Identity-Based Key Aggregate Cryptosystem from Multilinear Maps

Sikhar Patranabis and Debdeep Mukhopadhyay

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
{sikhar.patranabis, debdeep}@cse.iitkgp.ernet.in

Abstract. The key-aggregate cryptosystem (KAC) proposed by Chu et al. in 2014 offers a solution to the flexible access delegation problem in shared data environments such as the cloud. KAC allows a data owner, owning N classes of encrypted data, to securely grant access to any subset S of these data classes among a subset $\hat{S}$ of data users, via a single low overhead *aggregate key* $K_{\mathcal{S}}$. Existing constructions for KAC are efficient in so far they achieve constant size ciphertexts and aggregate keys. But they resort to a public parameter that has size linear in the number of data classes N, and require $O(M'M)$ secure channels for distribution of aggregate keys in a system with M' data owners and M data users. In this paper, we propose three different multilinear-map based KAC constructions that have at most polylogarithmic overhead for both ciphertexts and public parameters, and generate constant size aggregate keys. We further demonstrate how the aggregate keys may be efficiently broadcast among any arbitrary size subset of M data users using only $O(M' + M)$ secure channels, in a system with M' data owners. Our constructions are secure in the generic multilinear group model and are fully collusion resistant against any number of colluding parties. In addition, they naturally give rise to *identity based* secure access delegation schemes.

Keywords: Key-Aggregate Cryptosystem, Identity-based, Online Data Sharing, Multilinear Maps, Collusion-resistant

1 Introduction

The recent advent of cloud computing has led to unforeseen amounts of data being shared online with wide-ranging applications. There exists today a massive demand for scalable and efficiently implementable online data sharing schemes that provide formal guarantees of security and resistance against multi-party collusion attacks. The major challenge in designing such a system is solving the *online access delegation problem* [DMMM$^+$12,CCT$^+$14] in which a data owner owning N different classes of encrypted data, wishes to grant decryption rights to an arbitrary subset $\mathcal{S}$ of these data classes to a subset $\hat{\mathcal{S}}$ of authorized data users. Note that a data class in this context refers to a collection of similar data objects with identical access permissions.

A recently proposed solution to the online access delegation problem is the key-aggregate cryptosystem (KAC) [CCT$^+$14,PSM15]. KAC allows a data user to delegate decryption rights for *any* arbitrary set of of data classes $\mathcal{S}$ into a *single low overhead aggregate decryption key*. The aggregate key can then be distributed among a subset $\hat{S}$ of data users with appropriate access rights. The other major advantage of KAC is that it does not assume any pre-defined data hierarchy as in [BBG05,**?**] and can be adapted for any data organization mechanism. The efficiency of any KAC construction is measured in terms of the *ciphertext size* (storage overhead) and the *aggregate key size* (distribution overhead). A low overhead KAC construction is one in which both the ciphertext overhead and the key aggregate overhead is upper bounded by a logarithmic function in the number of data classes as well as the number of data users that the system can handle.

Relation of KAC with Broadcast Encryption. KAC may essentially be considered as a dual notion of broadcast encryption [BGW05,BWZ14a]. In broadcast encryption, a single ciphertext is broadcast among multiple users, each of whom may decrypt the same using their own individual private keys. In KAC, a single aggregate key is distributed among multiple users and may be used to decrypt ciphertexts encrypted with respect to different classes. For broadcast encryption, the focus is on having shorter ciphertexts and low overhead individual decryption keys, while in KAC, the focus is in having short ciphertexts and low overhead *aggregate keys*.

There exists, however, significant differences in the fundamental constructs for broadcast encryption and key aggregate encryption. Broadcast encryption essentially involves two classes of parties - the broadcaster who broadcasts the secret key, and the data users who decrypt the broadcast message. On the other hand, KAC involves three parties - the data owner who encrypts and puts the data in the online sharing environment, the data users who access the data by decrypting it, and the trusted third party that generates the aggregate key. In addition, the security framework for KAC constructions as well as the notions for collusion resistance, anticipated in [CCT+14] and introduced concretely [PSM15], are very different from that for the broadcast encryption techniques [BGW05,BWZ14a]. This motivates the dedicated study of KAC constructions separately from broadcast encryption.

Existing KAC Constructions in the Literature. Since KAC has only recently been introduced, there exist only a handful of constructions that achieve full collusion resistance while maintaining low ciphertext and aggregate key overhead. The first concrete construction for KAC was proposed in [CCT+14]. Although [CCT+14] lays out the basic KAC framework for shared data environments, it only anticipates the security notions for the same without stating any concrete proofs or security results. The first concrete game-based security framework for KAC was introduced in [PSM15], which was then used to prove the existing KAC construction to be CPA secure in the standard model. The constructions proposed in both [CCT+14] and [PSM15] achieve *constant size* overhead for both the ciphertext and the aggregate key. However, both the aforementioned constructions use a public parameter that has *linear* size in the number of data classes N. In addition, for a system with M' data owners and M data users, these schemes require $O(M'M)$ secure channels for distribution of the aggregate keys.

1.1 Our Contributions

In this paper, we propose three novel KAC constructions using multilinear maps with at most poly-logarithmic overhead for all system parameters, ciphertexts and aggregate keys:

- Our first KAC construction uses an asymmetric $O(\log N)$ multilinear map to support N data classes. The scheme has a public parameter overhead of $O(\log N)$ group elements, and produces short ciphertexts and aggregate keys comprising of $O(1)$ group elements. The scheme is proved to be non-adaptively secure in the generic multilinear group model based on a well-known complexity assumption.

- Our second KAC construction supports N data classes using a more general symmetric $O(\log N)$ multilinear map, and has similar overheads as the first construction for ciphertexts, aggregate keys as well as the public parameter. The symmetric map setting allows for non-adaptive security proofs in the generic multilinear group model based on a simpler complexity assumption, as compared to the asymmetric setting. However, as a flip side, it must be ensured that each data class index $i \in \{1, \cdots, N\}$ can be efficiently mapped to integers $\hat{i} \in \{1, \cdots, O(N \log N)\}$, where all $\hat{i}$ have the same Hamming weight l.

- Our third KAC construction has two major differences with the first and second constructions. It is *adaptively* secure in the generic multilinear group model, and provides tighter bounds on the

group size parameters. The trade-off is in the blow-up of the ciphertext size, which is $O(\log N)$ group elements instead of $O(1)$. However, a polylogarithmic overhead for the ciphertext is acceptable since the overall system overhead still remains polylogarithmic.

- We also demonstrate how each of the three KAC constructons may be efficiently combined with broadcast encryption schemes [BGW05,BWZ14a] so that the aggregate keys may be securely broadcast to the target subset of data users without the need for secure channels. We demonstrate that this extension to the basic KAC framework gives rise to data sharing schemes that require $O(M' + M)$ secure channels (instead of $O(M'M)$ secure channels in previous KAC constructions) for systems with M' data owners and M data users. Moreover, the extension does not cause any blow-up in the size of the system parameters.

1.2 Other Related Work

One of the most popular techniques for access control in online data storage is to use a pre-defined hierarchy of secret keys [ADSFM12,BBG05,**?**] in the form of a tree-like structure, where access to the key corresponding to any node implicitly grants access to all the keys in the subtree rooted at that node. Compact key encryption for the symmetric key setting has been used in [BCHL09] to solve the problem of concisely transmitting large number of keys in the broadcast scenario. However, symmetric key sharing via a secured channel is costly and not always practically viable for many applications on the cloud. Efficient public key based encryption methods such as identity based encryption (IBE) [BF03] and attribute based encryption (ABE) [GPSW06] focus principally on efficient decryption key distribution. However, these schemes do not focus on the possibility of key aggregation for multi-class data environments. Proxy re-encryption is another technique to achieve fine-grained access control and scalable user revocation in unreliable clouds [AFGH06]. Proxy re-encryption essentially transfers the responsibility for secure key storage from the delegatee to the proxy and may be susceptible to collusion attacks.

2 Preliminaries

In this section, we formally define the key-aggregate cryptosystem (KAC) framework as well its security. For clarity of understanding, we present the definition in two parts. The first part defines a basic KAC framework that focuses on generating small aggregate keys for arbitrarily large subsets of data classes. The second part extends this basic framework by combining it with broadcast encryption systems to distribute the aggregate key among multiple data users.

2.1 Key-Aggregate Cryptosystem : The Basic Version

The basic KAC is an ensemble of five poly-time randomized algorithms that are described next:

SetUp($\mathcal{ID}$): A data owner can classify her data into one or more classes belonging an identity space $\mathcal{ID}$. The function sets up the key-aggregate cryptosystem for the identity space $\mathcal{ID}$. Outputs the public parameter $param$.

KeyGen(): Outputs a master-secret key msk and the corresponding public key PK. A unique tuple (msk, PK) is generated for each data owner. The master secret key msk is known only to the trusted third party that generates the aggregate keys.

Encrypt($param, PK, i, \mathcal{M}$): Takes as input the public key parameter PK, the data class $i \in \mathcal{ID}$ and the plaintext message $\mathcal{M}$. Outputs the corresponding ciphertext $\mathcal{C}$, which is stored online in the

shared environment.

Extract$(param, msk, \mathcal{S})$: Takes as input the master secret key and a polynomial size subset of data classes $\mathcal{S} \subseteq \mathcal{ID}$. Computes the aggregate key $K_{\mathcal{S}}$ for all encrypted data/messages classified into any class in $\mathcal{S}$.

Decrypt$(param, \mathcal{C}, i, \mathcal{S}, K_{\mathcal{S}})$: Takes as input the ciphertext $\mathcal{C}$, the data class i and the aggregate key $K_{\mathcal{S}}$ corresponding to a subset $\mathcal{S}$. If $i \notin \mathcal{S}$, output $\perp$. Otherwise, outputs the decrypted message $\mathcal{M}$. The **Decrypt** function is invoked by a data user with the appropriate credentials to access one or more classes of data owned by the data owner. Note that the **Decrypt** operation for a given data user requires the explicit knowledge of the subset $\mathcal{S}$ of data classes that the corresponding user can access. This is of course a valid requirement since each user is expected to be aware of the subset $\mathcal{S}$ of data classes that she can access.

Correctness. For correctness, we require that the decryption algorithm always succeeds in decrypting a correctly encrypted plaintext message m. Formally, correctness of KAC may be described as follows. For any valid identity space $\mathcal{ID}$, any set $\mathcal{S} \subseteq \mathcal{ID}$, any index $i \in \mathcal{S}$, and any plaintext message m, we must have

$$Pr[\textbf{Decrypt}(\mathcal{C}, i, \mathcal{S}, K_{\mathcal{S}}) = \mathcal{M} | \mathcal{E}] = 1$$

where $\mathcal{E}$ is the event described as the conjunction of the following atomic events:

$$param \leftarrow \textbf{SetUp}(\mathcal{ID}), (msk, PK) \leftarrow \textbf{KeyGen}(),$$
$$\mathcal{C} \leftarrow \textbf{Encrypt}(param, PK, i, \mathcal{M}), K_{\mathcal{S}} \leftarrow \textbf{Extract}(msk, \mathcal{S})$$

2.2 Security Definitions

We define a formal framework for proving active chosen ciphertext security of KAC. We begin by introducing a game between a non-adaptive attack algorithm $\mathcal{A}$ and a challenger $\mathcal{B}$, both of whom are given $\mathcal{ID}$, the data class identity space, as input. The game proceeds through the following stages.

SetUp: Challenger $\mathcal{B}$ sets up the KAC system. In particular, $\mathcal{B}$ generates the public parameter $param$, the master secret key msk and the public key PK. Of these, $param$ and PK are furnished to $\mathcal{A}$.

Query Phase 1: Algorithm $\mathcal{A}$ adaptively issues decryption queries $q_1, \cdots, q_w$. Here a decryption query comprises of the tuple $(\mathcal{C}, v)$, where $v \in \mathcal{ID}$ is the data class of the message encrypted as $\mathcal{C}$. The challenger has to respond a valid decryption of the ciphertext.

Commit: $\mathcal{A}$ adaptively commits to a set $\mathcal{S} \subset \mathcal{ID}$ of data classes that it wishes to attack. Since collusion attacks are allowed in our framework, $\mathcal{B}$ furnishes $\mathcal{A}$ with the aggregate key $K_{\overline{\mathcal{S}}}$ that allows $\mathcal{A}$ to decrypt any data class $v \notin \mathcal{S}$. Next, $\mathcal{B}$ randomly chooses a data class $i \in \mathcal{S}$ and provides it to $\mathcal{A}$.

Challenge: $\mathcal{A}$ picks at random two messages $\mathcal{M}_0$ and $\mathcal{M}_1$ from the set of possible plaintext messages and provides them to $\mathcal{B}$. To generate the challenge, $\mathcal{B}$ randomly picks $b \in \{0, 1\}$, and sets the challenge to $\mathcal{A}$ as $(\mathcal{C}^*, \mathcal{M}_0, \mathcal{M}_1)$, where $\mathcal{C}^* = \textbf{Encrypt}(PK, i, \mathcal{M}_b)$.

Query Phase 2: $\mathcal{A}$ continues to adaptively issue decryption queries $q_{w+1}, \cdots, q_{Q_D}$ where a decryption query comprises of the tuple $(\mathcal{C}, v)$, but is now subject to the restriction $\mathcal{C} \neq \mathcal{C}^*$. $\mathcal{B}$ responds as in query phase 1.

Guess: $\mathcal{A}$ outputs a guess b' of b. If $b' = b$, $\mathcal{A}$ wins the game.

The game above models an attack in the real world setting where users who do not have authorized access to the subset $\mathcal{S}$ collude to try and expose a message in this subset. We now formally define the security notions for KAC. Let $Adv_{\mathcal{A},|\mathcal{ID}|}$ denote the probability that $\mathcal{A}$ wins the game.

Definition 2.1. A KAC construction is $(\epsilon, \mathcal{ID}, Q_D)$ adaptively secure under a chosen ciphertext attack (that is, adaptively CCA-secure) if, for all adaptive probabilistic poly-time algorithms $\mathcal{A}$ that can make a total of Q_D decryption queries, we have that $|Adv_{\mathcal{A},|\mathcal{ID}|} - \frac{1}{2}| < \epsilon$.

Definition 2.2. A KAC construction is $(\epsilon, \mathcal{ID})$ adaptively secure under a chosen plaintext attack (that is, adaptively CPA-secure) if it is $(\epsilon, \mathcal{ID}, 0)$ adaptively CCA secure.

We also define two weaker notions of security in the non-adaptive setting. In particular, non-adaptive security is achieved in the scenario when $\mathcal{A}$ is required to commit to the set $\mathcal{S}$ before seeing the public parameters. We refer to such an adversary as a non-adaptive adversary. This leads to the following definitions.

Definition 2.3. A KAC construction is $(\epsilon, \mathcal{ID}, Q_D)$ non-adaptively secure under a chosen ciphertext attack (that is, non-adaptively CCA-secure) if, for all non-adaptive probabilistic poly-time algorithms $\mathcal{A}$ that can make a total of Q_D decryption queries, we have that $|Adv_{\mathcal{A},|\mathcal{ID}|} - \frac{1}{2}| < \epsilon$.

Definition 2.4. A KAC construction is $(\epsilon, \mathcal{ID})$ non-adaptively secure under a chosen plaintext attack (that is, non-adaptively CPA-secure) if it is $(\epsilon, \mathcal{ID}, 0)$ non-adaptively CCA secure.

2.3 Extensions to The Basic Version : Broadcasting Aggregate Keys

The basic KAC framework described in Section 2.1 assumes that the aggregate key is securely broadcast to the target set of users via secure channels. In a system with M' data owners and M data users, this would require as many as $O(M'M)$ secure channels. In addition, storing the aggregate keys also warrants significant amounts of secure storage. In this paper, we augment the basic KAC framework to address these issues. Our approach is to broadcast the aggregate key among the target set of data users using public-key based broadcast encryption schemes [BGW05,BWZ14a].

The Basic Idea. Quite intuitively, the crux of the extended construction lies in combining the aggregate key $K_{\mathcal{S}}$ with the broadcast secret for the target subset of data users $\hat{\mathcal{S}}$. The main challenge in achieving this combination is the fact that the aggregate key and the broadcast secret generally lie in two different mathematical groups, and cannot be trivially combined using a simple group operation. Pairing them, on the other hand, makes it difficult to retrieve each individually and leads to an extremely cumbersome decryption process. To retain the simplicity of the scheme, we introduce the use of an additional secret key *dsk*, known only to the trusted third party. The secret key is incorporated into both the ciphertext $\mathcal{C}$, as well as the aggregate key $K_{\mathcal{S}}$, such that the knowledge of the public key PK and the public parameter *param* leaks no information about *dsk*. The aggregate key is then lifted to the appropriate group so that it may be combined with the broadcast secret to produce the *broadcast aggregate key* $K_{(\mathcal{S},\hat{\mathcal{S}})}$. Since *dsk* is only known to the trusted third party, the extended scheme has the following major augmentations:

1. The **Encrypt** algorithm in the basic KAC framework is now broken up into two separate algorithms - **OwnerEncrypt** where the data owner partially encrypts the plaintext message, and **FinalEncrypt** where the trusted third party takes the partial encryption and incorporates the knowledge of *dsk* in it to produce the final ciphertext. Note that it is to be made sure that the partial ciphertext does not leak any knowledge of the plaintext to the trusted third party.

2. The aggregate key $K_{\mathcal{S}}$ generated by the **Extract** algorithm is passed as input to a **Broadcast** algorithm, which then generates the corresponding broadcast aggregate key $K_{\left(\mathcal{S},\hat{\mathcal{S}}\right)}$ for the target subset of users $\hat{\mathcal{S}}$.

We point out that our proposed combination technique is generic; in particular, it is independent of the nature of the mathematical constructs (such as bilinear or multilinear maps) used in an actual construction instance of the extended KAC framework.

Let $\mathcal{ID}_1$ and $\mathcal{ID}_2$ denote the identity spaces for the data classes and the data users respectively. We formally define the combined scheme, referred to as the *extended* KAC framework, using the following set of algorithms.

SetUp$(\mathcal{ID}_1, \mathcal{ID}_2)$: Same as the basic KAC framework.

OwnerKeyGen(): In addition to the public key PK and the master-secret key msk, also outputs a distribution secret key dsk. The tuple (msk, dsk) is known only to the trusted third party.

OwnerEncrypt$(param, PK, i, \mathcal{M})$: Takes as input the data class $i \in \mathcal{ID}_1$ and the plaintext message $\mathcal{M}$. Outputs a *partially encrypted* ciphertext $\mathcal{C}'$. This ciphertext is not placed directly in the online shared environment. Instead, it is passed on to the **FinalEncrypt** operation executed by the trusted third party, described next.

FinalEncrypt$(\mathcal{C}', msk, dsk)$: Takes as input the partially encrypted ciphertext $\mathcal{C}'$ and outputs the final ciphertext $\mathcal{C}$, which is then placed in the shared data environment. This additional step is essential for supporting the broadcast of aggregate keys, as will be evident from the actual constructions presented later.

UserKeyGen$(param, msk, \hat{i})$: Takes as input the index $\hat{i} \in \mathcal{ID}_2$ for a user and outputs the corresponding secret key $d_{\hat{i}}$.

Extract$(param, msk, dsk, \mathcal{S})$: Takes as input the master secret key, the distribution secret key and a polynomial size subset of data classes $\mathcal{S} \subseteq \mathcal{ID}_1$. Computes the aggregate key $K_{\mathcal{S}}$ for all encrypted data/messages classified into any class in $\mathcal{S}$.

Broadcast$(param, K_{\mathcal{S}}, \hat{\mathcal{S}}, PK)$: Takes as input the aggregate key $K_{\mathcal{S}}$, the polynomial size target subset of users $\hat{\mathcal{S}} \subseteq \mathcal{ID}_2$. Outputs a single *broadcast aggregate key* $K_{\left(\mathcal{S},\hat{\mathcal{S}}\right)}$ that allows any user $\hat{i} \in \hat{\mathcal{S}}$ to decrypt all encrypted data/messages classified into any class $i \in \mathcal{S}$.

Decrypt$(param, \mathcal{C}, K_{\left(\mathcal{S},\hat{\mathcal{S}}\right)}, i, \hat{i}, d_{\hat{i}}, \mathcal{S}, \hat{\mathcal{S}})$: The decryption algorithm now takes, besides the ciphertext $\mathcal{C}$ and the corresponding data class $i \in \mathcal{S}$, a valid user id $\hat{i} \in \hat{\mathcal{S}}$. It also takes as input the broadcast aggregate key $K_{\left(\mathcal{S},\hat{\mathcal{S}}\right)}$ and the secret key $d_{\hat{i}}$. The algorithm outputs the decrypted message.

Note that the only secure channel requirements here are for the transmission of the secret key $d_{\hat{i}}$ output by **UserKeyGen**, and the transmission of the partially encrypted ciphertext from the data owner to the trusted third party. Since this requires only one one secure channel per data user and one secure channel pr data owner, the number of secure channels necessary is $O(M' + M)$ in a system with M' data owners and M data users. This is a significant reduction from the $O(M'M)$ secure channel requirement in the basic construction. Moreover, the extended construction does not require any secure storage for the aggregate keys, since the broadcast aggregate keys can be stored on the public cloud itself.

2.4 Security of Extended KAC: A Game Based Framework

We also define the formal framework for proving the security of the extended KAC via the following game between an attack algorithm $\mathcal{A}$ and a challenger $\mathcal{B}$:

SetUp: Challenger $\mathcal{B}$ sets up the KAC system. In particular, $\mathcal{B}$ generates the public parameter $param$, the master secret key msk, the distribution secret key dsk and the public key PK. Of these, $param$ and PK are furnished to $\mathcal{A}$.

Query Phase 1: Algorithm $\mathcal{A}$ adaptively issues decryption queries $q_1, \cdots, q_w$. Here a decryption query comprises of the tuple $(\mathcal{C}, v)$, where $v \in \mathcal{ID}$ is the data class of the message encrypted as $\mathcal{C}$. The challenger has to respond a valid decryption of the ciphertext.

Commit: $\mathcal{A}$ adaptively commits to a set $\mathcal{S} \subset \mathcal{ID}_1$ of data classes and a corresponding set $\hat{\mathcal{S}} \subset \mathcal{ID}_2$ of authorized data users (with access to $\mathcal{S}$) that it wishes to attack. Since collusion attacks are allowed in our framework, $\mathcal{B}$ furnishes $\mathcal{A}$ with all the private user keys $d_{\hat{j}}$ for $\hat{j} \notin \hat{\mathcal{S}}$. In addition, $\mathcal{A}$ is also provided with the broadcast aggregate key $K_{\left(\overline{\mathcal{S}}, \hat{\mathcal{S}}\right)}$ that allows any user in $\hat{\mathcal{S}}$ to decrypt any ciphertext class $v \notin \mathcal{S}$.

Challenge: $\mathcal{A}$ picks at random two messages $\mathcal{M}_0$ and $\mathcal{M}_1$ from the set of possible plaintext messages and provides them to $\mathcal{B}$. To generate the challenge, $\mathcal{B}$ randomly picks $b \in \{0, 1\}$, and sets the challenge to $\mathcal{A}$ as $(\mathcal{C}^*, \mathcal{M}_0, \mathcal{M}_1)$, where $\mathcal{C}^* = \mathbf{Encrypt}(PK, i, \mathcal{M}_b)$.

Query Phase 2: $\mathcal{A}$ continues to adaptively issue decryption queries $q_{w+1}, \cdots, q_{Q_D}$ where a decryption query comprises of the tuple $(\mathcal{C}, v)$, but is now subject to the restriction $\mathcal{C} \neq \mathcal{C}^*$. $\mathcal{B}$ responds as in query phase 1.

Guess: $\mathcal{A}$ outputs a guess b' of b. If $b' = b$, $\mathcal{A}$ wins the game.

The game above models an attack involving two different kinds of collusion. The first collusion is by all users not in $\hat{\mathcal{S}}$ who collude to try and expose an aggregate key that is broadcast for users in $\hat{\mathcal{S}}$ only. The second collusion is by users in $\hat{\mathcal{S}}$ who collude (by compromising the knowledge of the aggregate key for different subsets) to try and expose a message class in $\mathcal{S}$.

We next define the security of extended KAC. Let $Adv'_{\mathcal{A},|\mathcal{ID}_1|,|\mathcal{ID}_2|}$ denote the probability that $\mathcal{A}$ wins the above game.

Definition 2.5. An extended KAC construction with the ability to broadcast aggregate keys is $(\epsilon, \mathcal{ID}_1, \mathcal{ID}_2, Q_D)$ non-adaptively secure under a chosen ciphertext attack (that is, non-adaptively CCA-secure) if, for all non-adaptive probabilistic poly-time algorithms $\mathcal{A}$ that can make a total of Q_D decryption queries, we have that $|Adv'_{\mathcal{A},|\mathcal{ID}_1|,|\mathcal{ID}_2|} - \frac{1}{2}| < \epsilon$.

Definition 2.6. An extended KAC construction is $(\epsilon, \mathcal{ID}_1, \mathcal{ID}_2)$ non-adaptively secure under a chosen plaintext attack (that is, non-adaptively CPA-secure) if it is $(\epsilon, \mathcal{ID}_1, \mathcal{ID}_2, 0)$ non-adaptively CCA secure.

2.5 Multilinear Maps

In this section, we provide a brief overview of multilinear maps. Our description of multilinear maps is based on the *graded encoding scheme* used in several candidate multilinear map constructions [GGH13].

Symmetric Multilinear Maps. A standard symmetric multilinear map consists of the following pair of algorithms.

$\mathbf{SetUp}'(1^\lambda, m)$: Sets up an m-linear map by outputting an m-tuple of groups $< \mathbb{G}_1, \mathbb{G}_2, \cdots, \mathbb{G}_m >$ of prime order q (where q is a λ bit prime), along with the respective generator $g_i \in \mathbb{G}_i$ for $1 \leq i \leq m$. In standard notation, $\mathbb{G}_1$ is the source group, $\mathbb{G}_m$ is the target group, and $\mathbb{G}_2, \cdots, \mathbb{G}_{m-1}$ are the intermediate groups.

$e_{i,j}(h_1, h_2)$: Takes as input $h_1 \in \mathbb{G}_i$ and $h_2 \in \mathbb{G}_j$, and outputs $h_3 \in \mathbb{G}_{i+j}$ such that

$$(h_1 = g_i^a, h_2 = g_j^b) \Rightarrow h_3 = g_{i+j}^{ab}$$

In this paper, we follow the standard notation used in the literature to omit the subscripts and simply refer to this multilinear map as e. Further, e may be generalized to multiple inputs as $e(h_1, \cdots, h_k) = e(h_1, e(h_2, \cdots, h_k))$. Note that g_i^a is sometimes referred to as the level-i *encoding* of a. The scalar a itself may therefore be referred to as the level 0 encoding of itself. Although symmetric multilinear maps are simple to understand, there currently exist no reported candidate constructions for the same. Almost all candidate constructions [GGH13,CLT13] realize asymmetric multilinear maps, which we describe next.

Asymmetric Multilinear Maps. We adopt the same definition of asymmetric multilinear maps presented in [GGH13]. According to this definition, in asymmetric multilinear maps, the groups are indexed by integer vectors. Formally, a standard asymmetric multilinear map consists of the following algorithms.

$\mathbf{SetUp}''(1^\lambda, \mathbf{n})$: Takes as input a vector $\mathbf{n} \in \mathbb{Z}^l$. Sets up an $\mathbf{n}$-linear map by outputting an n-tuple of groups $< \mathbb{G}_{\mathbf{1}}, \mathbb{G}_{\mathbf{2}}, \cdots, \mathbb{G}_{\mathbf{n}} >$ of prime order q (where q is a λ bit prime), along with the respective generators $g_{\mathbf{v}} \in \mathbb{G}_{\mathbf{v}}$ for $\mathbf{1} \leq \mathbf{v} \leq \mathbf{n}$(comparison is defined component-wise). Further, let $\mathbf{x}_i$ be the ith *standard* basis vector (with 1 at position i and 0 at each other position). In standard notation, $\mathbb{G}_{\mathbf{x}_i}$ is the ith source group, $\mathbb{G}_{\mathbf{v}}$ is the target group, and the rest are the intermediate groups.

$e_{\mathbf{v_1},\mathbf{v_2}}(h_1, h_2)$: Takes as input $h_1 \in \mathbb{G}_{\mathbf{v_1}}$ and $h_2 \in \mathbb{G}_{\mathbf{v_2}}$, and outputs $h_3 \in \mathbb{G}_{\mathbf{i+j}}$ such that

$$(h_1 = g_{\mathbf{v_1}}^a, h_2 = g_{\mathbf{v_2}}^b) \Rightarrow h_3 = g_{\mathbf{i+j}}^{ab}$$

Again, we omit the subscripts and simply refer to this multilinear map as e, which may be generalized to multiple inputs as $e(h_1, \cdots, h_k) = e(h_1, e(h_2, \cdots, h_k))$.

In the forthcoming discussions, we present our KAC constructions assuming that the ideal multilinear maps based on the graded encoding scheme described above exist and are efficiently computable. We do this to make the analysis simple and easy to follow. We point out, however, that current candidates for multilinear maps in the cryptographic literature deviate from these ideal notions. In these candidates, group elements lack unique representations due to the presence of a noise term that tends to grow with repeated group/multilinear operations. This, however, is not a major setback since the KAC constructions presented in this paper can be instantiated using multilinear maps with the properties listed below:

- The representation of an element should be statistically independent of the group and multilinear operations that led to that element.
- It is possible to extract a *canonical* representation of an element in the target group given any representation of that element using the *zero-test parameter*.

- The party setting up the multilinear map has sufficient *trapdoor* information to compute g^{α^x} for a non-random α and exponentially large x.
- It is possible to generate asymmetric multilinear maps for any positive integer vector $\mathbf{n} \in \mathbb{Z}^l$.
- It should be possible to design the parameters of our system such that the noise growth during the execution of our scheme does not lead to erroneous computations.

The two foremost candidate multilinear map constructions [GGH13,CLT13] possess the aforementioned properties. Unfortunately, both these constructions, as well as attempted fixes for them [GHMS14,BWZ14b] have been cryptanalyzed either in parts or completely [CHL$^+$15,CLT14]. A more recent proposition by Gentry et al. is the graph-induced multilinear map based on non-ideal lattices [GGH15], which naturally gives rise to asymmetric multilinear maps. Even this construction has been broken by Coron et al. [**?**], who demonstrate how to recover an equivalent private key for this scheme (even in the presence of safeguards).

Given the extensive cryptanalysis of the existing multilinear map constructions, most of the intractable problems related to multilinear maps are currently secure only in the generic group model [BBG05]. We present a brief overview of the generic group model next.

2.6 Generic Multilinear Maps

Just as multilinear maps are an extension of bilinear maps, the generic multilinear map model is an extension of the generic bilinear map model [BBG05]. We describe the model here for completeness. In this model, the group $\mathbb{G}_{\mathbf{v}}$ (where $\mathbf{v} \in \mathbb{Z}_l$) is represented by a random injective function $\xi : \mathbb{Z}_q \times \mathbb{Z}^l \to \{0,1\}^n$ [BWZ14a]. Suppose that the target vector is $\mathbf{n} \in \mathbb{Z}^l$. Any algorithm in the generic multilinear map model is said to interact with the map using the tuple of algorithms (**Encode**, **Mult**, **Pair**) described below.

Encode$(x, \mathbf{v})$: Takes as input a non-negative integer vector $\mathbf{v} \leq \mathbf{m}$ and outputs $\xi(x, \mathbf{v})$.

Mult$(\xi_1, \xi_2, \diamond)$: Takes as input $\xi_1 = \xi(x_1, \mathbf{v})$, $\xi_2 = \xi(x_2, \mathbf{v})$ and $\diamond \in \{+, -\}$. Outputs $\xi(x_1 \diamond x_2, \mathbf{v})$.

Pair(ξ_1, ξ_2): Takes as input $\xi_1 = \xi(x_1, \mathbf{v}_1)$ and $\xi_2 = \xi(x_2, \mathbf{v}_2)$ where $\mathbf{v}_1 + \mathbf{v}_2 = \mathbf{v} \leq \mathbf{m}$. Outputs $\xi(x_1.x_2, \mathbf{v})$.

Note that if the inputs are not valid, each off the above algorithms returns $\perp$. Also, **Mult** and **Pair** here are assumed to be oracles to compute the induced group multiplication and multilinear map operations.

3 KAC Using Asymmetric Multilinear Maps

In this section, we present the first construction of identity-based KAC based on asymmetric multilinear maps. Our goal in this scheme is to ensure that the size of the public parameter is $O(\log N)$ group elements, while the ciphertext and the aggregate key comprise of $O(1)$ group elements. The construction is presented in two parts - the basic version followed by its extension for aggregate key distribution among multiple users.

The Basic Idea. Let $N = 2^m - 1$ for some integer m, and let $\mathbf{m}$ be the $m+1$ length vector consisting of all ones. We use an asymmetric multilinear map with the target group $\mathbb{G}_{2\mathbf{m}}$. Note that if we pair two elements in the group $\mathbb{G}_{\mathbf{m}}$, we get an element in $\mathbb{G}_{2\mathbf{m}}$ by the definition of asymmetric multilinear maps. Let $Y_i = g_{\mathbf{m}}^{\alpha^i}$, where $\alpha \in \mathbb{Z}_q$. Recall that $\mathbf{x}_j$ is the jth *standard* basis vector (with 1 at position j

and 0 at each other position) and $\mathbb{G}_{\mathbf{x}_j}$ is the jth source group with generator $g_{\mathbf{x}_i}$. Also, let $X_j = g_{\mathbf{x}_j}^{\alpha^{(2^j)}}$ for $0 \leq j \leq m-1$, $X_m = g_{\mathbf{x}_m}^{\alpha^{(2^m+1)}}$ and $X'_m = g_{\mathbf{x}_m}^{\alpha^{(2^m)}}$. We make the following claims.

Claim 3.1. Given an i such that $0 \leq i \leq N$, Y_i can be computed from the set of parameters $(X_0, \cdots, X_m)$.

Proof. Let $i = \sum_{j=0}^{m-1} i_j 2^j$. We have

$$\begin{aligned} Y_i &= g_{\mathbf{m}}^{\alpha^i} \\ &= g_{\mathbf{m}}^{\alpha^{\left(\sum_{j=0}^{m-1} i_j 2^j\right)}} \\ &= e\left(g_{\mathbf{x}_0}^{\alpha^{(2^j)} i_0} g_{\mathbf{x}_0}^{1-i_0}, \cdots, g_{\mathbf{x}_{m-1}}^{\alpha^{(2^j)} i_{m-1}} g_{\mathbf{x}_{m-1}}^{1-i_{m-1}}, g_{\mathbf{x}_m}\right) \\ &= e\left(X_0^{i_0} g_{\mathbf{x}_0}^{1-i_0}, \cdots, X_{m-1}^{i_{m-1}} g_{\mathbf{x}_{m-1}}^{1-i_{m-1}}, g_{\mathbf{x}_m}\right) \end{aligned}$$

Claim 3.2. Given i such that $N+2 \leq i \leq 2N$, Y_i can be computed from the set of parameters $(X_0, \cdots, X_m)$.

Proof. Let $i' = i - (2^m + 1) = \sum_{j=0}^{m-1} i'_j 2^j$. Then, we have

$$\begin{aligned} Y_i &= g_{\mathbf{m}}^{\alpha^i} \\ &= g_{\mathbf{m}}^{\alpha^{\left(\left(\sum_{j=0}^{m-1} i'_j 2^j\right)+(2^m+1)\right)}} \\ &= e\left(g_{\mathbf{x}_0}^{\alpha^{(2^j)} i'_0} g_{\mathbf{x}_0}^{1-i'_0}, \cdots, g_{\mathbf{x}_{m-1}}^{\alpha^{(2^j)} i'_{m-1}} g_{\mathbf{x}_{m-1}}^{1-i'_{m-1}}, g_{\mathbf{x}_m}^{\alpha^{(2^m+1)}}\right) \\ &= e\left(X_0^{i'_0} g_{\mathbf{x}_0}^{1-i'_0}, \cdots, X_{m-1}^{i'_{m-1}} g_{\mathbf{x}_{m-1}}^{1-i'_{m-1}}, X_m\right) \end{aligned}$$

We now make the following important observation.

Observation 3.3. *Given the set of parameters $(X_0, \cdots, X_m)$ and without the knowledge of $X'_m = g_{\mathbf{x}_m}^{\alpha^{(2^m)}}$, it is difficult to compute the value of Y_{N+1}.*

This follows trivially from the fact that 2^m cannot be expressed as the sum of other lower powers of 2. Thus we successfully embed a parameter set comprising of $O(N)$ group elements into another parameter set comprising of $O(\log N)$ group elements, without revealing any secret information. We next present the construction of the basic single data-owner KAC using this framework.

Assumption 3.4. For simplicity, we assume in the forthcoming discussion that our plaintext messages are embedded as elements in the group $\mathbb{G}_{2\mathbf{m}}$. We discuss in Appendix A how we may modify our scheme to relax this assumption.

3.1 Construction for the Basic KAC Framework

We first present a basic construction for the KAC scheme that focuses on generating aggregate keys. Assume that a data owner owning N classes of data wishes to furnish a data user with a *single* low overhead aggregate key that grants the user with decryption rights for any data class $i \in \mathcal{S}$, where $\mathcal{S}$

is any arbitrary subset of $\{1, \cdots, N\}$. For the moment we assume that the aggregate key is received by the data owner from a trusted third party who sets up the overall system. We later show how this construction may be extended using public-key based broadcast encryption to distribute the aggregate key to multiple data users.

Assume that $\mathbf{SetUp}''(1^\lambda, \mathbf{m})$ is the setup algorithm for an asymmetric multilinear map, where groups have prime order q (where q is a λ bit prime) and $\mathbb{G}_{\mathbf{m}}$ is the target group. Our first basic identity-based KAC, for a single data owner with $N = 2^m - 1$ data classes, consists of the following algorithms.

SetUp$(1^\lambda, m)$: Take as input the length m of identities and the group order parameter λ. Set $\mathcal{ID} = \{0,1\}^m \backslash \{0\}^m$ as the identity space. Let $\mathbf{m}$ be the $m+1$ length vector consisting of all ones. Also, let $param'' \leftarrow \mathbf{SetUp}''(1^{\lambda'}, 2\mathbf{m})$ be the public parameters for a multilinear map, with $\mathbb{G}_{2\mathbf{m}}$ being the target group and λ' is the adequate group size parameter for achieving λ bit security. Choose a random $\alpha \in \mathbb{Z}_q$. Set $X_j = g_{\mathbf{x}_j}^{\alpha^{(2^j)}}$ for $0 \leq j \leq m-1$ and $X_m = g_{\mathbf{x}_m}^{\alpha^{(2^m+1)}}$. Output the public parameter tuple $param$ as

$$param = (param'', \{X_j\}_{j \in \{0,\cdots,m\}})$$

Discard α after $param$ has been output.

KeyGen(): Randomly pick $\gamma \in \mathbb{Z}_q$. Set the master secret key $msk = \gamma$ and the public key $PK = g_{\mathbf{m}}^{\gamma}$. Output the tuple (msk, PK).

Encrypt$(param, PK, i, \mathcal{M})$: Take as input a message $\mathcal{M} \in \mathbb{G}_{2\mathbf{m}}$ belonging to class $i \in \mathcal{ID}$. Randomly choose $t \in \mathbb{Z}_q$. Recall that $Y_i = g_{\mathbf{m}}^{\alpha^i}$ and can be computed as per the formulation in Claim 3.1 for $1 \leq i \leq N$. Output the ciphertext $\mathcal{C}$ as

$$\mathcal{C} = \left(g_{\mathbf{m}}^t, (PK.Y_i)^t, \mathcal{M}.g_{2\mathbf{m}}^{t\alpha^{(2^m)}}\right)$$

where $g_{2\mathbf{m}}^{t\alpha^{(2^m)}}$ is computed as $(e(Y_{2^m-1}, Y_1))^t$.

Extract$(param, msk, \mathcal{S})$: For the input subset of data class indices $\mathcal{S}$, the aggregate key is computed as

$$K_{\mathcal{S}} = \prod_{v \in \mathcal{S}} Y_{2^m-v}^{msk}$$

Note that this is indirectly equivalent to setting $K_{\mathcal{S}}$ to $\prod_{v \in \mathcal{S}} PK^{\alpha^{2^m-v}}$.

Decrypt$(param, \mathcal{C}, i, \mathcal{S}, K_{\mathcal{S}})$: If $i \notin \mathcal{S}$, output $\perp$. Otherwise, set

$$a_{\mathcal{S}} = \left(\prod_{v \in \mathcal{S}, v \neq i} Y_{2^m-v+i}\right) \text{ and } b_{\mathcal{S}} = \left(\prod_{v \in \mathcal{S}} Y_{2^m-v}\right)$$

Let $\mathcal{C} = (c_0, c_1, c_2)$. Output the decrypted message as

$$\hat{\mathcal{M}} = c_2 \frac{e(K_{\mathcal{S}}.a_{\mathcal{S}}, c_0)}{e(b_{\mathcal{S}}, c_1)}$$

Correctness. To see that the scheme is correct, that is, $\hat{\mathcal{M}} = \mathcal{M}$, put $c_0 = g_{\mathbf{m}}^t$, $c_1 = (PK.Y_i)^t$ and $c_2 = \mathcal{M}.g_{2\mathbf{m}}^{t\alpha^{(2^m)}}$. Then we have

$$
\begin{aligned}
\hat{\mathcal{M}} &= c_2 \frac{e(K_{\mathcal{S}}.a_{\mathcal{S}}, c_0)}{e(b_{\mathcal{S}}, c_1)} \\
&= c_2 \frac{e(\prod_{v\in\mathcal{S}} Y^{\gamma}_{2^m-v}.\prod_{v\in\mathcal{S},v\neq i} Y_{2^m-v+i}, g^t_{\mathbf{m}})}{e(\prod_{v\in\mathcal{S}} Y_{2^m-v}, (PK.Y_i)^t)} \\
&= c_2 \frac{e(\prod_{v\in\mathcal{S},v\neq i} Y_{2^m-v+i}, g^t_{\mathbf{m}})}{e(\prod_{v\in\mathcal{S}} Y_{2^m-v}, Y_i^t)} \\
&= \frac{\mathcal{M}.g^{t\alpha^{(2^m)}}_{2\mathbf{m}}}{e(Y_{2^m}, g^t_{\mathbf{m}})} \\
&= \mathcal{M}
\end{aligned}
$$

3.2 Security of the Proposed KAC

We state and prove the non-adaptive CPA security of our proposed KAC scheme in the generic group model. We briefly state the complexity assumption that is to be used to prove the security of the proposed KAC scheme.

The Hybrid Diffie-Hellman Exponent Assumption. Let $param''$ is generated by $\mathbf{SetUp''}(1^{\lambda'}, 2\mathbf{m})$, where $\mathbf{m}$ is the $m+1$ length vector consisting of all ones and λ' is the adequate group size parameter for achieving λ bit security. Choose $\alpha \in \mathbb{Z}_q$ at random (where q is a λ'-bit prime), and let $X_j = g^{\alpha^{(2^j)}}_{\mathbf{x}_j}$ for $0 \leq j \leq m-1$. Also, define $X_m = g^{\alpha^{(2^m+1)}}_{\mathbf{x}_m}$. Choose a random $t \in \mathbb{Z}_q$, and let $V = g^t_{\mathbf{m}}$. The decisional m-Hybrid Diffie Hellman Exponent (HDHE) problem as defined as follows. Given the tuple $(param'', \{X_j\}_{j\in\{0,\cdots,m\}}, V, Z)$, distinguish if Z is $g^{t\alpha^{(2^m)}}_{2\mathbf{m}}$ or a random element of $\mathbb{G}_{2\mathbf{m}}$.

Definition 3.5. The decisional m-HDHE assumption holds for SetUp'' if, for any polynomial m and a probabilistic poly-time algorithm $\mathcal{A}$, $\mathcal{A}$ has negligible advantage in solving the m-HDHE problem.

We now state and prove the following theorem.

Theorem 3.6. *Let* **SetUp'''** *be the setup algorithm for an asymmetric multilinear map, and let the decisional m-Hybrid Diffie-Hellman Exponent assumption holds for* **SetUp''**. *Then our proposed basic KAC for $N = 2^m - 1$ data classes presented in Section 3.1 is non-adaptively CPA secure.*

Proof. Let $\mathcal{A}$ be a poly-time adversary such that $|Adv_{\mathcal{A},N} - \frac{1}{2}| > \epsilon$ for the proposed KAC system parameterized with an identity space $\mathcal{ID}$ of size $N = 2^m - 1$. Here ϵ is a non-negligible positive constant. We build an algorithm $\mathcal{B}$ that has advantage at least ϵ in solving the decisional m-HDHE problem for **SetUp'''**. $\mathcal{B}$ takes as input a random m-HDHE challenge $(param'', \{X_j\}_{j\in\{0,\cdots,m\}}, V, Z)$ where:

- $param'' \leftarrow$**SetUp''**$(1^{\lambda'}, 2\mathbf{m})$ (λ' is the adequate group size parameter for achieving λ bit security)
- $X_j = g^{\alpha^{(2^j)}}_{\mathbf{x}_j}$ for $0 \leq j \leq m-1$
- $X_m = g^{\alpha^{(2^m+1)}}_{\mathbf{x}_m}$
- $V = g^t_{\mathbf{m}}$ for a random $t \in \mathbb{Z}_q$ (q being a λ' bit prime)
- Z is either $g^{t\alpha^{(2^m)}}_{2\mathbf{m}}$ or a random element of $\mathbb{G}_{2\mathbf{m}}$

$\mathcal{B}$ then proceeds as follows.

Commit: $\mathcal{B}$ runs $\mathcal{A}$ and receives the set $\mathcal{S}$ of data classes that $\mathcal{A}$ wishes to be challenged on. $\mathcal{B}$ then randomly chooses a data class $i \in \mathcal{S}$ and provides it to $\mathcal{A}$.

SetUp: $\mathcal{B}$ should generate the public $param$ the public key PK and the aggregate key $K_{\overline{\mathcal{S}}}$, and provide them to $\mathcal{A}$. They are generated as follows.

- $param$ is set as $(param'', \{X_j\}_{j \in \{0,\cdots,m\}})$.
- PK is set as $g_{\mathbf{m}}^u / Y_i$ where u is chosen uniformly at random from $\mathbb{Z}_q$ and Y_i is computed as mentioned in Claim 3.1. Note that this is equivalent to setting $msk = (u - \alpha^i)$.
- $\mathcal{B}$ then computes

$$K_{\overline{\mathcal{S}}} = \prod_{v \notin \mathcal{S}} \frac{Y_{2^m - v}^u}{Y_{2^m - v + i}}$$

  Observe that $K_{\overline{\mathcal{S}}} = \prod_{v \notin \mathcal{S}} PK^{\alpha^{2^m - v}}$, as desired. Moreover, $\mathcal{B}$ is aware that $i \notin \overline{\mathcal{S}}$ (implying $i \neq v$), and hence has all the resources to compute $K_{\overline{\mathcal{S}}}$.

Since the $g_{\mathbf{m}}$, α, u and t values are chosen uniformly at random, *all the parameters and the keys have an identical distribution to that in the actual construction.*

Challenge: $\mathcal{A}$ picks at random two messages $\mathcal{M}_0$ and $\mathcal{M}_1$ from the set of possible plaintext messages in $\mathbb{G}_{2\mathbf{m}}$, and provides them to $\mathcal{B}$. $\mathcal{B}$ randomly picks $b \in \{0, 1\}$, and sets the challenge as $(\mathcal{C}, \mathcal{M}_0, \mathcal{M}_1)$, where

$$\mathcal{C} = (V, V^u, \mathcal{M}_b.Z)$$

We claim that when $Z = g_{2\mathbf{m}}^{t\alpha^{(2^m)}}$ (i.e. the input to $\mathcal{B}$ is a valid m-HDHE tuple), then $(\mathcal{C}, \mathcal{M}_0, \mathcal{M}_1)$ is a valid challenge to $\mathcal{A}$ as in a real attack. To see this, observe that

$$\begin{aligned} V = g_{\mathbf{m}}^t \text{ and } V^u &= (g_{\mathbf{m}}^u)^t = (PK.Y_i)^t \\ \mathcal{M}_b.Z &= \mathcal{M}_b.g_{2\mathbf{m}}^{t\alpha^{(2^m)}} \end{aligned}$$

Thus, by definition, $\mathcal{C}$ is a valid encryption of the message $\mathcal{M}_b$ in class i and hence, $(\mathcal{C}, \mathcal{M}_0, \mathcal{M}_1)$ is a valid challenge to $\mathcal{A}$.

Guess: The adversary $\mathcal{A}$ outputs a guess b' of b. If $b' = b$, $\mathcal{B}$ outputs 0 (indicating that $Z = g_{2\mathbf{m}}^{t\alpha^{(2^m)}}$). Otherwise, it outputs 1 (indicating that Z is a random element in $\mathbb{G}_{2\mathbf{m}}$).

We conclude that $\mathcal{B}$ has the same advantage ϵ as $\mathcal{A}$, which must therefore be negligible, as desired. This completes the proof of Theorem 3.6. Finally, we note here that in the absence of any provably secure multilinear map construction in the cryptographic literature, the m-HDHE assumption currently only holds in the generic group model. Boneh, Boyen and Goh [BBG05] introduced a general technique to prove computational lower bounds on the difficulty of breaking Diffie-Hellman-type complexity assumptions in a generic bilinear group model. An extension of these techniques can be used to prove that the m-HDHE assumption holds in the generic group model. Hence our proposed KAC construction is provably CPA secure in the generic group model. We point out, however, that should a secure multilinear map construction be achieved in the future that causes the m-HDHE assumption to hold in an actual group structure, the above proof would escalate to the standard model. □

CCA Security. The CPA secure construction of Section 3.1 may be efficiently combined with a signature scheme to obtain a CCA secure construction, albeit in the generic group model. For details, refer Appendix B.

3.3 An Extended KAC Construction : Broadcasting the Aggregate Keys

We now extend the basic KAC construction to tackle the problem of aggregate key distribution in data sharing environments with multiple users. Assume that there are N data classes and M users in the system. For simplicity, let $N = M$, that is, we assume the identity spaces $\mathcal{ID}_1$ and $\mathcal{ID}_2$ to be identical and denoted simply as $\mathcal{ID}$. To achieve this, one can sufficiently pad all class identity and user identity strings to have the same number of bits. The data owner grants access to a subset $\mathcal{S}$ of her data classes to a subset $\hat{\mathcal{S}}$ of the data users in the system. Here, both $\mathcal{S}$ and $\hat{\mathcal{S}}$ are arbitrary subset of $\{1, \cdots, N\}$, not necessarily equal. We show how the construction from Section 3.1 may be efficiently combined with the public-key based broadcast encryption scheme proposed in [BWZ14a] to achieve a fully identity-based public key solution to this problem.

Construction. Let $N = 2^m - 1$ and **SetUp'''** be as described before. The crux of the generalized scheme lies in the combination of the aggregate key with the broadcast encryption secret, although though they lie in different groups. Note that we do not need any additional parameters for incorporating broadcast encryption. Also note that generalization does not significantly blow up the overhead for any component of the system. In particular, the generalized scheme also consists of parameters that have size at most logarithmic in the number of data (and user) classes N. This allows N to be exponentially large. Hence, the generalized system is fully identity-based with each data class and each user associated with a unique identity string $id \in \{0,1\}^*$. The class index i and the user index $\hat{i}$ (where $1 \leq i, \hat{i} \leq N$) are obtained by hashing the corresponding id strings.

SetUp$(1^\lambda, m)$: Same as the basic construction in Section 3.1.

OwnerKeyGen(): Randomly pick $\gamma_1, \gamma_2, \gamma_3 \in \mathbb{Z}_q$. Set the master secret key msk to (γ_1, γ_2) and the public key $PK = (g_{\mathbf{m}}^{\gamma_1}, g_{\mathbf{m}}^{\gamma_2})$. Additionally, set the distribution secret key $dsk = \gamma_3$. Output the tuple (msk, PK, dsk).

OwnerEncrypt$(param, PK, i, \mathcal{M})$: Take as input a message $\mathcal{M} \in \mathbb{G}_{\mathbf{2m}}$ belonging to class $i \in \mathcal{ID}$. Randomly choose $t \in \mathbb{Z}_q$. Also, let $PK = (PK_1, PK_2)$. Output the partially encrypted ciphertext $\mathcal{C}$ as

$$\mathcal{C}' = \left(g_{\mathbf{m}}^t, PK_1^t, (PK_1.Y_i)^t, \mathcal{M}.g_{\mathbf{2m}}^{t\alpha^{(2^m)}}\right)$$

Note that the additional group element PK_1^t in the tuple blows up the ciphertext overhead by only a constant factor. The partially encrypted ciphertext $\mathcal{C}'$ is then passed on to the **FinalEncrypt** operation described next.

FinalEncrypt$(\mathcal{C}', msk, dsk)$: Let the partially encrypted ciphertext $\mathcal{C}'$ be of the form (c_0, c_1', c_2, c_3) and $msk = (msk_1, msk_2)$. Computes outputs the final ciphertext $\mathcal{C}$ as:

$$\mathcal{C} = (c_0, c_1, c_2, c_3)$$

where

$$c_1 = \frac{c_1'}{g_{\mathbf{m}}^{msk_1.dsk}}$$

The final ciphertext is now placed in the shared data environment. Note that the trusted third party gains no knowledge of the secret parameter t used in the partial encryption step, neither does she gain any knowledge of the plaintext message $\mathcal{M}$.

UserKeyGen$(param, msk, \hat{i})$: Let $msk = (msk_1, msk_2)$. Output the secret key for data user $\hat{i}$ as

$$d_{\hat{i}} = Y_{\hat{i}}^{msk_2}$$

Extract$(param, msk, dsk, \mathcal{S})$: Let $msk = (msk_1, msk_2)$. For the input subset of data class indices $\mathcal{S}$, the aggregate key is computed as

$$K_{\mathcal{S}} = \prod_{v \in \mathcal{S}} (Y_{2^m - v})^{msk_1 . dsk}$$

Note that this aggregate key will be passed as input to the **Broadcast** algorithm described next, and will not be transmitted to the target data users directly.

Broadcast$(param, K_{\mathcal{S}}, \hat{\mathcal{S}}, PK)$: Broadcasts the aggregate key $K_{\mathcal{S}}$ to all users in $\hat{\mathcal{S}}$ as follows. Let $PK = (PK_1, PK_2)$. Randomly choose $\hat{t} \in \mathbb{Z}_q$ and set

$$b_{\hat{\mathcal{S}}} = \left(\prod_{\hat{v} \in \hat{\mathcal{S}}} Y_{2^m - \hat{v}} \right)$$

Output

$$K_{(\mathcal{S}, \hat{\mathcal{S}})} = \left(g_{\mathbf{m}}^{\hat{t}}, \left(PK_2 . b_{\hat{\mathcal{S}}} \right)^{\hat{t}}, \mathcal{K} \right)$$

where

$$\mathcal{K} = \left(\left(g_{2\mathbf{m}}^{\hat{t}\alpha^{(2^m)}} \right) . \left(e(K_{\mathcal{S}}, g_{\mathbf{m}}) \right) \right)$$

Here, $g_{2\mathbf{m}}^{\hat{t}\alpha^{(2^m)}}$ is computed as $(e(Y_{2^m - 1}, Y_1))^{\hat{t}}$. Note that the actual group element corresponding to $K_{\mathcal{S}}$ is difficult to recover from $\mathcal{K}$. However, as we demonstrate next, this knowledge is not explicitly necessary for decryption.

Decrypt$(param, \mathcal{C}, K_{(\mathcal{S}, \hat{\mathcal{S}})}, i, \hat{i}, d_{\hat{i}}, \mathcal{S}, \hat{\mathcal{S}})$: If $i \notin \mathcal{S}$ or $\hat{i} \notin \hat{\mathcal{S}}$, output $\perp$. Otherwise, set

$$a_{\hat{\mathcal{S}}} = \left(\prod_{\hat{v} \in \hat{\mathcal{S}}, \hat{v} \neq \hat{i}} Y_{2^m - \hat{v} + \hat{i}} \right) \ , \ a_{\mathcal{S}} = \left(\prod_{v \in \mathcal{S}, v \neq i} Y_{2^m - v + i} \right)$$

$$\text{and } b_{\mathcal{S}} = \left(\prod_{v \in \mathcal{S}} Y_{2^m - v} \right)$$

Let $\mathcal{C} = (c_0, c_1, c_2, c_3)$ and $K_{(\mathcal{S}, \hat{\mathcal{S}})} = (\hat{k}_0, \hat{k}_1, \hat{k}_2)$. Output the decrypted message as

$$\hat{\mathcal{M}} = c_3 . \hat{k}_2 . \left(\frac{e(b_{\mathcal{S}}, c_1) e(a_{\mathcal{S}}, c_0)}{e(b_{\mathcal{S}}, c_2)} \right) . \left(\frac{e(d_{\hat{i}} . a_{\hat{\mathcal{S}}}, \hat{k}_0)}{e(Y_{\hat{i}}, \hat{k}_1)} \right)$$

Correctness of this scheme may be easily proven. We demonstrate next that this scheme is non-adaptively CPA secure in the standard model.

3.4 Security of Extended KAC

We state and prove the non-adaptive CPA security of the extended multi-user KAC. We first describe the complexity assumption that is used to prove security.

The Extended Hybrid Diffie-Hellman Exponent Assumption. Let $param''$ is generated by $\mathbf{SetUp''}(1^{\lambda'}, 2\mathbf{m})$, where $\mathbf{m}$ is the $m+1$ length vector consisting of all ones and λ' is the adequate group size parameter for achieving λ bit security. Choose $\alpha \in \mathbb{Z}_q$ at random (where q is a λ-bit prime), and let $X_j = g_{\mathbf{x}_j}^{\alpha^{(2^j)}}$ for $0 \leq j \leq m-1$. Also, define $X_m = g_{\mathbf{x}_m}^{\alpha^{(2^{m+1})}}$. Choose random $t, \hat{t} \in \mathbb{Z}_q$, and let $V_1 = g_{\mathbf{m}}^t$ and $V_2 = g_{\mathbf{m}}^{\hat{t}}$. The decisional m-Extended Hybrid Diffie Hellman Exponent (EHDHE) problem as defined as follows. Given the tuple

$$\left(param'', \{X_j\}_{j\in\{0,\cdots,m\}}, (V_1, V_2), (Z_1, Z_2)\right)$$

distinguish if (Z_1, Z_2) is $\left(g_{2\mathbf{m}}^{t\alpha^{(2^m)}}, g_{2\mathbf{m}}^{\hat{t}\alpha^{(2^m)}}\right)$ or a random element in $\mathbb{G}_{2\mathbf{m}} \times \mathbb{G}_{2\mathbf{m}}$.

Definition 3.7. The decisional m-EHDHE assumption holds for SetUp'' if, for any polynomial m and a probabilistic poly-time algorithm $\mathcal{A}$, $\mathcal{A}$ has negligible advantage in solving the m-EHDHE problem.

We now state and prove the following theorem:

Theorem 3.8. *Let* **SetUp'''** *be the setup algorithm for an asymmetric multilinear map, and let the decisional m-EHDHE assumption holds for* **SetUp''**. *Then the extended multi-user KAC for $N = 2^m - 1$ data classes is non-adaptively CPA secure.*

Proof. Let $\mathcal{A}$ be a poly-time adversary such that $|Adv_{\mathcal{A},N} - \frac{1}{2}| > \epsilon$ for the extended KAC parameterized with an identity space $\mathcal{ID}$ of size $N = 2^m - 1$. Here ϵ is a non-negligible positive constant. We build an algorithm $\mathcal{B}$ that has advantage at least ϵ in solving the decisional m-EHDHE problem for **SetUp''**. $\mathcal{B}$ takes as input a random m-EHDHE challenge $(param'', \{X_j\}_{j\in\{0,\cdots,m\}}, (V_1, V_2), (Z_1, Z_2))$ where:

- $param'' \leftarrow$**SetUp''**$(1^{\lambda'}, 2\mathbf{m})$, where λ' is the adequate group size parameter for achieving λ bit security.
- $X_j = g_{\mathbf{x}_j}^{\alpha^{(2^j)}}$ for $0 \leq j \leq m-1$
- $X_m = g_{\mathbf{x}_m}^{\alpha^{(2^{m+1})}}$
- $(V_1, V_2) = \left(g_{\mathbf{m}}^t, g_{\mathbf{m}}^{\hat{t}}\right)$ for a random $t \in \mathbb{Z}_q$ (q being a λ' bit prime)
- (Z_1, Z_2) is either $\left(g_{2\mathbf{m}}^{t\alpha^{(2^m)}}, g_{2\mathbf{m}}^{\hat{t}\alpha^{(2^m)}}\right)$ or a random element of $\mathbb{G}_{2\mathbf{m}} \times \mathbb{G}_{2\mathbf{m}}$.

$\mathcal{B}$ then proceeds as follows.

Commit: $\mathcal{B}$ runs $\mathcal{A}$ and receives the set $\mathcal{S}$ of data classes and the set $\hat{\mathcal{S}}$ of data users that $\mathcal{A}$ wishes to be challenged on. $\mathcal{B}$ then randomly chooses a data class $i \in \mathcal{S}$ and provides it to $\mathcal{A}$.

SetUp: $\mathcal{B}$ sets the following parameters and provides them to $\mathcal{A}$.

- $param$ is set as $(param'', \{X_j\}_{j\in\{0,\cdots,m\}})$.
- PK is set as

$$PK = (PK_1, PK_2) = \left(\frac{g_{\mathbf{m}}^{\gamma_1}}{Y_i}, \frac{g_{\mathbf{m}}^{\gamma_2}}{\prod_{\hat{v}\in\hat{\mathcal{S}}} Y_{2^m-\hat{v}}}\right)$$

  where γ_1, γ_2 are chosen uniformly at random from $\mathbb{Z}_q$, and Y_i is computed as mentioned in Claim 3.1.

Note that this is equivalent to setting msk as

$$msk = (msk_1, msk_2) = \left(\gamma_1 - \alpha^i, \gamma_2 - \sum_{\hat{v} \in \hat{\mathcal{S}}} \alpha^{2^m - \hat{v}} \right)$$

$\mathcal{B}$ chooses a random $\gamma_3 \in \mathbb{Z}_q$ and implicitly sets the value of the secret distribution key dsk to $t - \gamma_3$ (recall that $V_1 = g_{\mathbf{m}}^t$). We see later how this implicit definition manifests in the actual game.

Once again, since the $g_{\mathbf{m}}$, α, γ_1, γ_2, and $\hat{t}$ values are uniformly random, *all parameters and keys have an identical distribution to that in the actual construction.*

Query Phase: A is allowed to query secret keys for users $\hat{i} \notin \mathcal{S}$. $\mathcal{B}$ responds with

$$d_{\hat{i}} = \frac{Y_{\hat{i}}^{\gamma_2}}{\prod_{\hat{v} \in \hat{\mathcal{S}}} Y_{2^m - \hat{v} + \hat{i}}}$$

Observe that $d_{\hat{i}} = Y_{\hat{i}}^{msk_2}$, as desired. In addition, $\mathcal{A}$ may also query for $K_{\left(\overline{\mathcal{S}}, \hat{\mathcal{S}}\right)}$. This query models a collusion scenario where users in the set $\mathcal{S}$ itself may also collude to leak information about data classes not in $\mathcal{S}$. In response, $\mathcal{B}$ computes

$$K_{\overline{\mathcal{S}}} = \prod_{v \notin \mathcal{S}} \frac{Y_{2^m - v}^u}{Y_{2^m - v + i}}$$

and sets

$$\overline{\mathcal{K}} = Z_2 . e(K_{\overline{\mathcal{S}}}, V_1 . g_{\mathbf{m}}^{-\gamma_3})$$

Note that the implicit definition of dsk is used here. Finally, $\mathcal{B}$ provides $\mathcal{A}$ with the aggregate key

$$K_{\left(\overline{\mathcal{S}}, \hat{\mathcal{S}}\right)} = \left(V_2, V_2^{\gamma_2}, \overline{\mathcal{K}}\right)$$

It can be easily shown that whenever $Z_2 = g_{2\mathbf{m}}^{\hat{t}\alpha^{(2^m)}}$, this is a valid aggregate key that allows any user in $\hat{\mathcal{S}}$ to decrypt any class $i \notin \mathcal{S}$.

Challenge: $\mathcal{A}$ picks at random two messages $\mathcal{M}_0$ and $\mathcal{M}_1$ from the set of possible plaintext messages in $\mathbb{G}_{2\mathbf{m}}$, and provides them to $\mathcal{B}$. $\mathcal{B}$ randomly picks $b \in \{0, 1\}$, and sets the challenge as $(\mathcal{C}, \mathcal{M}_0, \mathcal{M}_1)$, where

$$\mathcal{C} = (V_1, PK_1^{\gamma_3}, V_1^{\gamma_1}, \mathcal{M}_b . Z_1)$$

As before, whenever $Z_1 = g_{2\mathbf{m}}^{t\alpha^{(2^m)}}$, $(\mathcal{C}, \mathcal{M}_0, \mathcal{M}_1)$ is a valid challenge to $\mathcal{A}$, as in a real attack.

Guess: $\mathcal{A}$ outputs a guess b' of b. If $b' = b$, $\mathcal{B}$ outputs 0 (indicating that $(Z_1, Z_2) = \left(g_{2\mathbf{m}}^{t\alpha^{(2^m)}}, g_{2\mathbf{m}}^{\hat{t}\alpha^{(2^m)}}\right)$). Otherwise, it outputs 1 (indicating that (Z_1, Z_2) is a random element in $\mathbb{G}_{2\mathbf{m}} \times \mathbb{G}_{2\mathbf{m}}$).

We conclude that $\mathcal{B}$ has the same advantage ϵ as $\mathcal{A}$, which must therefore be negligible. This completes the proof of Theorem 3.8. Once again, we note here that the m-EHDHE assumption currently only holds in the generic group model. Hence our proposed extended KAC construction is provably CPA secure in the generic group model. Should a secure multilinear map construction be achieved in the future that causes the m-EHDHE assumption to hold in an actual group structure, the above proof would also escalate to the standard model. □

3.5 Privacy of Data Owners

In any online data sharing environment with multiple data owners, data privacy is an essential requirement. In particular, the aggregate key supplied by one data owner should not leak information about another data owner to an unauthorized user. This problem however does not arise in our construction if a new parallel instance of the KAC construction in Section 3.3 is run for each data owner. Each instance can handle N data classes and can cater to N data users. In order to distinguish between data classes belonging to different instances, each data class is assigned a double index (i_1, i_2), where i_1 is the instance index, and i_2 is the class index specific to the instance. Each instance i_1 is characterized by its own master secret key msk^{i_1}, public key PK^{i_1} and distribution key dsk^{i_1}. The main advantage of this approach is that all the parallel instances can share the same public $param$, which needs to be setup exactly once by the system administrator. Also note that the number of unique ordered tuples $\left(msk^{i_1}, PK^{i_1}, dsk^{i_1}\right)$ is q^3. For $q = O(N)$, a single setup can support an exponentially large number of data owners. Finally, if a data owner wishes to store more than N classes of data or cater to more than N data users, she may be allocated more than one instance of the KAC construction in Section 3.3.

Finally, we note that our construction is agnostic of the manner in which the data owner organizes her data. In particular, our construction is easily adaptable for hierarchical data structures, since a data owner could create an aggregate key corresponding to all the data classes rooted at any internal node, and then broadcast it to the target user group.

3.6 An Identity-Based Data Sharing Scheme

The extended KAC construction presented above can be used to design fully identity-based online data sharing schemes with low overhead parameters. To see this, observe that each of our constructions can support N data classes and N data users, while using parameters with only $O(\log N)$ group elements. Thus, N is allowed to be exponentially large and in particular, as large as the range of a collision resistant hash function H. This makes our KAC constructions fully identity-based, since each data class and each data user can be associated with a unique identity $id \in \{0,1\}^*$. The index number can be automatically set by hashing the corresponding identity id to $H(id) \in \{1, \cdots, N\}$. Quite evidently, an identity-based KAC offers much flexibility. The aggregate key to any subset of data classes may be computed by simply hashing the corresponding class identities. At the same time, this aggregate key may be broadcast to any set of authorized users by hashing their public identities.

4 KAC Using Symmetric Multilinear Maps

In this section, we present a KAC construction based on traditional symmetric multilinear maps. We use the same idea presented in the earlier construction, that is, we use a symmetric multilinear map to ensure that the necessary parameters can be derived from a small number of public elements in the source group of the map. In this construction, the parameter $Y_i = g_m^{\alpha^i}$, while $X_j = g_1^{\alpha^{2^j}}$. However, unlike in the asymmetric setting where the same elements cannot be paired together, in the symmetric setting one could pair X_{m-1} with itself, and then pair it with g_1 $m-2$ times, to obtain Y_{N+1}. To overcome this we use a technique proposed in [BWZ14a] that restricts the bit representations of all identities in $\mathcal{ID}$ to a single Hamming weight class. This actually allows computing the necessary Y_i without leaking the value of Y_{N+1}. The idea is illustrated in the following discussion.

The Basic Idea. Let $Y_i = g_{m-1}^{\alpha^i}$ and $\hat{Y}_i = g_l^{\alpha^i}$ where $l \leq m$. Set $X_j = g_1^{\alpha^{(2^j)}}$ for $i = 0, 1, \cdots, m$. Further, let $HW(i)$ denote the Hamming weight of the bit representation of i. We now make the following claims.

Claim 4.1. One can compute $g_{HW(i)}^{\alpha^i}$ for $1 \leq i \leq 2^m - 2$. In particular, one can compute $\hat{Y}_i$ for $1 \leq i \leq 2^m - 2$ such that $HW(i) = l$.

Proof. Compute $g_{HW(i)}^{\alpha^i}$ by pairing together all X_j such that the jth bit of i is 1. Since i has at most m bits, the necessary X_j are available.

Claim 4.2. One can compute Y_i and Y_{2^m-1-i} for $1 \leq i \leq 2^m - 2$ such that $HW(i) = l$.

Proof. Note that for all i such that $1 \leq i \leq 2^m - 2$, $HW(i) \leq m - 1$. Hence, one can compute $g_{HW(i)}^{\alpha^i}$ by Claim 4.1 and then pair it with $g_{m-HW(i)-1}$ (if $HW(i) \leq m - 2$) to obtain Y_i. Also, compute $g_{HW(2^m-1-i)}^{\alpha^{2^m-1-i}}$ as per Claim 4.1. Note that $HW(2^m - 1 - i) = m - l$ if $HW(i) = l$. Thus, we basically computed $g_{m-l}^{\alpha^{2^m-1-i}}$. Then, we pair it with g_{l-1} (obtained by pairing g_1 $(l-1)$ times) to obtain Y_{2^m-1-i}.

Claim 4.3. One can compute $Y_{2^m-1-v+i}$ for $1 \leq i, v \leq 2^m - 2$, $i \neq v$ where $HW(i) = HW(v) = l$.

Proof. Let T_1 denote the set of these bit positions that are 1 in the bit representation of i, and T_2 denote the set of bit positions that are 1 in the bit representation of $2^m - 1 - v$. Clearly, $|T_1| = l$ and $|T_2| = m - l$. Now, note that that $T_1 \bigcap T_2 = \phi$ iff $i = v$ which is not allowed. So $\exists j' \in T_1 \bigcap T_2$. Then, we can write

$$2^m - 1 - v + i = \left(\sum_{j \in T_1 \setminus \{j'\}} 2^j \right) + \left(\sum_{j \in T_2 \setminus \{j'\}} 2^j \right) + 2^{j'+1}$$

Note that this is a sum of $m - 1$ powers of two. This in turn allows us to compute

$$Y_{2^m-1-v+i} = e\left(\{X_j\}_{j \in T_1 \setminus \{j'\}}, \{X_j\}_{j \in T_2 \setminus \{j'\}}, X_{j'+1} \right)$$

which is a pairing of $(m-1)$ X_j terms, as desired.

Assumption 4.4. For simplicity, we assume in the forthcoming discussion that our plaintext messages are embedded as elements in the group $\mathbb{G}_{m+l-1}$. For relaxations, refer Appendix A.

4.1 An Extended KAC Construction with Broadcast Aggregate Keys

We present an extended KAC construction using symmetric multilinear maps with the ability to broadcast aggregate keys for arbitrarily large subsets of data classes $\mathcal{S}$ to arbitrarily large subsets of data users $\hat{\mathcal{S}}$. Our scheme handles N data classes and N users, as in the extended KAC construction presented in Section 3.5.

Construction. Let $N = 2^m - 2$ and **SetUp′** sets up a symmetric multilinear map. Recall that $Y_i = g_{m-1}^{\alpha^i}$ and $\hat{Y}_i = g_l^{\alpha^i}$, where $1 \leq i \leq N$ and $l \leq m$. Our scheme can handle N data classes and N data users.

SetUp$(1^\lambda, (m, l))$: Set up the KAC system for $\mathcal{ID}$ consisting of all m bit class identities with Hamming weight l, that is $N = |\mathcal{ID}| = \binom{m}{l}$. Since $1 \leq l \leq m - 1$, we have $N \leq 2^{m-2}$. Let $param' \leftarrow$ **SetUp′**$(1^{\lambda'}, m+l-1)$ be the public parameters for a symmetric multilinear map, with $\mathbb{G}_{m+l-1}$ being the target group and λ' is the adequate group size parameter for achieving λ bit security. Choose a random $\alpha \in \mathbb{Z}_q$. Set $X_j = g_1^{\alpha^{(2^j)}}$ for $0 \leq j \leq m$. Output the public parameter tuple $param$ as

$$param = (param', \{X_j\}_{j \in \{0, \cdots, m\}})$$

Discard α after $param$ has been output.

OwnerKeyGen(): Randomly pick $\gamma_1, \gamma_2, \gamma_3 \in \mathbb{Z}_q$. Set the master secret key $msk = (\gamma_1, \gamma_2)$, the public key $PK = (g_l^{\gamma_1}, g_l^{\gamma_2})$ and the distribution key $dsk = \gamma_3$. Output the tuple (msk, PK, dsk).

OwnerEncrypt$(param, PK, i, \mathcal{M})$: Take as input a message $\mathcal{M} \in \mathbb{G}_{m+l-1}$ belonging to class $i \in \mathcal{ID}$. Randomly choose $t \in \mathbb{Z}_q$. Output the partially encrypted ciphertext $\mathcal{C}'$ as

$$\mathcal{C}' = \left(g_l^t, PK_1^t, (PK_1.\hat{Y}_i)^t, \mathcal{M}.g_{m+l-1}^{t\alpha^{(2^m-1)}}\right)$$

The partially encrypted ciphertext $\mathcal{C}'$ is then passed on to the **FinalEncrypt** operation described next.

FinalEncrypt$(\mathcal{C}', msk, dsk)$: Let the partially encrypted ciphertext $\mathcal{C}'$ be of the form (c_0, c_1', c_2, c_3) and $msk = (msk_1, msk_2)$. Computes outputs the final ciphertext $\mathcal{C}$ as:

$$\mathcal{C} = (c_0, c_1, c_2, c_3)$$

where

$$c_1 = \frac{c_1'}{g_1^{msk_1.dsk}}$$

The final ciphertext is now placed in the shared data environment. Once gain, note that the trusted third party gains no knowledge of the secret parameter t used in the partial encryption step, neither does she gain any knowledge of the plaintext message $\mathcal{M}$.

UserKeyGen$(param, msk, \hat{i})$: Let $msk = (msk_1, msk_2)$. Output the secret key for data user $\hat{i}$ as

$$d_{\hat{i}} = Y_{\hat{i}}^{msk_2}$$

Extract$(param, msk, dsk, \mathcal{S})$: Let $msk = (msk_1, msk_2)$. For the input subset of data class indices $\mathcal{S}$, the aggregate key is computed as

$$K_{\mathcal{S}} = \prod_{v \in \mathcal{S}} (Y_{2^m-1-v})^{msk_1.dsk}$$

Recall that Y_{2^m-1-v} can be computed as per Claim .3 for $j \in \mathcal{ID}$. Note once again that this aggregate key will be passed as input to the **Broadcast** algorithm described next, and will not be transmitted to the target data users directly.

Broadcast$(param, K_{\mathcal{S}}, \hat{\mathcal{S}}, PK)$: Broadcasts the aggregate key $K_{\mathcal{S}}$ to all users in $\hat{\mathcal{S}}$ as follows. Let $PK = (PK_1, PK_2)$. Randomly choose $\hat{t} \in \mathbb{Z}_q$ and set

$$b_{\hat{\mathcal{S}}} = \left(\prod_{\hat{v} \in \hat{\mathcal{S}}} Y_{2^m-1-\hat{v}}\right)$$

Output

$$K_{(\mathcal{S},\hat{\mathcal{S}})} = \left(g_l^{\hat{t}}, \left(g_{m-1}^{msk_2}.b_{\hat{\mathcal{S}}}\right)^{\hat{t}}, \mathcal{K}\right)$$

where

$$\mathcal{K} = \left(\left(g_{m+l-1}^{\hat{t}\alpha^{(2^m-1)}} \right) . (e(K_{\mathcal{S}}, g_l)) \right)$$

Decrypt$(param, \mathcal{C}, K_{(\mathcal{S},\hat{\mathcal{S}})}, i, \hat{i}, d_{\hat{i}}, \mathcal{S}, \hat{\mathcal{S}})$: If $i \notin \mathcal{S}$ or $\hat{i} \notin \hat{\mathcal{S}}$, output $\perp$. Otherwise, set

$$a_{\hat{\mathcal{S}}} = \left(\prod_{\hat{v} \in \hat{\mathcal{S}}, \hat{v} \neq \hat{i}} Y_{2^m - 1 - \hat{v} + \hat{i}} \right) \ , \ a_{\mathcal{S}} = \left(\prod_{v \in \mathcal{S}, v \neq i} Y_{2^m - 1 - v + i} \right)$$

$$\text{and } b_{\mathcal{S}} = \left(\prod_{v \in \mathcal{S}} Y_{2^m - 1 - v} \right)$$

Let $\mathcal{C} = (c_0, c_1, c_2, c_3)$ and $K_{(\mathcal{S},\hat{\mathcal{S}})} = (\hat{k}_0, \hat{k}_1, \hat{k}_2)$. Output the decrypted message as

$$\hat{\mathcal{M}} = c_3 . \hat{k}_2 . \left(\frac{e(b_{\mathcal{S}}, c_1) e(a_{\mathcal{S}}, c_0)}{e(b_{\mathcal{S}}, c_2)} \right) . \left(\frac{e(d_{\hat{i}} . a_{\hat{\mathcal{S}}}, \hat{k}_0)}{e(Y_{\hat{i}}, \hat{k}_1)} \right)$$

Correctness of this scheme may be easily proven. We introduce the complexity assumption for the proof of security of this scheme here.

The Extended Multilinear Diffie-Hellman Exponent Assumption. Let $param'$ is generated by $\mathbf{SetUp}'(1^{\lambda'}, m + l - 1)$. Choose $\alpha \in \mathbb{Z}_q$ at random (where q is a λ'-bit prime), and let $X_j = g_1^{\alpha^{(2^j)}}$ for $0 \leq j \leq m$. Choose random $t, \hat{t} \in \mathbb{Z}_q$, and let $(V_1, V_2) = \left(g_l^t, g_l^{\hat{t}} \right)$. The decisional (m, l)-Extended Multilinear Diffie Hellman Exponent (EMDHE) problem as defined as follows. Given the tuple

$$\left(param', \{X_j\}_{j \in \{0, \cdots, m\}}, (V_1, V_2), (Z_1, Z_2) \right)$$

distinguish if (Z_1, Z_2) is $\left(g_{m+l-1}^{t\alpha^{(2^m-1)}}, g_{m+l-1}^{\hat{t}\alpha^{(2^m-1)}} \right)$ or a random element of $\mathbb{G}_{m+l-1} \times \mathbb{G}_{m+l-1}$.

Definition 4.5. The decisional (m, l)-Extended Multilinear Diffie-Hellman Exponent (EMDHE) assumption holds for $\mathbf{SetUp}'$ if, for any polynomial m and a probabilistic poly-time algorithm $\mathcal{A}$, $\mathcal{A}$ has negligible advantage in solving the (m, l)-EMDHE problem.

Theorem 4.6. *Let* $\mathbf{SetUp}'$ *be the setup algorithm for an symmetric multilinear map, and let the decisional* (m, l)*-EHDHE assumption holds for* $\mathbf{SetUp}'$*. Then the extended multi-user KAC supporting* N *data classes and* N *data users is non-adaptively CPA secure, where* $N = \binom{m}{l}$*.*

The detailed proof of Theorem 4.6. is very similar to that of Theorem 3.8 in Section 3.4 and is hence avoided. Like the m-HDHE assumption, the (m, l)-EMDHE assumption too currently only holds in the generic group model. Hence our proposed KAC construction using symmetric multilinear maps is provably CPA secure in the generic group model.Finally, we can easily extend this KAC construction for non-adaptive CCA security with full collusion resistance. We present a basic CCA secure KAC construction using symmetric multilinear maps in Appendix C.

5 An Adaptively Secure KAC in the Generic Multilinear Map Model

In this section, we present a fully collusion resistant KAC construction that is adaptively CPA secure in the generic multilinear map model for standard group order parameter q. But first, we state the motivation behind the third construction.

Need for Tighter Security Bounds. We have mentioned before that the m-HDHE and the (m, l)-MDHE assumptions (along with their extended versions) are hard in the generic multilinear map model. However, the presence of high degree exponents such as α^{2^m} in these assumptions means that the adversary can construct polynomials with degree as high as $m2^m$ in the secret α. As pointed out in [BWZ14a], this means the upper bound on the advantage of a generic adversary making at most t queries is only $\approx \left(t^2 2^m/q\right)$. This in turn implies that, althouth the KAC constructions described in Sections 3 and 4 may be proven to be *adaptively* secure in the generic group model, they require the group order parameter $\lambda' \approx 3\lambda$ (instead of the normal λ) or $q = 2^{3\lambda}$ to achieve λ-bit security. This motivates us to investigate the possibility of a KAC construction that is adaptively secure in the generic multilinear map model for $q = 2^\lambda$. We next present the idea behind this construction.

The Basic Idea. Our third KAC construction has two fundamental differences with the first two constructions:

1. The third construction uses a set of public parameters that are not derived from a single scalar α. Instead, each parameter is derived from a separate random scalar. This is essentially done to avoid the high degree exponents in the previous constructions, and is similar to the technique proposed in [BWZ14a]. The trick of embedding linearly many group elements in a logarithmic size public parameter is still used. However, the presence of multiple scalars implies that we cannot use the same embedding technique used in the first two constructions. Instead, we use a Naor-Reingold-style pseudorandom function (PRF)[NR04].

2. The ciphertext in this construction cannot consist of $O(1)$ group elements, since that would make decryption impossible. At the same time, we cannot allow the ciphertext size to exceed polylogarithmic, since that would render the KAC construction inefficient. To achieve this, we ensure that the necessary ciphertext components for decryption are the outputs of another Naor-Reingold-style PRF, and the decrypter can generate precisely these outputs by puncturing the PRF at suitable points, similar to as described in [**?**].

5.1 A Basic Construction

Let $Setup'(1^\lambda, m)$ be the setup algorithm for an m-linear map with groups of prime order q (q being a λ bit prime) and the target group $\mathbb{G}_m$. Our third and final identity-based KAC consists of the following algorithms.

SetUp$(1^\lambda, m)$: Set up the KAC system for $\mathcal{ID}$ consisting of all m bit class identities. Let $param' \leftarrow SetUp'(1^\lambda, m+1)$ be the public parameters for a symmetric multilinear map, with $\mathbb{G}_{m+1}$ being the target group. For $j = 0, \cdots, m-1$ and $b = 0, 1$, generate random $\alpha_{j,b} \in \mathbb{Z}_q$ and let $X_{j,b} = g_1^{\alpha_{j,b}}$. Finally, randomly pick $x \in \mathbb{Z}_q$ and set $W = g_1^x$. Output the public parameter as

$$param = \left(param', \{X_{j,b}\}_{j \in \{0,\cdots,m-1\}, b \in \{0,1\}}, W\right)$$

Discard all $\alpha_{j,b}$ after $param$ has been output. The secret parameter x is retained, but is known only to the trusted third party.

Claim 5.1. For any class index number $i \in \mathcal{ID}$, one can compute $Y_i = g_m^{\prod_{j=0}^{m-1} \alpha_{j,i_j}}$ where i_j is the j-th bit in the binary representation of i.

Proof. Compute Y_i as $e(X_{0,i_0}, X_{1,i_1}, \cdots, X_{m-1,i_{m-1}})$. Note that each Y_i value is essentially the output of a Naor-Reingold-style PRF.

KeyGen(): Randomly pick $\gamma \in \mathbb{Z}_q$. Set the master secret key $msk = \gamma$ and the public key $PK = g_m^{\gamma}$. Output the tuple (msk, PK). Again, msk is known only to the trusted third party.

Encrypt$(param, PK, i, \mathcal{M})$: Randomly choose $t \in \mathbb{Z}_q$ and set

$$\begin{aligned} c_0 &= g_1^r \\ c_{j+1} &= X_{j,(1-u_j)}^t \text{ for } j = 0, 1, \cdots, m-1 \\ c_{m+1} &= (PK.Y_i)^t \\ c_{m+2} &= \mathcal{M}.\left(g_{m+1}^{\gamma x}\right)^t = \mathcal{M}.\left(e(PK, W)\right)^t \end{aligned}$$

Finally, output the ciphertext as

$$\mathcal{C} = (\{c_j\}_{j \in \{0, \cdots, m+2\}})$$

Claim 5.2. For any class index $v \neq i$, one can compute Y_v^t given $\mathcal{C}$.

Proof. Let $\mathcal{C} = (c_0, \cdots, c_{m+2})$. Since $v \neq i$, there exists a bit position $j' \in \{0, \cdots, m-1\}$ such that $v_{j'} = 1 - i_{j'}$. This allows one to compute

$$Y_v^t = e(\{X_{j,v_j}\}_{j \in \{0, \cdots, j-1\} \setminus \{j'\}}, c_{j+1})$$

because:

$$\begin{aligned} Y_v^t &= g_m^{t.\prod_{j=0}^{m-1} \alpha_{j,v_j}} \\ &= g_m^{t\alpha_{j',v_{j'}} \cdot \prod_{j \neq j'} \alpha_{j,v_j}} \\ &= g_m^{t\alpha_{j',1-i_{j'}} \cdot \prod_{j \neq j'} \alpha_{j,v_j}} \\ &= e(\{X_{j,v_j}\}_{j \in \{0, \cdots, m-1\} \setminus \{j'\}}, c_{j'+1}) \end{aligned}$$

Note that for a given i, each Y_v^t for $v \neq i$ and random t is also the output of a Naor-Reingold style PRF. Moreover, the PRF is punctured at i to generate Y_v^t for $v \neq i$ *without* the knowledge of Y_i^t.

Extract$(param, x, \mathcal{S})$: We slightly alter the semantics of the **Extract** operation here in the sense that it uses the secret x instead of msk. For the input subset of data class indices $\mathcal{S}$, the aggregate key is computed as

$$K_{\mathcal{S}} = \left(\prod_{v \in \mathcal{S}} Y_v\right)^x$$

Decrypt$(param, \mathcal{C}, i, \mathcal{S}, K_{\mathcal{S}})$: If $i \notin \mathcal{S}$, output $\perp$. Otherwise, use the result from Claim 5.2 to set

$$a_{\mathcal{S}} = \left(\prod_{v \in \mathcal{S}, v \neq i} Y_v^t\right)$$

Let $\mathcal{C} = (c_0, \cdots, c_{m+1}, c_{m+2})$. Output the decrypted message as

$$\hat{\mathcal{M}} = c_{m+2} \frac{e(K_{\mathcal{S}}, c_0)}{e(c_{m+1}.a_{\mathcal{S}}, W)}$$

Correctness, Security and Extensions. The correctness of this scheme may be easily shown as follows:

$$\begin{aligned}\hat{\mathcal{M}} &= c_{m+2}\frac{e(K_{\mathcal{S}}, c_0)}{e(c_{m+1}.a_{\mathcal{S}}, W)} \\ &= c_{m+2}\frac{e\left(\left(\prod_{v\in\mathcal{S}} Y_v\right)^x, g_1^t\right)}{e\left(\left(g_m^{\gamma}.Y_i.\prod_{v\in\mathcal{S},v\neq i} Y_v\right)^t, g_1^x\right)} \\ &= \frac{c_{m+2}}{e((g_m^{\gamma})^t, g_1^x)} \\ &= \mathcal{M}\end{aligned}$$

Unlike the previous constructions, there does not seem to exist any well known hardness assumption (even in the generic group model) that may be reduced to the third construction. Nevertheless, the construction is provably secure in the generic group model (see Appendix D.1) in the *adaptive* setting, with tighter bounds on the group order parameters. An extension to this construction for broadcasting the aggregate key $K_{\mathcal{S}}$ to a target subset of data users is also straightforward to achieve.

6 Conclusions and Open Problems

We presented the first identity-based key-aggregate cryptosystem (KAC) for access delegation to arbitrarily large subsets of data classes shared online, among any number of authorized data users. We proposed three different $O(\log N)$-way multilinear map-based constructions that support N data classes and N data users, with low overhead for ciphertexts, aggregate keys and all public and private parameters. We proved the generic group model based security and full collusion resistance of each of the schemes under different security assumptions. For broadcasting the aggregate keys among multiple users, we showed how to efficiently combine the stand-alone KAC constructions with broadcast encryption schemes. This in turn significantly reduces the secure channel requirement for KAC constructions from those in the existing literature.

We leave as an open problem the question of building identity-based KAC constructions that are secure in the standard model. Another interesting problem is to build efficiently revocable cryptosystems for data sharing with traitor tracing properties. In particular, it would be interesting to build a cryptosystem that allows revoking a user's access rights without the need for re-encrypting all the shared data. Finally, a possible future work is to present an instantiation of our proposed scheme using a secure multilinear map construction, and test the performance of the resulting system in actual data sharing environments such as the cloud.

References

[ACP09] Michel Abdalla, Céline Chevalier, and David Pointcheval. Smooth projective hashing for conditionally extractable commitments. In *Advances in Cryptology-CRYPTO 2009*, pages 671–689. Springer, 2009.

[ADSFM12] Giuseppe Ateniese, Alfredo De Santis, Anna Lisa Ferrara, and Barbara Masucci. Provably-secure time-bound hierarchical key assignment schemes. *Journal of cryptology*, 25(2):243–270, 2012.

[AFGH06] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Transactions on Information and System Security (TISSEC)*, 9(1):1–30, 2006.

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *Advances in Cryptology–EUROCRYPT 2005*, pages 440–456. Springer, 2005.

[BCHL09] Josh Benaloh, Melissa Chase, Eric Horvitz, and Kristin Lauter. Patient controlled encryption: ensuring privacy of electronic medical records. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 103–114. ACM, 2009.

[BF03] Dan Boneh and Matthew Franklin. Identity-based encryption from the weil pairing. *SIAM Journal on Computing*, 32(3):586–615, 2003.

[BGW05] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *Advances in Cryptology–CRYPTO 2005*, pages 258–275. Springer, 2005.

[BWZ14a] Dan Boneh, Brent Waters, and Mark Zhandry. Low overhead broadcast encryption from multilinear maps. In *Advances in Cryptology–CRYPTO 2014*, pages 206–223. Springer, 2014.

[BWZ14b] Dan Boneh, David J Wu, and Joe Zimmerman. Immunizing multilinear maps against zeroizing attacks. *IACR Cryptology ePrint Archive*, 2014:930, 2014.

[CCT+14] Cheng-Kang Chu, Sherman SM Chow, Wen-Guey Tzeng, Jianying Zhou, and Robert H Deng. Key-aggregate cryptosystem for scalable data sharing in cloud storage. *Parallel and Distributed Systems, IEEE Transactions on*, 25(2):468–477, 2014.

[CHK04] Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In *Advances in Cryptology-Eurocrypt 2004*, pages 207–222. Springer, 2004.

[CHL+15] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In *Advances in Cryptology–EUROCRYPT 2015*, pages 3–12. Springer, 2015.

[Clo] CloudPro. Dropbox goes big on security with Enterprise offering.

[CLT13] Jean-Sébastien Coron, Tancrede Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In *Advances in Cryptology–CRYPTO 2013*, pages 476–493. Springer, 2013.

[CLT14] Jean-Sebastien Coron, Tancrede Lepoint, and Mehdi Tibouchi. Cryptanalysis of two candidate fixes of multilinear maps over the integers. *IACR Cryptology ePrint Archive*, 2014:975, 2014.

[CS02] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *Advances in CryptologyEurocrypt 2002*, pages 45–64. Springer, 2002.

[DMMM+12] Idilio Drago, Marco Mellia, Maurizio M Munafo, Anna Sperotto, Ramin Sadre, and Aiko Pras. Inside dropbox: understanding personal cloud storage services. In *Proceedings of the 2012 ACM conference on Internet measurement conference*, pages 481–494. ACM, 2012.

[GGH13] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *Eurocrypt*, volume 7881, pages 1–17. Springer, 2013.

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In *Theory of Cryptography*, pages 498–527. Springer, 2015.

[GHMS14] Craig Gentry, Shai Halevi, Hemanta K Maji, and Amit Sahai. Zeroizing without zeroes: Cryptanalyzing multilinear maps without encodings of zero. *IACR Cryptology ePrint Archive*, 2014:929, 2014.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98. Acm, 2006.

[Kil88] Joe Kilian. Founding crytpography on oblivious transfer. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 20–31. ACM, 1988.

[Mos10] Dana Moshkovitz. An alternative proof of the schwartz-zippel lemma. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 17, page 34, 2010.

[NR04] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. *Journal of the ACM (JACM)*, 51(2):231–262, 2004.

[Pau] Ian Paul. Google Drive: The Pros and Cons.

[PSM15] Sikhar Patranabis, Yash Shrivastava, and Debdeep Mukhopadhyay. Dynamic key-aggregate cryptosystem on elliptic curves for online data sharing. In *Progress in Cryptology–INDOCRYPT 2015*, pages 25–44. Springer, 2015.

A Relaxing Assumptions 3.4 and 4.4

In the two KAC constructions presented so far, we have assumed that all plaintext messages $\mathcal{M}$ may be efficiently embedded as elements in the respective target multilinear groups. However, embedding any general class of data as group elements is extremely challenging and requires *public samplability* - a feature that makes a multilinear map constructions insecure. However, a workaround may be readily proposed. We first note that in any ciphertext output by **Encrypt**, the message $\mathcal{M}$ is essentially multiplied with a random secret group element ρ. Rather than embedding $\mathcal{M}$ as a group element, we propose hashing ρ using a collision resistant hash function H, and then outputting $\mathcal{M} \odot H(\rho)$ in the ciphertext (here $\odot$ denotes an appropriate operator). In order to ensure that the constructions are still provably secure in the standard model, we propose that H be chosen from the family of *smooth projective hash functions* [CS02], that do not require the use of random oracles to prove security. Smooth projective hash functions are very efficient to construct and can be designed to be collision-resistant [ACP09], making them an ideal choice for our constructions.

B CCA Secure Basic KAC Using Asymmetric Multilinear Maps

In this section we demonstrate how to extend the basic identity-based KAC construction (for single owner - single user scenario) to obtain non-adaptive chosen ciphertext security in the generic group model, while maintaining full collusion resistance. The proof ideas stem from [CCT+14,PSM15]. We have the following additional requirements for achieving CCA security:

- A signature scheme $(SigKeyGen, Sign, Verify)$.
- A collision resistant hash function for mapping verification keys to $\mathbb{Z}_q$.

For simplicity of presentation, we assume here that the signature verification keys are encoded as elements of $\mathbb{Z}_q$. We avoid any further mention of the hash function in the forthcoming discussion, since it is implicitly assumed that any signature value we refer to is essentially the hash value corresponding to the original signature.

Construction. It is to be noted that unlike non-adaptive CPA security, non-adaptive CCA security for our proposed KAC under the m-HDHE assumption requires that the system handles at most $N-1$ data classes, where $N = 2^m - 1$. The reason for this will be apparent in the proof. hence for consistency of notation, we describe here the construction of the CCA-secure KAC for $N-1$ data classes. Recall that $\mathbf{SetUp}''(1^\lambda, \mathbf{m})$ is the setup algorithm for an asymmetric multilinear map, where groups have prime order q (where q is a λ bit prime).

$\mathbf{SetUp}(1^\lambda, m)$: Takes as input the length m of identities and the group order parameter λ. Let $\mathcal{ID} = \{0,1\}^m \backslash (\{0\}^m \cup \{1\}^m)$ be the data class identity space with $N-1$ classes. Also, let $\mathbf{m}$ be the $m+1$ length vector consisting of all ones. Let $param'' \leftarrow SetUp''(1^\lambda, 2\mathbf{m})$ be the public parameters for a multilinear map, with $\mathbb{G}_{2\mathbf{m}}$ being the target group. Next, choose a random $\alpha \in \mathbb{Z}_q$. Set $X_j = g_{\mathbf{x}_j}^{\alpha^{(2^j)}}$ for $0 \leq j \leq m-1$ and $X_m = g_{\mathbf{x}_m}^{\alpha^{(2^m+1)}}$. Output the public parameter tuple $param$ as

$$param = (param'', \{X_j\}_{j \in \{0,\cdots,m\}})$$

Discard α after $param$ has been output.

$\mathbf{KeyGen}()$: Same as in the construction of Section 3.1.

Encrypt$(param, PK, i, \mathcal{M})$: Run the $SigKeyGen$ algorithm to obtain a signature signing key K_{SIG} and a verification key $V_{SIG} \in \mathbb{Z}_q$. Randomly choose $t \in \mathbb{Z}_q$. Compute

$$\mathcal{C}' = (g_{\mathbf{m}}^t, (PK.Y_i.Y_{2^m-1}^{V_{SIG}})^t, \mathcal{M}.g_{2\mathbf{m}}^{t\alpha^{(2^m)}})$$

and output the ciphertext as

$$\mathcal{C} = (\mathcal{C}', Sign(\mathcal{C}', K_{SIG}), V_{SIG})$$

Extract$(param, msk, \mathcal{S})$: Same as in the construction of Section 3.1.

Decrypt$(param, \mathcal{C}, i, \mathcal{S}, K_{\mathcal{S}})$: Let $\mathcal{C} = ((c_0, c_1, c_2), \sigma, V_{SIG})$. Verify that σ is a valid signature of (c_0, c_1, c_2) under the key V_{SIG}. If not, output $\perp$. Also, if $i \notin \mathcal{S}$, output $\perp$. Otherwise, set

$$SIG_{\mathcal{S}} = \prod_{v \in \mathcal{S}} Y_{2^{m+1}-1-v}^{V_{SIG}}$$
$$a_{\mathcal{S}} = \prod_{v \in \mathcal{S}, v \neq i} Y_{2^m-v+i}$$
$$b_{\mathcal{S}} = \prod_{v \in \mathcal{S}} Y_{2^m-v}$$

Note that these can be computed as $1 \leq i, v \leq N-1 (= 2^m - 2)$. *This is precisely why we allow only* $N-1$ *data classes.* Next, pick a random $w \in \mathbb{Z}_q$ and set

$$\hat{h}_1 = (K_{\mathcal{S}}.SIG_{\mathcal{S}}.a_{\mathcal{S}}.(PK.Y_i.Y_{2^m-1}^{V_{SIG}})^w)$$
$$\hat{h}_2 = (b_{\mathcal{S}}.g_{\mathbf{m}}^w)$$

Output the decrypted message

$$\hat{\mathcal{M}} = c_2 \frac{e(\hat{h}_1, c_0)}{e(\hat{h}_2, c_1)}$$

The proof of correctness of this scheme is presented below.

$$\begin{aligned}
\hat{\mathcal{M}} &= c_3 \frac{e(\hat{h}_1, c_1)}{e(\hat{h}_2, c_2)} \\
&= c_3 . \left(\frac{e\left(K_{\mathcal{S}}.SIG_{\mathcal{S}}.a_{\mathcal{S}}, g_{\mathbf{m}}^t\right)}{e\left(b_{\mathcal{S}}, (PK.Y_i.Y_{2^m-1}^{V_{SIG}})^t\right)} \right) . \left(\frac{e\left(\left(PK.Y_i.Y_{2^m-1}^{V_{SIG}}\right)^w, g_{\mathbf{m}}^t\right)}{e\left(g_{\mathbf{m}}^w, (PK.Y_i.Y_{2^m-1}^{V_{SIG}})^t\right)} \right) \\
&= c_3 . \left(\frac{e\left(K_{\mathcal{S}}, g_{\mathbf{m}}^t\right)}{e\left(b_{\mathcal{S}}, PK^t\right)} \right) . \left(\frac{e\left(SIG_{\mathcal{S}}, g_{\mathbf{m}}^t\right)}{e\left(b_{\mathcal{S}}, \left(Y_{2^m-1}^{V_{SIG}}\right)^t\right)} \right) . \left(\frac{e\left(a_{\mathcal{S}}, g_{\mathbf{m}}^t\right)}{e\left(b_{\mathcal{S}}, Y_i^t\right)} \right) \\
&= c_3 \frac{e(\prod_{v \in \mathcal{S}, v \neq i} Y_{2^m-v+i}, g_{\mathbf{m}}^t)}{e(\prod_{v \in \mathcal{S}} Y_{2^m-v}, Y_i^t)} \\
&= \frac{\mathcal{M}.g_{2\mathbf{m}}^{t\alpha^{(2^m)}}}{e(Y_{2^m}, g_{\mathbf{m}}^t)} \\
&= \mathcal{M}
\end{aligned}$$

Note that the overhead for the ciphertext, aggregate key, public parameters, and the private and public keys, remains unchanged. The main change from the original scheme is in the fact that decryption requires a randomization value $w \in \mathbb{Z}_q$.

Claim B.1. *For a given $i \in \mathcal{S}$, the pair $(\hat{h}_1, \hat{h}_2)$ is chosen from the following distribution*

$$\left((Y_{2^m})^{-1} \cdot \left(PK.Y_i.Y_{2^m-1}^{V_{SIG}}\right)^x, (g_{\mathbf{m}})^x\right)$$

where x is uniformly randomly chosen from $\mathbb{Z}_q$.

Proof. We have

$$\begin{aligned}\hat{h}_2 &= (b_{\mathcal{S}}.g_{\mathbf{m}}^w) \\ &= g_{\mathbf{m}}^{(w+(\sum_{v\in\mathcal{S}} \alpha^{2^m-v}))} \\ &= (g_{\mathbf{m}})^x\end{aligned}$$

Also, observe the following:

$$\begin{aligned}\hat{h}_1 &= (K_{\mathcal{S}}.SIG_{\mathcal{S}}.a_{\mathcal{S}}) \cdot \left(PK.Y_i.Y_{2^m-1}^{V_{SIG}}\right)^w \\ &= (Y_{2^m})^{-1} (K_{\mathcal{S}}.SIG_{\mathcal{S}}.a_{\mathcal{S}}.Y_{2^m}) \cdot \left(PK.Y_i.Y_{2^m-1}^{V_{SIG}}\right)^w \\ &= (Y_{2^m})^{-1} \left(PK.Y_i.Y_{2^m-1}^{V_{SIG}}\right)^{\left(\sum_{v\in\mathcal{S}} \alpha^{2^m-v}\right)} \cdot \left(PK.Y_i.Y_{2^m-1}^{V_{SIG}}\right)^w \\ &= (Y_{2^m})^{-1} \left(PK.Y_i.Y_{2^m-1}^{V_{SIG}}\right)^{(w+\left(\sum_{v\in\mathcal{S}} \alpha^{2^m-v}\right))} \\ &= (Y_{2^m})^{-1} \left(PK.Y_i.Y_{2^m-1}^{V_{SIG}}\right)^x\end{aligned}$$

This randomization slows down the decryption by a factor of two, but is vital from the point of view of CCA-security. Note that the distribution $(\hat{h}_1, \hat{h}_2)$ depends *only on the data class i for the message $\mathcal{M}$ to be decrypted* and is *completely independent of the subset $\mathcal{S}$ used to encrypt it.*

CCA Security. We next prove the non-adaptive CCA security of this scheme. Note that a signature scheme $(SigKeyGen, Sign, Verify)$ is said to be (ϵ, q_S) strongly existentially unforgeable if no poly-time adversary, making at most q_S signature signature queries, fails to produce some new message-signature pair with probability at least ϵ. For a more complete description, refer [CHK04].

Theorem B.2. *Let **Setup**$''$ be the setup algorithm for an asymmetric multilinear map, and let the decisional m-Hybrid Diffie-Hellman Exponent assumption holds for SetUp$''$. Also, assume that the signature scheme is strongly existentially unforgeable. Then the modified KAC construction for $N-1$ data classes presented above is non-adaptively CCA secure.*

Proof. Once again, let $\mathcal{A}$ be a poly-time adversary such that $|Adv_{\mathcal{A},N-1} - \frac{1}{2}| > \epsilon_1 + \epsilon_2$ for the proposed KAC system parameterized with an identity space $\mathcal{ID}$ of size $N-1 = 2^m - 2$. Let the signature scheme is (ϵ_2, q_S) strongly existentially unforgeable. We build an algorithm $\mathcal{B}$ that has advantage at least ϵ_1 in solving the decisional m-HDHE problem for **Setup**$''$. $\mathcal{B}$ takes as input a random m-HDHE challenge $(param'', \{X_j\}_{j\in\{0,\cdots,m\}}, V, Z)$ where:

- $param'' \leftarrow SetUp''(1^\lambda, 2\mathbf{m})$
- $X_j = g_{\mathbf{x}_j}^{\alpha^{(2^j)}}$ for $0 \le j \le m-1$
- $X_m = g_{\mathbf{x}_m}^{\alpha^{(2^{m+1})}}$
- $V = g_{\mathbf{m}}^t$ for a random $t \in \mathbb{Z}_q$, q being a λ bit prime
- Z is either $g_{2\mathbf{m}}^{t\alpha^{(2^m)}}$ or a random element of $\mathbb{G}_{2\mathbf{m}}$

$\mathcal{B}$ then proceeds as follows.

Commit: $\mathcal{B}$ runs $\mathcal{A}$ and receives the set $\mathcal{S}^*$ of data classes that $\mathcal{A}$ wishes to be challenged on. $\mathcal{B}$ then randomly chooses a data class $i \in \mathcal{S}^*$ and provides it to $\mathcal{A}$.

SetUp: $\mathcal{B}$ should generate the public *param*, public key PK and the aggregate key $K_{\overline{\mathcal{S}^*}}$ and provide them to $\mathcal{A}$. Algorithm $\mathcal{B}$ first runs the $SigKeyGen$ algorithm to obtain a signature signing key K^*_{SIG} and a corresponding verification key $V^*_{SIG} \in \mathbb{Z}_q$. The various items to be provided to $\mathcal{A}$ are generated as follows.

- *param* is set as $(param'', \{X_i\}_{i\in\{0,\cdots,m\}})$.
- PK is set as $(g^u_{\mathbf{m}}) / \left(Y_i.Y_{2^m-1}^{V^*_{SIG}}\right)$ where u is chosen uniformly at random from $\mathbb{Z}_q$ and Y_i, Y_{2^m-1} are computed as mentioned in Claim 3.1. Note that this is equivalent to setting $msk = u - \alpha^i - V^*_{SIG}\alpha^{2^m-1}$.
- $\mathcal{B}$ then computes
$$K_{\overline{\mathcal{S}^*}} = \prod_{v\notin\mathcal{S}^*} \frac{Y^u_{2^m-v}}{(Y_{2^m-v+i}).(Y^{V^*_{SIG}}_{2^{m+1}-1-v})}$$
Observe that $K_{\overline{\mathcal{S}^*}} = \prod_{v\notin\mathcal{S}^*} PK^{\alpha^{2^m-v}}$, as desired. Moreover, $\mathcal{B}$ is aware that $i \notin \overline{\mathcal{S}^*}$ (implying $i \neq v$), and hence has all the resources to compute $K_{\overline{\mathcal{S}^*}}$.

Since the $g_{\mathbf{m}}$, α, u and t values are chosen uniformly at random, *all parameters and keys have an identical distribution to that in the actual construction.*

Query Phase 1: Algorithm $\mathcal{A}$ now issues decryption queries. Let $(\mathcal{C}, v)$ be a decryption query $\mathcal{C}$ is obtained by $\mathcal{A}$ using some subset $\mathcal{S}$ containing v. However, $\mathcal{B}$ is not given the knowledge of $\mathcal{S}$. Let $\mathcal{C} = ((c_0, c_1, c_2), \sigma, V_{SIG})$. Algorithm $\mathcal{B}$ first runs $Verify$ to check if the signature σ is valid on (c_0, c_1, c_2) using V_{SIG}. If invalid, $\mathcal{B}$ returns $\perp$. If $V_{SIG} = V^*_{SIG}$, $\mathcal{B}$ outputs a random bit $b \in \{0, 1\}$ and *aborts* the simulation. Otherwise, the challenger picks a random $x \in \mathbb{Z}_q$. It then sets

$$\begin{aligned}
\hat{h}_0 &= Y_{2^m-1}^{(V_{SIG}-V^*_{SIG})}.Y_v.Y_i^{-1} \\
\hat{h'}_0 &= (Y_{v+1}/Y_{i+1})^{\frac{1}{(V_{SIG}-V^*_{SIG})}} \\
\hat{h}_2 &= g^x_{\mathbf{m}}.Y_1^{\frac{1}{(V_{SIG}-V^*_{SIG})}} \\
\hat{h}_1 &= \left(\hat{h}_2\right)^u . \left(\hat{h}_0\right)^x . \left(\hat{h'}_0\right)
\end{aligned}$$

Note that $\hat{h'}_0$ can be computed following Claim 3.1 as $1 \leq i, v \leq 2^m - 2$. $\mathcal{B}$ responds with $\mathcal{M}' = c_2 \frac{e(\hat{h}_1, c_0)}{e(\hat{h}_2, c_1)}$.

Claim B.3. *$\mathcal{B}$'s response is exactly as in a real attack scenario, that is, for some x' chosen uniformly at random from $\mathbb{Z}_q$, we have*

$$\hat{h}_1 = (Y_{2^m})^{-1} . \left(PK.Y_v.Y_{2^m-1}^{V_{SIG}}\right)^{x'} \text{ and } \hat{h}_2 = g^{x'}_{\mathbf{m}}$$

Proof. Set $x' = x + \frac{\alpha}{(V_{SIG}-V^*_{SIG})}$. Since x is uniform in $\mathbb{Z}_q$, so is x'. Now, observe that

$$\begin{aligned}
\hat{h}_2 &= g^x_{\mathbf{m}}.Y_1^{\frac{1}{(V_{SIG}-V^*_{SIG})}} \\
&= g^x_{\mathbf{m}}.g_{\mathbf{m}}^{\frac{\alpha}{(V_{SIG}-V^*_{SIG})}} \\
&= g^{x'}_{\mathbf{m}}
\end{aligned}$$

Next, observe the following:

$$\begin{aligned}
\hat{h}_1 &= \left(\hat{h}_2\right)^u . \left(\hat{h}_0\right)^x . \left(\hat{h'}_0\right) \\
&= \left(g_{\mathbf{m}}^u\right)^{x'} . \left(Y_{2^m-1}^{x(V_{SIG}-V_{SIG}^*)}\right) . \left(Y_v/Y_i\right)^{\left(x+\frac{\alpha}{(V_{SIG}-V_{SIG}^*)}\right)} \\
&= \left(PK.Y_i.Y_{2^m-1}^{V_{SIG}^*}\right)^{x'} . \left(Y_{2^m-1}^{x(V_{SIG}-V_{SIG}^*)}\right) . \left(Y_v/Y_i\right)^{x'} \\
&= \left(PK.Y_v.Y_{2^m-1}^{V_{SIG}}\right)^{x'} . \left(Y_{2^m-1}^{(x-x')(V_{SIG}-V_{SIG}^*)}\right) \\
&= \left(PK.Y_v.Y_{2^m-1}^{V_{SIG}}\right)^{x'} . \left(Y_{2^m-1}^{-\alpha}\right) \\
&= \left(Y_{2^m}\right)^{-1} . \left(PK.Y_v.Y_{2^m-1}^{V_{SIG}}\right)^{x'}
\end{aligned}$$

Thus, $\mathcal{B}$'s response is identical to $\mathbf{Decrypt}(\mathcal{C}, v, \mathcal{S}, K_{\mathcal{S}})$, even though $\mathcal{B}$ does not possess the knowledge of the subset $\mathcal{S}$ used by $\mathcal{A}$ to obtain $\mathcal{C}$.

Challenge: $\mathcal{A}$ picks at random two messages $\mathcal{M}_0$ and $\mathcal{M}_1$ from the set of possible plaintext messages in $\mathbb{G}_{\mathbf{2m}}$, and provides them to $\mathcal{B}$. $\mathcal{B}$ randomly picks $b \in \{0, 1\}$, and sets

$$\begin{aligned}
\mathcal{C} &= (V, V^u, \mathcal{M}_b.Z) \\
\mathcal{C}^* &= (\mathcal{C}, Sign(\mathcal{C}, K_{SIG}^*), V_{SIG}^*)
\end{aligned}$$

The challenge posed to $\mathcal{A}$ is $(\mathcal{C}^*, \mathcal{M}_0, \mathcal{M}_1)$. It can be easily shown that when $Z = g_{\mathbf{2m}}^{t\alpha^{(2^m)}}$ (i.e. the input to $\mathcal{B}$ is a valid m-HDHE tuple), then this is a valid challenge to $\mathcal{A}$ as in a real attack.

Query Phase 2: Same as in query phase 1.

Guess: The adversary $\mathcal{A}$ outputs a guess b' of b. If $b' = b$, $\mathcal{B}$ outputs 0 (indicating that $Z = g_{\mathbf{2m}}^{t\alpha^{(2^m)}}$). Otherwise, it outputs 1 (indicating that Z is a random element in $\mathbb{G}_{\mathbf{2m}}$).

We now bound the probability that $\mathcal{B}$ aborts the simulation as a result of one of the decryption queries by $\mathcal{A}$. We claim that $Pr[\mathbf{abort}] < \epsilon_2$; otherwise one can use $\mathcal{A}$ to forge signatures with probability at least ϵ_2. A very brief proof of this may be stated as follows. We may construct a simulator that knows the secret u and receives K_{SIG}^* as a challenge in an existential forgery game. $\mathcal{A}$ can then cause an abort by producing a query that leads to an existential forgery under K_{SIG}^* on some ciphertext. Our simulator uses this forgery to win the existential forgery game. Only one chosen message query is made by the adversary during the game to generate the signature corresponding to the challenge ciphertext. Thus, $Pr[\text{abort}] < \epsilon_2$.

We conclude that $\mathcal{B}$ has the same advantage ϵ as $\mathcal{A}$, which must therefore be negligible, as desired. This completes the proof of Theorem B.2. □

Similar CCA secure extensions can also be made to the generalized KAC construction for the multi-user scenario.

C A CCA Secure Basic KAC using Symmetric Multilinear Maps

We now demonstrate how to extend the identity-based KAC construction using multilinear maps to obtain non-adaptive chosen ciphertext security. We again make use of the signature scheme coupled with the collusion resistant hash function that we introduced in Appendix B. In this CCA secure construction, we force the class index value i to be strictly less than $2^m - 2$ instead of $2^m - 1$ in the

CPA secure construction.

SetUp$(1^\lambda, m)$: **SetUp**$(1^\lambda, (m, l))$: Set up the KAC system for $\mathcal{ID}$ consisting of all m bit class identities with Hamming weight l, that is $N = |\mathcal{ID}| = \binom{m}{l}$. Since $1 \leq l \leq m-1$, we have $N \leq 2^{m-2}$. Let $param' \leftarrow SetUp'(1^\lambda, m+l-1)$ be the public parameters for a symmetric multilinear map, with $\mathbb{G}_{m+l-1}$ being the target group. Choose a random $\alpha \in \mathbb{Z}_q$. Set $X_j = g_1^{\alpha^{(2^j)}}$ for $0 \leq j \leq m$. Output the public parameter tuple $param$ as

$$param = (param', \{X_j\}_{j \in \{0,\cdots,m\}})$$

Discard α after $param$ has been output.

KeyGen(): Randomly pick $\gamma \in \mathbb{Z}_q$. Set $msk = \gamma$ and $PK = g_l^\gamma$. Output the tuple (msk, PK).

Encrypt$(param, PK, i, \mathcal{M})$: Run the $SigKeyGen$ algorithm to obtain a signature signing key K_{SIG} and a verification key $V_{SIG} \in \mathbb{Z}_q$. Randomly choose $t \in \mathbb{Z}_q$. Compute

$$\mathcal{C}' = (g_l^t, (PK.Y_i.g_{m-1}^{V_{SIG}})^t, \mathcal{M}.g_{m+l-1}^{t\alpha^{(2^m-1)}})$$

and output the ciphertext as

$$\mathcal{C} = (\mathcal{C}', Sign(\mathcal{C}', K_{SIG}), V_{SIG})$$

Extract$(param, msk, \mathcal{S})$: Let $msk = (msk_1, msk_2)$. For the input subset of data class indices $\mathcal{S}$, the aggregate key is computed as

$$K_{\mathcal{S}} = \prod_{v \in \mathcal{S}} (Y_{2^m-1-v})^{msk_1}$$

Recall that Y_{2^m-1-v} can be computed as per Claim 4.3 for $j \in \mathcal{ID}$. Note that this is equivalent to setting $K_{\mathcal{S}}$ to $\prod_{v \in \mathcal{S}} \left(g_{m-1}^{msk}\right)^{\alpha^{2^m-1-v}}$.

Decrypt$(param, \mathcal{C}, i, \mathcal{S}, K_{\mathcal{S}})$: Let $\mathcal{C} = ((c_0, c_1, c_2), \sigma, V_{SIG})$. Verify that σ is a valid signature of (c_0, c_1, c_2) under the key V_{SIG}. If not, output $\perp$. Also, if $i \notin \mathcal{S}$, output $\perp$. Otherwise, set

$$SIG_{\mathcal{S}} = \prod_{v \in \mathcal{S}} g_{m-1}^{V_{SIG}}$$
$$a_{\mathcal{S}} = \prod_{v \in \mathcal{S}, v \neq i} Y_{2^m-1-v+i}$$
$$b_{\mathcal{S}} = \prod_{v \in \mathcal{S}} Y_{2^m-1-v}$$

Next, pick a random $w \in \mathbb{Z}_q$ and set

$$\hat{h}_1 = (K_{\mathcal{S}}.SIG_{\mathcal{S}}.a_{\mathcal{S}}.(PK.Y_i.g_{m-1}^{V_{SIG}})^w)$$
$$\hat{h}_2 = (b_{\mathcal{S}}.g_{m-1}^w)$$

Output the decrypted message

$$\hat{\mathcal{M}} = c_2 \frac{e(\hat{h}_1, c_0)}{e(\hat{h}_2, c_1)}$$

The proof of correctness of this scheme is presented in Appendix B. Note that the overhead for the ciphertext, aggregate key, public parameters, and the private and public keys, remains unchanged. The main change from the original scheme is in the fact that decryption requires a randomization value $w \in \mathbb{Z}_q$.

Claim C.1. *For a given $i \in \mathcal{S}$, the pair $(\hat{h}_1, \hat{h}_2)$ is chosen from the following distribution*

$$\left((Y_{2^m-1})^{-1} . \left(PK.Y_i.g_{m-1}^{V_{SIG}} \right)^x , (g_{m-1})^x \right)$$

where x is uniformly randomly chosen from $\mathbb{Z}_q$.
Proof. Similar to the proof in Appendix B.

Once again, note that the distribution $(\hat{h}_1, \hat{h}_2)$ depends *only on the data class i for the message $\mathcal{M}$ to be decrypted* and is *completely independent of the subset $\mathcal{S}$ used to encrypt it.* We next prove the non-adaptive CCA security of this scheme.

Theorem C.2. *Let **Setup**$'$ be the setup algorithm for a symmetric multilinear map, and let the decisional (m, l)-Multilinear Diffie-Hellman Exponent assumption holds for SetUp$'$. Then our proposed construction of KAC for N data classes presented in this section is non-adaptively CCA secure for $N = \binom{m}{l}$, where each data class number $i < 2^m - 2$.*

Proof. Let $\mathcal{A}$ be a poly-time adversary such that $|Adv_{\mathcal{A},N'} - \frac{1}{2}| > \epsilon$ for the proposed KAC system parameterized with an identity space $\mathcal{ID}'$ of size N', and ϵ is a non-negligible positive constant. We build an algorithm $\mathcal{B}$ that has advantage at least ϵ in solving the decisional (m, l)-MDHE problem for **Setup**$'$. $\mathcal{B}$ takes as input a random (m, l)-MDHE challenge tuple $(param', \{X_j\}_{j\in\{0,\cdots,m\}}, V, Z)$, where:

- $param' \leftarrow SetUp'(1^\lambda, m + l - 1)$
- $X_j = g_1^{\alpha^{(2^j)}}$ for $0 \le j \le m$
- $V = g_l^t$ for a random $t \in \mathbb{Z}_q$ (where q is a λ bit prime)
- Z is either $g_{m+l-1}^{t\alpha^{(2^m-1)}}$ or a random element of $\mathbb{G}_{m+l-1}$.

$\mathcal{B}$ then proceeds as follows.

Commit: $\mathcal{B}$ runs $\mathcal{A}$ and receives the set $\mathcal{S}^*$ of data classes that $\mathcal{A}$ wishes to be challenged on. $\mathcal{B}$ then randomly chooses a data class $i \in \mathcal{S}^*$ and provides it to $\mathcal{A}$.

SetUp: $\mathcal{B}$ should generate the public $param$, public key PK and the aggregate key $K_{\overline{\mathcal{S}^*}}$ and provide them to $\mathcal{A}$. Algorithm $\mathcal{B}$ first runs the $SigKeyGen$ algorithm to obtain a signature signing key K_{SIG}^* and a corresponding verification key $V_{SIG}^* \in \mathbb{Z}_q$. The various items to be provided to $\mathcal{A}$ are generated as follows.

- $param$ is set as $(param'', \{X_i\}_{i\in\{0,\cdots,m\}})$.
- PK is set as $(g_l^u) / \left(Y_i.g_{m-1}^{V_{SIG}^*} \right)$ where u is chosen uniformly at random from $\mathbb{Z}_q$.
- $\mathcal{B}$ then computes

$$K_{\overline{\mathcal{S}^*}} = \prod_{v \notin \mathcal{S}^*} \frac{Y_{2^m-1-v}^u}{(Y_{2^m-1-v+i}).(g_{m-1}^{V_{SIG}^*})}$$

Since the $g_{\mathbf{m}}$, α, u and t values are chosen uniformly at random, *all the parameters and keys have an identical distribution to that in the actual construction.*

Query Phase 1: Algorithm $\mathcal{A}$ now issues decryption queries. Let $(\mathcal{C}, v)$ be a decryption query $\mathcal{C}$ is obtained by $\mathcal{A}$ using some subset $\mathcal{S}$ containing v. However, $\mathcal{B}$ is not given the knowledge of $\mathcal{S}$. Let $\mathcal{C} = ((c_0, c_1, c_2), \sigma, V_{SIG})$. Algorithm $\mathcal{B}$ first runs $Verify$ to check if the signature σ is valid on

(c_0, c_1, c_2) using V_{SIG}. If invalid, $\mathcal{B}$ returns $\perp$. If $V_{SIG} = V^*_{SIG}$, $\mathcal{B}$ outputs a random bit $b \in \{0,1\}$ and *aborts* the simulation. Otherwise, the challenger picks a random $x \in \mathbb{Z}_q$. It then sets

$$\hat{h}_0 = Y^{(V_{SIG} - V^*_{SIG})}.Y_v.Y_i^{-1}$$
$$\hat{h'}_0 = (Y_{v+1}/Y_{i+1})^{\frac{1}{(V_{SIG} - V^*_{SIG})}}$$
$$\hat{h}_2 = g^x_{\mathbf{m}}.Y_1^{\frac{1}{(V_{SIG} - V^*_{SIG})}}$$
$$\hat{h}_1 = \left(\hat{h}_2\right)^u . \left(\hat{h}_0\right)^x . \left(\hat{h'}_0\right)$$

$\mathcal{B}$ responds with $\mathcal{M}' = c_2 \frac{e(\hat{h}_1, c_0)}{e(\hat{h}_2, c_1)}$.

Claim C.3. *$\mathcal{B}$'s response is exactly as in a real attack scenario, that is, for some x' chosen uniformly at random from $\mathbb{Z}_q$, we have*

$$\hat{h}_1 = (Y_{2^m - 1})^{-1} . \left(PK.Y_v.g^{V_{SIG}}_{m-1}\right)^{x'} \text{ and } \hat{h}_2 = g^{x'}_{m-1}$$

Proof. Similar to the proof in Appendix B.

Thus, by the result in Claim C.2, $\mathcal{B}$'s response is identical to $\mathbf{Decrypt}(\mathcal{C}, v, \mathcal{S}, K_{\mathcal{S}})$, even though $\mathcal{B}$ does not possess the knowledge of the subset $\mathcal{S}$ used by $\mathcal{A}$ to obtain $\mathcal{C}$.

Challenge: $\mathcal{A}$ picks at random two messages $\mathcal{M}_0$ and $\mathcal{M}_1$ from the set of possible plaintext messages in $\mathbb{G}_{\mathbf{2m}}$, and provides them to $\mathcal{B}$. $\mathcal{B}$ randomly picks $b \in \{0,1\}$, and sets

$$\mathcal{C} = (V, V^u, \mathcal{M}_b.Z)$$
$$\mathcal{C}^* = (\mathcal{C}, Sign(\mathcal{C}, K^*_{SIG}), V^*_{SIG})$$

The challenge posed to $\mathcal{A}$ is $(\mathcal{C}^*, \mathcal{M}_0, \mathcal{M}_1)$. It can be easily shown that when $Z = g^{t\alpha^{(2^m - 1)}}_{m+l-1}$ (i.e. the input to $\mathcal{B}$ is a valid m-MDHE tuple), then this is a valid challenge to $\mathcal{A}$ as in a real attack.

Query Phase 2: Same as in query phase 1.

Guess: The adversary $\mathcal{A}$ outputs a guess b' of b. If $b' = b$, $\mathcal{B}$ outputs 0 (indicating that $Z = g^{t\alpha^{(2^m - 1)}}_{m+l-1}$). Otherwise, it outputs 1 (indicating that Z is a random element in $\mathbb{G}_{m+l-1}$).

It can be easily proved that the probability that $\mathcal{B}$ aborts the simulation as a result of one of the decryption queries by $\mathcal{A}$ is less than ϵ_2 (from the existential unforgability property of the signature scheme). We conclude that $\mathcal{B}$ has the same advantage ϵ as $\mathcal{A}$, which must therefore be negligible, as desired. This completes the proof of Theorem C.3. □

Similar CCA secure extensions can also be made to the generalized KAC construction for the multi-user scenario.

D The Third KAC Construction

D.1 Security In The Generic Multilinear Map Model

In this section, we prove that the basic version of our third KAC construction described in Section 5.1 is adaptively CPA secure in the generic multilinear map model . In particular, we demonstrate that with a prime group order parameter $q \approx 2^\lambda$, the scheme achieves λ-bit security. We state and prove the following theorem.

Theorem D.1. *Any generic adversary $\mathcal{A}$ that can make at most a polynomial number of queries to* (**Encode**,**Mult**,**Pair**) *has negligible advantage in breaking the adaptive security of the KAC construction presented in Section* 5.1*, provided that* $1/q$ *is negligible.*

Proof. Let $\mathcal{A}$ be an adaptive adversary under the generic model and let $\mathcal{B}$ be a challenger that plays the following game with $\mathcal{A}$:

SetUp: Challenger $\mathcal{B}$ sets up the system for $\mathcal{ID}$ consisting of all m bit class identities. $\mathcal{B}$ generates random $\alpha_{j,b} \in \mathbb{Z}_q$ for $j = 0, \cdots, m-1$, and $b = 0, 1$. $\mathcal{B}$ also generates random $\gamma, x \in \mathbb{Z}_q$. $\mathcal{A}$ receives the following:

- $X_{j,b} = \xi(\alpha_{j,b}, 1)$ for $j = 0, \cdots, m-1$, and $b = 0, 1$
- $W = \xi(x, 1)$
- $PK = (\xi(\gamma, m))$

Oracle Query Phase. $\mathcal{A}$ adaptively issues queries to (**Encode**,**Mult**,**Pair**).

Commit. Algorithm $\mathcal{A}$ commits to a set $\mathcal{S} \subset \mathcal{ID}$ of data classes that it wishes to attack. Since collusion attacks are allowed in our framework, $\mathcal{B}$ furnishes $\mathcal{A}$ with the aggregate key $K_{\overline{\mathcal{S}}}$ computed as

$$K_{\overline{\mathcal{S}}} = \xi\left(x \sum_{v \notin \mathcal{S}} \prod_{j=0}^{m-1} \alpha_{j,v_j}, m\right)$$

$\mathcal{B}$ also chooses $i \in \mathcal{S}$ and communicates the same to $\mathcal{A}$.

Challenge: To make a challenge query, $\mathcal{A}$ randomly generates $\hat{m}_0, \hat{m}_1 \in \mathbb{Z}_2 q$ and provides these to $\mathcal{B}$. $\mathcal{B}$ in turn randomly chooses $t \in \mathbb{Z}_q$ and a random $\hat{b} \in \{0, 1\}$, and sets

$$\begin{aligned} c_0 &= \xi(t, 1) \\ c_{j+1} &= \xi(t\alpha_{j,(1-i_j)}) \text{ for } j = 0, 1, \cdots, m-1 \\ c_{m+1} &= \xi(t(\gamma + \prod_{j=0}^{m-1} \alpha_{j,i_j}), m) \\ c_{m+2} &= \xi(\hat{m}_{\hat{b}} + \gamma x t, m+1) \end{aligned}$$

Finally, $\mathcal{B}$ sets

$$\mathcal{C} = (\{c_j\}_{j \in \{0, \cdots, m+2\}})$$

and provides the challenge to $\mathcal{A}$ as $(\mathcal{C}, \xi(\hat{m}_0, m+1), \xi(\hat{m}_1, m+1))$.

Guess: $\mathcal{A}$ outputs a guess $\hat{b}'$ of $\hat{b}$. If $\hat{b}' = \hat{b}$, $\mathcal{A}$ wins the game.

We now assume that instead of choosing random values for the set of parameters

$$(\{\alpha_{j,b}\}_{j \in \{0, \cdots, m-1\}, b \in \{0,1\}}, \gamma, x, t, \hat{m}_0, \hat{m}_1)$$

the algorithm $\mathcal{B}$ treats them as formal variables and maintains a list of tuples $L = \{(p, l, \epsilon)\}$, where p is a polynomial in these formal variables, l is the group index and $\epsilon \in \{0, 1\}^n$. The game now proceeds through the following stages:

- $\mathcal{B}$ initializes the list with the following tuples:

  – $(\alpha_{j,b}, 1, \xi_{2j+b})$ for randomly generated $\xi_{2j+b} \in \{0,1\}^n$, where $j \in \{0, \cdots, m-1\}$ and $b \in \{0,1\}$
  – $(\gamma, m, \xi 2m)$ for a randomly generated $\xi_{2m} \in \{0,1\}^n$
  – $(x, 1, \xi_{2m+1})$ for a randomly generated $\xi_{2m+1} \in \{0,1\}^n$

  Thus intially, $|L| = 2m+2$. $\mathcal{B}$ supplies the set of strings $\{\xi_j\}_{j \in \{0,\cdots,2m+2\}}$ to $\mathcal{A}$.

- $\mathcal{A}$ is allowed at most a polynomial number of (**Encode**,**Mult**,**Pair**) queries to which $\mathcal{B}$ responds as follows:

  – **Encode**(x, l): If $x \notin \mathbb{Z}_q$ or $i \notin \{1, \cdots, m+1\}$, $\mathcal{B}$ returns $\perp$. Otherwise, $\mathcal{B}$ looks for a tuple $(p_x, l, \xi) \in L$, where p_x is a constant polynomial equal to x. If such a tuple exists, $\mathcal{B}$ returns ξ. Otherwise, $\mathcal{B}$ generates a random $\xi \in \{0,1\}^n$. The tuple (p_x, l, ξ) is added to L and $\mathcal{B}$ responds with ξ.

  – **Mult**$(\xi_{k_0}, \xi_{k_1}, b)$: $\mathcal{B}$ searches in L for the pair of tuples $(p_{k_0}, l_{k_0}, \xi_{k_0})$ and $(p_{k_1}, l_{k_1}, \xi_{k_1})$. Unless both of them exist, $\mathcal{B}$ returns $\perp$. Also, if both tuples are available but $l_{k_0} \neq l_{k_1}$, $\mathcal{B}$ returns $\perp$. Otherwise, $\mathcal{B}$ searches in L for a tuple of the form (p_k, l_k, ξ_k), where $p_k = p_{k_0} + (-1)^b p_{k_1}$ and $l_k \equiv l_{k_0} \equiv l_{k_1}$. If such a tuple is found, $\mathcal{B}$ responds with ξ_k. If not, $\mathcal{B}$ generates a random $\xi_k \in \{0,1\}^n$, augments L by adding the tuple (p_k, l_k, ξ_k) and returns ξ_k.

  – **Pair**(ξ_{k_0}, ξ_{k_1}): $\mathcal{B}$ searches in L for the pair of tuples $(p_{k_0}, l_{k_0}, \xi_{k_0})$ and $(p_{k_1}, l_{k_1}, \xi_{k_1})$. Unless both of them exist, $\mathcal{B}$ returns $\perp$. Also, if both tuples are available but $l_{k_0} + l_{k_1} > m+1$, $\mathcal{B}$ returns $\perp$. Otherwise, $\mathcal{B}$ searches in L for a tuple of the form (p'_k, l'_k, ξ'_k), where $p'_k = p_{k_0}.p_{k_1}$ and $l'_k \equiv l_{k_0} + l_{k_1}$. If such a tuple is found, $\mathcal{B}$ responds with ξ'_k. If not, $\mathcal{B}$ generates a random $\xi'_k \in \{0,1\}^n$, augments L by adding the tuple (p'_k, l'_k, ξ'_k) and returns ξ'_k.

  We stress the fact that in each query to any of the three algorithms, *at most one* new tuple is added to L, and no tuple can have index $j > m+1$. Note that $\mathcal{B}$ can make the string responses ξ to $\mathcal{A}$ arbitrarily long, and thus, hard to guess. Hence, without loss of generality, we assume that all **Mult** and **Pair** queries made by $\mathcal{A}$ are precisely on strings furnished by $\mathcal{B}$.

- $\mathcal{A}$ is allowed to commit to a set $\mathcal{S}$ and query for the collusion aggregate key $K_{\overline{\mathcal{S}}}$. In response, $\mathcal{B}$ adds the tuple $\left(x \sum_{v \notin \mathcal{S}} \prod_{j=0}^{m-1} \alpha_{j,v_j}, m, \xi\right)$ for randomly generated $\xi \in \{0,1\}^n$, which is given as response to $\mathcal{A}$. $\mathcal{B}$ also chooses $i \in \mathcal{S}$ and communicates the same to $\mathcal{A}$.

- Finally, $\mathcal{A}$ is allowed to make a single encryption query on the class $i \in \mathcal{S}$. $\mathcal{B}$ creates a new formal variables t and $\hat{m}$ and adds the following tuples to its list L:

  – $(t, 1, \xi)$
  – $(t\alpha_{j,(1-i_j)}, 1, \xi_j)$ for $j = 0, 1, \cdots, m-1$
  – $(t(\gamma + \prod_{j=0}^{m-1} \alpha_{j,i_j}), m, \xi_m)$
  – $(\hat{m} + \gamma x t, m+1, \xi_{m+1})$
  – $(\hat{m}_0, m+1, \xi_{m+2})$
  – $(\hat{m}_1, m+1, \xi_{m+3})$

  Once again ξ and $\xi_j, j \in \{0, \cdots, m+3\}$ are randomly generated strings in $\{0,1\}^n$ that are provided to $\mathcal{A}$ as response.

- $\mathcal{A}$ outputs a random $\hat{b}' \in \{0,1\}$.

Table 1: Upper Bounds on Contributions to Length of L

Query Stage	**Maximum Contribution to $\|L\|$**
SetUp	$2m+2$
Oracle Query Phase	$Q_e + Q_m + Q_p$
Commit	1
Challenge	$m+5$
Total	$Q_e + Q_m + Q_p + 3m + 8$

- At this point, $\mathcal{B}$ chooses random values for $\alpha_{j,b}, \gamma, x, t$ and asks $\mathcal{A}$ for two random messages m_0 and m_1. $\mathcal{B}$ also randomly chooses , $\hat{b} \in \{0,1\}$ and sets $\hat{m} = \hat{m}_{\hat{b}}$.

We denote by Υ a *false polynomial equality* event where, for two random tuples (p, l, ξ) and (p', l', ξ') in the list L such that $l = l'$, we have $p(\alpha_{j,b}, \cdots) = p'(\alpha_{j,b}, \cdots)$ even though $p \not\equiv p'$. In case an instance of Υ occurs, $\mathcal{B}$ fails to simulate the oracle perfectly. We say that $\mathcal{A}$ wins the game if $\hat{b} = \hat{b'}$ or an instance of the event Υ occurs.

We now look at the probability that a random choice of values for the formal variables

$$(\alpha_{j,b}, \gamma, x, t, m_0, m_1) \in \mathbb{Z}_q^{(2m+5)}$$

results in the event Υ. First, we make the following observations.

Observation D.2. *The maximum degree of any polynomial in L is at most $m+1$.*

Observation D.3. *Substituting the formal variable $\hat{m}$ with $\hat{m}_{\hat{b}}$ results in no instances of the false polynomial equality event Υ.*

It follows from the Swartz-Zippel lemma [Mos10] that the probability that a randomly chosen pair of polynomials in L evaluate to the same value for a random choice of variable values, is upper bounded by $(m+1)/q$. Next, assume that $\mathcal{A}$ makes Q_E queries to **Encode**, Q_M queries to **Mult** and Q_P queries to **Pair** during **Oracle Query Phase**. Table 1 summarizes the maximum possible contributions to $|L|$ by the tuples added by $\mathcal{B}$ at different query stages. Note that $|L|$ is upper bounded by $(Q_e + Q_m + Q_p + 3m + 8)$. It easily follows that the probability of a false polynomial equality event Υ is upper bounded as

$$Pr(\Upsilon) \leq \left(Q_e + Q_m + Q_p + 3m + 8\right)^2 (m+1)/2q$$

If the event Υ does not occur, $\mathcal{B}$ simulates the oracle in response to $\mathcal{A}$'s queries perfectly and, from $\mathcal{A}$'s view, $\hat{b}$ is independent as it was chosen after the simulation. Hence we have

$$Pr[\hat{b} = \hat{b'} \mid \overline{\Upsilon}] = 1/2$$

This in turn gives us the following relations:

$$Pr[\hat{b} = \hat{b'}] \geq Pr[\hat{b} = \hat{b'} \mid \overline{\Upsilon}] Pr[\overline{\Upsilon}] = \frac{1 - Pr[\Upsilon]}{2}$$

$$Pr[\hat{b} = \hat{b'}] \leq Pr[\hat{b} = \hat{b'} \mid \overline{\Upsilon}] Pr[\Upsilon] + Pr[\Upsilon] = \frac{1 + Pr[\Upsilon]}{2}$$

From this, it is straightforward to conclude that the advantage of the generic adversary $\mathcal{A}$ may be upper bounded as follows:

$$\begin{aligned} |Adv_{\mathcal{A},2^m} - \frac{1}{2}| &= |Pr[\hat{b} = \hat{b'}]| \\ &\leq Pr[\Upsilon]/2 \\ &= \left(Q_e + Q_m + Q_p + 3m + 8\right)^2 (m+1)/4q \end{aligned}$$

For Q_e, Q_m, Q_p, m polynomial in the security parameter λ, this quantity is negligible provided that $1/q$ is negligible, or in particular, $q \approx 2^\lambda$, as desired. This completes the proof of Theorem D.1. □

E Applications of KAC

In this section, we point out some specific applications in which KAC proves to be a very efficient solution. Some of these are mentioned in [CCT+14]. The applications are mentioned for readers not entirely familiar with the utility of KAC.

Online Collaborative Data Sharing. The foremost application of KAC is in secure data sharing for collaborative applications. Applications such as Google Drive [Pau] and Dropbox [Clo] allow users to share their data on the cloud and delegate access rights to multiple users to specific subsets of their whole data. Even government and corporate organizations require secure data sharing mechanisms for their daily operations. KAC can be easily set up to function on top of standard data sharing applications to provide security and flexibility. Data classes may be viewed as folders containing similar files. The fact that our proposed KAC is identity based means that each folder can have its own unique ID chosen by the data owner. Also, the fact that the ciphertext overhead is only logarithmic in the number of data classes implies that space requirement for any data owner is optimal. Finally, the aggregate key also has low overhead and can be transmitted via a secure channel such as a password protected mail service. Since KAC is easily extensible to multiple data owners, the system is practically deployable for a practical data sharing environment. The other advantage of KAC is that once a system is setup with a set of multilinear maps and public parameters, the same setup with the same set of public parameters can be reused by multiple teams within the same organization. Since data owned by each individual owner is insulated from access by users who do not have the corresponding aggregate key, and each data owner has her own tuple of public, private and authentication keys, a single KAC can support multiple data sharing units, while guaranteeing the same underlying security. This saves the cost of setting up new multilinear maps and public parameters each time.

Distribution of Product License and/or Activation Keys. Suppose a company owns a number of products, and intends to distribute the license files (or activation keys) corresponding to these to different users. The KAC framework allows them to put these keys on the cloud in an encrypted fashion, and distribute an aggregate key corresponding to the license files for multiple products to legally authenticated customers as per their requirements. The legal authentication comes from the fact the user who buys multiple products from the company is given the authentication key and the aggregate key that allows her to decrypt the license file for each product. Since both these keys are of constant size, distributing these to users is easier than providing a separate license file to each user.

Patient controlled encryption (PCE). Patient controlled encryption (PCE) is a recent concept that has been studied in the literature [BCHL09]. PCE allows a patient to upload her own medical data on the cloud and delegate decryption rights to healthcare personnel as per her requirement. KAC acts as an efficient solution to this problem by allowing patients to define their own hierarchy of medical data and delegate decryption rights to this data to different specialists/medical institutions using aggregate keys in an efficient fashion. Given the multitude of sensitive digital health records existent in today's world, storing this data in local/personal machines is not a viable solution and the cloud seems the best alternative. KAC thus provides a two-way advantage in this regard. Not only does it allow people from across the globe to store their health data efficiently and safely, but also allows them to envisage the support of expert medical care from across the globe.