{VK}}_{|\tilde{S}|}, \tilde{M}_{|\tilde{S}|})\}$ and the elements of S consist of $\{(\mathrm{VK}'_1, M'_1), \ldots, (\mathrm{VK}'_{|S'|}, M'_{|S'|})\}$. The process produces a signature σ on the multiset $S = \tilde{S} \cup S'$, where $\cup$ is a multiset union.

Verify(PP, S, σ**).** The verification algorithm takes a input the public parameters, a multiset S of verification key/message pairs and a purported aggregate signature σ. It outputs true or false to indicate whether verification succeeded.

Correctness The correctness property states that all valid aggregate signatures will pass the verification algorithm, where a valid aggregate is defined recursively as an aggregate signature derived by an application of the aggregation algorithm on two valid inputs or the signing algorithm. More formally, for all integers $\lambda, \ell, k \geq 1$, all $\mathrm{PP} \in \mathrm{Setup}(1^\lambda, \ell)$, all $(\mathrm{VK}_i, \mathrm{SK}_i) \in \mathrm{KeyGen}(\mathrm{PP})$ for $i = 1$ to k, $\mathrm{Verify}(\mathrm{PP}, S, \sigma) = 1$, if σ is a *valid* aggregate for multiset S under PP. We say that an aggregate signature σ is *valid* for multiset S if: (1) $S = \{(\mathrm{VK}_i, M)\}$ for some $i \in [1, k]$, $M \in \{0,1\}^\ell$ and $\sigma \in \mathrm{Sign}(\mathrm{PP}, \mathrm{SK}_i, M)$; or (2) there exists multisets $S', \tilde{S}$ where $S = S' \cup \tilde{S}$ and valid aggregate signatures $\sigma', \tilde{\sigma}$ on them respectively such that $\sigma \in \mathrm{Aggregate}(\mathrm{PP}, \tilde{S}, S', \tilde{\sigma}, \sigma')$.

B.1 Security Model for Aggregate Signatures

We define the adaptive security game as in [13, 2]. Informally, it should be computationally infeasible for any adversary to produce a forgery implicating an honest signer, even when the adversary can control all other keys involved in the aggregate and can mount a chosen-message attack on the honest signer. This is defined using a game between a challenger and an adversary $\mathcal{A}$ with respect to scheme $\Pi =$ (Setup, KeyGen, Sign, Aggregate, Verify).

– **Unforg**$(\Pi, \mathcal{A}, \lambda, \ell)$:

Setup. The challenger runs $\mathrm{Setup}(1^\lambda, \ell)$ to obtain PP and $\mathrm{KeyGen}(\mathrm{PP})$ to obtain $(\mathrm{VK}, \mathrm{SK})$. It sends $(\mathrm{PP}, \mathrm{VK})$ to $\mathcal{A}$.

Queries. Proceeding adaptively, $\mathcal{A}$ requests signatures under VK on ℓ-bit messages of his choice.

Response. Finally, $\mathcal{A}$ outputs a multiset S^* of verification key/message pairs and a purported aggregate signature σ^*.

We say the adversary "wins" or that the output of this experiment is 1 if: (1) $\mathrm{Verify}(\mathrm{PP}, S^*, \sigma^*) = 1$ and (2) there exists an element $(\mathrm{VK}, M^*) \in S^*$ such that M^* was not queried for a signature by the adversary. Otherwise, the output is 0. Define $\mathbf{Forg}_{\mathcal{A}}$ as the probability that $\mathbf{Unforg}(\Pi, \mathcal{A}, \lambda, \ell) = 1$, where the probability is over the coin tosses of the Setup, KeyGen, and Sign algorithms and of $\mathcal{A}$.

Definition B.1 (Adaptive Unforgeability) *An aggregate signature scheme* Π *is existentially* unforgeable *with respect to adaptive chosen-message attacks if for all probabilistic polynomial-time adversaries* $\mathcal{A}$*, the function* $\boldsymbol{Forg}_A$ *is negligible in* λ*.*

Selective Security. We consider a selective variant to **Unforg** where there is an Init phase before the Setup phase, wherein $\mathcal{A}$ gives to the challenger the forgery message $M^* \in \{0,1\}^\ell$. This message M^* cannot be queried for a signature and yet (VK, M^*) must appear in S^*.

Distinct Message Security. We consider a distinct message variant to **Unforg**, where the game is the same as above, but we change how we define the experiment output. The output of the experiment is 1 if and only if: (1) it was 1 in the **Unforg** game, and (2) the message M^* was not associated with any other signer. That is, for all $(VK_i, M_i) \in S^*$, if $VK_i \neq \text{VK}$, then $M_i \neq M^*$. (The forgery message M^* could be associated with the key VK multiple times. This is allowed.)

C Transforming Distinct Message Security into Standard Security

In this section, we show how to transform any aggregate signature scheme proved in the distinct message security game into one which is secure in the standard security game. This will apply to both the selective and full security games. We remind the reader that distinct message security is not used in the ID-based setting, so we consider only regular signatures here.

The transformation essentially captures and modularizes idea of Boneh, Gentry, Lynn and Shacham [13], which was formally captured by Bellare, Namprempre, and Neven [2], of hashing the public key and message to get an output that is plugged in as the message for the core scheme. To execute this transformation the message length, ℓ, of the core scheme must be as large as the output of a collision-resistant hash function. We give the construction.

Let Π =(Setup, KeyGen, Sign, Aggregate, Verify) be an aggregate signature scheme for message length ℓ. Let $H : \{0,1\}^* \to \{0,1\}^\ell$ be a collision resistant hash function. We build a second aggregate signature scheme Π' derived from Π as follows. The public parameters now include the description of H. Keys are generated as before. To sign a message $M \in \{0,1\}^*$ for the user associated with verification key VK, first compute $M' = H(\text{VK}, M)$ and then run the regular signing algorithm on M'. Aggregation works the same as before. The verification algorithm re-computes $M'_i = H(\text{VK}_i, M_i)$ for each $(\text{VK}_i, M_i) \in S$ and treats these as the messages in the regular verification algorithm.

Lemma C.1 (Distinct Message to Standard Transformation) *If* Π *is an adaptively (resp., selectively),* distinct message *secure aggregate signature scheme for message length* ℓ *and* $H : \{0,1\}^* \to \{0,1\}^\ell$ *is a collision-resistant hash function, then* Π' *as defined above is an adaptively (reps., selectively) secure aggregate signature scheme.*

Proof. We argue that any PPT adversary $\mathcal{A}'$ against Π' can be turned into a PPT adversary $\mathcal{A}$ that breaks either Π in the distinct message game or finds a collision in H. Let σ^* be the forgery on S^* submitted by the adversary at the end of the standard security game. Let $M'_1, \ldots, M'_{|S|}$ be derived as $M'_i = H(\text{VK}_i, M_i)$ for each entry (VK_i, M_i) in S^*. We consider two cases.

First, suppose there exists some M'_i, M'_j such that $M'_i = M'_j$ and yet $(\text{VK}_i, M_i) \neq (\text{VK}_j, M_j)$. Then, the simulator can use the adversary to find a collision in the hash function since $H(\text{VK}_i, M_i) = H(\text{VK}_j, M_j)$ and the inputs are not equal.

Otherwise, it must be the case that if $M'_i = M'_j$, then $(\mathrm{VK}_i, M_i) = (\mathrm{VK}_j, M_j)$. Thus, the adversary is not violating the distinct message property since there cannot be $\mathrm{VK}_i \neq \mathrm{VK}_j$ where $M'_i = M'_j$. The simulator can reduce to the security of the underlying distinct message secure scheme. □

There are alternatives to proving security in the distinct message setting and then applying the above transformation, which have been explored in prior works. One possibility is to require public keys to be registered with some authority, where registration is contingent upon proving knowledge of the secret key to the authority. Verification only proceeds if the public key includes a registration certificate from the authority. Alternatively, one could include a non-interactive zero knowledge proof of knowledge of the private key as part of the private key. Verification only proceeds after the verifier checks the NIZKs. While we choose the distinct message plus transformation route, we expect these other alternatives would be viable with only minor technical modifications.

D Security of the Base Construction

We provide three claims on the security of the generic, base construction from Section 4.1. The proofs for the translation to the GGH framework follow along the same lines from these and the proof of the ID-Based construction in the GGH framework.

D.1 Security against Selective or Subexponential Advantage Attacks

The k-Multilinear Computational Diffie-Hellman (k-MDDH) assumption was defined in Section 4.3. We establish full security of our basic aggregate signature scheme conditioned on the k-MCDH assumption holding against subexponential advantage. This follows immediately from the following theorem and a standard complexity leveraging argument:

Theorem D.1 (Selective Security of Base Construction) *The aggregate signature scheme for message length ℓ in Section 4.1 is selectively secure in the distinct message unforgeability game under the $(\ell+1)$-MCDH assumption.*

Corollary D.2 *The aggregate signature scheme for message length ℓ in Section 4.1 is fully secure in the distinct message unforgeability game under the $(\ell+1)$-MCDH assumption against subexponential advantage.*

Proof. This follows immediately from a complexity leveraging argument: the security parameter λ is chosen to ensure that $2^{\lambda^{\epsilon_0}} >> 2^{\ell}$, where $2^{-\lambda^{\epsilon_0}}$ is the maximum probability of success allowed in the k-MCDH assumption against subexponential advantage. Now, to establish full security, the simulator performs exactly as in the selective security proof, but first it simply guesses the message that will be forged (instead of expecting the adversary to produce this message). Because this guess will be correct with probability at least $2^{-\ell}$, and the security parameter λ is chosen carefully, full security with polynomial advantage (or even appropriately defined subexponential advantage) implies an attacker on the k-MCDH assumption with subexponential advantage. □

Proof. **(of Theorem D.1)** We show that if there exists a PPT adversary $\mathcal{A}$ that can break the selective security of the aggregate signature scheme in the distinct message unforgeability game with probability ϵ for message length ℓ and security parameter λ, then there exists a PPT simulator

that can break the $(\ell+1)$-MCDH assumption for security parameter λ with probability ϵ. The simulator takes as input a MCDH instance $(g, g^{c_1}, \ldots, g^{c_k})$ together with the group descriptions where $k = \ell + 1$. The simulator plays the role of the challenger in the game as follows.

Init. Let $M^* \in \{0,1\}^\ell$ be the forgery message output by $\mathcal{A}$.

Setup. The simulator chooses random $x_1, \ldots, x_\ell \in \mathbb{Z}_p$. Let $A_{i,m_i^*} = g^{c_i}$ and $A_{i,\bar{m_i^*}} = g^{x_i}$. Set the challenge verification key as $\mathrm{VK}^* = g^{c_k}$. We remark that the parameters are distributed independently and uniformly at random as in the real scheme.

Queries. The simulator can sign any of $\mathcal{A}$'s message requests using the multilinear map. Let $M \in \{0,1\}^\ell$ be the request. The key point is that since $M \neq M^*$ there will be at least one i where x_i is used (and x_i is known to the simulator). So at most ℓ of the $k = \ell + 1$ parameters associated with c_i will need to be accumulated to make σ which is doable. That is, letting β be the first index such that $m_\beta \neq m_\beta^*$, compute the pairing of all $\ell - 1$ elements A_{i,m_i} where $i \neq \beta$ together with VK^* and denote this σ'. This requires $\ell - 1$ pairings resulting in an element in $\mathbb{G}_{k-1}$. Next compute $\sigma = \sigma'^{x_\beta}$ and return σ. Signatures are unique and perfectly distributed as in the real game.

Response. Eventually, $\mathcal{A}$ outputs an aggregate signature σ^* on multiset S^* where $(\mathrm{VK}^*, M^*) \in S^*$. The simulator will extract from this a solution to the MCDH problem. This works by iteratively computing all the other signatures in S^* and then dividing them out of the aggregate until only one or more signatures on (VK^*, M^*) remain. That is, the simulator takes an aggregate for S^* and computes an aggregate signature for S' where S' has one less verification key/message pair than S at each step. These signatures will be peeled off in one of two ways. If the signature is under key VK^*, then it can be computed as in the query phase. If the signature is under key $\mathrm{VK} \neq \mathrm{VK}^*$, then due to the distinct message restriction, we know $M \neq M^*$. Thus, there is some β where $m_\beta \neq m_\beta^*$. The signature can be computed similarly to the query phase by pairing VK with all A_{i,m_i} where $i \neq \beta$ and then raising the result to x_β. To help see why this works, recall that these signatures are unique.

Eventually, we have an aggregate σ' on $t \geq 1$ instances of (VK^*, M^*). We have that $e(\sigma', g) = e(H(M^*), VK^*)^t = g_k^{t\prod_{i=1}^k c_i}$ and thus $\sigma' = g_{k-1}^{t\prod_{i=1}^k c_i}$. The simulator computes $\sigma'^{1/t}$ (recall that t is not 0 mod p) which gives $g_{k-1}^{\prod_{i=1}^k c_i}$ and this is given as the solution to the MCDH problem.

As remarked in the Setup and Query phase, the responses of the challenger are distributed identically to the real unforgeability game. The simulator succeeds whenever $\mathcal{A}$ does. □

D.2 Security against Adaptive Attacks

We now give an adaptive proof of security under a *polynomial* assumption (as opposed to the subexponential advantage assumption necessary to achieve full security given previously). We will employ a parameterized assumption family, where the choice of assumption depends on the adversary's behavior. It can be viewed as a modification/adaptation of the computation Bilinear-Diffie Hellman Assumption (introduced by Boneh, Boyen, and Goh [11]) to the multilinear map setting.

Assumption D.3 ((n,k)-Modified Multilinear Computational Diffie-Hellman Exponent) *The (n,k)-Modified Multilinear Computational Diffie-Hellman Exponent ((n,k)-MMCDHE) problem states the following: A challenger runs $\mathcal{G}(1^\lambda, k)$ to generate groups and generators of order p. Then it picks random $a, b, c \in \mathbb{Z}_p$.*

The assumption then states that given

$$g = g_1, g^b, \ \forall i \in [1,n] \ g^{a^i c}, \quad \forall i \neq n \in [1,2n] \ (g_{k-2})^{a^i c^{k-1}}$$

it is hard to compute $(g_{k-1})^{a^n c^{k-1} b}$ with better than negligible advantage (in security parameter λ).

The above assumption is only defined for $k \geq 3$ due to the reference of the g_{k-2} generator.

Intuitively, our proof will follow in the Waters [41] framework. Waters gave a technique for partitioning approximately a (hidden) $1/Q$ fraction of messages to be useful as challenge forgeries and the other $1 - 1/Q$ to be messages a reduction algorithm could create signatures on. [5] Typically, one sets Q to be the maximum number of queries made by the adversary, although in this case, we must also add in the messages involved in the adversary's forgery aggregate signature. We will use a multiplicative analog of the technique which is closer to the VRF analysis of Hohenberger and Waters [30].

Theorem D.4 (Adaptive Security of Base Construction) *The aggregate signature scheme for message length ℓ in Section 4.1 is adaptively secure in the distinct message unforgeability game under the $(4Q(\ell+2), \ell+1)$-MMCDHE assumption, where Q is the number of signing queries made by the adversary plus one minus the number of distinct messages in the forgery aggregate.*

Proof. We show that if there exists a PPT adversary $\mathcal{A}$ that can break the adaptive security of the aggregate signature scheme in the distinct message unforgeability game with probability ϵ for message length ℓ, security parameter λ, making at most Q signing queries and Q' distinct messages in the forgery aggregate, then there exists a PPT simulator that can break the (n,k)-MMCDHE assumption for security parameter λ with probability $\geq \frac{3\epsilon}{64Q(\ell+1)}$.

The simulator takes as input an MMCDHE instance

$$(g, g^b, \ \forall i \in [1,n] \ g^{a^i c}, \ \forall i \neq n \in [1,2n] \ (g_{k-2})^{a^i c^{k-1}})$$

together with the group descriptions where $n = 4Q(\ell+2)$ and $k = \ell+1$. The simulator's challenge is to compute $(g_{k-1})^{a^n c^{(k-1)} b} = (g_{k-1})^{a^n c^\ell b}$. The simulator plays the role of the challenger in the game as follows.

Setup. The simulator first sets an integer $z = 4Q$ and chooses an integer t uniformly at random between 0 and ℓ. Recall that Q is the number of queries made by the adversary plus one minus the number of distinct messages in S^* that forms the forgery aggregate and ℓ is the message bit-length. It then chooses random integers $r_{1,0}, r_{1,1}, \ldots, r_{\ell,0}, r_{\ell,1}, r'$ between 0 and $z-1$. Additionally, the simulator chooses random values $s_{1,0}, s_{1,1}, \ldots, s_{\ell,0}, s_{\ell,1} \in {\mathbb{Z}_p}^*$. These values are all kept internal to the simulator. Intuitively, the r values will be used to embed the challenge, while the s values will be used as blinding factors to present the proper distribution to the adversary.

[5] There exists some variants of this technique [3, 29, 28] with different loss tradeoffs. We believe these tradeoffs are applicable to our setting, but we choose to stick closest to the original analysis.

Let m_i denote the ith bit of M. For $M \in \{0,1\}^\ell$, define the functions:

$$C(M) = zt + r' + \sum_{i=1}^{\ell} r_{i,m_i} \quad , \quad J(M) = \prod_{i=1}^{\ell} s_{i,m_i}$$

For $M \in \{0,1\}^\ell$, define the binary function:

$$K(M) = \begin{cases} 0 & \text{if } r' + \sum_{i=1}^{\ell} r_{i,m_i} \equiv 0 \mod z; \\ 1 & \text{otherwise.} \end{cases}$$

If the function K outputs 1 on a given message, then we know that the simulator will be able to correctly produce a signature on this message. If the function outputs 0, then the simulator may or may not be able to do it. This function will be used in later analysis.

The simulator sets the public parameters as $A_{1,0} = (g^{a^{(zt+r'+r_{1,0})}c})^{s_{1,0}}$, $A_{1,1} = (g^{a^{(zt+r'+r_{1,1})}c})^{s_{1,1}}$, and $A_{i,0} = (g^{a^{r_{i,0}}c})^{s_{i,0}}$ and $A_{i,1} = (g^{a^{r_{i,1}}c})^{s_{i,1}}$ for $i = 2$ to ℓ. The simulator can compute these values from the challenge input since all powers of a in the exponent are at most $zt + 2(z-1) = 4Q(\ell+2) - 2 < n$ for any possible choice of r', r_i, t. It sets the challenge verification key as $\mathrm{VK}^* = g^b$. It passes the public information to the adversary. We remark that the parameters are distributed independently and uniformly at random as in the real scheme.

Queries. The adversary will ask for signatures under the challenge verification key. On message input M, the simulator first checks if $C(M) = n$ and aborts if this is true. Otherwise, it outputs the signature as

$$\sigma = e(g^b, g_{k-2}^{a^{C(M)}c^{k-1}})^{J(M)} = g_{k-1}^{ba^{C(M)}c^\ell J(M)}.$$

Given the above settings, we can verify that for any value of $M \in \{0,1\}^\ell$, the maximum value of $C(M)$ is $z\ell + (\ell+1)(z-1) < 2z(\ell+1) = 2n$. Thus, if $C(M) \neq n$, then the simulator can correctly compute the signature.

Response. Eventually, the adversary will output a multiset S^* of verification key/message pairs and a purported aggregate signature σ^* such that:

1. $\mathrm{Verify}(\mathrm{PP}, S^*, \sigma^*) = 1$, and
2. there exists an element $(\mathrm{VK}, M^*) \in S^*$ such that M^* was not one of the adversary's signature query inputs, and
3. the message M^* is not signed by any other signer. (This is the distinct message requirement.)

If $C(M^*) \neq n$, then the simulator will abort. The goal is that the forgery message will fall into the "hole" of the assumption and that all other messages with not.

If $C(M^*) = n$, then the simulator will now work to extract a forgery on M^* from the aggregate by calculating the other signatures and then removing them from the aggregate. These can come both from the challenge signer and other signers. The simulator does this as follows:

Signatures from other signers. For (VK', M) where $\mathrm{VK}' = g^{b'} \neq \mathrm{VK}$, if $C(M) = n$, then the simulator aborts. Otherwise, it computes the signature as

$$\sigma = e(\mathrm{VK}', g_{k-2}^{a^{C(M)}c^{k-1}})^{J(M)} = g_{k-1}^{b'a^{C(M)}c^{\ell}J(M)}.$$

Signatures from challenge signer. For (VK, M) where $M \neq M^*$, if $C(M) = n$, then the simulator aborts. Otherwise, it computes the signature as

$$\sigma = e(g^{b}, g_{k-2}^{a^{C(M)}c^{k-1}})^{J(M)} = g_{k-1}^{ba^{C(M)}c^{\ell}J(M)}.$$

Extracting the response. Once these signatures are calculated they can be removed from the aggregate by division, resulting in an aggregate on $w \geq 1$ (non-multiple of p) signatures by the challenge signer on M^*. The uniqueness of this scheme dictates that this aggregate is:

$$\sigma' = ((g_{k-1})^{ba^{C(M^*)}c^{\ell}})^{J(M^*)w} = ((g_{k-1})^{ba^{n}c^{k-1}})^{J(M^*)w}$$

and raising σ' to $1/(J(M^*)w)$ results in $(g_{k-1})^{ba^{n}c^{k-1}}$, the solution to the MMCDHE instance. Recall that w is not a multiple of the group order p. $J(M^*)$ is a product of elements from ${\mathbb{Z}_p}^*$ where p is prime and therefore will also have an inverse modulo p.

A Series of Games Analysis. We now argue that any successful adversary $\mathcal{A}$ against our scheme will have success in the game presented by the simulator. To do this, we first define a sequence of games, where the first game models the real security game and the final game is exactly the view of the adversary when interacting with the simulator. We then show via a series of claims that if $\mathcal{A}$ is successful in Game j, then it will also be successful in Game $j+1$.

Game 1: This game is defined to be the same as the distinct message unforgeability game.

Game 2: The same as Game 1, with the exception that we keep a record of each signing query made by $\mathcal{A}$ concatenated together with each *distinct* message in the forgery multiset S^* minus M^*. We'll denote $\vec{M} = (M_0, M_1, M_2, ..., M_Q)$. Without loss of generality, let $M_0 = M^*$. At the end of the game, we set $z = 4Q$ and choose random integers $\vec{r} = (r_{1,0}, r_{1,1}, \ldots, r_{\ell,0}, r_{\ell,1}, r')$ between 0 and $z-1$ and a random integer t between 0 and ℓ. We define the regular abort function:

$$\mathsf{regabort}(\vec{M}, \vec{r}, t) = \begin{cases} 1 & \text{if } C(M^*) \neq n \ \bigvee_{i=1}^{Q} \ K(M_i) = 0; \\ 0 & \text{otherwise.} \end{cases}$$

This function evaluates to 0 if the queries and forgery messages will not cause a regular abort by the simulator for the given choice of simulation values. Consider the probability over all simulation values for the given set of queries and forgery messages as $\zeta(\vec{M}) = \Pr_{\vec{r},t}[\mathsf{regabort}(\vec{M}, \vec{r}, t) = 0]$.

As in [41], the simulator estimates $\zeta(\vec{M})$ as ζ' by evaluating $\tau(\vec{M}, \vec{r}, t)$ with fresh random $\vec{r}, t$ values a total of $O(\epsilon^{-2}\ln(\epsilon^{-1})\zeta_{\mathsf{min}}^{-1}\ln(\zeta_{\mathsf{min}}^{-1}))$ times, where $\zeta_{\mathsf{min}} = \frac{1}{8Q(\ell+1)}$. This does not require running the adversary again.

The adversary's success in the game is determined as follows:

1. *Regular Abort.* If $\mathsf{regabort}(\vec{M}, \vec{r}, t) = 1$, then the adversary wins.
2. *Balancing (Artificial) Abort.* Let $\zeta_{\mathsf{min}} = \frac{1}{8Q(\ell+1)}$ as derived from Claim D.5. If $\zeta' \geq \zeta_{\mathsf{min}}$, the simulator will abort with probability $\frac{\zeta' - \zeta_{\mathsf{min}}}{\zeta'}$ (not abort with probability $\frac{\zeta_{\mathsf{min}}}{\zeta'}$). If it aborts, then the adversary wins.
3. Otherwise, the adversary wins if and only if it outputs a valid forgery.

Game 3: The same as Game 2, with the exception that the simulator tests if any abort conditions are satisfied, with each new query or response from the adversary, and if so, follows the abort procedure immediately instead of waiting until the end.

Game 3 is exactly the view of the adversary when interacting with the simulator. We will shortly prove that if $\mathcal{A}$ succeeds in Game 1 with probability ϵ, then it succeeds in Game 3 with probability $\geq \frac{3\epsilon}{64Q(\ell+1)}$.

Claims Regarding the Probability of Aborting. We now establish one claim which was referenced above and two claims which will be needed shortly. Our first claim helps us establish a minimum probability that a given set of queries/forgery sets do not cause a *regular* abort. We use this minimum during our balancing abort in Game 2, to "even out" the probability of an abort over all possible queries/forgery sets. In the next two claims, we employ Chernoff Bounds to establish upper and lower bounds for *any* abort (regular or balancing) for any adversary behavior. The latter two claims will be used in the analysis of the adversary's probability of success in Game 2.

Proofs of these claims are similar to related arguments in [41, 30], but we include them here for completeness.

Claim D.5 *Let* $\zeta_{\mathsf{min}} = \frac{1}{8Q(\ell+1)}$. *For any vector* $\vec{M}$, $\zeta(\vec{M}) \geq \zeta_{\mathsf{min}}$.

Proof. In other words, the probability of the simulation *not* triggering a general abort is at least ζ_{min}. This analysis follows that of [41], which we reproduce here for completeness. Without loss of generality, we can assume the adversary always makes the maximum Q queries and number of distinct messages in the output forgery set (since the probability of not aborting increases with fewer queries/smaller output set size). Fix an arbitrary $\vec{M} = (M^*, M_1, \ldots, M_Q) \in \{0,1\}^{(Q+1)\times\ell}$.

Then, with the probability over the choice of $\vec{r}, t$, we have that $\Pr[\overline{\mathsf{abort}} \text{ on } \vec{M}]$ is

$$
\begin{aligned}
&= \Pr[\bigwedge_{i=1}^{Q} K(M_i) = 1 \ \wedge \ C(M^*) = n)] && (1)\\
&= (1 - \Pr[\bigvee_{i=1}^{Q} K(M_i) = 0]) \Pr[(zt + r' + \sum_{i=1}^{\ell} r_{i,m_i^*} = n) | \bigwedge_{i=1}^{Q} K(M_i) = 1] && (2)\\
&\geq (1 - \sum_{i=1}^{Q} \Pr[K(M_i) = 0]) \Pr[(zt + r' + \sum_{i=1}^{\ell} r_{i,m_i^*} = n) | \bigwedge_{i=1}^{Q} K(M_i) = 1] && (3)\\
&= (1 - \frac{Q}{z}) \cdot \Pr[(zt + r' + \sum_{i=1}^{\ell} r_{i,m_i^*} = n) \mid \bigwedge_{i=1}^{Q} K(M_i) = 1] && (4)\\
&= \frac{1}{\ell+1} \cdot (1 - \frac{Q}{z}) \cdot \Pr[K(M^*) = 0 \mid \bigwedge_{i=1}^{Q} K(M_i) = 1] && (5)\\
&= \frac{1}{\ell+1} \cdot (1 - \frac{Q}{z}) \cdot \frac{\Pr[K(M^*) = 0] \cdot \Pr[\bigwedge_{i=1}^{Q} K(M_i) = 1] \mid K(M^*) = 0]}{\Pr[\bigwedge_{i=1}^{Q} K(M_i) = 1]]} && (6)\\
&\geq \frac{1}{(\ell+1)z} \cdot (1 - \frac{Q}{z}) \cdot \Pr[\bigwedge_{i=1}^{Q} K(M_i) = 1] \mid K(M^*) = 0] && (7)\\
&= \frac{1}{(\ell+1)z} \cdot (1 - \frac{Q}{z}) \cdot (1 - \Pr[\bigvee_{i=1}^{Q} K(M_i) = 0] \mid K(M^*) = 0]) && (8)\\
&\geq \frac{1}{(\ell+1)z} \cdot (1 - \frac{Q}{z}) \cdot (1 - \sum_{i=1}^{Q} \Pr[K(M_i) = 0] \mid K(M^*) = 0]) && (9)\\
&= \frac{1}{(\ell+1)z} \cdot (1 - \frac{Q}{z})^2 && (10)\\
&\geq \frac{1}{(\ell+1)z} \cdot (1 - \frac{2Q}{z}) && (11)\\
&= \frac{1}{8Q(\ell+1)} && (12)
\end{aligned}
$$

Equations 4 and 7 derive from $\Pr[K(M) = 0] = \frac{1}{z}$ for any query M. Equation 5 gets a factor of $\frac{1}{\ell+1}$ from the simulator taking a guess of t. Equation 6 follows from Bayes' Theorem. Equation 10 follows from the pairwise independence of the probabilities that $K(M) = 0, K(M') = 0$ for any pair of queries $M \neq M'$, since they will differ in at least one random r_j value. Equation 12 follows from our setting of $z = 4Q$. □

Claim D.6 *For any vector $\vec{M}$, the probability that there is an abort (i.e., regular or balancing) is $\geq 1 - \zeta_{\mathsf{min}} - \frac{3}{8}\zeta_{\mathsf{min}}\epsilon$.*

Proof. Let $\zeta_x = \zeta(\vec{M})$ be the probability that the set of queries/forgery messages $\vec{M}$ do not cause a regular abort. In Game 2, $T = O(\epsilon^{-2} \ln(\epsilon^{-1}) \zeta_{\mathsf{min}}^{-1} \ln(\zeta_{\mathsf{min}}^{-1}))$ samples are taken to approximate this

value as ζ'_x. By Chernoff Bounds, we have that for all $\vec{M}$,

$$\Pr[T\zeta'_x < T\zeta_x(1-\frac{\epsilon}{8})] < e^{-[128\epsilon^{-2}\ln((\epsilon/8)^{-1})\zeta_{\mathsf{min}}^{-1}\ln(\zeta_{\mathsf{min}}^{-1})(\zeta_{\mathsf{min}})(\epsilon/8)^2/2]},$$

which reduces to

$$\Pr[\zeta'_x < \zeta_x(1-\frac{\epsilon}{8})] < \zeta_{\mathsf{min}}\frac{\epsilon}{8}.$$

The probability of not aborting is equal to the probability of not regular aborting (RA) times the probability of not artificial aborting (AA). Recall that for a measured ζ'_x an artificial abort will not happen with probability $\zeta_{\mathsf{min}}/\zeta'_x$. The probability of aborting is therefore

$$\begin{aligned}
\Pr[\mathsf{abort}] &= 1-\Pr[\overline{\mathsf{abort}}] = 1-\Pr[\overline{\mathsf{RA}}]\Pr[\overline{\mathsf{AA}}] = 1-\zeta_x\Pr[\overline{\mathsf{AA}}] \\
&\geq 1-\zeta_x(\zeta_{\mathsf{min}}\frac{\epsilon}{8}+\frac{\zeta_{\mathsf{min}}}{\zeta_x(1-\epsilon/8)}) \\
&\geq 1-(\zeta_{\mathsf{min}}\frac{\epsilon}{8}+\frac{\zeta_{\mathsf{min}}}{1-\epsilon/8}) \\
&\geq 1-(\frac{\zeta_{\mathsf{min}}\epsilon}{8}+\zeta_{\mathsf{min}}(1+\frac{2\epsilon}{8})) \\
&\geq 1-\zeta_{\mathsf{min}}-\zeta_{\mathsf{min}}\frac{3\epsilon}{8}
\end{aligned}$$

□

Claim D.7 *For any vector $\vec{M}$, the probability that there is no abort (i.e., regular or balancing) is $\geq \zeta_{\mathsf{min}} - \frac{1}{4}\zeta_{\mathsf{min}}\epsilon$.*

Proof. Let $\zeta_x = \zeta(\vec{M})$ be the probability that the set of queries/forgery messages $\vec{M}$ do not cause a regular abort. In Game 2, $T = O(\epsilon^{-2}\ln(\epsilon^{-1})\zeta_{\mathsf{min}}^{-1}\ln(\zeta_{\mathsf{min}}^{-1}))$ samples are taken to approximate this value as ζ'_x. By Chernoff Bounds, we have that for all $\vec{M}$,

$$\Pr[T\zeta'_x > T\zeta_x(1+\frac{\epsilon}{8})] < e^{-[256\epsilon^{-2}\ln((\epsilon/8)^{-1})\zeta_{\mathsf{min}}^{-1}\ln(\zeta_{\mathsf{min}}^{-1})](\zeta_{\mathsf{min}})(\epsilon/8)^2/4]},$$

which reduces to

$$\Pr[\zeta'_x > \zeta_x(1+\frac{\epsilon}{8})] < \zeta_{\mathsf{min}}\frac{\epsilon}{8}.$$

Recall that for a measured ζ'_x an artificial abort (AA) will not happen with probability $\zeta_{\mathsf{min}}/\zeta'_x$. Therefore, for any $\vec{M}$, the $\Pr[\overline{\mathsf{AA}}] \geq (1-\frac{\zeta_{\mathsf{min}}\epsilon}{8})\frac{\zeta_{\mathsf{min}}}{\zeta_x(1+\epsilon/8)}$. It follows that

$$\Pr[\overline{\mathsf{abort}}] \geq \zeta_x(1-\frac{\zeta_{\mathsf{min}}\epsilon}{8})\frac{\zeta_{\mathsf{min}}}{\zeta_x(1+\epsilon/8)} \geq \zeta_{\mathsf{min}}(1-\frac{\epsilon}{8})^2 \geq \zeta_{\mathsf{min}}(1-\frac{1}{4}\epsilon).$$

□

Analyzing $\mathcal{A}$'s Probability of Winning in Each Game. Define $\mathcal{A}$'s probability of success in Game x as $\mathsf{Adv}_{\mathcal{A}}[\text{Game } x]$. We reason about the probability of $\mathcal{A}$'s success in the series of games as follows.

Lemma D.8 *If* $\mathsf{Adv}_{\mathcal{A}}[\text{Game 1}] = \epsilon$, *then* $\mathsf{Adv}_{\mathcal{A}}[\text{Game 2}] \geq \frac{3 \cdot \epsilon}{64Q(\ell+1)}$.

Proof. We begin by observing that $\mathsf{Adv}_{\mathcal{A}}[\text{Game 2}]$ is

$$
\begin{aligned}
&= \mathsf{Adv}_{\mathcal{A}}[\text{Game 2}|\mathsf{abort}] \cdot \Pr[\mathsf{abort}] + \mathsf{Adv}_{\mathcal{A}}[\text{Game 2}|\overline{\mathsf{abort}}] \cdot \Pr[\overline{\mathsf{abort}}] && (13)\\
&= \Pr[\mathsf{abort}] + \mathsf{Adv}_{\mathcal{A}}[\text{Game 2}|\overline{\mathsf{abort}}] \cdot \Pr[\overline{\mathsf{abort}}] && (14)\\
&= \Pr[\mathsf{abort}] + \Pr[\mathcal{A} \text{ forges } |\overline{\mathsf{abort}}] \cdot \Pr[\overline{\mathsf{abort}}] && (15)\\
&= \Pr[\mathsf{abort}] + \Pr[\mathcal{A} \text{ forges}] \cdot \Pr[\overline{\mathsf{abort}}|\mathcal{A} \text{ forges}] && (16)\\
&= \Pr[\mathsf{abort}] + \epsilon \cdot \Pr[\overline{\mathsf{abort}}|\mathcal{A} \text{ forges}] && (17)\\
&\geq (1 - \zeta_{\mathsf{min}} - \zeta_{\mathsf{min}}\frac{3\epsilon}{8}) + \epsilon \cdot (\zeta_{\mathsf{min}} - \zeta_{\mathsf{min}}\frac{\epsilon}{4}) && (18)\\
&\geq \frac{3 \cdot \epsilon \cdot \zeta_{\mathsf{min}}}{8} && (19)\\
&= \frac{3 \cdot \epsilon}{64Q(\ell+1)} && (20)
\end{aligned}
$$

Equation 14 follows from the fact that, in the case of abort, $\mathcal{A}$ always wins. It would be very convenient if we could claim that $\mathsf{Adv}_{\mathcal{A}}[\text{Game 2} \mid \overline{\mathsf{abort}}] = \mathsf{Adv}_{\mathcal{A}}[\text{Game 1}]$, but unfortunately, this is false. The event that $\mathcal{A}$ wins Game 2 and the event of an abort are not independent; however, we have inserted the balancing abort condition in the attempt to lessen the dependence between these events. Equation 15 simply states that, when there is no abort, $\mathcal{A}$ wins if and only if it forges correctly. Equation 16 follows from Bayes' Theorem. In Equation 17, we observe that $\Pr[\mathcal{A} \text{ forges}]$ is exactly $\mathcal{A}$'s success in Game 1.

Now, the purpose of our balancing abort is to even the probability of aborting, for all queries and outputs of $\mathcal{A}$, to be roughly ζ_{min}. This will also get rid of the conditional dependence on $\mathcal{A}$ forging. There will be a small error, which must be taken into account. We set $\zeta_{\mathsf{min}} = \frac{1}{8Q(\ell+1)}$ from Claim D.5. We know, for all queries/outputs, that $\Pr[\mathsf{abort}] \geq 1 - \zeta_{\mathsf{min}} - \frac{3}{8}\zeta_{\mathsf{min}}\epsilon$ from Claim D.6 and that $\Pr[\overline{\mathsf{abort}}] \geq \zeta_{\mathsf{min}} - \frac{1}{4}\zeta_{\mathsf{min}}\epsilon$ from Claim D.7. Plugging these values into Equations 18 and 20 establishes the lemma. □

Lemma D.9 $\mathcal{A}_D[\text{Game 3}] = \mathcal{A}_D[\text{Game 2}]$.

Proof. We make the explicit observation that these games are equivalent by observing that their only difference is the time at which the regular aborts occur. The artificial abort stage is identical. All public parameters and signatures provided by the simulator have the same distribution. □

□

E The Base Aggregate Construction in the GGH framework

We now describe how to modify the construction of Section 4.1 to use the GGH [22] graded algebras analogue of multilinear maps. The translation of our scheme above is straightforward to the GGH

setting. Please note that we use the same notation developed in [22], with some minor changes: Firstly, we use the canonical encoding function cenc provided by the GGH framework more than once at each level of the encoding, but only a globally fixed constant number of times per level. This is compatible with the GGH encoding [22], and allows for a simpler exposition of our scheme and proof. Also, **for ease of notation on the reader, we suppress repeated params arguments that are provided to every algorithm.**. Thus, for instance, we will write $\alpha \leftarrow \mathsf{samp}()$ instead of $\alpha \leftarrow \mathsf{samp}(\mathsf{params})$. Note that in our scheme, there will only ever be a single uniquely chosen value for params throughout the scheme, so there is no cause for confusion. Finally, we use the variant of the GGH framework with "strong" zero-testing, where the zero test statistically guarantees that a vector is a valid encoding of zero if it passes the zero test. For further details on the GGH framework, please refer to [22].

Setup($1^\lambda, \ell$) The trusted setup algorithm takes as input the security parameter as well as the length ℓ of messages. It then runs $(\mathsf{params}, \mathbf{p}_{zt}) \leftarrow \mathsf{InstGen}(1^\lambda, 1^{k=\ell+1})$. Recall that params will be implicitly given as input to all GGH-related algorithms below.

Next, it generates elements $(A_{1,0}, A_{1,1}), \ldots, (A_{\ell,0}, A_{\ell,1})$, each equal to a fresh invocation of $\mathsf{cenc}_1(1, \mathsf{samp}())$.

These will be used to compute a function H mapping ℓ bit messages to level $k-1$ encodings. This function serves as the analog of the full domain hash function of the BGLS [13] construction. Let $m_1, \ldots, m_\ell$ be the bits of message M. It is computed iteratively as

$$H_1(M) = A_{1,m_1} \quad \text{for } i \in [2, \ell] \;\; H_i(M) = H_{i-1}(M) \cdot A_{i,m_i}.$$

We define $H(M) = \mathsf{cenc}_2(k-1, H_\ell(M))$.

The public parameters, PP, consist of the $\mathsf{params}, \mathbf{p}_{zt}$ plus:

$$(A_{1,0}, A_{1,1}), \ldots, (A_{\ell,0}, A_{\ell,1})$$

Note that params includes a level 1 encoding of 1, which we denote as g.

KeyGen(PP) The key generation algorithm first chooses random $\alpha = \mathsf{samp}()$. It outputs the public verification key as

$$\mathrm{VK} = \mathsf{cenc}_2(1, \alpha).$$

The secret key SK is α.

Sign($\mathrm{PP}, \mathrm{SK}, M \in \{0,1\}^\ell$) The signing algorithm computes the signature as

$$\sigma = \mathsf{cenc}_3(k-1, H(M) \cdot \alpha).$$

This serves as an aggregate signature for the (single element) multiset $S = \{(\mathrm{VK}, M)\}$.

Aggregate($\mathrm{PP}, \tilde{S}, S', \tilde{\sigma}, \sigma'$). The aggregation algorithm simply computes the output signature σ as $\sigma = \tilde{\sigma} + \sigma'$. The serves as a signature on the multiset $S = \tilde{S} \cup S'$, where $\cup$ is a *multiset union.*

Verify(PP, S, σ)**.** The verification algorithm parses S as $\{(\mathrm{VK}_1, M_1), \ldots, (\mathrm{VK}_{|S|}, M_{|S|})\}$. It rejects if the multiplicity of any public key, message pair is greater than 2^λ. We don't expect this to naturally occur much in practice.

The algorithm then proceeds to check the signature by setting $\tau = \mathsf{cenc}_2(1, g)$, and testing:

$$\mathsf{isZero}\left(\mathbf{p}_{zt}, \tau \cdot \sigma - \sum_{i=1,\ldots,|S|} H(M_i) \cdot \mathrm{VK}_i\right)$$

and accepts if and only if the zero testing procedure outputs true. Recall that g above is a canonical level 1 encoding of 1 that is included in params, part of the public parameters.

Correctness. Correctness follows from the same argument as for the "basic" aggregate signature scheme in the generic multilinear setting.

Proof of Security. In Section 5.2, we generalize this construction to provide ID-based aggregate signatures in the GGH framework. We provide a proof of selective security for the ID-based version of this scheme in the GGH framework, based on a variant of the MCDH assumption. Since our main focus is the ID-based aggregate signature scheme, we omit the formal proof of selective security for this scheme, but we note that it would be essentially identical to the proof of the ID-based scheme that we give in Section 5.3.

Efficiency and Tradeoffs. An aggregate signature is one level $k-1$ encoding, independent of the number of messages aggregated. In a multilinear setting, the space to represent an encoding might grow with k (which is $\ell + 1$). Indeed, this happens in the GGH [22] graded algebra translation. One way to mitigate this is to differ the message alphabet size in a tradeoff of computation versus storage. The above construction uses a binary message alphabet. If it used an alphabet of 2^d symbols, then the aggregate signature could be an ℓ/d level encoding, with $\ell/d - 1$ multiplications required to compute it, at the cost of the public parameters requiring $2^d\ell$ encodings in order to define the hash function H.

F Proof of Security for the Generic ID-Based Construction

The k-MCDH assumption is defined in Appendix D.1.

Theorem F.1 (Selective Security of ID-Based Construction) *The ID-based aggregate signature scheme for message length ℓ and identity length n in Section 5.1 is selectively secure in the unforgeability game in Section 3 under the $(\ell + n)$-MCDH assumption.*

Proof. We show that if there exists a PPT adversary $\mathcal{A}$ that can break the selective security of the ID-based aggregate signature scheme in the unforgettability game with probability ϵ for message length ℓ, identity length n and security parameter λ, then there exists a PPT simulator that can break the $(\ell + n)$-MCDH assumption for security parameter λ with probability ϵ. The simulator takes as input a MCDH instance $(g, g^{c_1}, \ldots, g^{c_k})$ together with the group descriptions where $k = \ell + n$. Let m_i denote the ith bit of M and id_i denote the ith bit of $\mathcal{I}$. The simulator plays the role of the challenger in the game as follows.

Init. Let $\mathcal{I}^* \in \{0,1\}^n$ and $M^* \in \{0,1\}^\ell$ be the forgery identity/message pair output by $\mathcal{A}$.

Setup. The simulator chooses random $x_1, \ldots, x_\ell, y_1, \ldots, y_n \in \mathbb{Z}_p$. For $i = 1$ to ℓ, let $A_{i,m_i^*} = g^{c_{i+n}}$ and $A_{i,\bar{m_i^*}} = g^{x_i}$. For $i = 1$ to n, iet $B_{i,\mathrm{id}_i^*} = g^{c_i}$ and $B_{i,\bar{\mathrm{id}_i^*}} = g^{y_i}$. We remark that the parameters are distributed independently and uniformly at random as in the real scheme.

Queries. Conceptually, the simulator will be able to create keys or signatures for the adversary, because his requests will differ from the challenge identity or message in at least one bit. More specifically,

1. Create New Key: The simulator begins with an index $i = 1$ and an empty sequence of index/identity/private key triples T. On input an identity $\mathcal{I} \in \{0,1\}^n$, if $\mathcal{I} = \mathcal{I}^*$, the simulator records $(i, \mathcal{I}^*, \perp)$ in T. Otherwise, the simulator computes the secret key as follows. Let β be the first index such that $\mathrm{id}_i \neq \mathrm{id}_i^*$. Use $n-2$ pairings on the B_{i,id_i} values to compute $s = (g_{n-1})^{\prod_{i=1,\ldots,n \wedge i \neq \beta} b_{i,\mathrm{id}_i}}$. Then compute $\mathrm{SK}_{\mathcal{I}} = s^{y_\beta} = (g_{n-1})^{\prod_{i=1,\ldots,n} b_{i,\mathrm{id}_i}}$. Record $(i, \mathcal{I}, \mathrm{SK}_{\mathcal{I}})$ in T. Secret keys are unique and perfectly distributed as in the real game.
2. Corrupt User: On input an index $i \in [1, |T|]$, the simulator returns to the adversary the triple $(i, \mathcal{I}_i, \mathrm{SK}_{\mathcal{I}_i}) \in T$. It returns an error if T is empty or i is out of range. Recall that i cannot be associated with $\mathcal{I}^*$ in this game.
3. Sign: On input an index $i \in [1, |T|]$ and a message $M \in \{0,1\}^\ell$, the simulator obtains the triple $(i, \mathcal{I}_i, \mathrm{SK}_{\mathcal{I}_i}) \in T$ or returns an error if it does not exist. If $\mathcal{I}_i \neq \mathcal{I}^*$, then the simulator signs M with $\mathrm{SK}_{\mathcal{I}_i}$ in the usual way.

   If $\mathcal{I}_i = \mathcal{I}^*$, then we know $M \neq M^*$. Let β be the first index such that $m_\beta \neq m_\beta^*$. Use $\ell - 2$ pairings on the A_{i,m_i} values to compute $\sigma' = (g_{\ell-1})^{\prod_{i=1,\ldots,\ell \wedge i \neq \beta} a_{i,m_i}}$. Next, compute $\sigma'' = \sigma'^{x_i} = (g_{\ell-1})^{\prod_{i=1,\ldots,\ell} a_{i,m_i}}$. Use $n-1$ pairings on the B_{i,id_i} values to compute $\gamma = (g_n)^{\prod_{i=1,\ldots,n} b_{i,\mathrm{id}_i}}$. Finally, compute $\sigma = e(\gamma, \sigma'') = (g_{k-1})^{(\prod_{i \in [1,n]} b_{i,\mathrm{id}_i})(\prod_{i \in [1,\ell]} a_{i,m_i})}$. Return σ to $\mathcal{A}$. Signatures are unique and perfectly distributed as in the real game.

Response. Eventually, $\mathcal{A}$ outputs an aggregate signature σ^* on multiset S^* where $(\mathcal{I}^*, M^*) \in S^*$. The simulator will extract from this a solution to the MCDH problem. This works by iteratively computing all the other signatures in S^* and then dividing them out of the aggregate until only one or more signatures on $(\mathcal{I}^*, M^*)$ remain. That is, the simulator takes an aggregate for S^* and computes an aggregate signature for S' where S' has one less verification key/message pair than S at each step. These signatures will be computed as in the query phase.

Eventually, we have an aggregate σ' on $t \geq 1$ instances of $(\mathcal{I}^*, M^*)$. We have that $e(\sigma', g) = H(\mathcal{I}^*, M^*)^t = (g_k)^{t(\prod_{i \in [1,n]} b_{i,\mathrm{id}_i^*})(\prod_{i \in [1,\ell]} a_{i,m_i^*})} = (g_k)^{t \prod_{i=1}^k c_i}$ and thus $\sigma' = (g_{k-1})^{t \prod_{i=1}^k c_i}$. The simulator computes $\sigma'^{1/t}$ (recall that t is not 0 mod p) which gives $(g_{k-1})^{\prod_{i=1}^k c_i}$ and this is given as the solution to the MCDH problem.

As remarked in the Setup and Query phase, the responses of the challenger are distributed identically to the real unforgeability game. The simulator succeeds whenever $\mathcal{A}$ does. □