Scalable Key-Aggregate Cryptosystem for Secure Online Data Sharing on the Cloud

Sikhar Patranabis, Yash Shrivastava and Debdeep Mukhopadhyay
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
{sikhar.patranabis, yash.shrivastava, debdeep}@cse.iitkgp.ernet.in

Abstract—Online data sharing for increased productivity and efficiency is one of the primary requirements today for any organization. The advent of cloud computing has pushed the limits of sharing across geographical boundaries, and has enabled a multitude of users to contribute and collaborate on shared data. However, protecting online data is critical to the success of the cloud, which leads to the requirement of efficient and secure cryptographic schemes for the same. Data owners would ideally want to store their data/files online in an encrypted manner, and delegate decryption rights for some of these to users, while retaining the power to revoke access at any point of time. An efficient solution in this regard would be one that allows users to decrypt multiple classes of data using a single key of constant size that can be efficiently shared via a secure channel. In this paper, we address this problem by proposing a dynamic, scalable and efficient key aggregate encryption scheme with provable security and user revocation properties. We lay special focus on how the scheme can be actually deployed on the cloud for multiple data owners, with hierarchical data organization. We present simulation results to prove the efficiency of the scheme and compare its performance with other existing cryptosystems for online data sharing in literature.

Index Terms—Cloud Computing, Data Sharing, Data Security, Key-Aggregate Cryptosystem, Provable Security, Scalability, Revocation

1 Introduction

The recent advent of cloud computing has pushed the limits of data sharing capabilities for numerous applications that transcend geographical boundaries and involve millions of users. Governments and corporations today treat data sharing as a vital tool for enhanced productivity. Cloud computing has revolutionized education, healthcare and social networking. Perhaps the most exciting use case for cloud computing is its ability to allow multiple users across the globe share and exchange data, while saving the pangs of manual data exchanges, and avoiding the creation of redundant or out-of-date documents. Social networking sites have used the cloud to create a more connected world where people can share a variety of data including text and multimedia. Collaborative tools commonly supported by cloud platforms and are extremely popular since they lead to improved productivity and synchronization of effort. The impact of cloud computing has also pervaded the sphere of healthcare, with smpartphone applications that allow remote monitoring and even diagnosis of patients. In short, cloud computing is changing various aspects of our lives in unprecedented ways.

Despite all its advantages, the cloud is susceptible to privacy and security attacks, that are a major hindrance to its wholesome acceptance as the primary means of data sharing in todays world. According to a survey carried out by IDC Enterprise Panel in August 2008 [1], Cloud users regarded security as the top challenge with 75% of surveyed users worried about their critical business and IT systems being vulnerable to attack. While security threats from external agents are widespread, malicious service providers must also be taken into consideration. Since online data almost always resides in shared environments (for instance, multiple virtual machines running on the same physical device), ensuring security and privacy on the cloud is a non trivial task. When talking about security and privacy of data in the cloud, it is important to lay down the requirements that a data sharing service must provide in order to be considered secure. We list down here some of the most primary requirements that a user would want in a cloud-based data sharing service:

- *Data Confidentiality*: Unauthorized users (including the cloud service provider), should not be able to access the data at any given time. Data should remain confidential in transit, at rest and on backup media.
- *User revocation*: The data owner must be able to revoke any user's access rights to data the without affecting other authorized users in the group.
- *Scalability and Efficiency*: Perhaps the biggest challenge faced by data management on the cloud is maintaining scalability and efficiency in the face of immensely large user bases and dynamically changing data usage patterns.
- *Collusion between entities*: Any data sharing service in the cloud must ensure that even when certain malicious entities collude, they should still not be able to access any of the data in an unauthorized fashion.

A traditional way of ensuring data privacy is to depend on the server to enforce access control mechanisms [2]. This methodology is prone to privilege escalation attacks in shared data environments such as the cloud, where data

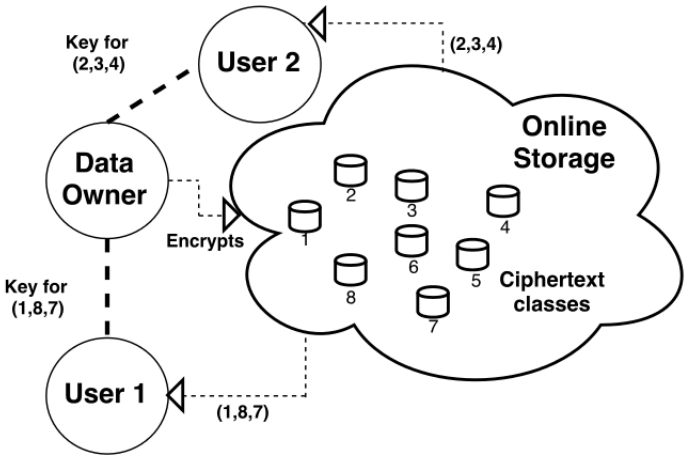

Fig. 1: Example of Online Data Sharing

corresponding to multiple users could reside on the same server. Current technology for secure online data sharing comes in two major flavors - trusting a third party auditor [3], or using the user's own key to encrypt her data while preserving anonymity [4]. In either case, a user would want a reliable and efficient cryptographic scheme in place, with formal guarantees of security, high scalability and ease of use. The main challenge in designing such a cryptosystem lies in effective sharing of encrypted data. A data sharing scheme on the cloud is only successful if data owners can delegate the access rights to their data efficiently to multiple users, who can then access the data directly from the cloud servers. Figure 1 describes a realistic online data sharing set-up on the cloud. Assume that the data owner is using an online data sharing service such as Microsoft OneDrive [5] to store her research documents. She wishes to add an additional layer of security for her data by storing them in an encrypted fashion. Now, she intends to share specific subsets of these documents with different collaborators. For this, she needs to provide each of her collaborators with decryption rights to specific classes of the data that they are authorized to access. She might also wish to dynamically update the delegated access rights based on changes to the data/credibility issues. The challenge therefore is to provide her with a secure and efficient online *partial* data sharing scheme that allows updates to user access rights on the fly.

A näive (and extremely inefficient) solution is to have a different decryption key for each ciphertext class, and share them accordingly with users via secured channels. This scheme is not practically deployable for two major reasons. Firstly, the number of secret keys would grow with the number of data classes. Secondly, any user revocation event would require Alice to entirely re-encrypt the corresponding subset of data, and distribute the new set of keys to the other existing valid users. This makes the scheme inefficient and difficult to scale. Since the decryption key in public key cryptosystems is usually sent via a secure channel, smaller key sizes are desirable. Moreover, resource constrained devices such as wireless sensor nodes and smart phones cannot afford large expensive storage for the decryption keys either. Not many of the present cryptosystems seem to address this issue of reducing the decryption key size for online data sharing schemes, as we describe next.

1.1 Related Work

In this section we present a brief overview of public and private key cryptographic schemes in literature for secure online data sharing. While many of them focus on key aggregation in some form or the other, very few have the ability to provide constant size keys to decrypt an arbitrary number of encrypted entities. One of the most popular techniques for access control in online data storage is to use a pre-defined hierarchy of secret keys 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 [6], [7], [8], [9], [10], [7]. A major disadvantage of hierarchical encryption schemes is that granting access to only a selected set of branches within a given subtree warrants an increase in the number of granted secret keys. This in turn blows up the size of the key shared. Compact key encryption for the symmetric key setting has been used in [11] 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. Proxy re-encryption is another technique to achieve fine-grained access control and scalable user revocation in unreliable clouds [12], [13]. However, proxy re-encryption essentially transfers the responsibility for secure key storage from the delegatee to the proxy and is susceptible to collusion attacks. It is also important to ensure that the transformation key of the proxy is well protected, and every decryption would require a separate interaction with the proxy, which is inconvenient for applications on the cloud.

1.2 The Key-Aggregate Encryption Scheme

The most efficient proposition pertaining to our problem statement, to the best of our knowledge, is made in [14]. The proposition is to allow Alice to combine the decryption power of multiple data classes into a single key of constant size. Thus, while each class of data is encrypted using a different public key, a single decryption key of constant

size is sufficient to decrypt any subset of these classes. This system is popularly known as the key-aggregate cryptosystem (KAC), and derives its roots from the seminal work by Boneh *et.al.* [15] that allows broadcasting of data (encrypted by the same public key) among multiple users, each of whom possess their own private keys for decryption. In KAC, when a user demands for a particular subset of the available classes of data, the data owner computes an aggregate key which integrates the power of the individual decryption keys corresponding to each class of data. However, KAC as proposed in [14] suffers from three major drawbacks, each of which we address in this paper.

1) Firstly, no concrete proofs of cryptographic security for KAC are provided by the authors of [14]. Formal proofs are considered to be of paramount importance in establishing the security of any cryptographic scheme, since they define the specific adversarial models against which the scheme is secure.
2) Secondly, with respect to user access rights, KAC is a static scheme in the sense that once a user is in possession of the aggregate key corresponding to a subset of the data classes, the owner cannot dynamically revoke the user's access rights without entirely changing the public-private key pairs corresponding to each data class in that subset. This is extremely expensive and impractical. Since dynamic changes in access rights is extremely common in online data storage, this scenario needs to be tackled.
3) The KAC proposed in [14] uses Type-1 symmetric bilinear pairings. However, such pairings are usually not efficiently implementable, especially in resource constrained environments such as the cloud.
4) Finally, the two-hierarchical public key extension of KAC proposed in [14] does not allow users to extend the concept of key aggregation within her own class of data. A user must define a different data class altogether whenever she wishes to have a different access policy for some of her new data. Since hierarchical organization of data is very common in nay online data sharing application, allowing a user to use hierarchical organizations beyond level two within her own data classes makes the system more practical and efficient.

1.3 Our Contributions

The main contributions of this paper can be enumerated as follows:

1) In this paper we propose a dynamic key-aggregate cryptosystem (KAC) with user revocation for online data sharing on the cloud using efficiently implementable asymmetric bilinear pairings.
2) We formally prove this basic KAC to be secure against chosen plaintext attack (CPA) and chosen ciphertext attack (CCA) models. To the best of our knowledge, no formal proofs of security for KAC existed in literature before.
3) We demonstrate how the basic KAC can be extended to handle multiple data classes with a hierarchical organization for deployment in practical scenarios.
4) We present simulation results to demonstrate that the proposed KAC actually scales better than hierarchical cryptosystems in terms of space and time complexity requirements.

2 Preliminaries

We begin by formally defining the dynamic key-aggregate cryptosystem (KAC). We then outline the CPA and CCA securities for this scheme. Finally, we state and prove the complexity assumptions for the proposed KAC schemes.

2.1 Key-Aggregate Cryptosystem (KAC) : The Framework

Our proposed dynamic KAC is the same as that in [14] with an additional secret key for user authentication. KAC is an ensemble of five randomized polynomial-time algorithms. The system administrator is responsible for setting up the public parameters via the **SetUp** operation. These parameters are fixed in the sense that they remain fixed over time for a given set-up. A data owner willing to share her data using this system registers to receive her own public key, private key and authentication key, generated using the **KeyGen** operation. The data owner is responsible for classifying each of her data files/messages into a specific class i. Each message is accordingly encrypted by an **Encrypt** operation and stored online in the cloud. When delegating the decryption rights to a specific subset of message classes, the data owner uses the **Extract** operation to generate a constant-size *aggregate decryption key* unique to that subset. This aggregate key, along with the secret authentication key, must be sent to any user with decryption rights to the data via a secure channel. Finally, the user can use the aggregate key and the authentication key to decrypt any file belonging to one the corresponding subset of classes. Please not that the master secret key and the decryption key are not the same, as the master secret key is never known to anyone except the data owner.

We now describe in details each of the five algorithms involved in KAC in terms of their inputs and outputs:

1) **SetUp**$(1^\lambda, n)$: Takes as input the number of data classes n and the security level parameter λ. Outputs the public parameter $param$.
2) **KeyGen**(): Outputs the public key PK, the master-secret key msk and the authentication key U.
3) **Encrypt**(PK, i, m): Takes as input the public key parameter PK, the data class i and the plaintext data m. Outputs the corresponding ciphertext $\mathcal{C}$.
4) **Extract**$(msk, \mathcal{S})$: Takes as input the master secret key and a subset of data classes $\mathcal{S} \subset \{1, 2, \cdots, n\}$. Computes the aggregate key $K_\mathcal{S}$ for all encrypted messages belonging to these subset of classes.
5) **Decrypt**$(\mathcal{C}, i, \mathcal{S}, K_\mathcal{S}, U)$: Takes as input the ciphertext $\mathcal{C}$, the data class i the aggregate key $K_\mathcal{S}$ corresponding to a subset $\mathcal{S}$ and the authentication key U. Outputs the decrypted data m.

The main difference between this scheme and the one proposed in [14] is in the additional user authentication key U. In case the data owner wishes to revoke the decryption rights of some user, she changes the authentication key

and communicates the new authentication key to all users who still have the decryption rights to one or more subsets of data classes. The main advantage of this approach is that this redistribution is independent of subset information since the authentication key is the same for all data classes belonging to a particular owner. The data owner does not need to recompute and distribute the aggregate key for every possible subset of data classes, which is a potentially exponential time effort in the number of data classes. Thus our scheme allows more flexible user revocation than the KAC proposed in [14].

2.2 Security of KAC : A Formal Outline

In this paper, we propose a formal framework for proving the security of KAC against both chosen plaintext and chosen ciphertext attacks. Our framework takes into account collusion attacks where multiple rogue users may collude together by compromising the knowledge of the aggregate key for different subsets, as well as the secret authentication key. We begin by introducing a game between an attack algorithm $\mathcal{A}$ and a challenger $\mathcal{B}$, both of whom are given n, the total number of ciphertext classes, as input. The game proceeds through the following stages:

1) **Init:** Algorithm $\mathcal{A}$ begins by outputting a set $\mathcal{S}^* \subset \{1, 2, \cdots, n\}$ of receivers that it wishes to attack. Challenger $\mathcal{B}$ randomly chooses a ciphertext class $i \in \mathcal{S}^*$.
2) **SetUp**: Challenger $\mathcal{B}$ sets up the KAC system generates the public parameter $param$, the public key PK, the master secret key msk and the authentication key U. Since collusion attacks are allowed in our framework, $\mathcal{B}$ furnishes $\mathcal{A}$ with both U the aggregate key $K_{\overline{\mathcal{S}}^*}$ that allows $\mathcal{A}$ to decrypt any ciphertext class $j \notin \mathcal{S}^*$.
3) **Query Phase 1**: Algorithm $\mathcal{A}$ adaptively issues decryption queries $q_1, \cdots, q_v$ where a decryption query comprises of the triple $(j, \mathcal{S}, \mathcal{C})$. Here $S \subseteq \mathcal{S}^*$ and $j \in \mathcal{S}$. The challenger responds with **Decrypt**$(\mathcal{C}, j, \mathcal{S}, K_{\mathcal{S}}, U)$.
4) **Challenge**: To generate the challenge, $\mathcal{B}$ picks at random two messages m_0 and m_1 from the set of possible plaintext messages belonging to class i. She randomly picks $b \in \{0, 1\}$, and sets the challenge to $\mathcal{A}$ as $(\mathcal{C}^*, m_0, m_1)$, where $\mathcal{C}^*$ = **Encrypt**(PK, i, m_b).
5) **Query Phase 2**: Algorithm $\mathcal{A}$ continues to adaptively issue decryption queries $q_{v+1}, \cdots, q_{Q_D}$ where a decryption query comprises of the triple $(j, \mathcal{S}, \mathcal{C})$. Once again, $S \subseteq \mathcal{S}^*$ and $j \in \mathcal{S}$, but now $\mathcal{C} \neq \mathcal{C}^*$. The challenger responds as in phase 1.
6) **Guess**: The adversary $\mathcal{A}$ 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. Note that the adversary $\mathcal{A}$ is non-adaptive; it chooses $\mathcal{S}$, and obtains the aggregate decryption key for all ciphertext classes outside of $\mathcal{S}$, before it even sees the public parameters $param$ or the public key PK. Let $Adv_{\mathcal{A},n}$ denote the probability that $\mathcal{A}$ wins the game when the challenger is given n as input. We next define the security of KAC against chosen ciphertext attacks (CCA) and chosen plaintext attacks (CPA) as follows:

- **CCA Security:** We say that a key-aggregate encryption system is (τ, ϵ, n, q_D) CCA secure if for all non-adaptive τ-time algorithms $\mathcal{A}$ that can make a total of q_D decryption queries, we have that $|Adv_{\mathcal{A},n} - \frac{1}{2}| < \epsilon$.
- **CPA Security:** We say that a key-aggregate encryption system is (τ, ϵ, n) semantically secure if it is $(\tau, \epsilon, n, 0)$ CCA secure.

2.3 Bilinear Pairings

In this paper, we make several references to bilinear non-degenerate mappings on elliptic curve sub-groups, popularly known in literature as *pairings*. Hence we begin by providing a brief background on bilinear pairing based schemes on elliptic curve subgroups. A pairing is a bilinear map defined over elliptic curve subgroups. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two (additive) cyclic elliptic curve subgroups of the same prime order q. Let $\mathbb{G}_T$ be a multiplicative group, also of order q with identity element 1. Also, let P and Q be points on the elliptic curve that generate the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. A mapping $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \longrightarrow \mathbb{G}_T$ is said to be a bilinear ma[if it satisfies the following properties:

- Bilinear: For all $P' \in \mathbb{G}_1, Q' \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}_{\shortparallel}$, we have $\hat{e}(aP_1, bQ_1) = \hat{e}(P_1, Q_1)^{ab}$.
- Non-degeneracy: If for all $P_i \in \mathbb{G}_1, \hat{e'}(P_1, Q_1) = 1$ then $Q_1 = \prime$. Alternatively, if P and Q be the generators for $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively where neither group only contains the point at infinity, then $\hat{e'}(P, Q) \neq 1$
- Computability: There exists an efficient algorithm to compute $\hat{e'}(R, S) \forall R \in \mathbb{G}_1, S \in \mathbb{G}_2$

From a cryptographic standpoint, pairings can be broadly classified into three categories as follows (for more details refer [16]):

- *Type-1 symmetric pairings*: In such pairings $\mathbb{G}_1 = \mathbb{G}_2$. These pairings are defined exlusively over supersingular curves and require subgroups of extremely large order.
- *Type-2 asymmetric pairings*: In such pairings $\mathbb{G}_1 \neq \mathbb{G}_2$, but there exists an efficiently computable homomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$.
- *Type-3 asymmetric pairings*: In such pairings $\mathbb{G}_1 \neq \mathbb{G}_2$, and there exists no efficiently computable homomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$.

The case where $\mathbb{G}_1 \neq \mathbb{G}_2$ but there exists efficiently computable homomorphisms in either direction is equivalent to type one pairings. There also exist Type-4 pairings over non-cyclic subgroups [16], but they are not pertinent to our discussions on KAC. The original proposal for KAC in [14] is based on Type-1 symmetric pairings. However, from a practical deployment perspective on the cloud, Type-1 pairings are usually not efficient enough to be deployed due to their specific curve requirements and large group size demands. So, in this paper, we focus on the use of efficiently implementable Type-2 and Type-3 pairings such as the Tate pairing [17], the Eta pairing [18], the Ate pairing

[19] and even advanced pairings that are extremely efficient such as the R-Ate pairing [19]. We next state the complexity assumptions that allow us to formally prove the security of our proposed KAC schemes based on such pairings.

2.4 Security of KAC : The Complexity Assumptions

We now introduce the complexity assumptions used in this paper. These complexity assumptions form the basis for the security of our proposed KAC schemes. We begin by introducing some notations that are used throughout this paper. We assume the existence of equi-prime order (q) elliptic curve subgroups $\mathbb{G}_1$ and $\mathbb{G}_2$, along with their generators P and Q and an efficiently computable isomorphism $\psi : \mathbb{G}_2 \longrightarrow \mathbb{G}_1$ such that $\psi(aQ) = aP$ for any $a \in Z_q$. We also assume the existence of a multiplicative cyclic group $\mathbb{G}_T$, also of order q with identity element 1. Let α be a randomly chosen element in $\mathbb{Z}_q$. For any point R in either $\mathbb{G}_1$ or $\mathbb{G}_2$, let $R_x = \alpha^x R$, where x is an integer. We denote by $Y_{R,\alpha,l}$ the set of $2l-1$ points $(R_1, R_2, \cdots, R_l, R_{l+2}, \cdots, R_{2l})$. Note that the term R_{l+1} is missing. Next, we assume the existence of two efficiently computable bilinear pairings - a symmetric Type-1 pairing $e : \mathbb{G}_2 \times \mathbb{G}_2 \longrightarrow \mathbb{G}_T$ and an asymmetric Type-2 pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \longrightarrow \mathbb{G}_T$. Finally, we denote by ϕ a mapping from $\mathbb{G}_T$to $\mathbb{G}_T$ such that $\phi(e(H_1, H_2)) = \hat{e}(\psi(H_1), (H_2))$ for all $H_1, H_2 \in \mathbb{G}_2$. We now state two hard problems which form the complexity assumptions for the security of KAC, The first of these is described in [15] and the second is proposed by us in this paper. For simplicity, we focus on the decision version of both the hard problems.

- *The decision l-BDHE problem*: Given an input $(I = (H, Q, Y_{Q,\alpha,l}), Z)$, where $H \in \mathbb{G}_2$ and $Z \in \mathbb{G}_T$, and the bilinear pairing e, decide if $Z = e(Q_{l+1}, H)$.
- *The decision (l,l)-BDHE problem*: Given an input $(I' = (H', P, Q, Y_{P,\alpha,l}, Y_{Q,\alpha,l}), Z')$, where $H' \in \mathbb{G}_2$ and $Z' \in \mathbb{G}_T$, and the bilinear pairing $\hat{e}$, decide if $Z' = \hat{e}(P_{l+1}, H')$.

Let $\mathcal{A}$ be a τ-time algorithm that takes an input challenge for l-BDHE and outputs a decision bit $b \in \{0, 1\}$. We say that $\mathcal{A}$ has advantage ϵ in solving the decision l-BDHE problem if

$$|Pr[\mathcal{A}(I, e(Q_{l+1}, H)) = 0] - Pr[\mathcal{A}(I, T) = 0]| \geq \epsilon \quad (1)$$

where the probability is over random choice of $H \in \mathbb{G}_2$, $T \in \mathbb{G}_T$ and $\alpha \in \mathbb{Z}_q$, and random bits used by $\mathcal{A}$. We refer to the distribution on the left as $\mathcal{L}_{\text{BDHE}}$ and the distribution on the right as $\mathcal{R}_{\text{BDHE}}$. Again, let $\mathcal{B}$ be a τ-time algorithm that takes an input challenge for (l,l)-BDHE and outputs a decision bit $b \in \{0, 1\}$. We say that $\mathcal{B}$ has advantage ϵ in solving the decision (l,l)-BDHE problem if

$$|Pr[\mathcal{B}(I', \hat{e}(P_{l+1}, H')) = 0] - Pr[\mathcal{B}(I', T) = 0]| \geq \epsilon \quad (2)$$

where the probability is over random choice of $H' \in \mathbb{G}_2$, $T \in \mathbb{G}_T$ and $\alpha \in \mathbb{Z}_q$, and random bits used by $\mathcal{B}$. We now refer to the distribution on the left as $\mathcal{L}'_{\text{BDHE}}$ and the distribution on the right as $\mathcal{R}'_{\text{BDHE}}$. We next state the following definitions.

Definition 1. The decision (τ, ϵ, l)-BDHE assumption holds in $\mathbb{G}_2$ if no τ-time algorithm has advantage at least ϵ in solving the decision l-BDHE problem in $\mathbb{G}_2$.

Definition 2. The decision (τ, ϵ, l, l)-BDHE assumption holds in $(\mathbb{G}_1, \mathbb{G}_2)$ if no τ-time algorithm has advantage at least ϵ in solving the decision (l,l)-BDHE problem in $(\mathbb{G}_1, \mathbb{G}_2)$.

We relate the two complexity assumptions by claiming that decision (τ, ϵ, l, l)-BDHE assumption holds in $(\mathbb{G}_1, \mathbb{G}_2)$ if the decision (τ, ϵ, l)-BDHE assumption for elliptic curves holds in $\mathbb{G}_2$. It can be easily proved as follows. Let $\mathcal{B}$ be a τ-time adversary that has advantage greater than ϵ in solving the decision (l,l)-BDHE problem in $(\mathbb{G}_1, \mathbb{G}_2)$. We build a τ-time algorithm $\mathcal{A}$ that has advantage at least ϵ in solving the l-BDHE problem in $\mathbb{G}_1$. Algorithm A takes as input a random l-BDHE challenge $(I = (H, Q, Y_{Q,\alpha,l}), Z)$, where $H \in \mathbb{G}_2$ and $Z \in \mathbb{G}_T$. It computes $(P, Y_{P,\alpha,l})$ from $Q, (Q, \alpha, l)$ using the efficiently computable isomorphism ψ. It also computes $Z' = \phi(Z)$. It then passes on the challenge $(I' = (H, P, Q, Y_{P,\alpha,l}, Y_{Q,\alpha,l}), Z')$ to $\mathcal{B}$, who outputs $b \in \{0, 1\}$. Algorithm $\mathcal{A}$ returns b. It is easy to see that (I', Z') is a valid (l,l)-BDHE challenge whenever (I, Z) is a valid l-BDHE challenge. This completes the proof. Note that we have assumed here that ϕ is an efficiently computable mapping. In reality, it may be as difficult to compute ϕ as the discrete log problem in $\mathbb{G}_2$ [20]. But since we have used a stronger assumption, the (l,l)-BDHE assumption is at least as strong as the l-BDHE assumption.

We finish this section by referring to a standard result widely presented in the cryptographic literature on pairings [20]. Given any protocol Protocol-2 described using a Type-2 pairing, and with a security proof for the same with respect to some problem $\mathcal{P}$-2, there exists a natural transformation of Protocol-2 to a Protocol-3 that uses a Type-3 pairing, a natural transformation of $\mathcal{P}$-2 to a problem $\mathcal{P}$-3, and a natural transformation of the security proof to one for Protocol-3 with respect to $\mathcal{P}$-3. Moreover, Protocol-3 is at least as efficient as Protocol-2, and $\mathcal{P}$-3 is equally as hard as $\mathcal{P}$-2 (for appropriately chosen parameters). In other words, as long as the efficiently computable homomorphism ψ does not have an explicit use in the protocol, both Protocol-2 and Protocol-3 can be used without any compromise on the security. In our description of KAC, we use precisely this result to argue that it is efficiently implementable. While the security proofs presented use the (l,l)-BDHE assumption that seems to be inherently specific to Type-2 pairings, in reality the proposed KAC scheme can use efficiently implementable Type-3 pairings as well. In the forthcoming discussions, we refer simply to asymmetric pairings without specifying whether they are Type-2 or Type-3.

3 A Basic KAC for Single Data Owner

In this section, we present the design of our proposed dynamic key-aggregate cryptosystem. Unlike the KAC proposed in [14], our proposed scheme is based on asymmetric bilinear pairings that are practical and efficiently implementable and do not require non-singular elliptic curves. For the basic case, we assume a single data owner who stores her data in n different classes, with no hierarchical

organization within each class. In later discussions, we show how the system can be scaled to multiple users, and also to allow hierarchical constructions within each class. Our scheme ensures that the ciphertext and aggregate key are of constant size, while the public parameter size is linear in the number of data classes n. We prove the scheme to secure against CPA, and also show how to extend it to achieve CCA security as well.

3.1 Construction

We present the basic construction of our proposed KAC. As mentioned in Section 2, we assume the existence of equi-prime order (q such that $2^\lambda \leq q \leq 2^{\lambda+1}$) elliptic curve subgroups $\mathbb{G}_1$ and $\mathbb{G}_2$, along with their generators P and Q. We also assume the existence of a multiplicative cyclic group $\mathbb{G}_T$, also of order q with identity element 1. Finally, we assume there exists an asymmetric bilinear pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \longrightarrow \mathbb{G}_T$. The notations used in the forthcoming discussion are already introduced in Section 2.

SetUp$(1^\lambda, n)$: Randomly pick $\alpha \in \mathbb{Z}_q$. Output the system parameter as $param = (P, Q, Y_{P,\alpha,n}, Y_{Q,\alpha,n})$. Discard α.

KeyGen(): Randomly pick $\gamma, t \in \mathbb{Z}_q$. Set the master secret key msk to γ. Let $PK_1 = \gamma P$ and $PK_2 = \gamma Q$. Set the public key $PK = (PK_1, PK_2)$. Finally set the user authentication key $U = tQ$.

Encrypt(PK, i, m): For a message $m \in \mathbb{G}_T$ belonging to class $i \in \{1, 2, \cdots, n\}$, randomly choose $r \in \mathbb{Z}_q$ and let $t' = t + r \in \mathbb{Z}_q$. Output the ciphertext $\mathcal{C}$ as

$$\mathcal{C} = (c_1, c_2, c_3) = (rQ, t'(PK_2 + Q_i), m\hat{e}(P_n, t'Q_1))$$

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

$$K_\mathcal{S} = msk \sum_{j \in \mathcal{S}} P_{n+1-j}$$

Note that this is indirectly equivalent to setting $K_\mathcal{S}$ to $\sum_{j\in\mathcal{S}} \alpha^{n+1-j} PK_1$.

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

$$\begin{aligned} a_\mathcal{S} &= \sum_{j\in\mathcal{S}, j\neq i} P_{n+1-j+i} \\ b_\mathcal{S} &= \sum_{j\in\mathcal{S}} P_{n+1-j} \\ \hat{m} &= c_3\hat{e}(K_\mathcal{S} + a_\mathcal{S}, U + c_1)/(\hat{e}(b_\mathcal{S}, c_2)) \end{aligned}$$

Return the decrypted message $\hat{m}$.

The proof of correctness of the basic KAC scheme is presented next.

$$\begin{aligned} \hat{m} &= c_3 \frac{\hat{e}(K_\mathcal{S} + \sum_{j\in\mathcal{S}, j\neq i} P_{n+1-j+i}, U + c_1)}{\hat{e}(\sum_{j\in\mathcal{S}} P_{n+1-j}, c_2)} \\ &= c_3 \frac{\hat{e}(\sum_{j\in\mathcal{S}} \gamma P_{n+1-j} + \sum_{j\in\mathcal{S}, j\neq i} P_{n+1-j+i}, t'Q)}{\hat{e}(\sum_{j\in\mathcal{S}} P_{n+1-j}, t'(PK_2 + Q_i))} \\ &= c_3 \frac{\hat{e}(\sum_{j\in\mathcal{S}} \gamma P_{n+1-j}, t'Q)\hat{e}(\sum_{j\in\mathcal{S}}(P_{n+1-j+i}) - P_{n+1}, t'P)}{\hat{e}(\sum_{j\in\mathcal{S}} P_{n+1-j}, t'PK_2)\hat{e}(\sum_{j\in\mathcal{S}} P_{n+1-j}, t'Q_i))} \\ &= c_3 \frac{\hat{e}(\sum_{j\in\mathcal{S}} P_{n+1-j+i}, t'Q)}{\hat{e}(P_{n+1}, t'Q)\hat{e}(\sum_{j\in\mathcal{S}} P_{n+1-j}, t'Q_i))} \\ &= c_3 \frac{\hat{e}(\sum_{j\in\mathcal{S}} P_{n+1-j+i}, t'Q)}{\hat{e}(P_{n+1}, t'Q)\hat{e}(\sum_{j\in\mathcal{S}} P_{n+1-j+i}, t'Q))} \\ &= m \frac{\hat{e}(P_n, t'Q_1)}{\hat{e}(P_{n+1}, t'Q)} \\ &= m \end{aligned}$$

3.2 Performance, Efficiency and User Access Control

The decryption time for any subset of ciphertext classes $\mathcal{S}$ is essentially dominated by the computation of $a_\mathcal{S}$ aand $b_\mathcal{S}$. However, if a user has already computed $a_{\mathcal{S}'}$ for a subset S' that has significant overlap with S, then she can easily compute the desired value by at most $|\mathcal{S} - \mathcal{S}'|$ operations. For similar subsets S and S', this value is expected to be fairly small. For subsets of very large size($n-r, r \ll n$), an advantageous approach could be to pre-compute $\sum_{j=1}^{j=n} P_{n+1-j+i}$ corresponding to $i = 1$ to n, which would allow the user to decrypt using only r group operations, and would require only r elements of $param$. Similar optimizations would also hold for the encryption operation where pre-computation of $\sum_{j=1}^{j=n} P_{n+1-j}$ is useful for large subsets.

Our proposed scheme also allows an user to control user access to her data by updating the authentication key U. Any user with the wrong authentication key fails to decrypt any encrypted message even if she has the corresponding aggregate key for a subset containing the data. The advantage of this scheme is that unlike in the proposition in [14], user access revocation does not require the data owner to entirely recompute the aggregate key corresponding to all possible subsets of data, which is a very expensive process. In our proposition, the data owner simply distributes the updated authentication key to all users whose access rights are not be revoked. However, in both the earlier and current propositions, the messages must be re-encrypted according to the updated authentication information on the server.

3.3 CPA Security of the Basic KAC

We now formally prove the CPA security of the basic KAC. We begin by stating the following theorem.

Theorem 1. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be bilinear elliptic curve subgroups of prime order q. For any pair of positive integers $n', n(n' > n)$, the basic KAC is (τ, ϵ, n') CPA secure if the decision (τ, ϵ, n, n)-BDHE assumption holds in $(\mathbb{G}_1, \mathbb{G}_2)$.

Proof: Let $\mathcal{A}$ be a τ-time adversary such that $|Adv_{\mathcal{A},n'} - \frac{1}{2}| > \epsilon$ for a KAC system parameterized with a given n. We build an algorithm $\mathcal{B}$ that has advantage at least ϵ in solving the (n, n)-BDHE problem in $(\mathbb{G}_1, \mathbb{G}_2)$. Algorithm $\mathcal{B}$ takes as input a random (n, n)-BDHE challenge $(I' = (H', P, Q, Y_{P,\alpha,n}, Y_{Q,\alpha,n}), Z')$ (where Z' is either $\hat{e}(P_{n+1}, H')$ or a random value in $\mathbb{G}_T$), and proceeds

as follows.

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

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

- $param$ is set as $((P, Q, Y_{P,\alpha,n}, Y_{Q,\alpha,n}))$.
- msk is set as some u randomly chosen from $\mathbb{Z}_q$.
- Set $PK = (PK_1, PK_2)$, where PK_1 and PK_2 are computed as $uP - P_i$ and $uQ - Q_i$ respectively.
- $K_{\overline{\mathcal{S}}}$ is set as

$$K_{\overline{\mathcal{S}}} = \sum_{j \notin \mathcal{S}} (uP_{n+1-j} - (P_{n+1-j+i}))$$

  Note that $K_{\overline{\mathcal{S}}}$ is equal to $\sum_{j \notin \mathcal{S}} \alpha^{n+1-j} PK_1$, which is an indirect way of computing as stated in the scheme. Moreover, $\mathcal{B}$ is aware that $i \notin \overline{\mathcal{S}}$ (implying $i \neq j$), and hence has all the resources to compute $K_{\overline{\mathcal{S}}}$.
- Choose a random $t \in \mathbb{Z}_q$, and set $U = tQ$.

Since P, Q α, U and the u values are chosen uniformly at random, *the public parameters and the public key have an identical distribution to that in the actual construction*.

Challenge: To generate the challenge, $\mathcal{B}$ picks at random two messages m_0 and m_1 from the set of possible plaintext messages belonging to class i. She randomly picks $b \in \{0, 1\}$, and sets the challenge as $(\mathcal{C}, m_0, m_1)$, where $\mathcal{C} = (H' - U, uH', m_b.Z')$. We claim that when $Z' = \hat{e}(P_{n+1}, H')$ (i.e. the input to $\mathcal{B}$ is a valid (n, n)-BDHE tuple), then $(\mathcal{C}, m_0, m_1)$ is a valid challenge to $\mathcal{A}$ as in a real attack. To see this, we write $H' = t'Q$ for some unknown $t' \in \mathbb{Z}_q$, and let $r = t' - t$. Then we have

$$\begin{aligned} H' - U &= rQ \\ uH' &= t'(uQ) = t'(PK_2 + Q_i) \\ m_b.Z' &= m_b \hat{e}(P_{n+1}, t'Q) \end{aligned}$$

Thus, by definition, $\mathcal{C}$ is a valid encryption of the messsage m_b in class i and hence, $(\mathcal{C}, m_0, m_1)$ 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 $Z' = \hat{e}(P_{n+1}, H')$). Otherwise, it outputs 1 (indicating that Z' is a random element in $\mathbb{Z}_T$).

We now analyze the probability that $\mathcal{B}$ gives a correct output. If (I', Z') is sampled from $\mathcal{R}'_{\text{BDHE}}$, we have $Pr[\mathcal{B}(I', Z') = 0] = \frac{1}{2}$, while if (I', Z') is sampled from $\mathcal{L}'_{\text{BDHE}}$, $|Pr[\mathcal{B}(I', Z')] - \frac{1}{2}| = |Adv_{\mathcal{A},n'} - \frac{1}{2}| \geq \epsilon$. This implies that $\mathcal{B}$ has advantage at least ϵ in solving the (n, n)-BDHE problem in $(\mathbb{G}_1, \mathbb{G}_2)$. This concludes the proof of Theorem 1. Note that this proof does not use the random oracle model. In the next section, we demonstrate how the basic KAC scheme can be modified and strengthened to achieve CCA security as well.

4 Chosen Ciphertext Secure Basic KAC

We now demonstrate how to extend the basic KAC proposed in Section 3 to obtain chosen ciphertext security. The resulting KAC system is CCA secure again *without using random oracles*. To the best of our knowledge, this is the first CCA secure KAC proposed in literature. The extension methodology is inspired by ideas presented in [15].

4.1 Additional Requirements for CCA Security

We have the following additional requirements for the CCA secure basic KAC:

- 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.

4.2 Construction

We will demonstrate in the following section that the security of CCA-secure Dynamic KAC for n data classes is based on the $(n+1, n+1)$-BDHE assumption, instead of the (n, n)-BDHE assumption as for the basic scheme. For consistency of notation, we describe here the CCA-secure dynamic KAC for $n - 1$ users, such that the security assumption is still the (n, n)-BDHE assumption as before.

SetUp$(1^\lambda, n-1)$: Randomly pick $\alpha \in \mathbb{Z}_q$. Output the system parameter as $param = (P, Q, Y_{P,\alpha,n}, Y_{Q,\alpha,n}))$. Discard α. Note that we still output $Y_{P,\alpha,n}$ and $Y_{Q,\alpha,n}$ instead of $Y_{P,\alpha,n-1}$ or $Y_{Q,\alpha,n-1}$.

KeyGen(): Same as in the basic scheme of Section 3.

Encrypt(PK, i, m): Run the $SigKeyGen$ algorithm to obtain a signature signing key K_{SIG} and a verification key $V_{SIG} \in \mathbb{Z}_q$. Then, randomly choose $r \in \mathbb{Z}_q$ and let $t' = t + r \in \mathbb{Z}_q$. Set

$$\begin{aligned} c_1 = rQ \quad \text{and} \quad c_2 &= t'(PK_2 + Q_i + V_{SIG} Q_n) \\ c_3 &= m\hat{e}(P_n, t'Q_1) \\ \mathcal{C} &= ((c_1, c_2, c_3), Sign(\mathcal{C}', K_{SIG}), V_{SIG}) \end{aligned}$$

Output the ciphertext $\mathcal{C}$.

Extract$(msk, \mathcal{S})$: Same as in Section 3.

Decrypt$(\mathcal{C}, i, \mathcal{S}, K_{\mathcal{S}}, U)$: Let $\mathcal{C} = (\mathcal{C}', \sigma, V_{SIG})$. Verify that σ is a valid signature of $\mathcal{C}'$ under the key V_{SIG}. If not, output $\perp$. Also, if $i \notin \mathcal{S}$, output $\perp$. Otherwise, pick a random $w \in \mathbb{Z}_q$ and set

$$\begin{aligned} SIG_{\mathcal{S}} &= \sum_{j \in \mathcal{S}} V_{SIG} P_{2n+1-j} \\ a_{\mathcal{S}} &= \sum_{j \in \mathcal{S}, j \neq i} P_{n+1-j+i} \\ b_{\mathcal{S}} &= \sum_{j \in \mathcal{S}} P_{n+1-j} \end{aligned}$$

Note that these can be computed as $1 \leq i, j \leq n-1$. Next, set two entities $\hat{d_1}$ and $\hat{d_2}$ as

$$\begin{aligned} \hat{d_1} &= K_{\mathcal{S}} + SIG_{\mathcal{S}} + a_{\mathcal{S}} + w(PK_1 + P_i + V_{SIG}P_n) \\ \hat{d_2} &= b_{\mathcal{S}} + wP \end{aligned}$$

Output the decrypted message

$$\hat{m} = c_3 \frac{\hat{e}(\hat{d_1}, U + c_1)}{\hat{e}(\hat{d_2}, c_2)}$$

The proof of correctness of this scheme is very similar to the proof for the basic dynamic KAC scheme presented in Section 3. Note that the ciphertext size is still constant and everything else, including the public and private parameters, as well as the aggregate key, remains unchanged. The main change from the original scheme is in the fact that decryption requires a randomization value $w \in \mathbb{Z}_q$. This randomization makes sure that that the pair $(\hat{d_1}, \hat{d_2})$ is chosen from the following distribution

$$(x(PK_1 + P_i + V_{SIG}P_n) - P_{n+1}, xP) \tag{3}$$

where x is chosen uniformly from $\mathbb{Z}_q$. To verify this claim, set $x = w + \sum_{j \in \mathcal{S}} \alpha^{n+1-j}$. Moreover, since w is uniformly random in $\mathbb{Z}_q$, so is x. *This randomization is a vital aspect from the point of view of CCA-security*. Note that the distribution $(\hat{d_1}, \hat{d_2})$ depends on the ciphertext class i for the message m to be decrypted.

4.3 Proof of CCA-Security

We begin by recalling that a signature scheme $(SigKeyGen, Sign, Verify)$ is said to be (τ, ϵ, q_S) strongly existentially unforgeable if no τ-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 [21].

Theorem 2. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be bilinear elliptic curve subgroups of prime order q. For any positive integer n, the modified basic KAC is $(\tau, \epsilon_1 + \epsilon_2, n-1, q_D)$ CCA-secure if the decision (τ, ϵ, n, n)-BDHE assumption holds in $(\mathbb{G}_1, \mathbb{G}_2)$ and the signature scheme is $(t, \epsilon_2, 1)$ strongly existentially unforgeable.

Proof: Let $\mathcal{A}$ be a τ-time adversary such that $|Adv_{\mathcal{A},n-1} - \frac{1}{2}| > \epsilon_1 + \epsilon_2$. We build an algorithm $\mathcal{B}$ that has advantage at least ϵ_1 in solving the (n,n)-BDHE problem in $\mathbb{G}$. Algorithm $\mathcal{B}$ takes as input a random (n,n)-BDHE challenge $(I' = (H', P, Q, Y_{P,\alpha,l}, Y_{Q,\alpha,l}), Z')$ (where Z' is either $\hat{e}(P_{n+1}, H')$ or a random value in $\mathbb{G}_T$), and proceeds as follows.

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

SetUp: $\mathcal{B}$ should generate the public $param$, public key PK, the authentication key U, 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 $((P, Q, Y_{P,\alpha,n}, Y_{Q,\alpha,n}))$.
- msk is set as some u randomly chosen from $\mathbb{Z}_q$.
- Set $PK = (PK_1, PK_2)$, where PK_1 and PK_2 are computed as $uP - {V^*}_{SIG}P_n - P_i$ and $uQ - {V^*}_{SIG}Q_n - Q_i$ respectively.
- $K_{\overline{\mathcal{S}^*}}$ is set as

  $$K_{\overline{\mathcal{S}^*}} = \sum_{j \notin \mathcal{S}^*} (uP_{n+1-j} - {V^*}_{SIG}P_{2n+1-j} - P_{n+1-j+i})$$

  Note that $K_{\overline{\mathcal{S}^*}}$ is equal to $\sum_{j \notin \mathcal{S}^*} \alpha^{n+1-j} PK_1$, in accordance with the specification provided by the scheme. Moreover, $\mathcal{B}$ is aware that $i \notin \overline{\mathcal{S}^*}$ (implying $i \neq j$). Also, $j \neq n$ as $1 \leq j \leq n-1$. Hence, $\mathcal{B}$ has all the resources to compute $K_{\overline{\mathcal{S}^*}}$.
- Choose a random $t \in \mathbb{Z}_q$, and set $U = tQ$.

Since P, α, U and the u values are chosen uniformly at random, *the public parameters and the public key have an identical distribution to that in the actual construction*.

Query Phase 1: Algorithm $\mathcal{A}$ now issues decryption queries. Let $(j, \mathcal{S}, \mathcal{C})$ be a decryption query where $\mathcal{S} \subseteq \mathcal{S}^*$ and $j \in \mathcal{S}$. Let $\mathcal{C} = ((c_1, c_2, c_3), \sigma, V_{SIG})$. Algorithm $\mathcal{B}$ first runs $Verify$ to check if the signature σ is valid on (c_1, c_2, c_3) 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

$$\begin{aligned} \hat{d_0} &= (V_{SIG} - {V^*}_{SIG})P_n + P_j - P_i \\ \hat{d_0'} &= (V_{SIG} - {V^*}_{SIG})^{-1}(P_{j+1} - P_{i+1}) \\ \hat{d_2} &= xP + (V_{SIG} - {V^*}_{SIG})^{-1} P_1 \\ \hat{d_1} &= u\hat{d_2} + xd_0 + d_0' \end{aligned}$$

$\mathcal{B}$ responds with $K = c_3 \frac{\hat{e}(\hat{d_1}, c_1 + U)}{\hat{e}(\hat{d_2}, c_2)}$. To see that this response is as in a real attack game, set $x' = x + \alpha(V_{SIG} - {V^*}_{SIG})^{-1}$ and observe that $\hat{d_2} = x'P$ and $\hat{d_1} = x'(PK_1 + P_i + V_{SIG}P_n) - P_{n+1}$. Furthermore, since x is uniform in $\mathbb{Z}_q$, x' is also uniform in $\mathbb{Z}_q$. Thus, $\mathcal{B}$'s response is identical to **Decrypt**$(\mathcal{C}, j, \mathcal{S}, K_{\mathcal{S}}, U)$ as required.

Challenge:To generate the challenge, $\mathcal{B}$ picks at random two messages m_0 and m_1 from the set of possible plaintext messages belonging to class i. She randomly picks $b \in \{0,1\}$, and sets

$$\begin{aligned} \mathcal{C} &= (H' - U, uH', m_b.Z) \\ \mathcal{C}^* &= (\mathcal{C}, Sign(\mathcal{C}, {K^*}_{SIG}), {V^*}_{SIG}) \end{aligned}$$

The challenge posed to $\mathcal{A}$ is $(\mathcal{C}^*, m_0, m_1)$. We claim that when $Z = \hat{e}(P_{n+1}, H')$, $(\mathcal{C}^*, m_0, m_1)$ is a valid challenge to $\mathcal{A}$ as in a real attack. To see this, we write $H' = t'P$ for some unknown $t' \in \mathbb{Z}_q$, and let $r = t' - t$. Then we have

$$\begin{aligned} H' - U &= rP \\ uH' &= t'(PK_2 + Q_i + {V^*}_{SIG}Q_n) \\ m_b.Z' &= m_b\hat{e}(P_{n+1}, t'Q) \end{aligned}$$

Thus, by definition, $\mathcal{C}^*$ is a valid encryption of m_b in class i and hence, $(\mathcal{C}^*, m_0, m_1)$ is a valid challenge to algorithm $\mathcal{A}$.

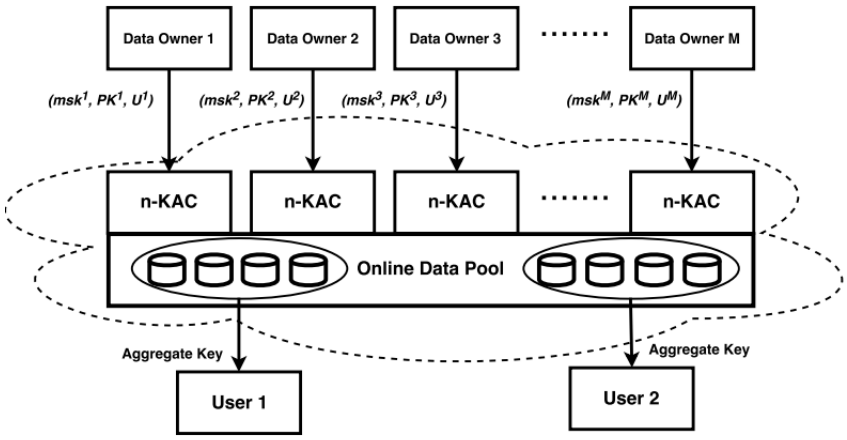

Fig. 2: Generalized Multi-Data Owner KAC

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 $Z' = \hat{e}(P_{n+1}, H)$). Otherwise, it outputs 1 (indicating that Z' is a random element in $\mathbb{Z}_T$).

Quite evidently, if (I', Z') is sampled from $\mathcal{R}'_{\text{BDHE}}$, $Pr[\mathcal{B}(I', Z') = 0] = \frac{1}{2}$. Let **abort** be the event that $\mathcal{B}$ aborted the simulation. Now when (I', Z') is sampled from $\mathcal{L}'_{\text{BDHE}}$, we have

$$|Pr[\mathcal{B}(I', Z')] - \frac{1}{2}| > (\epsilon_1 + \epsilon_2) - Pr[\textbf{abort}]$$

This essentially implies that $\mathcal{B}$ has advantage at least $\epsilon_1 + \epsilon_2 - Pr[\textbf{abort}]$ in solving the (n, n)-BDHE problem in $(\mathbb{G}_1, \mathbb{G}_2)$.

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 $Pr[\textbf{abort}] < \epsilon_2$; 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 master secret key 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, $Pr[\text{abort}] < \epsilon_2$, implying $\mathcal{B}$ has advantage at least ϵ_1 in solving the (n, n)-BDHE problem in $(\mathbb{G}_1, \mathbb{G}_2)$. This completes the proof of Theorem 2.

5 General KAC for Multiple Data Owners

The basic version of KAC discussed so far accommodates a single data owner with n data classes. In a practical deployment scenario, we would ideally want a system where multiple users can store their own data, and delegate users with decryption rights to their own data classes. The challenge in designing such a system is to ensure that data owned by different data owners should be isolated and the aggregate key for one data owner should not reveal the data for another data owner. This concept is referred to as *local aggregation*. We tackle this issue by proposing a *two-tier* construction that uses the cryptographic security provided by the basic KAC and extends it to accommodate multiple users.

5.1 The Basic Idea : A Two-Tier Construction

The fundamental idea underlying the proposed construction is summarized in Figure 2. An underlying layer of KAC parameterized with n forms the core of the architecture. Any user who registers in the system with a public key-private key pair gets her own copy of the underlying n basic KAC on the cloud where she can store her own encrypted data, and delegate decryption rights to users, as was the case with the basic KAC. If a user wishes to store more than n classes of data, she registers another public key-private key pair. We refer to each of the individual owner units as *data superclasses*. The main advantages of using this approach are as follows:

- Each superclass of data in the system is characterized by its own public key, master secret key and authentication key.
- The system achieves local aggregation within each superclass in the sense that decryption rights to any subset of data classes in the same superclass can be aggregated into a single constant-size key as before.
- A user must have the local aggregate key as well as the authentication key of a particular superclass to access the corresponding subset of data classes in that superclass. This ensures that data privacy for each data owner is independent of the other.
- The system is scalable since it allows new users to register and share their data without having to alter the underlying basic KAC set-up in any way. This also implies that the basic security assumption of the system is not affected by the increase in the number of data classes.

We next describe the construction of the two-tier scheme. The construction is inspired by the generalized broadcast encryption system described in [15]. Suppose at any point there are M different public key-private key pairs registered in the system, and the underlying basic KAC for each of

them accommodates n data classes. Each data class must now have a double index (i_1, i_2) where $1 \leq i_1 \leq M$ and $1 \leq i_2 \leq n$. This allows the overall setup to handle $N = Mn$ classes. However, it is important to note that all the instances can use the same public parameters. We further point out that while in [15], the two-tier construction offers a trade-off between the public parameter size and the ciphertext size, our two-tier scheme actually reduces the public parameter size from n to n, without increasing the size of the ciphertext.

5.2 Construction

We present the construction of two-tier multi-user KAC. Recall that we assume the existence of equi-prime order (q such that $2^\lambda \leq q \leq 2^{\lambda+1}$) elliptic curve subgroups $\mathbb{G}_1$ and $\mathbb{G}_2$, along with their generators P and Q. We also assume the existence of a multiplicative cyclic group $\mathbb{G}_T$, also of order q with identity element 1. Finally, we assume there exists an asymmetric bilinear pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \longrightarrow \mathbb{G}_T$. The notations used in the forthcoming discussion are already introduced in Section 2.

SetUp$(1^\lambda, n)$: Randomly pick $\alpha \in \mathbb{Z}_q$. Output the system parameter as $param = (P, Q, Y_{P,\alpha,n}, Y_{Q,\alpha,n}))$. Discard α.

KeyGen(): When a new data owner i_1 registers in the system, randomly pick $\gamma^{i_1}, t^{i_1} \in \mathbb{Z}_q$. Set the master secret key msk^{i_1} to γ^{i_1}. Let $PK_1^{i_1} = \gamma^{i_1} P$ and $PK_2^{i_1} = \gamma^{i_1} Q$. Set the public key $PK^{i_1} = (PK_1^{i_1}, PK_2^{i_1})$. Finally set the user authentication key $U^{i_1} = t^{i_1} Q$.

Encrypt$(PK^{i_1}, (i_1, i_2), m)$: For a message $m \in \mathbb{G}_T$ belonging to the class (i_1, i_2), randomly choose $r \in \mathbb{Z}_q$ and let $t' = t^{i_1} + r \in \mathbb{Z}_q$. Output the ciphertext $\mathcal{C}$ as

$$\mathcal{C} = (c_1, c_2, c_3) = (rQ, t'(PK_2^{i_1} + Q_{i_2}), m\hat{e}(P_n, t'Q_1))$$

Extract$(msk^{i_1}, \mathcal{S})$: For the subset of class indices $\mathcal{S}$, the aggregate key is computed as

$$K_{\mathcal{S}}^{i_1} = \sum_{(i_1,i_2)\in\mathcal{S}} msk^{i_1} P_{n+1-i_2}$$

Note that this is indirectly equivalent to setting $K_{\mathcal{S}}^{i_1}$ to $\sum_{(i_1,i_2)\in\mathcal{S}} \alpha^{n+1-i_2} PK_1^{i_1}$.

Decrypt$(\mathcal{C}, (i_1, i_2), K_{\mathcal{S}}^{i_1}, U^{i_1}, \mathcal{S})$: If $(i_1, i_2) \notin \mathcal{S}$, output $\perp$. Otherwise, set

$$\begin{aligned} a_{\mathcal{S}}^{i_1} &= \sum_{(i_1,j_2)\in\mathcal{S}, j_2\neq i_2} P_{n+1-j_2+i_2} \\ b_{\mathcal{S}}^{i_1} &= \sum_{(i_1,j_2)\in\mathcal{S}} P_{n+1-j_2} \\ \hat{m} &= c_3 \frac{\hat{e}(K_{\mathcal{S}}^{i_1} + a_{\mathcal{S}}^{i_1}, U^{i_1} + c_1)}{\hat{e}(b_{\mathcal{S}}^{i_1}, c_2)} \end{aligned}$$

Return the decrypted message $\hat{m}$.

Quite evidently, the size of the local aggregate key for each superclass is still of constant size. However, if a user wishes to access a subset of data classes that spans across multiple superclasses, then the size of the overall aggregate key grows accordingly. The correctness of this scheme can be proved exactly as for the basic KAC scheme.

5.3 Performance of General KAC

We look at the performance of the general KAC construction from the perspective of both parties involved - the data owner sharing her data and an user accessing the shared data. From the data owner's perspective, each superclass comprises of a basic KAC module parameterized with n, with constant size public, private and authentication keys. The encryption process can be speeded up using similar caching techniques described in Section 3.2. From the user's perspective, the system is an ensemble of basic KAC units - one per superclass. For a user willing to access data from k data owners, the aggregate key and authentication keys have size linear in k. It seems difficult to do better than this while maintaining data isolation and privacy for each data owner. As in encryption, the decryption process can also be expedited using appropriate caching techniques. It is important to note that the public $param$, which essentially forms the fundamental basic for the security of the general KAC, is a function of n only, and does not depend on the number of users M registered in the system.

An important aspect of this construction is to decide on an appropriate value for n. Keeping a small n value reduces the size of the public parameter. But then a data owner with a large number of data classes N would have to register with $\lceil N/n \rceil$ public, private and authentication key sets, which may not be desirable. Thus there exists a trade-off in using smaller and larger values of n in this scheme.

It is important to note that the security of the system depends on the (n, n)-BDHE assumption, as we show next.

5.4 Security of General KAC

For the CPA security of general multi-data owner KAC, we state the following theorem.

Theorem 3. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be bilinear elliptic curve subgroups of prime order q. For any pair of positive integers $n', n(n' > n)$, the two-tier KAC is (τ, ϵ, n') CPA secure if the decision (τ, ϵ, n, n)-BDHE assumption holds in $(\mathbb{G}_1, \mathbb{G}_2)$.

The proof of this theorem is very similar to the proof of Theorem 1 and is hence avoided. The generalized KAC can be modified to achieve CCA security as well. The procedure for obtaining the modified generalized KAC is very similar to that shown in Section 4.

It is important to note that in a practical environment, any data sharing scheme is expected to support a hierarchical organization of data, where different users are given access to different levels of the data. For example, consider an online data pool on the cloud storing an organization's data. The organization (which owns the data) would like to delegate access rights to this data among its employees. Different employees may be given access to different levels of this data depending on their respective credentials and designation. The most common solution to this is to have a pre-defined hierarchical framework for organizing the data, and have keys at different levels of the hierarchy. As pointed out in our previous discussion, highly irregular

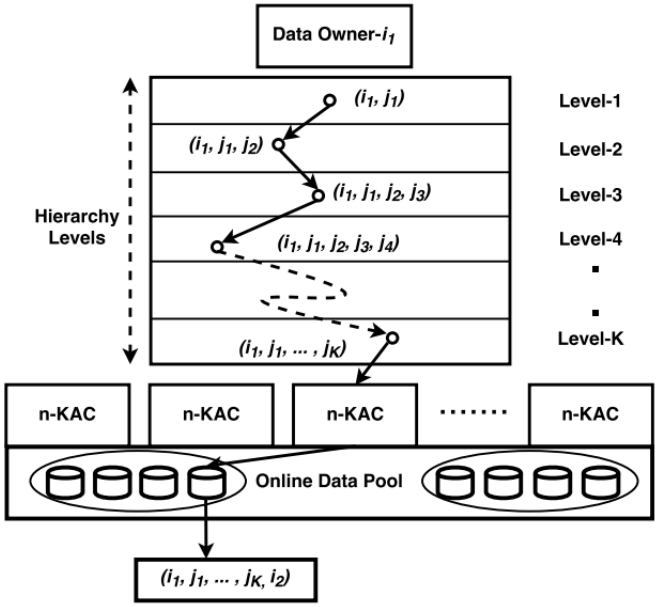

Fig. 3: Hierarchical KAC

data access patterns from multiple users could compromise the efficiency of such a hierarchical system by potentially blowing up the size of the key to be shared. Since key aggregation allows us to combine the decryption rights to multiple data classes into a single constant-sized entity, it seems KAC could be the most efficient solution even in this scenario. However, we show next via a small example that the generalized KAC in its original form may not be able to tackle this problem in the most efficient manner.

Although the generalized KAC is highly scalable and efficient, it has a drawback in the sense that it fails to support hierarchical organization of data within a single data owner system. Consider the scenario in which a data owner would like to have two distinct folders, namely F_1 and F_2, that are meant to be accessed by separate user groups. In the generalized KAC framework, she must then have different public, private and authentication keys for these folders. Now if she were to delegate decryption rights to some user for a subset of files both these folders, she would need to provide two aggregate keys, one for each folder. This is because the generalized KAC does not allow her to aggregate these two keys into a single key. If she wants to provide a single aggregate key for both the folders, she would need to have another aggregate system in place to combine these two individual keys. This process is cumbersome and becomes difficult to maintain as the number of hierarchy levels grow. The same disadvantage also persists with the KAC proposed in [14]. However, since data hierarchy is extremely common in file-based storage systems, we must adopt KAC so as to be able to provide the advantage of key aggregation for any arbitrarily complex hierarchical structure, without the data owner having to worry about the internal details of the aggregation. The modified KAC infrastructure must now support key aggregation at various levels of the hierarchy, while using the same security guarantees provided the fundamental single data owner KAC. This motivates the introduction of an augmented version of the general KAC in the next section, which we refer to as the *hierarchical* KAC.

6 A Hierarchical KAC

In this section, we introduce the third and last version of KAC that allows hierarchical data organization for multiple data owners. The similarity of this scheme with the general KAC proposed in Section 5 is in the fact that the fundamental unit of aggregation at the lowest level of hierarchy remains the same - a basic single user KAC parameterized by n. The main USP of the hierarchical KAC lies in its application of the key aggregation principle to multiple levels for each data owner. To the best of our knowledge, this is the first attempt to combine the ideas of hierarchical data storage with that of key aggregation for decryption. We begin by describing a naïve construction for the hierarchical KAC, and then look at a more elegant construction that overcomes some of the shortcomings of the naïve construction.

6.1 The Hierarchical Setting

We begin by describing a model for the hierarchical data organization that we use in this paper. Our model assumes a file-based data storage system, where the hierarchy is realized in the form of a nested folder-based organization of the data. Also, it is natural to assume that all data files belonging to the same data class are present in the same folder. For clarity of understanding, we view the overall organization as a tree-like structure, with the internal nodes representing folders, and the final data is stored in the leaf nodes. The parent-child relationship between any pair of nodes in this tree semantically represents *containment*, that is, the parent folder contains all its child folders/files. The data organization is summarized in Figure **??**. Our aim is to design a KAC that allows the user to efficiently and secretly delegate decryption rights to multiple nodes of this tree according to a user's credentials. The main challenge is to be able to perform key aggregation at different levels

of this tree without compromising on either performance or security. With this broad overview, we now introduce a formal description of the hierarchical data storage model, which we use in the forthcoming discussion on hierarchical KAC.

As in the generalized data storage model introduced in Section 5, suppose there are M users registered in the system. Each user i_1 has her own K-hierarchical KAC, with each hierarchy level, denoted by k^{i_1}, having $n_k^{i_1}$ classes. In our view of the hierarchical organization as a tree, the index of any data class essentially denotes the sequence of nodes encountered while traveling from the root to the file at the leaf node. Also, the *level* of a node in the hierarchy is the length of the path from the root to that node. Figure 3 summarizes this pictorially. Each message (at the leaf level of the hierarchy) now has a $K+2$ length index $(i_1, j_1, j_2 \cdots, j_K, i_2)$, while any other class (internal node) at level k of the hierarchy has an index of length $k+1$ given by $(i_1, j_1, \cdots, j_k)$. Here i_1 denotes the data owner id and i_2 denotes its index in the lowermost level of the hierarchy, as in the generalized scheme. Note that the total number of classes for data owner i_1 is given by $N^{i_1} = n \prod_{k=1}^{K} n_k^{i_1}$, and the total number of data classes handled by the system (assuming n_1 data owners at any point of time) is given by $\sum_{i_1=1}^{n_1} N^{i_1}$. For clarity of presentation, we initially describe the scheme by restricting the number of hierarchy levels to K for each user. We then discuss how the scheme may be slightly modified to allow users to decide on their own number of hierarchy levels and also how it can be managed dynamically.

6.2 A Naïve Hierarchical KAC

The first construction of hierarchical KAC that we present here is naïve in the sense that it is a simple extension of the generalized KAC. The idea is that each internal node is able to aggregate into a single key the decryption rights to any subset of nodes in the subtree rooted at that node. If the aggregate key for an internal node was to be constructed by simply concatenating the aggregate keys for the corresponding subtrees rooted at this tree, then the size of the aggregate key grows as we move up the hierarchy, potentially exponentially with the level of the node in the hierarchy. Since this is against the basic idea of key aggregation where any aggregate key must be of constant size, this approach, although simple, is not a viable one. A naïve alternative solution to this problem is to use a separate KAC at each level of the hierarchy, that aggregates the keys for any subset of nodes in the subtree rooted at any internal node at this level. This ensures that any aggregate key at a given node of the tree in this hierarchical structure is of constant size.

The naïve hierarchical KAC is simple to understand but inefficient to implement for the following reasons:

- The scheme requires to publish the public $param$ for the KAC at level k characterized by $n_k^{i_1}$ for each data owner. As the number of hierarchy levels increase, the size of $param$ also increases. This leads to massive storage requirements.
- There is a possibility that the $param$ for KAC at some internal node compromises the secret parameter for KAC at another internal node. In this case, although the standalone KAC at each node is still secure in the standard model, the overall system is compromised. So when constructing the hierarchical KAC, it is important to ensure this situation does not arise. This however is extremely difficult to ensure since the secret parameter α used to setup the KAC system is discarded after each **SetUp** operation.
- Alternatively, if the system decides to use KAC characterized by identical security parameter, say n, at all levels of the hierarchy, then we restrict the number of children any internal node can have to n, under the restriction that the $params$ must not change once published. This is not entirely practical since data owners may wish to dynamically update their data organization. On the other hand, even if we allow the $param$ to be changed, we still restrict each internal node to have the same number of children since the KAC at each internal node uses the same $param$. This is not a practical restriction either in a real world data sharing scenario.
- Finally, any decryption using an aggregate key

6.3 An Alternative Hierarchical KAC

In order to address the shortcomings of the naïve KAC, we propose an alternative hierarchical KAC with the following salient features:

- A data owner can organize her data in K hierarchical levels, with n_k data classes at the k^{th} level of hierarchy. The value of n_k can be decided by the data owner for each $k, 1 \leq k \leq K$.
- Each hierarchy level k is characterized by its own unique public key, private key and authentication key. This allows local aggregation of the key for different data classes at each level of the hierarchy.
- The security of the system is still provided by the basic KAC, on top of which the entire hierarchy is constructed. For consistency of notation, we assume that this basic KAC is still characterized by the parameter n.
- The ciphertext for hierarchical KAC is a tuple that consists of $K+2$ curve points. If K is a constant, then hierarchical KAC still has a constant ciphertext size.

For clarity of presentation, we initially describe the scheme by restricting the number of hierarchy levels to K for each user. We then discuss how the scheme may be slightly modified to allow users to decide on their own number of hierarchy levels and also how it can be managed dynamically.

We next describe the construction of the K-hierarchical KAC. Once again, as in the generalized construction of Section 5, suppose there are M users registered in the system. Each user i_1 has her own K-hierarchical KAC, with each hierarchy level, denoted by k^{i_1}, having $n_k^{i_1}$ classes. If one were to view the hierarchical organization as a tree, the index of any data class essentially denotes the sequence of nodes encountered while traveling from the root to the file at the leaf node. Figure 3 summarizes this pictorially.

Each message (at the leaf level of the hierarchy) now has a $K+2$ length index $(i_1, j_1, j_2 \cdots, j_K, i_2)$, while any other class (internal node) at level k of the hierarchy has an index of length $k+1$ given by $(i_1, j_1, \cdots, j_k)$. Here i_1 denotes the data owner id and i_2 denotes its index in the lowermost level of the hierarchy, as in the generalized scheme. Note that the total number of classes for data owner i_1 is given by $N^{i_1} = n\prod_{k=1}^{K} n_k^{i_1}$, and the total number of data classes handled by the system (assuming n_1 data owners at any point of time) is given by $\sum_{i_1=1}^{n_1} N^{i_1}$.

6.4 Construction

We present the construction of hierarchical multi-user KAC. Please note that $\mathcal{U}$ denotes the entire set of message classes in the system.

SetUp$(1^\lambda, n)$: Randomly pick $\alpha \in \mathbb{Z}_q$. Output the system parameter as $param = (P, Q, Y_{P,\alpha,n}, Y_{Q,\alpha,n}))$. Discard α.

KeyGen(): When a new data owner i_1 registers in the system, randomly pick $\gamma_{k,j}^{i_1}, t_{k,j}^{i_1} \in \mathbb{Z}_q$ for $1 \leq k \leq K$ and $1 \leq j \leq n_k^{i_1}$. Set the master secret key $msk_{k,j}^{i_1}$ to $\gamma_{k,j}^{i_1}$. Let $PK_{1,k,j}^{i_1} = \gamma_{k,j}^{i_1} P$ and $PK_{2,k,j}^{i_1} = \gamma_{k,j}^{i_1} Q$. Finally set the user authentication key $U_{k,j}^{i_1} = t_{k,j}^{i_1} Q$. Output the collection of all these individual keys as the cumulative keys msk^{i_1}, PK^{i_1} and U^{i_1} for data owner i_1.

Encrypt$(PK^{i_1}, (i_1, j_1, \cdots, j_K, i_2), m)$: For a message $m \in \mathbb{G}_T$ belonging to the class $(i_1, j_1, \cdots, j_K, i_2)$, randomly choose $r \in \mathbb{Z}_q$ and let $t' = \sum_{k=1}^{K} t_{k,j_k}^{i_1} + r \in \mathbb{Z}_q$. We define the following entities for $1 \leq k \leq K$

$$W^{i_1, j_1, \cdots, j_k} = \sum_{(x=1)}^{k} msk_{x,j_x}^{i_1} \tag{4}$$

Next we define $c_2^k = t'(W^{i_1, j_1, \cdots, j_k})Q + t'Q_{i_2}$ and set

$$\begin{aligned} c_1 &= rQ \\ c_2 &= (c_2^1, \cdots, c_2^K) \\ c_3 &= m\hat{e}(P_n, t'Q_1) \end{aligned}$$

Output the ciphertext $\mathcal{C} = (c_1, c_2, c_3)$.

Extract$(msk^{i_1}, \mathcal{S})$: As mentioned earlier, the aggregate key is now defined at each level of the hierarchy. The aggregate key corresponding to subset $\mathcal{S}$ for any class at level k with index $(i_1, j_1, j_2, \cdots, j_k)$ is computed as follows. As in the previous KAC definitions, we set

$$a_{\mathcal{S}}^{i_1, j_1, j_2, \cdots, j_k} = \sum_{(i_1, j_1 \cdots, j_k, l_{k+1}, \cdots, l_K, x_2) \in \mathcal{S}, x_2 \neq i_2} P_{n+1-x_2+i_2}$$

$$b_{\mathcal{S}}^{i_1, j_1, j_2, \cdots, j_k} = \sum_{(i_1, j_1 \cdots, j_k, l_{k+1}, \cdots, l_K, x_2) \in \mathcal{S}} P_{n+1-x_2}$$

Next, output the aggregate key as

$$K_{\mathcal{S}}^{i_1, j_1, \cdots, j_k} = (W^{i_1, j_1, \cdots, j_k}) b_{\mathcal{S}}^{i_1, j_1, j_2, \cdots, j_k}$$

Recall the definition of $W^{i_1, j_1, \cdots, j_k}$ proposed earlier. The aggregate key gives access to all message classes of the form $(i_1, j_1 \cdots, j_k, l_{k+1}, \cdots, l_K, i_2)$, in the subtree rooted at the internal class node $(i_1, j_1, \cdots, j_k)$, that are present in the subset $\mathcal{S}$. To see this, note that the outer summation in both X and Y is over all the messages classes that are included in the subtree rooted in this node. This also implies while that a node higher up in the hierarchy covers more number of classes, computing the aggregate key for it requires more time.

Decrypt$(\mathcal{C}, (i_1, j_1, \cdots, j_K, i_2), K_{\mathcal{S}}^{i_1}, U^{i_1}, \mathcal{S})$: Suppose this class is covered by a node at the k^{th} level and the corresponding aggregate key is $K_{\mathcal{S}}^{i_1, j_1, \cdots, j_k}$. Now, let

$$L_{\mathcal{S}}^{i_1, j_1, j_2, \cdots, j_k} = |\{(i_1, j_1 \cdots, j_k, l_{k+1}, \cdots, l_K, x_2) \in \mathcal{S}, x_2 = i_2\}|$$

Also, let $u = \sum_{k=1}^{K} U_{k,j_k}^{i_1}$. Return the decrypted message

$$\hat{m} = c_3 \left(\frac{\hat{e}(K_{\mathcal{S}}^{i_1, j_1, \cdots, j_k} + a_{\mathcal{S}}^{i_1, j_1, \cdots, j_k}, u + c_1)}{\hat{e}(b_{\mathcal{S}}^{i_1, j_1, \cdots, j_k}, c_2^k)} \right)^{\frac{1}{L_{\mathcal{S}}^{i_1, j_1, j_2, \cdots, j_k}}}$$

The proof of correctness of this scheme is very similar to that of the earlier schemes.

6.5 Performance and Efficiency Issues

We have already mentioned in Section 3.2 certain pre-computations that can be used to optimize the running time of encryption and decryption operations. Suitable extensions of these techniques may be applied to improve the performance of the hierarchical KAC as well. Note that in our proposed hierarchical KAC, there are a two main classes of computed parameters that essentially dominate the time required for the encryption and decryption operations, as shown below.

Class-1: $W^{i_1, j_1, j_2, \cdots, j_k}$
Class-2: $(a_{\mathcal{S}}^{i_1, j_1, j_2, \cdots, j_k}, b_{\mathcal{S}}^{i_1, j_1, j_2, \cdots, j_k}, L_{\mathcal{S}}^{i_1, j_1, j_2, \cdots, j_k})$

It is interesting to while all parameters are node-specific, Class-2 parameters are also subset specific. Let N be the total number of data classes, and ϑ be the amount of storage for a single group element (assumed to be a constant). Recall that the basic KAC underlying the hierarchical construction is characterized by the parameter n. We further assume for simplicity that the hierarchy has a regular tree-based structure, with each node parenting a constant number of children nodes. We now make the following observations.

Observation 1. Pre-computing and storing the Class-1 parameter for each node in the hierarchy requires an overall space complexity of $\Theta(\frac{N\vartheta}{n})$ space. Further, the computation of this parameter for any random internal node requires $\mathcal{O}(\frac{N}{n})$ amount of time in the worst case.

Observation 2. For a given subset $\mathcal{S}$, pre-computing and storing the Class-2 parameter set for each node leads to an overall space requirement of $\Theta(\frac{N\vartheta}{n})$, while the computation of these parameters for a random internal node requires $\mathcal{O}(\frac{N}{n})$ amount of time in the worst case.

The observations are quite straightforward to establish. We now focus on their implications on the computational efficiency of the scheme.

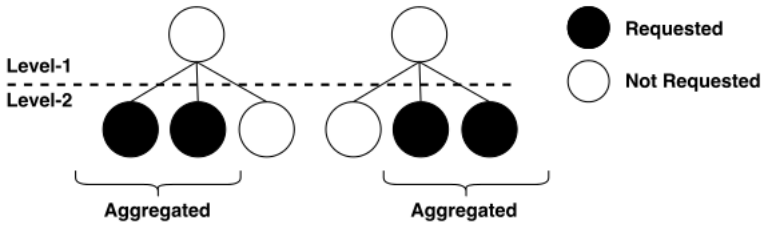

Fig. 4: A Practical Request Scenario in the Hierarchical Setting

- If the Class-1 parameter was pre-computed and stored for each node, we require polynomial storage, with constant time encryption. If we compute all parameters on the fly, encryption becomes polynomial-time in the worst case. To achieve a trade-off, one might store the parameter values for nodes at evenly spaced levels, and compute the rest on the fly from these intermittent stored values.
- For the computation of the Class-2 parameters, we can use extensions of the subset caching based techniques discussed in Section 3.2 to improve performance. Note that here, it is not possible to pre-compute the parameters at all nodes for all possible subsets of data classes. So it is probably to cache the parameters at each node only for larger subsets, while for the smaller subsets they can be computed on the fly.

We end this discussion on the performance of hierarchical KAC that the basic operations for the computation of these parameters are all associative, and can be speeded up by using parallel implementations on multi-core processors.

6.6 Security of Hierarchical KAC

For the CPA security of hierarchical KAC, we state the following theorem.

Theorem 4. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be bilinear elliptic curve subgroups of prime order q. For any pair of positive integers $n', n(n' > n)$, the hierarchical KAC is (τ, ϵ, n') CPA secure if the decision (τ, ϵ, n, n)-BDHE assumption holds in $(\mathbb{G}_1, \mathbb{G}_2)$.

Once again, the proof of this theorem is very similar to the proof of Theorem 1 and is hence avoided. Additionally, the hierarchical KAC scheme can also be easily modified to obtain a CCA-secure hierarchical KAC along similar lines to the discussion in Section 4.

6.7 Owner-Defined Hierarchy

The hierarchical KAC described above fixes the number of hierarchy levels to a pre-defined quantity K. This means any path in the hierarchy tree from the root to the length is of length $K + 2$. However, in a realistic scenario, it is easy to see that a data owner may want to have her own hierarchical construction and customize the number of levels in the hierarchy. The existing construction of hierarchical KAC can be easily adopted for this scenario by taking either of the following steps:

- The system may have a pre-defined upper bound K on the number of hierarchy levels. A data owner willing to go beyond this level must register again with a different set of public, private and access keys to share the additional data classes. A data owner with fewer hierarchy levels than K can have some *dummy nodes* that merely act as placeholders in the overall hierarchy.
- The other option is to have variable size indexes for different nodes depending on the depth of the hierarchical structure to which it belongs. As and when the hierarchy levels change, the length of index for each node in the hierarchy also changes. This system would require updating the index for each and every data class node in the hierarchy every time a new level is introduced or an old level is deleted. But it does avoid the use of dummy rounds, and allows the data owner the flexibility to increase the number of levels in the hierarchy.

6.8 Advantage over Hierarchical Cryptosystems

We end this section by pointing out that a major advantage of our proposed hierarchical KAC is that it avoids a significant pitfall of several existing hierarchical encryption based schemes [22], [7]. In standard tree based hierarchical systems, granting access to the key corresponding to any node implicitly grants access to all the keys in the subtree rooted at that node. This means granting access to a selected set of nodes in a given subtree would blow up the key-size to be the same as the number of nodes. This is avoided in our two-tier scheme, since decryption rights to any number of nodes (data classes) rooted at a particular node of the hierarchy tree may be aggregated into a single key corresponding to that node. Figure 4 summarizes this phenomenon for a simple two-level hierarchical system. In the situation depicted, a tree-based hierarchy system would require 4 constant size decryption keys, while our proposed hierarchical scheme would require only 2 constant-size decryption keys. The savings is much more pronounced as the number of hierarchy levels increase, as depicted in the following section via simulation studies in practical deployment scenarios.

7 Simulation Studies : Performance of proposed KAC

8 Applications of KAC

The key-aggregate encryption systems described in this paper are primarily meant for data sharing on the cloud. In this section, we point out some specific applications in which KAC proves to be a very efficient solution.

- **Online sharing of data for collaborative research** is extremely popular today in a number of educational institutions, corporations and government agencies. Much of this data could be sensitive and hence, needs to be stored in a secure fashion. KAC allows members of a team to share their own data on the cloud in an encrypted fashion, and delegate decryption rights specifically to those members of the team who are entitled to access the same. Also, in many organizations a team working on a project is expected to have a hierarchical organization, and as such storing the data in a hierarchical fashion with different access rights for different members is a practical requirement that the hierarchical KAC addresses.
  The other advantage of KAC is that the same setup with the same public parameters can be *reused* by multiple teams within the same organization and even across organizations. Since data owned by each individual owner is insulated from access by users who do not have the corresponding aggregate key, a single KAC can support multiple systems, while guaranteeing the same underlying security.
- **Distribution of product license and/or activation keys** is another useful application of KAC. Suppose a company wons a number of products, and intends to distribute the license/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/activation keys 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)** is a recent concept that has been studied in the literature [11]. 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.

9 Conclusions

In this paper, we have proposed a provably secure scalable key-aggregate cryptosystem (KAC) online data sharing on the cloud. The main aim of this cryptosystem is to allow data owners to delegate decryption rights to any subset encrypted files shared on the cloud using a constant-sized *aggregate key*. We have described the construction of a basic KAC unit that allows a single data owner to share her data online using constant sized ciphertext messages and linear-sized public parameters. We have proved this basic scheme to be CPA secure, and have also demonstrated how it may be suitably modified to achieve CCA security as well. To the best of our knowledge, this is the first KAC in the cryptographic literature that achieves CCA security. We have then demonstrated how this system can be actually deployed on the cloud securely and efficiently to accommodate multiple data owners. We laid particular focus on the performance, efficiency and scalability of the system. Finally, we have proposed a novel hierarchical KAC that allows a data owner to store her data in a hierarchical fashion on the cloud. Decryption can be performed via local aggregation at each hierarchy level. We have presented simulation results on an actual implementation of the system using bilinear Tate pairings, that showed that the proposed hierarchical KAC outperforms standard hierarchical key-based cryptosystems on performance and efficiency. We have also laid down guidelines on the best hierarchical construction for optimum performance in the KAC framework. We have also discussed possible application areas where KAC could be practically deployed.

References

[1] IDC Enterprise Panel. It cloud services user survey, pt. 3: What users want from cloud services providers, august 2008.

[2] Sherman SM Chow, Yi-Jun He, Lucas CK Hui, and Siu Ming Yiu. Spice–simple privacy-preserving identity-management for cloud environment. In *Applied Cryptography and Network Security*, pages 526–543. Springer, 2012.

[3] Cong Wang, Sherman S.-M. Chow, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for secure cloud storage. Cryptology ePrint Archive, Report 2009/579, 2009. http://eprint.iacr.org/.

[4] Sherman SM Chow, Cheng-Kang Chu, Xinyi Huang, Jianying Zhou, and Robert H Deng. Dynamic secure cloud storage with provenance. In *Cryptography and Security: From Theory to Applications*, pages 442–464. Springer, 2012.

[5] Erik C Shallman. Up in the air: Clarifying cloud storage protections. *Intell. Prop. L. Bull.*, 19:49, 2014.

[6] Wen-Guey Tzeng. A time-bound cryptographic key assignment scheme for access control in a hierarchy. *Knowledge and Data Engineering, IEEE Transactions on*, 14(1):182–188, 2002.

[7] 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.

[8] Ravinderpal S Sandhu. Cryptographic implementation of a tree hierarchy for access control. *Information Processing Letters*, 27(2):95–98, 1988.

[9] Yan Sun and KJ Liu. Scalable hierarchical access control in secure group communications. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 1296–1306. IEEE, 2004.

[10] Mikhail J Atallah, Marina Blanton, Nelly Fazio, and Keith B Frikken. Dynamic and efficient key management for access hierarchies. *ACM Transactions on Information and System Security (TISSEC)*, 12(3):18, 2009.

[11] 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.

[12] 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.

[13] Qin Liu, Chiu C Tan, Jie Wu, and Guojun Wang. Reliable re-encryption in unreliable clouds. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pages 1–5. IEEE, 2011.

[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.

[15] 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.

[16] Steven D Galbraith, Kenneth G Paterson, and Nigel P Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[17] Gerhard Frey, Michael Müller, and Hans-Georg Rück. The tate pairing and the discrete logarithm applied to elliptic curve cryptosystems. *Information Theory, IEEE Transactions on*, 45(5):1717–1719, 1999.

[18] Florian Hess, Nigel P Smart, and Frederik Vercauteren. The eta pairing revisited. *Information Theory, IEEE Transactions on*, 52(10):4595–4602, 2006.

[19] Jean-Luc Beuchat, Jorge E González-Díaz, Shigeo Mitsunari, Eiji Okamoto, Francisco Rodríguez-Henríquez, and Tadanori Teruya. High-speed software implementation of the optimal ate pairing over barreto–naehrig curves. In *Pairing-Based Cryptography-Pairing 2010*, pages 21–39. Springer, 2010.

[20] Sanjit Chatterjee, Darrel Hankerson, and Alfred Menezes. On the efficiency and security of pairing-based protocols in the type 1 and type 4 settings. In *Arithmetic of Finite Fields*, pages 114–134. Springer, 2010.

[21] 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.

[22] Selim G Akl and Peter D Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM Transactions on Computer Systems (TOCS)*, 1(3):239–248, 1983.