Key-Aggregate Searchable Encryption with Constant-Size Trapdoors for Fine-Grained Access Control in the Cloud

Sikhar Patranabis and Debdeep Mukhopadhyay

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

Abstract. Fine-grained access control, especially in shared data environments such as the cloud, is a common scenario. Suppose a data owner encrypts and stores a collection of N documents in the cloud, and wishes to grant access to a subset $\mathcal{S}$ of these to a data user. The data user must therefore be able to perform search operations only on the subset $\mathcal{S}$ of documents, and nothing else. For example, the user may want to store the documents in separate folders pertaining to work, family, personal etc., and selectively grant search access to one or more folders among different users. This is a commonly encountered in document sharing environments such as Dropbox and Google Drive. Most existing searchable encryption schemes today assume that a user has search permissions over the entire document collection, and are hence not designed explicitly to address such a controlled-search scenario. The only previous work in this direction by Cui et al. possesses inherent security weaknesses that can be exploited to trivially break the privacy of their system.
In this paper, we provide a novel, efficient and secure solution to this problem by introducing a new public-key searchable encryption primitive referred to as the key-aggregate searchable encryption (KASE). KASE is secure under well-defined cryptographic assumptions, and requires optimal network communication between the server and the data owners/data users. KASE is the first public key searchable encryption scheme to achieve performance and security at par with the most efficient symmetric key searchable encryption constructions. Additionally, while all existing schemes produce trapdoors that grow linearly with the size of the document collection, KASE produces constant-size trapdoors that are independent of the document collection size. KASE is also efficient and scalable with respect to data updates, making it ideal for use in dynamic cloud-based search applications.

Keywords: Searchable Encryption, Access Control, Data Privacy, Trapdoor Privacy

1 Introduction

The advent of cloud computing today brings with it a host of desirable solutions to the long-standing demand for a platform that allows efficient ubiquitous sharing and agile independent access to large volumes of data owned by corporations, governments and individuals alike. Despite the many advantages that the cloud offers, there exist a plethora of security concerns surrounding possible leakages of sensitive data such as customer transactions, search histories, credit card numbers, corporation and government policies, and personally identifiable information. Consequently, most clients prefer to encrypt their own data before uploading it to cloud-based data centers. While encryption helps mitigate concerns regarding data privacy, it often makes data searching and data mining operations difficult. A trivial solution is to send all the encrypted data back to the data owner, who can then perform the desired operations by decrypting the data, and return the results to the client. In most cases, however, this is time consuming and cost-ineffective, especially if the data owner only has access to devices with limited bandwidth and storage capabilities. It also under-utilizes the computational ability of the cloud, which is usually much greater than that of individual devices. A more pragmatic solution is to build a system that allows analytics on the encrypted data itself, without the need for decryption, and without compromising the security of the system. In other words, we need cryptosystems that allow *searchable encryption*, or more simply, keyword search over encrypted document collections and databases.

Searchable Encryption. Traditionally, searchable encryption of document collections is provisioned by means of a *secure searchable index* [1, 2] - a data structure created by the data owner and stored on a remote server (such as a cloud service provider) who is not trusted. Given a keyword, the server should be able to efficiently search the index and return the list of matching document identifiers. Unfortunately, revealing the actual keyword to the server leaks sensitive information about the underlying document collection. This is tackled by embedding the information of the keyword within a *trapdoor* and providing the same to the server. The trapdoor allows the server to search without gaining any knowledge of the underlying keyword, and thus prevents the aforementioned information leakage. Trapdoor generation usually requires access to a master secret key, and only users with access to valid trapdoors are able to instruct the server to perform keyword searches. Without valid trapdoors, an external agent should not gain any information about the underlying document collection. This notion of security is usually termed as *data-privacy* of the searchable encryption system. There exists yet another notion of security, known as *trapdoor privacy* that requires that any trapdoor generated by the system should not reveal any information about the underlying keyword to the server. A *secure* searchable encryption scheme must guarantee both the above notions of security.

1.1 Related Work

The study of searchable encryption in cryptographic literature can be broadly categorized into two categories - searching on *symmetric key encrypted data* and searching on *public key encrypted data*. Both models assume the existence of an untrusted server that stores the encrypted data, and a bunch of data users that wish to access, search or modify the data without revealing any information to the server. There, however, exist subtle differences between the two, as described next.

Symmetric Key Searchable Encryption. Symmetric key searchable encryption assumes that the data owner uses a secret key of her choice to encrypt the data and build the searchable index for the same, which is then put on the server. The data owner may organize her data efficiently using data structures of her choice. Only users with whom the owner has shared the secret key can generate trapdoors for searching the encrypted data. Such schemes are usually deterministic in nature, with intensive data pre-processing operations and highly efficient search algorithms [3, 1, 4, 2]. Also, the symmetric setting assumes that the adversary does not have access to an encryption oracle, making it inherently simple to achieve strong data and trapdoor privacy guarantees [2, 5].

Public Key Searchable Encryption. Public key searchable encryption, on the other hand, assumes the presence of a central agent or a trusted third party that is responsible for setting up the system and publishing a public key. The central agent is the only one to hold the master secret key. Any data owner can use the public key to encrypt her data and build the corresponding searchable index to be put on the server. Any user willing to search for a keyword must submit the same to the central agent and receive the corresponding trapdoor, which is then given to the server for performing the search operation. The original work on public-key searchable encryption by Boneh et al. [6] assumes that the user's *access pattern* (which documents contain a keyword) is revealed to the adversary. A stronger security notion that hides even the access pattern was introduced in [7]. However, their construction suffers from a search overhead that is proportional to the square-root of the data size, and is hence too slow for practical purposes. A more efficient version of public-key searchable encryption was introduced by Bellare et al. in [8]. Although their construction achieves optimal search times, it is an inherently deterministic scheme achieving weaker security guarantees than desirable, especially in shared environments such as the cloud. While the public key setting allows for data privacy guarantees, the fact that the adversary has access to an encryption oracle makes trapdoor privacy hard to model [9].

1.2 The Need for Controlled Access in Data-Sharing Environments

Fina-grained access control, especially in shared data environments such as the cloud, is a common scenario. Suppose a data owner encrypts and stores a collection of N documents in the cloud, and wishes to grant access to a subset $\mathcal{S}$ of these to a data user. The data user must therefore be able to perform search operations only on the subset $\mathcal{S}$ of documents, and nothing else. This is a commonly encountered in document sharing environments such as Dropbox and Google Drive, where a large number of data users with different access

Table 1: A Per-Query Performance Comparison of KASE with Symmetric Searchable Encryption Schemes. N denotes the number of documents in the collection, Δ is the total number of keywords in the document collection and Δ_{avg} is the average number of keywords per document. We omit the security parameter for simplicity.

Properties	[3, 1, 4, 2]	[2]	[5]	KASE (Unoptimized)	KASE (Optimized)
Server Computation	$\mathcal{O}(N)$	$\mathcal{O}(N)$	$\mathcal{O}(N)$	$\mathcal{O}(N)$	$\mathcal{O}(N)$
Server Storage	$\mathcal{O}(N \cdot \Delta)$	$\mathcal{O}(N \cdot \Delta)$	$\mathcal{O}(N \cdot \Delta_{\text{avg}})$	$\mathcal{O}(N \cdot \Delta)$	$\mathcal{O}(N \cdot \Delta_{\text{avg}})$
Number of Rounds	1	1	1	1	1
Communication	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Controlled-Access	No	No	No	Yes	Yes
Trapdoor Overhead	$\mathcal{O}(N)$	$\mathcal{O}(N)$	$\mathcal{O}(N)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Adaptive Data Privacy	No	Yes	Yes	Yes	Yes
Adaptive Trapdoor Privacy	No	Yes	Yes	Yes	Yes

permissions might want to search on a single owner's document collection. The dynamic growth in the number of data owners and data users on the cloud, as well as the ever-increasing amount of data stored online and changes to access permissions, are issues to be tackled efficiently and in a secure manner. Most existing searchable encryption schemes in the current literature assume that a user has search capabilities over the entire document collection, and are hence not necessarily the most efficient solutions when adapted to controlled-search scenarios. In particular, incorporating different access permissions for each document within a single searchable index, and generating controlled-search trapdoors that restrict search capabilities to a specific subset of documents in the collection, is non-trivial and challenging. Recent attempts in this direction include the work of Cui et al. [10]. However, the definitions and constructions presented in [10] are rather shallow and often do not conform to the standard norms of searchable encryption. Additionally, the authors of [10] do not provide any security framework in which their construction may be proven to be secure; nor do they present a clear description of the adversarial model assumed. Finally, their construction uses trapdoors that are *malleable* - a feature can be exploited by an adversary to trivially break the data privacy of the system. In this paper, we attempt to address these drawbacks by describing concrete cryptographic primitives for controlled-access searchable encryption with a well-defined security framework.

1.3 Our Contributions

In this paper, we provide a novel, efficient and secure solution to the problem of controlled-access in searchable encryption for data-sharing environments such the cloud. We introduce a new public-key searchable encryption primitive known as the Key-Aggregate Searchable Encryption (KASE). KASE allows a data owner to generate a controlled-search trapdoor that can restrict the search capabilities of a data user to a specific subset of documents in the collection. The detailed contributions of the paper are as follows:

– We present the key aggregate searchable encryption (KASE) - a public-key searchable encryption system for controlled-access environments. The KASE system involves three parties - a central system owner who is in charge of setting up the system, a data owner who builds a searchable index for her document collection and stores it on the server, and a data user who submits keywords to the system owner and receives the corresponding trapdoor for performing the search. KASE assumes an honest but curious server setting, which a popular model for modeling cloud environments [2, 3]. The salient feature of KASE is its controlled-access trapdoor, which can be used by a user to search only over the specific subset of documents she has access permissions to.

– We present separate data privacy and trapdoor privacy definitions for KASE. Our data privacy definitions are based on computational indistinguishability and assume poly-time adaptive adversaries with access to encryption and trapdoor oracles. Our trapdoor privacy definitions, on the other hand, are based on statistical indistinguishability and assume computationally unbounded adversaries.

– We present a concrete construction for KASE that satisfies our security definitions while achieving the desired controlled-search features. Our construction achieves searches in one communication round, and requires a $\mathcal{O}(N)$ amount of work from the server, where N is the number of documents. In addition, it requires $\mathcal{O}(1)$ storage on both the central system owner as well as the data users, and a $\mathcal{O}(N \cdot \varDelta)$ storage on the server, where $\varDelta$ is the total number of keywords in the document collection. Note that this is competitive with some of the most efficient symmetric searchable encryption systems in literature [3, 1, 4, 2]. Finally and most importantly, the trapdoor overhead of our construction is constant and independent of the size of the document subset it provides access to, which is asymptotically better than any searchable encryption construction in the literature.

– We finally present some optimizations to our KASE construction that reduce the storage requirements on the server even further to $\mathcal{O}(N \cdot \varDelta_{\text{avg}})$, where $\varDelta_{\text{avg}}$ is the average number of keywords per document. Note that this is optimal, and is competitive with the most efficient symmetric searchable encryption construction in the literature [5]. It also reduces the search work of the server to linear only in the number of documents that actually contain a keyword, which again is optimal. Additionally, the optimized version of KASE is both efficient and scalable with respect to modifications to the underlying document collection.

Table 1 compares our KASE constructions with the most efficient searchable encryption schemes in the symmetric key setting. Quite evidently, our construction is competitive with all existing constructions, with the added advantage of controlled-access trapdoors that have constant overhead.

2 Notations and Preliminaries

This section presents notations, preliminary definitions and background material on the cryptographic tools used in the rest of the paper.

2.1 Notations Used

We write $x \xleftarrow{R} \chi$ to represent that an element x is sampled uniformly at random from a set $\mathcal{X}$. The output a of a deterministic algorithm $\mathcal{A}$ is denoted by $x \leftarrow \mathcal{A}$ and the output a' of a randomized algorithm $\mathcal{A}'$ is denoted by $x' \xleftarrow{R} \mathcal{A}'$. We refer to $\lambda \in \mathbb{N}$ as the security parameter, and denote by $\mathsf{exp}(\lambda)$, $\mathsf{poly}(\lambda)$ and $\mathsf{negl}(\lambda)$ any generic (unspecified) exponential function, polynomial function and negligible function in λ respectively. Note that a function $f : \mathbb{N} \to \mathbb{N}$ is said to be negligible in λ if for every positive polynomial p, $f(\lambda) < 1/p(\lambda)$ when λ is sufficiently large. We denote by $\mathsf{Func}(n, m)$ the space of all functions from $\{0,1\}^n$ to $\{0,1\}^m$. We denote by $x_1||x_2$ the concatenation of strings x_1 and x_2. Finally, for $a, b \in \mathbb{Z}$ such that $a \leq b$, we denote by $[a, b]$ the set of integers lying between a and b (both inclusive).

2.2 Preliminary Definitions and Overview of Cryptographic Tools

Document Collection. Let $\boldsymbol{\varDelta} = (w_1, \cdots, w_k)$ be a dictionary of k =$\mathsf{poly}(\lambda)$ keywords, chosen from a keyword space $\mathcal{W}$ and arranged lexicographically, where $\mathcal{W}$ contains all strings of length n =$\mathsf{poly}(\lambda)$. Let $2^{\boldsymbol{\varDelta}}$ denote the space of all documents that contain words from $\boldsymbol{\varDelta}$. Also, let $\mathcal{ID}$ be a document identifier space containing all strings of length m =$\mathsf{poly}(\lambda)$. We denote by $\mathbf{D}$ a document collection $(D_1, \cdots, D_N) \subset 2^{\boldsymbol{\varDelta}}$ such that N =$\mathsf{poly}(\lambda)$, where each document D_j for $j \in [1, N]$ is associated with a unique identifier $id_j \in \mathcal{ID}$ and contains at most L =$\mathsf{poly}(\lambda)$ many keywords. We denote by $\mathcal{ID}_{\mathbf{D}}$ the set of identities of all documents in $\mathbf{D}$ and by $\boldsymbol{\varDelta}(\mathbf{D})$ the set of all keywords across all documents in $\mathbf{D}$. Finally, for any set $\mathcal{S} \subset \mathcal{ID}_{\mathbf{D}}$, we denote by $\delta_{w,\mathcal{S}}(\mathbf{D})$ the lexicographically ordered list of document identifiers in $\mathcal{S}$ that contain the keyword w.

Bilinear Maps. Let $\mathbb{G}$ and $\mathbb{G}_T$ be two multiplicative cyclic groups of prime order q, and let g be a generator for $\mathbb{G}$. A map $e : \mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G}_T$ is said to be *bilinear* has the following properties:

1. **Bilinearity:** $e\left(g^a, g^b\right) = e\left(g, g\right)^{ab}$ for all $a, b \in \mathbb{Z}_q$
2. **Non-degeneracy:** $e(g, g) \neq 1$

A group $\mathbb{G}$ is said to be a *bilinear group* [11] if the group operations in $\mathbb{G}$ can be computed efficiently, and there exists a corresponding group $\mathbb{G}_T$ along with an efficiently computable bilinear map $e : \mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G}_T$ as described above. In addition, there should exist an efficient algorithm $GroupGen(1^\lambda)$ that takes as input the security parameter λ, and outputs the tuple $(\mathbb{G}, \mathbb{G}_T, e, \mathcal{H})$ such that $\mathbb{G}$ and $\mathbb{G}_T$ are groups of prime order q (q being a λ bit), and $e : \mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G}_T$ is a bilinear map as described above.

We note here that efficient instantiations of bilinear maps over elliptic curves have been studied widely in the cryptographic literature. We now introduce a hardness assumption based on bilinear maps that we use to prove the security of our proposed construction in this paper.

The Decision $(\mathscr{S}, M)$-BDHES Problem. Let $\mathscr{S} \subset \mathbb{Z}$ and $M \in \mathbb{Z} \setminus (\mathscr{S} + \mathscr{S})$. Set $\alpha \xleftarrow{R} \mathbb{Z}_q$ and $b \xleftarrow{R} \{0, 1\}$. Set $Z_0 \leftarrow e(g, g)^{\alpha^M}$ and $Z_1 \xleftarrow{R} \mathbb{G}_T$. The decision $(\mathscr{S}, M)$-bilinear decision Diffie Hellman exponent sum (BDHES) problem is as follows. Given

$$(\{g^{\alpha^i}\}_{i \in \mathscr{S}}, Z_b)$$

guess b. *The decision $(\mathscr{S}, M)$-BDHES assumption states that for any probabilistic polynomial time algorithm $\mathcal{A}$, its advantage in solving the decision $(\mathscr{S}, M)$-BDHES problem is negligible in the security parameter λ.*

The decision $(\mathscr{S}, M)$-BDHES assumption was introduced in [12] for proving the security of adaptively secure broadcast encryption systems. While the BDHES Sum problem is not exactly a so-called *standard* hard problem, it is narrower than the generalized BDH exponent assumption defined by Boneh et al. in [13] and is fairly intuitive. For a more detailed discussion on the security of this assumption, refer Appendix B.

Min-Entropy. The min-entropy of a random variable Y is defined as $\mathbf{H}_\infty(Y) = -\log(\max_y \Pr[Y = y])$. A random variable Y is said to be a k-source if $\mathbf{H}_\infty(Y) \geq k$. A (T, k)-block-source is a random variable $\mathbf{Y} = (Y_1, \cdots, Y_T)$ where for each $i \in [1, T]$ and $y_1, \cdots, y_{i-1}$, it holds that $\mathbf{H}_\infty(Y_i | Y_1 = y_1, \cdots, Y_{i-1} = y_{i-1}) \geq k$.

Statistical Distance. The statistical distance between two random variables Y_1 and Y_2 over a finite domain Ω is defined as $\mathrm{SD}(Y_1, Y_2) = \frac{1}{2}\sum_{\omega \in \Omega} |\Pr[Y_1 = \omega] - \Pr[Y_2 = \omega]|$. Two random variables Y_1 and Y_2 are *statistically δ-close* if their statistical distance $\mathrm{SD}(Y_1, Y_2) \leq \delta$. Two distribution ensembles $\{Y_{1,\lambda}\}$ and $\{Y_{2,\lambda}\}$ for $\lambda \in \mathbb{N}$ are said to be *statistically indistinguishable* if it holds that $\mathrm{SD}(Y_{1,\lambda}, Y_{2,\lambda}) = \mathsf{negl}(\lambda)$. Finally, two distribution ensembles $\{Y_{1,\lambda}\}$ and $\{Y_{2,\lambda}\}$ for $\lambda \in \mathbb{N}$ are said to be *computationally indistinguishable* if for every probabilistic polynomial time (PPT) algorithm $\mathcal{A}$, it holds that:

$$\left|\Pr[\mathcal{A}\left(1^\lambda, y_1\right)] - \Pr[\mathcal{A}\left(1^\lambda, y_2\right)]\right| = \mathsf{negl}(\lambda)$$

where $y_1 \leftarrow Y_1$ and $y_2 \leftarrow Y_2$.

Universal Collection of Hash Functions. A collection $\mathscr{H}$ of functions $\mathcal{H} : \mathcal{X} \longrightarrow \mathcal{Y}$ is said to be *universal* if for any $x_1, x_2 \in \mathcal{X}$ such that $x_1 \neq x_2$, and any $\mathcal{H} \xleftarrow{R} \mathscr{H}$, it holds that $\Pr[\mathcal{H}(x_1) = \mathcal{H}(x_2)] = 1/|\mathcal{Y}|$.

Lemma 2.1 (Leftover Hash Lemma for Block Sources)[14–17]. *Let $\mathscr{H}$ be a universal collection of hash functions, and let $\mathbf{Y} = (Y_1, \cdots, Y_T)$ be an (T, k)-block source such that $k \geq \log|\mathcal{Y}| + 2\log(1/\epsilon) + \Theta(1)$ for any arbitrary positive constant ϵ. Then the ensemble distribution $(\mathcal{H}, \mathcal{H}(Y_1), \cdots, \mathcal{H}(Y_T))$, where $\mathcal{H} \xleftarrow{R} \mathscr{H}$, is statistically ϵT-close to the uniform distribution over $\mathscr{H} \times (\mathcal{Y})^T$.*

3 Our Proposition: Key-Aggregate Searchable Encryption

This section introduces the background and definitions for our proposed key-aggregate searchable encryption (KASE). We assume that the data owner wishes to encrypt and store a searchable index for such a document collection $\mathbf{D} = (D_1, \cdots, D_N)$, with identity space $\mathcal{ID}_{\mathbf{D}}$, remotely on an honest-but-curious server (e.g. a cloud server providing storage-as-a-service). The server has the ability to make searches on $\mathbf{D}$ using a trapdoor and return the indexes of the documents containing the corresponding keyword. We also assume that a controlled search scenario, where a user may be allowed to access and search over only a poly-size subset $\mathcal{S} \subset \mathcal{ID}_{\mathbf{D}}$ of document identities. Obviously, we have $|\mathcal{S}| \leq N$. Finally, we assume that the whole searchable encryption system is hosted by a central agent, who is a trusted third party in charge of generating the secret keys and trapdoors (the notion of such a trusted third party is common in several public key encryption systems such as identity based encryption [11]).

Definition 3.1 (Key Aggregate Searchable Encryption(KASE)). *A KASE system is an ensemble of the following algorithms:*

$\underline{\mathsf{SetUp}(1^\lambda, \mathcal{ID}, N)}$: *A randomized algorithm executed by the central agent. It takes as input the security parameter λ, the document identity space $\mathcal{ID}$ and the number of documents N in the collection to the encrypted. It outputs a set of public parameters $params$.*

$\underline{\mathsf{KeyGen}(params)}$: *A randomized algorithm executed by the central agent. It takes as input the public parameters $params$. It outputs a master secret key msk and a public key PK.*

$\underline{\mathsf{BuildIndex}(PK, \mathbf{D})}$: *A randomized algorithm executed by the document collection owner. It takes as input the public key PK and the document collection $\mathbf{D}$. It outputs the the searchable index $\mathbf{I}$ to be stored on the server.*

$\underline{\mathsf{GenTrpdr}(msk, w, \mathcal{S})}$: *A randomized algorithm executed by the central agent. It takes as input the master secret key msk, the keyword w and a poly-size subset $\mathcal{S} \subset \mathcal{ID}_{\mathbf{D}}$ over which the keyword is to be searched. It outputs the controlled-search trapdoor $\mathbf{T}_{w,\mathcal{S}}$.*

$\underline{\mathsf{Search}(\mathbf{I}, \mathbf{T}_{w,\mathcal{S}}, \mathcal{S})}$: *A deterministic algorithm executed by the server. Takes as input the searchable index $\mathbf{I}$ and the controlled-search trapdoor $\mathbf{T}_{w,\mathcal{S}}$ along with the subset $\mathcal{S}$ of document identifiers. It outputs a list $\mathbf{L}$ of lexicographically ordered document identifiers.*

Correctness. A KASE system is said to be correct if for all λ, for all document collections $\mathbf{D} \subset 2^{\mathbf{\Delta}}$ with N documents over the identity space $\mathcal{ID}$, for all subsets $\mathcal{S} \subset \mathcal{I}_{\mathbf{D}}$ and for all $params \xleftarrow{R} \mathsf{SetUp}(1^\lambda, \mathcal{ID}, N)$, $(msk, PK) \xleftarrow{R} \mathsf{KeyGen}(params)$, $\mathbf{I} \xleftarrow{R} \mathsf{BuildIndex}(PK, \mathbf{D})$, $\mathbf{T}_{w,\mathcal{S}} \xleftarrow{R} \mathsf{GenTrpdr}(msk, w, \mathcal{S})$ and $\mathbf{L} \leftarrow \mathsf{Search}(\mathbf{I}, \mathbf{T}_{w,\mathcal{S}})$, we have $\mathbf{L} = \delta_{w,\mathcal{S}}(\mathbf{D})$ (the lexicographically ordered list of identifiers in $\mathcal{S}$ corresponding to the documents that contain the keyword w).

3.1 Security Definitions for KASE

This section presents the security definitions for key-aggregate searchable encryption scheme. We begin with an informal description of the adversarial model and the security requirements, and then present the formal definitions using indistinguishability based frameworks.

The Adversarial Model. The adversarial model for a key-aggregate searchable encryption scheme is assumed to be as follows:

– The adversary has access to an encryption oracle, that is, it is able to create an encrypted searchable index on any polynomial size document collection of its choice.
– The adversary has access to a trapdoor generation oracle, that is, it can submit keyword queries and obtain trapdoors for the same over subsets of its choice.
– The adversary is able to search on any encrypted searchable index using the trapdoors generated by the trapdoor oracle.

Based on whether an adversary is restricted to make all its trapdoor queries in a *single pass* or in *multiple passes* based on the outcomes of the previous queries, it is classified as either *non-adaptive* or *adaptive.* Quite evidently, the adaptive adversarial model is intuitively stronger and bears maximum resemblance to real world scenarios.

Overview of the Security Requirements for KASE. Security definitions for searchable encryption traditionally require that an adversary learns nothing beyond the *access pattern*, that is, the document identifiers corresponding to a keyword search operation. Constructions with deterministically generated trapdoors, especially on the symmetric setting, further assume that the *search pattern* is also leaked [2], that is, the adversary is able to identify from two search outcomes if the same keyword has been queried for in both of them. In our definitions for KASE, however, we consider randomized trapdoor generation and hence, in the strongest security notion, the search pattern should not be considered as a trivial leakage. Thus, informally, our security definitions for KASE are intended to guarantee the following:

1. **Data Privacy:** Given an encrypted searchable index $\mathbf{I}$ corresponding to a document collection $\mathbf{D}$, an adaptive adversary making at most polynomial number of queries to the encryption and trapdoor oracles should learn nothing about $\mathbf{D}$ except for the search patterns corresponding to the trapdoors generated by the oracle.
2. **Trapdoor Privacy:** Given a trapdoor $\mathbf{T}_{w,\mathcal{S}}$ corresponding to some keyword w and a subset $\mathcal{S} \subset \mathcal{ID}_{\mathbf{D}}$, an adaptive adversary making at most polynomial number of queries to the encryption and trapdoor oracles should learn nothing about w.

Traditional security definitions for public-key searchable encryption such as the chosen keyword attack framework introduced by Goh in [1] take into account the former requirement but not the latter. Indeed, achieving trapdoor-private public-key searchable encryption schemes had been an open problem for quite some time. We point out here that trapdoor privacy is easier to achieve in the symmetric key setting, where the adversary does not have access to an encryption oracle. This is clearly reflected in the simplicity of trapdoor privacy definitions presented in [3, 1, 4, 2], where the adversary is merely restricted from making trapdoor queries corresponding to keywords that trivially distinguish the challenge document collection pair. Such definitions are not analogously applicable in the context of public-key searchable encryption, since an adversary could easily encrypt specially crafted document collections of its choice (different from the challenge document collections), and trivially break trapdoor privacy by using the generated trapdoors to run searches on these document collections. Stronger definitions are therefore necessary to guarantee trapdoor privacy in the public key setting. The first such definition was put forth by Boneh, Raghunathan and Segev in [9]. Their definition is based on statistical indistinguishability properties of the trapdoor. More specifically, given a trapdoor $\mathbf{T}_w$ corresponding to a keyword w, an adversary with access to the encryption oracle cannot distinguish with non-negligible advantage if w was generated from a statistical distribution of its choice, or from a uniformly random distribution. This is, to date, the strongest trapdoor privacy definition to be concretely realized in the context of public-key searchable encryption. Hence, we use the notion of statistical indistinguishability when presenting the trapdoor privacy definitions for KASE.

Formal Security Definitions for KASE. Our next task is to present formal definitions that capture the above security notions for KASE. We begin by introducing some auxiliary definitions:

Definition 3.2 (Search History). *A search history is a tuple of the form* $\mathcal{H} = \big(\mathbf{D}, \{(w_q, \mathcal{S}_q)\}_{q\in[1,Q]}\big)$, *where* $\mathbf{D}$ *is a poly-size document collection and each tuple* $(w_q, \mathcal{S}_q)$ *represents a trapdoor oracle query for a keyword* w_q *over a subset* $\mathcal{S}_q$ *of document identifiers.*

Definition 3.3 (Access Pattern). *Given a search history* $\mathcal{H} = \big(\mathbf{D}, \{(w_q, \mathcal{S}_q)\}_{q\in[1,Q]}\big)$, *the access pattern is defined as* $\tau(\mathcal{H}) = \{\delta_{w_q,\mathcal{S}_q}(\mathbf{D})\}_{q\in[1,Q]}$. *Note that the* $\tau(\mathcal{H})$ *can be easily deduced by running searches on an encrypted index* $\mathbf{I} \xleftarrow{R} \mathsf{BuildIndex}(PK, \mathbf{D})$ *using the collection of trapdoors* $\{\mathbf{T}_{w_q,\mathcal{S}_q} \xleftarrow{R} \mathsf{GenTrpdr}(msk, w_q, \mathcal{S}_q)\}_{q\in[1,Q]}$.

We are now in a position to formally present the security definitions for KASE. The definitions for data privacy and trapdoor privacy are presented separately. We assume *adaptive* adversaries that can issue trapdoor queries in multiple passes, and can base subsequent queries upon the responses to previous queries.

Definition 3.4 (Adaptive Data Privacy of a KASE system). *Let* λ *be the security parameter and let* $\mathcal{A}$ *be a probabilistic poly-time (PPT) algorithm that receives* λ, *a document identifier space* $\mathcal{ID}$ *and a number of documents* $N = \mathsf{poly}(\lambda)$ *as input.* $\mathcal{A}$ *plays the following game with a challenger:*

SetUp Phase. *The challenger generates params* $\xleftarrow{R}\mathsf{SetUp}(1^\lambda,\mathcal{ID},N)$ *and* $(msk,PK)\xleftarrow{R}\mathsf{KeyGen}(params)$. *It provides params and* PK *to* $\mathcal{A}$.

Trapdoor Query Phase-1. $\mathcal{A}$ *adaptively issues trapdoor queries of the form* $(w_q,\mathcal{S}_q)$ *for* $q\in[1,Q_1]$ *and* $\mathcal{S}_q\subset\mathcal{ID}$. *The challenger responds to each query with* $\mathbf{T}_q\xleftarrow{R}\mathsf{GenTrpdr}(msk,w_q,\mathcal{S}_q)$.

Challenge Phase. $\mathcal{A}$ *provides the challenger with two document collections* $\mathbf{D}_0=(D_{1,0},\cdots,D_{N,0})$ *and* $\mathbf{D}_1=(D_{1,1},\cdots,D_{N,1})$ *subject to the restriction that:*

$$\tau\left(\mathbf{D}_0,\{(w_q,\mathcal{S}_q)\}_{q\in[1,Q_1]}\right)=\tau\left(\mathbf{D}_1,\{(w_q,\mathcal{S}_q)\}_{q\in[1,Q_1]}\right)$$

The challenger picks $b\xleftarrow{R}\{0,1\}$ *and responds with* $\mathbf{I}^*\xleftarrow{R}\mathsf{BuildIndex}(PK,\mathbf{D}_b)$.

Trapdoor Query Phase-2. $\mathcal{A}$ *continues to adaptively issue trapdoor queries of the form* $(w_q,\mathcal{S}_q)$ *for* $q\in[Q_1,Q]$, *once again subject to the restriction that:*

$$\tau\left(\mathbf{D}_0,\{(w_q,\mathcal{S}_q)\}_{q\in[1,Q]}\right)=\tau\left(\mathbf{D}_1,\{(w_q,\mathcal{S}_q)\}_{q\in[1,Q]}\right)$$

The challenger responds as in Trapdoor Query Phase-1.

Guess Phase. $\mathcal{A}$ *outputs a guess* b' *for the random bit* b *chosen by the challenger.*

We say that a KASE construction is adaptively data private if for all $Q=\mathsf{poly}(\lambda)$ *and for all PPT algorithms* $\mathcal{A}$,

$$|Pr[b'=b]-Pr[b'\neq b]|\leq \mathsf{negl}(\lambda)$$

Note that the condition $\tau\left(\mathbf{D}_0,\{(w_q,\mathcal{S}_q)\}_{q\in[1,Q]}\right)=\tau\left(\mathbf{D}_1,\{(w_q,\mathcal{S}_q)\}_{q\in[1,Q]}\right)$ essentially models the fact that the search pattern is a trivial leakage and is not a cryptographic vulnerability of any KASE system; indeed, any searchable encryption scheme is expected to leak the search pattern. Consequently, the same should not be used to distinguish the document collection in the data privacy game.

Next, we present the trapdoor privacy definitions for KASE. These definitions are natural extensions of those presented in [9] and are presented here for completeness.

Definition 3.5 (Trapdoor Privacy of a KASE system). *Let* λ *be the security parameter, and let* T *and* k *be variables that can be computed as functions of* λ. *Let* $\mathcal{A}$ *be a computationally unbounded algorithm that receives* λ, *a document identifier space* $\mathcal{ID}$, *a number of documents* $N=\mathsf{poly}(\lambda)$, T *and* k *as input.* $\mathcal{A}$ *plays the following game with a challenger:*

SetUp Phase. *The challenger generates params* $\xleftarrow{R}\mathsf{SetUp}(1^\lambda,\mathcal{ID},N)$ *and* $(msk,PK)\xleftarrow{R}\mathsf{KeyGen}(params)$. *It provides params and* PK *to* $\mathcal{A}$. *The challenger also picks* $b\xleftarrow{R}\{0,1\}$.

Real-or-Random Queries. *At any point,* $\mathcal{A}$ *may generate* $\mathbf{W}=(W_1,\cdots,W_T)$ *and a poly-size subset* $\mathcal{S}\subset\mathcal{ID}$, *such that* $\mathbf{W}$ *is a circuit description of a* (T,k)*-block source over the space* $\mathcal{W}$ *of all keywords, and query the challenger with* $(\mathbf{W},\mathcal{S})$. *The challenger does the following:*

- *If* $b=0$, *it samples* $(w_1,\cdots,w_T)\xleftarrow{R}\mathbf{W}$.
- *If* $b=1$, *it samples* $(w_1,\cdots,w_T)\xleftarrow{R}\mathcal{W}^T$.

The challenger responds with $\{\mathbf{T}_t\xleftarrow{R}\mathsf{GenTrpdr}(msk,w_t,\mathcal{S})\}_{t\in[1,T]}$.

Trapdoor Queries. *At any point,* $\mathcal{A}$ *may issue a trapdoor query of the form* $(w,\mathcal{S})$ *for a poly-size subset* $\mathcal{S}\subset\mathcal{ID}$. *The challenger responds to each query with* $\mathbf{T}\xleftarrow{R}\mathsf{GenTrpdr}(msk,w,\mathcal{S})$.

Note that the both kinds of queries may be made by $\mathcal{A}$ *at any time and do not necessarily follow any specific order.*

Guess Phase. *$\mathcal{A}$ outputs a guess b' for the random bit b chosen by the challenger.*

We say that a KASE construction is statistically (T, k)-source trapdoor private if for all computationally unbounded algorithms $\mathcal{A}$ that make at most polynomially many (T, k)-block source real-or-random queries and at most polynomially many trapdoor queries,

$$|Pr[b' = b] - Pr[b' \neq b]| \leq \mathsf{negl}(\lambda)$$

Malleability of Trapdoors. We point out here that malleability of trapdoors leads to a straightforward attack on the data privacy of a KASE system. Given trapdoor $\mathbf{T}_q$ in response to a query $(w_q, \mathcal{S}_q)$ that does not trivially distinguish the document collections $\mathbf{D}_0$ and $\mathbf{D}_1$, the adversary could exploit the malleability property to create a new trapdoor $\mathbf{T}'_q$ corresponding to a different query $\left(w'_q, \mathcal{S}'_q\right)$ such that

$$\tau\left(\mathbf{D}_0, \left(w'_q, \mathcal{S}'_q\right)\right) \neq \tau\left(\mathbf{D}_1, \left(w'_q, \mathcal{S}'_q\right)\right)$$

This in turn trivially breaks the indistinguishability-based data privacy of the system.

Security Flaw in the Construction of [10]. The construction for controlled-search proposed in [10] by Cui et al. uses trapdoors of the form $\mathbf{T} = \mathcal{K}_{\mathcal{S}} \cdot \mathcal{H}(w)$, where $\mathcal{K}_{\mathcal{S}}$ is the controlled-access term independent of w and $\mathcal{H}$ is a hash function. Such a trapdoor is clearly malleable with respect to the keyword w, and can be used by an adversary to break the data privacy of their construction. In addition, the trapdoor generation process for the construction in [10] is *deterministic*, implying that their construction leaks the search pattern in addition to the access pattern. Finally, the fact that the trapdoors are deterministic trivially rules out the possibility of statistical trapdoor privacy.

4 Our Construction for Key-Aggregate Searchable Encryption

In this section, we propose a concrete construction for KASE. Recall that $\mathbf{D}$ refers to a document collection $(D_1, \cdots, D_N) \subset 2^{\boldsymbol{\Delta}}$ such that $N = \mathsf{poly}(\lambda)$, where each document D_j for $j \in [1, N]$ is associated with a unique identifier $id_j \in \mathcal{ID}$ and contains at most $L = \mathsf{poly}(\lambda)$ many keywords. Also, $\mathcal{ID}_{\mathbf{D}}$ denotes the set of identities of all documents in $\mathbf{D}$ and $\boldsymbol{\Delta}(\mathbf{D})$ denotes the set of all keywords across all documents in $\mathbf{D}$. Finally, for any set $\mathcal{S} \subset \mathcal{ID}$, $\delta_{w,\mathcal{S}}(\mathbf{D})$ denotes the lexicographically ordered list of document identifiers in $\mathcal{S} \cap \mathcal{ID}_{\mathbf{D}}$ that contain the keyword w. We also assume without loss of generality that the maximum size of a document id-subset $\mathcal{S}$ for which a user requests a trapdoor is N, since N is the total number of documents in the collection $\mathbf{D}$.

Overview of Our Construction. We start by giving an informal overview of our KASE construction. The construction has three main phases - system setup, searchable index creation and keyword search, as described below.

1. **System Setup.** The central agent, who is a trusted third party, first sets up the KASE system by generating the public parameters *params* depending on the security parameter λ, the document-identity space $\mathcal{ID}$, and the maximum number of documents N in a collection. This is followed by the generation of the master secret key msk and the public key PK.

2. **Searchable Index Creation.** A data owner can use the public key PK to create an encrypted searchable index $\mathbf{I}$ for her document collection $\mathbf{D}$, and store it online on a remote server. The searchable index $\mathbf{I}$ is implemented in our construction as a two-dimensional look-up table with the following properties:
   - Each row of the look-up table $\mathbf{I}$ corresponds to a keyword, while each column corresponds to a document in $\mathbf{D}$.
   - The row-index for a keyword $w_i \in \mathcal{W}$ is computed by applying a collision-resistant hash function $\mathcal{H}_1$ on w_i.

  - The column-index for a document with identifier $id_j \in \mathcal{ID}$ is similarly computed by applying a second collision-resistant hash function $\mathcal{H}_2$ on id_j.
  - Each entry $\mathbf{I}[\mathcal{H}_1(w_i)][\mathcal{H}_2(id_j)]$ encrypts a simple *yes/no* message for whether the document D_j contains the keyword w or not.

   The purpose of storing the *no* entries is to prevent a trivial leakage of the frequency distribution of different keywords from the server.

3. **Keyword Search.** A data user who wants to retrieve the documents that contain a keyword w submits a request for the corresponding trapdoor to the central agent. KASE models a controlled-access scenario where each user is allowed access to a specific subset $\mathcal{S} \subset \mathcal{ID}$ of document identifiers. The central agent enforces this access-control policy by providing the data user with the trapdoor $\mathbf{T}_{w,\mathcal{S}}$, which allows the data user to search only for documents with identities in the set $\mathcal{S}$. The data user submits this trapdoor to the online server. The trapdoor $\mathbf{T}_{w,\mathcal{S}}$ allows the server to:
   - identify the row number $\mathcal{H}_1(w)$ in $\mathbf{I}$ that corresponds to the keyword w
   - to decrypt the entries in the row $\mathbf{I}[\mathcal{H}_1(w)][\cdot]$ corresponding to the document identifiers in $\mathcal{S}$ and test for *yes/no*

   The server finally returns the list of document identifiers for which the table entries return *yes* upon decryption.

Efficient Storage and Access of the Searchable Index. We point out here that although the look-up table realizing the searchable index $\mathbf{I}$ is theoretically an exponential RAM, it only needs to store polynomially many valid entries. Hence, it may be implemented as a sparse table using indirect addressing. Similar implementations for efficient storage of sparse tables have been discussed in [2]. We briefly summarize the indirect addressing mechanism here for completeness. The idea is to maintain a list $\mathcal{L}$ of tuples of the form $\langle$`addr`, `val`$\rangle$ corresponding to the valid entries in $\mathbf{I}$, where the `addr` entry stores the row and column indices, and the `val` entry stores the actual content. The original row and column indices thus essentially represent *virtual addresses* in the table $\mathbf{I}$, and a mapping to the *actual address* in the list $\mathcal{L}$ exists only for the virtual addresses with valid entries. When a virtual address with a valid entry is looked-up, the system translates it to the actual address to locate the matching tuple in $\mathcal{L}$, and returns the corresponding `val` entry. On the other hand, when a virtual address with an invalid entry is looked-up, the system fails to map it to an actual address, and returns a random string.

Quite evidently, the virtual address space for the searchable index $\mathbf{I}$ in our KASE construction is $\boldsymbol{\Delta} \times \mathcal{ID}$, which is exponential in the security parameter λ. The actual address space, on the other hand, is $\boldsymbol{\Delta}(\mathbf{D}) \times N$, since $\mathbf{I}$ only needs to store one *yes/no* entry for each document-keyword pair in the document collection $\mathbf{D}$. Thus, using indirect addressing, $\mathbf{I}$ may be stored efficiently. In particular, the authors of [2] advocate the use of a special data structure called an FKS dictionary [18] that allows such efficient storage and look-up of sparse tables.

The Trapdoor Overhead. A major advantage of our construction is that the overhead for the trapdoor $\mathbf{T}_{w,\mathcal{S}}$ is *independent of the size of the document collection* $\mathbf{D}$ *as well as the size of the subset* $\mathcal{S}$ *that it allows access to.* All prior searchable encryption schemes [3, 1, 4, 2, 5] have trapdoor overhead proportional to the size of the document collection. The trapdoor $\mathbf{T}_{w,\mathcal{S}}$ in our construction consists of exactly two components:

- The first component allows the server to locate the row index corresponding to the keyword w
- The second component allows the server to decrypt the yes/no entries in the corresponding row for all the document identities in $\mathcal{S}$.

Of these, achieving a constant overhead for the second component is non-trivial, since it requires aggregating the access permissions to several document ids (polynomially many in our construction). The approach used in prior constructions [2, 5] was to create a separate trapdoor component for each document id to be searched. Our approach, on the other hand, is to multiplicatively combine the trapdoor components for each document id in the subset $\mathcal{S}$ into a single component that can be used precisely the ids in $\mathcal{S}$ and nothing else. To this end, we use the group structure underlying bilinear pairings.

4.1 The Detailed Construction

We now present the details of our first construction for KASE using bilinear maps. Let $GroupGen(1^\lambda)$ be an algorithm that takes as input the security parameter λ, and outputs the tuple $(\mathbb{G}, \mathbb{G}_T, e, \mathcal{H})$ such that $\mathbb{G}$ and $\mathbb{G}_T$ are groups of prime order q (q being a λ bit prime greater than N), and $e : \mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G}_T$ is a bilinear map. Recall that $\mathcal{W}$ is the space of all keywords and $\mathcal{ID}$ is the space of all document identifiers. We make use of the following collision-resistant hash functions in our construction:

- $\mathcal{H} : \mathcal{W} \longrightarrow \mathbb{G}^p$ for $p \geq \frac{2\lambda+\omega\log\lambda}{\log q}$. This hash function is used primarily for searchable index and trapdoor generation.
- $\mathcal{H}_1 : \mathcal{W} \longrightarrow \mathbb{Z}_q^p$, again for $p \geq \frac{2\lambda+\omega\log\lambda}{\log q}$. This hash function used for the row-index of the searchable index look-up table.
- $\mathcal{H}_2 : \mathcal{ID} \longrightarrow \mathbb{Z}_q$. This hash function is used for the column index of the searchable index look-up table.

Thus instead of producing a single output, the trapdoors $\mathcal{H}$ and $\mathcal{H}_1$ produces p independent output components. We can then take the product of these components to get a single value. While $p = 1$ suffices for correctness and data privacy, a larger value of p as described above is required to guarantee trapdoor privacy. We stress however that such a value of p is still $\mathcal{O}(1)$; for example, we could choose $p = 3$. We introduce some notations here for ease of presentation:

– Let $\mathcal{H}(w) = (h_1, \cdots, h_p)$. We use the notation $\mathcal{H}(w, y)$ to describe the y^{th} component h_y, where $y \in [1, p]$.
– Similarly, for $\mathcal{H}_1(w) = \left(h'_1, \cdots, h'_p\right)$, we use the notation $\mathcal{H}_1(w, y)$ to describe h'_y, where $y \in [1, p]$.

System Setup: (Performed by the central agent)

$\mathsf{SetUp}(1^\lambda, \mathcal{ID}, N)$: Run $(\mathbb{G}, \mathbb{G}_T, e) \xleftarrow{R} GroupGen(1^{2\lambda})$. Let $\mathcal{G}$ be a description of $(\mathbb{G}, \mathbb{G}_T, e)$. Pick $g_1, g_2 \xleftarrow{R} \mathbb{G}$ and $\alpha, \beta \xleftarrow{R} \mathbb{Z}_q$. Set $\hat{g}_1 \leftarrow g_1^\beta$ and $\hat{g}_2 \leftarrow g_2^\beta$. Set:

$$params \leftarrow \left(\mathcal{G}, \{g_1^{\alpha^j}, \hat{g}_1^{\alpha^j}\}_{j\in[0,N]}, \{g_2^{\alpha^j}, \hat{g}_2^{\alpha^j}\}_{j\in\{0,1\}}\right)$$

Output $params$.

$\mathsf{KeyGen}(params)$: Pick $\gamma, \nu \xleftarrow{R} \mathbb{Z}_q$. Set

$$msk \leftarrow (\gamma, \nu)$$
$$PK \leftarrow \left(params, g_2^\gamma, g_2^{\alpha\cdot\gamma}, \hat{g}_2^\nu, \hat{g}^{\alpha\cdot\nu}\right)$$

Output (msk, PK).

Searchable Index Creation: (Performed by the document collection owner)

$\mathsf{BuildIndex}(PK, \mathbf{D})$: Parse $\mathbf{D}$ to create $\boldsymbol{\Delta}(\mathbf{D})$. Initialize the searchable index $\mathbf{I}$ to empty. For each keyword w in $\boldsymbol{\Delta}(\mathbf{D})$ and each document $D_j \in \mathbf{D}$ with identity id_j, do the following:

(a) Let $i \leftarrow \mathcal{H}_2(id_j)$. Set :

$$c_0 \leftarrow \hat{g}_2^{(\alpha-i)\cdot t}\ ,\ c_1 \leftarrow g_2^{\gamma\cdot t}$$

(b) Next, for $y \in [1, p]$, set:

$$c_{2,y} \leftarrow e\left(\mathcal{H}(w, y), \hat{g}_2^{(\alpha-i)\cdot\nu}\right)^t$$

(c) If D_j contains the keyword w, set $c_3 \leftarrow e\left(\hat{g}_1^{\alpha^N}, g_2^{\alpha\cdot\gamma}\right)^t$. Otherwise, set $c_3 \xleftarrow{R} \mathbb{G}_T$.

(d) Let $T = \prod_{y=1}^{p} \mathcal{H}_1(w, y)$. Set $\mathbf{I}[T][i] \leftarrow \big(c_0, c_1, \{c_{2,y}\}_{y\in[1,p]}, c_3\big)$.

Finally, output $\mathbf{I}$.

Keyword Search: (Involves interaction between the central agent, the user and the remote server)

$\mathsf{GenTrpdr}(msk, w, \mathcal{S})$: Parse msk as (γ, ν) and $\mathcal{S}$ as $(id_1, \cdots, id_n)$ for some $n \leq N$. Set:

$$i_j \leftarrow \mathcal{H}_2(id_j) \text{ for } j \in [1, n]$$
$$i_j \leftarrow 2^\lambda + \mathcal{H}_2(\{1\}^j) \text{ for } j \in [n+1, N]$$

Now, let $P_\mathcal{S}(x) = \prod_{j=1}^{N}(x - i_j)$. Quite evidently, $P_\mathcal{S}(x) = \prod_{j=1}^{N}(x - i_j)$ forms a *unique signature polynomial* for the subset $\mathcal{S}$, that is, for two different subsets $\mathcal{S}$ and $\mathcal{S}'$, we have $P_\mathcal{S}(x) \not\equiv P_{\mathcal{S}'}(x)$.

Next, choose $(r_1, \cdots, r_p) \xleftarrow{R} \mathbb{Z}_q$ and set:

$$T_1 \leftarrow \prod_{y=1}^{p} \mathcal{H}_1(w, y)$$
$$T_2 \leftarrow \left(g_1^{\gamma\cdot P_\mathcal{S}(\alpha)}\right) \cdot \prod_{y=1}^{p} \left(\mathcal{H}(w, y)\right)^{\nu\cdot r_y}$$

Clearly, the first component allows the server to locate the row index corresponding to the keyword w, while the second component enforces controlled-search. This is exactly as discussed in the construction overview. Output $\mathbf{T}_{w,\mathcal{S}} = (T_1, T_2, r_1, \cdots, r_p)$.

Note that since $P_\mathcal{S}(x)$ has degree N, $g_1^{P_\mathcal{S}(\alpha)}$ can be computed from the public parameters. Also, the fact that the polynomial $P_\mathcal{S}(x)$ is unique to the subset $\mathcal{S}$ ensures that the trapdoor for a subset $\mathcal{S}$ cannot be used for searching over another subset $\mathcal{S}'$.

$\mathsf{Search}(\mathbf{I}, \mathbf{T}_{w,\mathcal{S}}, \mathcal{S})$: Parse $\mathbf{T}_{w,\mathcal{S}}$ as $(T_1, T_2, r_1, \cdots, r_p)$. Construct the set $\{i_1, \cdots, i_N\}$ for the subset $\mathcal{S}$ as described above. Initialize the list $\mathbf{L}$ to empty. For each $id_j \in \mathcal{S}$:

(a) Set $i \leftarrow \mathcal{H}_2(id_j)$ and parse $\mathbf{I}[T_1][i]$ as $\big(c_0, c_1, \{c_{2,y}\}_{y\in[1,p]}, c_3\big)$.

(b) Define $P^\star_\mathcal{S}(x) = x^{N+1} - (x - i) \cdot \prod_{j=1}^{N}(x - i_j)$ and set:

$$d_0 \leftarrow e\left(T_2, c_0\right) \ , \ d_1 \leftarrow e\left(\hat{g}_1^{P^\star_\mathcal{S}(\alpha)}, c_1\right)$$

Once again, since $P^\star_\mathcal{S}(x)$ has degree at most N, $\hat{g}_1^{P_\mathcal{S}(\alpha)}$ can be computed from the public parameters.

(c) If $\left(\prod_{y=1}^{p} c_{2,y}^{r_y}\right) \cdot c_3 = d_0 \cdot d_1$, add id_j to $\mathbf{L}$.

Output $\mathbf{L}$.

Correctness. We check that if some document D_j contains w, then $id_j \in \mathbf{L}$:

$$\begin{aligned}
d_0 \cdot d_1 &= e\left(T_2, c_0\right) \cdot e\left(\hat{g}_1^{P^\star_{\mathcal{S}}(\alpha)}, c_1\right) \\
&= e\left(\prod_{y=1}^{p} \left(\mathcal{H}(w,y)\right)^{\nu \cdot r_y}, \hat{g}_2^{(\alpha-i)\cdot t}\right) \cdot e\left(g_1^{\gamma \cdot P_{\mathcal{S}}(\alpha)}, \hat{g}_2^{(\alpha-i)\cdot t}\right) \cdot e\left(\hat{g}_1^{\alpha^{N+1}-(\alpha-i)\cdot P_{\mathcal{S}}(\alpha)}, g_2^{t\cdot\gamma}\right) \\
&= \left(\prod_{y=1}^{p} e\left(\mathcal{H}(w,y), \hat{g}_2^{(\alpha-i)\cdot\nu\cdot t}\right)^{r_y}\right) \cdot e\left(g_1, \hat{g}_2\right)^{t\cdot\gamma\cdot(\alpha-i)\cdot P_{\mathcal{S}}(\alpha) + t\cdot\gamma\cdot\left(\alpha^{N+1}-(\alpha-i)\cdot P_{\mathcal{S}}(\alpha)\right)} \\
&= \left(\prod_{y=1}^{p} c_{2,y}^{r_y}\right) \cdot c_3
\end{aligned}$$

Also, if a document D_j does not contain w, then the probability:

$$\Pr\left[\left(\prod_{y=1}^{p} c_{2,y}^{r_y}\right) \cdot c_3 = d_0 \cdot d_1\right] \leq \mathsf{negl}(\lambda)$$

as c_3 is any uniformly random element in $\mathbb{G}$.

Data Privacy of Our Construction. We state the following theorem for the adaptive data privacy of our first KASE construction:

Theorem 4.1 *Our first KASE construction is adaptively data private under the decision $(\mathscr{S}, M)$-BDHES assumption in the random oracle model.*

The detailed proof of this theorem is presented in Appendix C. We provide a brief overview of the proof technique here. The proof assumes the existence of a simulator algorithm $\mathcal{B}$ that receives as input an instance of the decision $(\mathscr{S}, M)$-BDHES problem as input. The proof then shows that $\mathcal{B}$ can simulate the role of the challenger in the adaptive data privacy game for our KASE construction with a PPT adversary $\mathcal{A}$. In particular, $\mathcal{B}$ simulates from its input instance the public parameters $params$, the public key PK, and the responses to trapdoor oracle queries made by $\mathcal{A}$. Eventually, $\mathcal{A}$ outputs two document collections $\mathbf{D}_0$ and $\mathbf{D}_1$ that are not trivially distinguishable from the search patterns induced by $\mathcal{A}$'s trapdoor queries. Once again, $\mathcal{B}$ uses its input instance to create a valid encrypted searchable index for the collection $\mathbf{D}_b$, where $b \xleftarrow{R} \{0,1\}$, and provides it to $\mathcal{A}$. Finally, $\mathcal{B}$ uses $\mathcal{A}$'s response to propose a solution to its input instance of the decision $(\mathscr{S}, M)$-BDHES problem. The proof argues that $\mathcal{B}$'s advantage in solving the decision $(\mathscr{S}, M)$-BDHES problem is precisely $\mathcal{A}$'s advantage in winning the adaptive data privacy game for KASE.

Trapdoor Privacy of Our Construction. We state the following theorem for the statistical trapdoor privacy of our first KASE construction:

Theorem 4.2 *Our first KASE construction is statistically trapdoor private in the random oracle model for all keywords drawn from (T,k)-block sources, for any $T = \mathit{poly}(\lambda)$ and $k \geq \lambda + \omega(\log\lambda)$.*

The detailed proof of this theorem is presented in Appendix D. We again present a brief overview of the proof technique here. Let $\mathcal{A}$ be a computationally unbounded adversary that makes at most polynomially many (T,k)-block source real-or-random queries and at most polynomially many trapdoor queries during the trapdoor privacy game. We prove here that $\mathcal{A}$'s view of the trapdoor distribution when the challenger chooses $b = 0$ is statistically indistinguishable from its view of the trapdoor distribution when the challenger chooses $b = 1$. We first show that if the hash functions $\mathcal{H} : \mathcal{W} \longrightarrow \mathbb{G}^p$ and $\mathcal{H}_1 : \mathcal{W} \longrightarrow \mathbb{Z}_q^p$ are modeled as random oracles, then the choice of $p \geq \frac{2\lambda + \omega \log\lambda}{\log q}$ ensures that the probability that each of these hash functions is injective over the domain $\mathcal{W}$ is lower bounded by $1 - 2^{-\omega \log\lambda}$. Since $\mathcal{A}$ is a computationally unbounded adversary, we assume without loss of generality that it has full knowledge of the secret parameters α, β, as well as the master secret key $msk = (\gamma, \nu)$, from the public parameters *params* and the public key PK. Finally, we argue that a *single-shot* adversary $\mathcal{A}$ that makes only one real-or-random oracle query during the trapdoor privacy game is polynomially equivalent to a *multi-shot* adversary $\mathcal{A}'$ that makes polynomially many such queries.

With the above assumptions in place, we formally define the view of the adversary $\mathcal{A}$ in the trapdoor privacy game, and prove using the universal hash lemma to prove that $\mathcal{A}$'s view of the trapdoor distribution when the challenger chooses $b = 0$ is statistically indistinguishable from its view of the trapdoor distribution when the challenger chooses $b = 1$.

5 Optimizing the Space Complexity for the Look-Up Table

In this section, we present some techniques for reducing the space complexity of the look-up table **I**. In particular, we provide a technique that allows us to store only the encrypted *yes* entries corresponding to matching keyword-document pairs, which is the most optimal storage possible for any searchable index. We begin by introducing some definitions based on our first KASE construction.

Definition 5.1 (Valid Searchable Index Entry). *A valid searchable index entry for our KASE construction is a tuple* $\left(c_0, c_1, \{c_{2,y}\}_{y\in[1,p]}, c_3\right) \in \mathbb{G} \times \mathbb{G} \times \mathbb{G}_T^p \times \mathbb{G}_T$*, for which there exists* $id \in \mathcal{ID}$*,* $w \in \mathcal{W}$ *and* $t \in \mathbb{Z}_q$*, such that for* $i = \mathcal{H}_2(id)$*, we have:*

$$c_0 \leftarrow \hat{g}_2^{(\alpha-i)\cdot t} \quad , \quad c_1 \leftarrow g_2^{\gamma\cdot t}$$
$$c_{2,y} \leftarrow e\left(\mathcal{H}(w, y), \hat{g}_2^{(\alpha-i)\cdot\nu}\right)^t \text{ for } y \in [1,p]$$
$$c_3 \leftarrow e\left(\hat{g}_1^{\alpha^N}, g_2^{\alpha\cdot\gamma}\right)^t$$

We denote by $\mathcal{V}_\lambda$ *the space of all valid searchable index entries for our KASE construction, where* λ *is the security parameter.*

Definition 5.2 (Dummy Searchable Index Entry). *A dummy searchable index entry for our KASE construction is a tuple* $\left(c_0, c_1, \{c_{2,y}\}_{y\in[1,p]}, c_3\right) \in \mathbb{G} \times \mathbb{G} \times \mathbb{G}_T^p \times \mathbb{G}_T$*, for which there exists* $id \in \mathcal{ID}$ *and* $t \in \mathbb{Z}_q$*, such that for* $i = \mathcal{H}_2(id)$*, we have:*

$$c_0 \leftarrow \hat{g}_2^{(\alpha-i)\cdot t} \quad , \quad c_1 \leftarrow g_2^{\gamma\cdot t}$$
$$c_{2,y} \xleftarrow{R} \mathbb{G}_T \text{ for } y \in [1,p]$$
$$c_3 \xleftarrow{R} \mathbb{G}_T$$

We denote by $\overline{\mathcal{V}}_\lambda$ *the space of all valid searchable index entries for our KASE construction, where* λ *is the security parameter.*

Let $\mathcal{D}$ be a PPT distinguisher takes as input a tuple $\left(c_0, c_1, \{c_{2,y}\}_{y\in[1,p]}, c_3\right) \in \mathbb{G} \times \mathbb{G} \times \mathbb{G}_T^p \times \mathbb{G}_T$ and outputs 0 (resp. 1) to indicate $\left(c_0, c_1, \{c_{2,y}\}_{y\in[1,p]}, c_3\right) \in \mathcal{V}_\lambda$ (resp. $\overline{\mathcal{V}}_\lambda$). We make the following claim.

Claim 5.1 *For all PPT distinguishers* $\mathcal{D}$*, we have:*

$$\left|Pr\left[\mathcal{D}\left(\left(c_0, c_1, \{c_{2,y}\}_{y\in[1,p]}, c_3\right) \in \mathcal{V}_\lambda\right) = 0\right] - Pr\left[\mathcal{D}\left(\left(c_0, c_1, \{c_{2,y}\}_{y\in[1,p]}, c_3\right) \in \overline{\mathcal{V}}_\lambda\right) = 0\right]\right| \leq \mathsf{negl}(\lambda)$$

Proof. The proof follows via a simple hybrid argument that we present here informally. Consider a tuple $\left(c_0, c_1, \{c_{2,y}\}_{y\in[1,p]}, c_3\right) \in \mathcal{V}_\lambda$, and substitute $c_{2,1}$ by $c'_{2,1} \xleftarrow{R} \mathbb{G}_T$. We claim that that a PPT distinguisher $\mathcal{D}$ cannot distinguish the tuples $\left(c_0, c_1, c_{2,1}, \{c_{2,y}\}_{y\in[2,p]}, c_3\right)$ and $\left(c_0, c_1, c'_{2,1}\{c_{2,y}\}_{y\in[2,p]}, c_3\right)$. To see this, observe that $c_{2,1} = e\left(\mathcal{H}(w, 1), c_0\right)^\nu$. Also, since $c'_{2,1}$ is uniformly random in $\mathbb{G}_T$, there must exist a ν' such that $c'_{2,1} = e\left(\mathcal{H}(w, 1), c_0\right)^{\nu'}$. Also, each component of $\mathcal{H}(w)$ is ideally expected to be independent and identically distributed (especially when $\mathcal{H}$ is modeled as a random oracle); hence no information that $\mathcal{D}$ gains from the other components $c_{2,y}$ for $y \in [2,p]$ aids it in distinguishing $c_{2,1}$ and $c'_{2,1}$. Hence, distinguishing $c_{2,1}$ from $c'_{2,1}$ would require $\mathcal{D}$ to solve the *discrete log problem* and check the value of ν. Similar arguments suffice to establish the indistinguishability of subsequent distributions, constructed by successively substituting $c_{2,y}$ by $c'_{2,y} \xleftarrow{R} \mathbb{G}_T$ for $y \in [2,p]$.

Finally, we substitute c_3 by $c'_3 \xleftarrow{R} \mathbb{G}_T$. Let us assume that $\mathcal{D}$ can distinguish the tuples $\left(c_0, c_1, \{c'_{2,y}\}_{y\in[1,p]}, c_3\right)$ and $\left(c_0, c_1, \{c'_{2,y}\}_{y\in[1,p]}, c'_3\right)$ with non-negligible advantage. Such a distinguisher $\mathcal{D}$ can actually be used to construct another efficient algorithm that can solve an instance of the decision $(\mathcal{S}, M)$-BDHES problem with the same advantage (see the proof of Theorem 4.1) in Appendix C for more details). This completes an informal proof of the claim.

An Efficient Implementation of the Look-Up Table. With the above discussion in place, we are now ready to present our optimal strategy for storing the searchable index look-up table $\mathbf{I}$. We advocate the following simple changes to our original construction:

– For each keyword w and each document identifier $id \in \mathcal{ID}_{\mathbf{D}}$, we store the entry $\mathbf{I}[\mathcal{H}_1(w)][\mathcal{H}_2(id)]$ if and only if $id \in \delta_{w,\mathcal{ID}_{\mathbf{D}}}(\mathbf{D})$. In other words, we store only valid entries in the look-up table $\mathbf{I}$.

– When the server looks up an address (i_1, i_2), we check if a valid entry corresponding to this address is present in the table or not. If yes, return the valid entry to the server. Otherwise, we sample an invalid entry from the space $\overline{\mathcal{V}}_\lambda$ as defined above using the document column index i_2, and return it to the server.

Handling the look-up requests from the server and sampling the invalid entries can be done by an efficient hardware engine that only has access to the public parameters and public key of the KASE system.

Performance Benefits. We note that the aforementioned strategy leads to a significant improvement in the storage requirement of the look-up table $\mathbf{I}$. While the space complexity in the original construction was $\mathcal{O}\left(|\boldsymbol{\Delta}(\mathbf{D})| \cdot N\right)$, it now reduces to $\mathcal{O}\left(\boldsymbol{\Delta}_{\mathrm{avg}}(\mathbf{D}) \cdot N\right)$, where $\boldsymbol{\Delta}_{\mathrm{avg}}(\mathbf{D})$ is the average number of keywords per document in the document collection $\mathbf{D}$. To the best of our knowledge, this is the most optimal space complexity to be achieved by any searchable encryption scheme in the literature [5]. Also, since the look-up table $\mathbf{I}$ now stores only the list of matching documents corresponding to each keyword, the work done by the server while searching is also linear only in the number of matching documents. Thus, the server only does an optimal amount of work per keyword search.

Security. It follows from Claim 5.1 that the data privacy of our original KASE construction is not weakened as a result of the aforementioned modifications. The only possible leakage to the adversary could be the number of entries per row and column of the look-up table. However, since both $\mathcal{H}_1$ and $\mathcal{H}_2$ are one-way functions, the row and column indexes reveal no information about the underlying keywords and document identifiers respectively.

Secure Updates to the Document Collection. We end by describing how the optimized KASE construction also facilitates efficient and secure updates to the document collection. Suppose at some point of time, the data owner wishes to augment the existing document collection $\mathbf{D}$ with a new set of documents $\mathscr{D}$. If the overall number of documents is less than or equal to the maximum limit N for which the system is built, the update process simply augments the existing searchable index $\mathbf{I}$ with the *valid* entries corresponding to each (keyword, document identifier) pair for the new set $\mathscr{D}$. *Note that this is sufficient* since the optimized KASE construction requires the searchable index to store only the *yes* entries.

We now focus on the scenario where an update to the document collection causes the overall number of documents to exceed the limit N. A straightforward solution is to to spawn a new document collection with the leftover documents beyond N and create a separate searchable index for the same. When a user submits a keyword search request over a set of documents spanning across multiple document collections, she must be issued separate trapdoors for each document collection. Assume without loss of generality that each update adds a constant c number of documents to the collection. Then, after U such update events, the expected number of searchable indexes in the system, as well as the expected number of trapdoors to be issued per query, thus increases as $\mathcal{O}\left(\frac{c \cdot U}{N}\right)$. This is similar to the update-handling proposition in [3], and is clearly not very scalable.

We propose a more efficient workaround to render the expected blow up in the number of searchable indexes, as well as the trapdoor size, independent of the number of update events. First of all, we may assume without loss of generality, that each new document added to the collection has a identifier different from that of the existing documents. Given that the overall document identity space is exponentially large, this seems a valid assumption. This leads to the following optimizations:

– We can do away with the need for multiple indexes by accommodating all newly added documents in the existing searchable index.
– It is possible to create a single trapdoor spawning across multiple document collections, provided that the document subset size does not exceed the limit N.

Thus if a user submits a keyword search request over a subset $\mathcal{S}$, the number of trapdoors to be issued is $\mathcal{O}\left(\frac{|\mathcal{S}|}{N}\right)$, which is independent of the number of updates to the document collection.

User Revocation. Cloud-based search applications often require efficient management of user access privileges while preserving the data privacy of the underlying document collection. In particular, dynamic revocation of a user's access rights to one or more document collections is commonly a encountered scenario. Once a user's access rights to a document collection is revoked, she should no longer be able to issue/access trapdoors for performing keyword searches over this collection. A straightforward way of achieving this is to combine our KASE construction with a broadcast encryption system with efficient revocation capabilities [13]. In particular, the trapdoor for the standalone KASE system may be combined with a secret key, and broadcast to the target user group with appropriate access privileges. The data and trapdoor privacy requirements for this augmented construction remains the same as the standalone version, that is, given access to an encryption oracle and a trapdoor generation oracle, the adversary should learn nothing about the underlying document collection except for the access pattern. The additional security guarantees related to access privilege management and revocation follow from the security of the broadcast encryption system.

6 Conclusion

In this paper, we presented the key aggregate searchable encryption (KASE) - a public-key searchable encryption system for controlled-access environments. The KASE system involves three parties - a central system owner who is in charge of setting up the system, a data owner who builds a searchable index for her document collection and stores it on the server, and a data user who submits keywords to the system owner and receives the corresponding trapdoor for performing the search. KASE assumes an honest but curious server setting, which a popular model for modeling cloud environments [2, 3]. The salient feature of KASE is its controlled-access trapdoor, which can be used by a user to search only over the specific subset of documents she has access permissions to. We formalized the security framework for KASE by providing separate data and trapdoor privacy definitions. Our data privacy definitions are based on computational indistinguishability and assume poly-time adaptive adversaries with access to encryption and trapdoor oracles. Our trapdoor privacy definitions, on the other hand, are based on statistical indistinguishability and assume computationally unbounded adversaries.

We presented a concrete construction for KASE that satisfies our security definitions while achieving the desired controlled-search features. Our construction assumes that each document is associated with a unique identity string, and allows generation of trapdoors that restrict a user's search capabilities to a polynomial-size subset of the document identity space. Our construction achieves searches in one communication round, and requires a $\mathcal{O}(N)$ amount of work from the server, where N is the number of documents. In addition, it requires $\mathcal{O}(1)$ storage on both the central system owner as well as the data users, and a $\mathcal{O}(N \cdot \Delta)$

storage on the server, where Δ is the total number of keywords in the document collection, which is competitive with some of the most efficient symmetric searchable encryption systems in literature. Finally and most importantly, the trapdoor overhead of our construction is constant and independent of the size of the document subset it provides access to, which is asymptotically better than any searchable encryption construction in the literature. We also presented some optimizations to our KASE construction that reduce the storage requirements on the server even further to $\mathcal{O}(N \cdot \Delta_{\mathrm{avg}})$, where Δ_{avg} is the average number of keywords per document. It also reduces the search work of the server to linear only in the number of documents that actually contain a keyword. To the best of our knowledge, this is the most optimal storage requirement and search complexity to be reported for any searchable encryption scheme in the literature. KASE is also efficient and scalable with respect to data updates, making it ideal for use in dynamic cloud-based search applications.

References

1. Eu-Jin Goh. Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216, 2003.
2. Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*, pages 79–88, 2006.
3. Yan-Cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Applied Cryptography and Network Security, Third International Conference, ACNS 2005, New York, NY, USA, June 7-10, 2005, Proceedings*, pages 442–455, 2005.
4. Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 14-17, 2000*, pages 44–55, 2000.
5. David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 353–373, 2013.
6. Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public Key Encryption with Keyword Search. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, pages 506–522, 2004.
7. Dan Boneh, Eyal Kushilevitz, Rafail Ostrovsky, and William E. Skeith III. Public key encryption that allows PIR queries. In *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, pages 50–67, 2007.
8. Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and efficiently searchable encryption. In *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, pages 535–552, 2007.
9. Dan Boneh, Ananth Raghunathan, and Gil Segev. Function-Private Identity-Based Encryption: Hiding the Function in Functional Encryption. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 461–478, 2013.
10. Baojiang Cui, Zheli Liu, and Lingyu Wang. Key-aggregate searchable encryption (KASE) for group data sharing via cloud storage. *IEEE Trans. Computers*, 65(8):2374–2385, 2016.
11. Dan Boneh and Matthew Franklin. Identity-based encryption from the weil pairing. *SIAM Journal on Computing*, 32(3):586–615, 2003.
12. Craig Gentry and Brent Waters. Adaptive Security in Broadcast Encryption Systems (with Short Ciphertexts). In *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, pages 171–188, 2009.
13. 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.
14. David Zuckerman. Simulating BPP Using a General Weak Random Source. *Algorithmica*, 16(4/5):367–391, 1996.
15. Benny Chor and Oded Goldreich. Unbiased Bits from Sources of Weak Randomness and Probabilistic Communication Complexity. *SIAM J. Comput.*, 17(2):230–261, 1988.
16. Kai-Min Chung and Salil P. Vadhan. Tight Bounds for Hashing Block Sources. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques, 11th International Workshop, APPROX 2008, and 12th International Workshop, RANDOM 2008, Boston, MA, USA, August 25-27, 2008. Proceedings*, pages 357–370, 2008.
17. Dan Boneh, Ananth Raghunathan, and Gil Segev. Function-Private Subspace-Membership Encryption and Its Applications. In *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I*, pages 255–275, 2013.
18. Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a Sparse Table with 0(1) Worst Case Access Time. *J. ACM*, 31(3):538–544, 1984.
19. Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, pages 440–456, 2005.

A The Generic Bilinear Group Model

Let $\mathbb{G}$ and $\mathbb{G}_T$ be two multiplicative cyclic groups of prime order q, such that there exists a bilinear map $e : \mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G}_T$. The generic bilinear group model [19] assumes that an adversary does not have direct access to the actual group representations of elements or the group/pairing operations, and must interact with a set of oracles for the same. The bilinear groups $\mathbb{G}$ and $\mathbb{G}_T$ are represented by a random injective function $\xi : \mathbb{Z}_q \times \{0,1\} \to \{0,1\}^n$ that maps pairs of elements from the additive group $\mathbb{Z}_q$ and the index space (0 for $\mathbb{G}$ and 1 for $\mathbb{G}_T$) to random strings of sufficiently large length. 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, i): Takes as input $x \in \mathbb{Z}_q$ and $i \in \{0,1\}$. Outputs $\xi(x, i)$.

– **Mult**$(\xi_1, \xi_2, \diamond)$: Takes as input $\xi_1 = \xi(x_1, i)$, $\xi_2 = \xi(x_2, i)$ and an operation $\diamond \in \{+, -\}$. Outputs $\xi(x_1 \diamond x_2, i)$.

– **Pair**(ξ_1, ξ_2): Takes as input $\xi_1 = \xi(x_1, 0)$ and $\xi_2 = \xi(x_2, 0)$. Outputs $\xi(x_1.x_2, 1)$.

The generic model is a useful tool to deduce lower bounds on the security of certain hardness assumptions, assuming that the adversary cannot exploit vulnerabilities in the group element representations. However, a lower bound on security in the generic model does not necessarily guarantee a lower bound on security in a specific group instance. It is useful to the extent that it provides us some insight on whether a problem is likely to be hard.

B On the Hardness of the Decision $(\mathscr{S}, M)$-BDHES Problem

We obtain a lower bound on the security of the decision $(\mathscr{S}, M)$-BDHES problem in the generic bilinear group model. Consider an algorithm $\mathcal{B}$ that plays the following game with a PPT adversary $\mathcal{A}$, both of whom receive the security parameter λ as input:

SetUp Phase. $\mathcal{B}$ generates $\mathscr{S} \subset \mathbb{Z}$, $M \in \mathbb{Z} \setminus (\mathscr{S} + \mathscr{S})$ and $\alpha \xleftarrow{R} \mathbb{Z}_q$. It then provides $\{g^{\alpha^i}\}_{i \in \mathscr{S}}$ to $\mathcal{A}$.

Oracle Query Phase-1. $\mathcal{A}$ adaptively issues queries to (**Encode**, **Mult**, **Pair**).

Challenge Phase. $\mathcal{B}$ picks $b \xleftarrow{R} \{0,1\}$. It sets $Z_b \leftarrow e(g,g)^{\alpha^M}$ and $Z_{1-b} \xleftarrow{R} \mathbb{G}_T$, and provides A with (Z_0, Z_1).

Oracle Query Phase-2. $\mathcal{A}$ continues to adaptively issue queries to (**Encode**, **Mult**, **Pair**).

Guess Phase. $\mathcal{A}$ outputs a guess b' for the random bit b chosen by the challenger.

As is the standard norm for generic group model proofs, we now assume that instead of choosing a random value for α at the beginning of the game, $\mathcal{B}$ treats it as a formal variable and maintains a list of tuples $L = \{(p, i, \epsilon)\}$, where p is a polynomial in α, $i \in \{0,1\}$ is the group index and $\epsilon \in \{0,1\}^n$ for n sufficiently large. The game now proceeds through the following stages:

1. $\mathcal{B}$ initializes the list with tuples of the form $\left(\alpha^j, 0, \xi_j\right)$ for randomly generated $\xi_j \in \{0,1\}^n$, where $j \in \mathscr{S}$. Thus initially $|L| = |\mathscr{S}|$.

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

- **Encode**(x, i): If $x \notin \mathbb{Z}_q$ or $i \notin \{0, 1\}$, $\mathcal{B}$ returns $\perp$. Otherwise, $\mathcal{B}$ looks for a tuple $(p_x, i, \xi) \in L$, 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, i, ξ) is added to L and $\mathcal{B}$ responds with ξ.

- **Mult**$(\xi_0, \xi_1, \diamond)$: $\mathcal{B}$ searches in L for the pair of tuples (p_0, i_0, ξ_0) and (p_1, i_1, ξ_1). Unless both of them exist, $\mathcal{B}$ returns $\perp$. Also, if both tuples are available but $i_0 \neq i_1$, $\mathcal{B}$ returns $\perp$. Otherwise, $\mathcal{B}$ searches in L for a tuple of the form (p, i, ξ), where $p = p_0 \diamond p_1$ (for $\diamond \in \{+, -\}$) and $i = i_0 = i_1$. If such a tuple is found, $\mathcal{B}$ responds with ξ. If not, $\mathcal{B}$ generates a random $\xi \in \{0, 1\}^n$, augments L by adding the tuple (p, i, ξ) and returns ξ.

- **Pair**(ξ_0, ξ_1): $\mathcal{B}$ searches in L for the pair of tuples $(p_0, 0, \xi_0)$ and $(p_1, 0, \xi_1)$. Unless both of them exist, $\mathcal{B}$ returns $\perp$. Otherwise, $\mathcal{B}$ searches in L for a tuple of the form $(p', 1, \xi')$, where $p' = p_0.p_1$. If such a tuple is found, $\mathcal{B}$ responds with ξ'. If not, $\mathcal{B}$ generates a random $\xi' \in \{0, 1\}^n$, augments L by adding the tuple $(p', 1, \xi')$ and returns ξ'.

3. $\mathcal{B}$ now adds the pair of tuples $(Z_0, 1, \xi_0^*)$ and $(Z_1, 1, \xi_1^*)$ to the list L, where $\xi_0^*, \xi_1^* \xleftarrow{R} \{0, 1\}^n$. It then provides the challenge (ξ_0^*, ξ_1^*) to $\mathcal{A}$.

4. $\mathcal{A}$ may continue to adaptively issue a polynomial number of (**Encode**,**Mult**,**Pair**) queries. $\mathcal{B}$ responds as in Phase-1.

5. Finally, $\mathcal{A}$ outputs a guess bit $b' \in \{0, 1\}$.

At this point, $\mathcal{B}$ samples a random value from $\mathbb{Z}_q$ for the formal variable α and a random bit $b \xleftarrow{R} \{0, 1\}$. It instantiates each of the polynomials in the list L with the value for α. Further, it substitutes Z_b with α^M and Z_{1-b} with some random $\beta \xleftarrow{R} \mathbb{Z}_q$. We say that $\mathcal{A}$ wins the game if $b = b'$ or $\mathcal{B}$ fails to simulate the (**Encode**,**Mult**,**Pair**) oracles successfully. Note that after instantiating the polynomials in the list L with a random value for α, $\mathcal{B}$ will have failed to simulate the oracles perfectly if and only if there exists at least one pair of tuples (p, i, ξ) and (p', i', ξ') in L such that $p \not\equiv p'$, $i = i'$, $\xi \neq \xi'$, and yet, $p(\alpha) = p'(\alpha)$. We refer to such an event as a false polynomial equality event Υ. Thus $\mathcal{A}$ wins the game if either $b = b'$ or an instance of Υ occurs.

If an instance of Υ does not occur, $\mathcal{B}$ simulates the oracle in response to $\mathcal{A}$'s queries perfectly. In this case, from $\mathcal{A}$'s view, b is independent of its own choice of b', as b was chosen by $\mathcal{B}$ after the simulation was over. Hence we have

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

This in turn gives us the following relations:

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

which leads to the following upper bound on the advantage of $\mathcal{A}$:

$$\left| Pr[b = b'] - \frac{1}{2} \right| \leq \frac{Pr[\Upsilon]}{2}$$

It remains to bound the probability that a random choice of value for the formal variable α results in an instance of Υ. Let d be the maximum value in $\mathscr{S}$. Then, by construction, the degree of any polynomial in L is upper bounded by $2d$ (the worst $\mathcal{A}$ could do is invoke the **Pair** oracle on the highest degree term). Also, the number of polynomials in L is upper bounded by $|\mathscr{S}| + Q_E + Q_M + Q_P$, where Q_E, Q_M and Q_P are the maximum number of queries made by $\mathcal{A}$ to the **Encode**, **Mult**, and **Pair** oracles respectively.

Thus, by the Schwartz-Zippel lemma, the probability of a false polynomial equality event may be upper bounded by $(|\mathscr{S}| + Q_E + Q_M + Q_P)^2 d/q$. From this, it is straightforward to conclude that the advantage of $\mathcal{A}$ may be upper bounded as $(|\mathscr{S}| + Q_E + Q_M + Q_P + Q_K)^2 d/2q$. For $|\mathscr{S}|, Q_E, Q_M, Q_P$ polynomial in the security parameter λ, this quantity is negligible provided that $q = \Omega(2^\lambda.d)$. Thus, for the decision BDHES assumption to hold, the group order must be a prime q that is at least $\Omega(2^\lambda)$ times larger than the maximum entry of the subset $\mathscr{S}$.

C Proof of Theorem 4.1

Let $d = 2^{\lambda/2} + N$ for $N = \mathsf{poly}(\lambda)$. Also, let q be a 2λ-bit prime. Quite evidently, $q = \Omega(2^\lambda.d)$. An algorithm $\mathcal{B}$ receives an instance of the decision $(\mathscr{S}, M)$-BDHES problem :

$$(\{g^{\alpha^i}\}_{i\in\mathscr{S}}, Z)$$

where

$$\mathscr{S} = \{0,1\} \cup [d+N, 2d+N-2] \cup [2d+2N, \lfloor 5d/2 \rfloor + \lfloor 5N/2 \rfloor - 1] \cup [3d+3N, 3d+4N] \cup [4d+4N, 5d+5N-3]$$

and Z is either $e(g,g)^M$ for $M = 5d + 5N - 1$ (in which case $\mathcal{B}$ should output 0) or a random element in $\mathbb{G}_T$ (in which case $\mathcal{B}$ should output 1). It interacts with a data-privacy adversary $\mathcal{A}$ as follows.

SetUp Phase. $\mathcal{B}$ picks $a_0, a_1, a_2, \nu \xleftarrow{R} \mathbb{Z}_q$ and sets:

$$g_1 \leftarrow g^{a_1.\alpha^{4d+4N}}, \ \hat{g}_1 \leftarrow g^{a_0.a_1.\alpha^{3d+3N}}, \ g_2 \leftarrow g^{a_2.\alpha^{d+N}}, \ \hat{g}_2 \leftarrow g^{a_0.a_2}$$

Note that this is implicitly equivalent to setting $\beta \leftarrow a_0\alpha^{-d-N}$. Next, $\mathcal{B}$ picks a random polynomial $f(x) \xleftarrow{R} \mathbb{Z}_q[x]$ of degree $d-3$ and formally sets $\gamma \leftarrow f(\alpha)$. Let $\mathcal{G}$ be a description of $(\mathbb{G}, \mathbb{G}_T, e)$. $\mathcal{B}$ sets:

$$\begin{aligned} params &\leftarrow \left(\mathcal{G}, g, \{g_1^{\alpha^i}, \hat{g}_1^{\alpha^i}\}_{i\in[0,N]}, \{g_2^{\alpha^i}, \hat{g}_2^{\alpha^i}\}_{i\in\{0,1\}}\right) \\ PK &\leftarrow (g_2^{\gamma}, g_2^{\alpha\cdot\gamma}, \hat{g}_2^{\nu}, \hat{g}_2^{\alpha\cdot\nu}) \end{aligned}$$

$\mathcal{B}$ provides *params* and PK to $\mathcal{A}$. Note that $\mathcal{B}$ can set each of the aforementioned components for both *params* and PK from its input instance. Since the choice of $\alpha, f(x), r$ and a_0, a_1, a_2 are all uniformly random, the distribution of *params* and PK are exactly as in the real world.

Trapdoor Query Phase-1. $\mathcal{A}$ adaptively issues trapdoor queries of the form $(w_q, \mathcal{S}_q)$ for $q \in [1, Q_1]$ and $\mathcal{S}_q \subset \mathcal{ID}$. Let $P_{\mathcal{S}}(x)$ be as described in the construction. $\mathcal{B}$ samples $(r_1, \cdots, r_p) \xleftarrow{R} \mathbb{Z}_q$ and sets:

$$\begin{aligned} T_1 &\leftarrow \prod_{y=1}^{p} \mathcal{H}_1(w,y) \\ T_2 &\leftarrow \left(g_1^{\gamma\cdot P_{\mathcal{S}}(\alpha)}\right) \cdot \left(\prod_{y=1}^{p} \mathcal{H}^{r_y}(w,y)\right)^{\nu} \end{aligned}$$

Note that since $P_{\mathcal{S}}(x)$ has degree N, $g_1^{P_{\mathcal{S}}(\alpha)}$ can be computed from the public parameters. $\mathcal{B}$ responds with $\mathbf{T}_q = (T_1, T_2, r_1, \cdots, r_p)$.

Challenge Phase. $\mathcal{A}$ provides $\mathcal{B}$ with two document collections $\mathbf{D}_0 = (D_{1,0}, \cdots, D_{N,0})$ and $\mathbf{D}_1 = (D_{1,1}, \cdots, D_{N,1})$ subject to the restriction that:

$$\tau\left(\mathbf{D}_0, \{(w_q, \mathcal{S}_q)\}_{q\in[1,Q_1]}\right) = \tau\left(\mathbf{D}_1, \{(w_q, \mathcal{S}_q)\}_{q\in[1,Q_1]}\right)$$

$\mathcal{B}$ picks $b \xleftarrow{R} \{0,1\}$. For each keyword w in $\boldsymbol{\Delta}(\mathbf{D}_b)$ and each document $D_{j,b} \in \mathbf{D}_b$ with identity $id_{j,b}$, it does the following:

(a) Let $i \leftarrow \mathcal{H}_2(id_j)$. $\mathcal{B}$ now picks a random polynomial $t(x) \xleftarrow{R} \mathbb{Z}_q[x]$ of degree $d+2N+3$ such that:

$$f(x)t(x)\mid_{d-2} = 1$$
$$f(x)t(x)\mid_j = 0 \text{ for } j \in [d-1, d+N-1] \cup [\lfloor 3d/2 \rfloor + \lfloor 3N/2 \rfloor - 1, 2d+2N-1]$$
$$(x-i)t(x)\mid_j = 0 \text{ for } j \in [1, d+N-1]$$

where $t(x)\mid_j$ denotes the coefficient of x^j in $t(x)$. $\mathcal{B}$ also sets:

$$r(x) \leftarrow x^{N+1} f(x)t(x) - x^{d+N-1}$$

(b) Next, it sets :

$$c_0^* \leftarrow \hat{g}_2^{(\alpha-i).t(\alpha)} \ , \ c_1^* \leftarrow g_2^{f(\alpha).t(\alpha)}$$

(c) Next, for $y \in [1,p]$, it sets:

$$c_{2,y}^* \leftarrow e\left(\mathcal{H}(w,y), c_0^{\nu}\right)$$

(d) If $D_{j,b}$ contains the keyword w, it sets:

$$c_3^* \leftarrow Z^{a_0.a_1.a_2}.e\left(g_1, \hat{g}_2\right)^{r(\alpha)}$$

otherwise it sets $c_3^* \xleftarrow{R} \mathbb{G}_T$.

(e) Let $T = \prod_{y=1}^{p} \mathcal{H}_1(w,y)$. $\mathcal{B}$ sets $\mathbf{I}^*[T][i] \leftarrow \left(c_0^*, c_1^*, \{c_{2,j}^*\}_{j\in[1,p]}, c_3^*\right)$.

Finally, it responds with $\mathbf{I}^*$.

Trapdoor Query Phase-2. $\mathcal{A}$ continues to adaptively issue trapdoor queries of the form $(w_q, \mathcal{S}_q)$ for $q \in [Q_1, Q]$, once again subject to the restriction that:

$$\tau\left(\mathbf{D}_0, \{(w_q, \mathcal{S}_q)\}_{q\in[1,Q]}\right) = \tau\left(\mathbf{D}_1, \{(w_q, \mathcal{S}_q)\}_{q\in[1,Q]}\right)$$

$\mathcal{B}$ responds as in *Trapdoor Query Phase-1.*

Guess Phase. $\mathcal{A}$ outputs a guess b' for the random bit b chosen by $\mathcal{B}$. If $b' = b$, $\mathcal{B}$ returns 0, else $\mathcal{B}$ returns 1.

Modeling the Hash Functions as Random Oracles. The hash functions in the above proof are simulated by $\mathcal{B}$ as random oracles that $\mathcal{A}$ can query at any time. The simulation is as follows:

Simulating $\mathcal{H}$: $\mathcal{B}$ maintains a list of tuples of the form $(w, \{g_y\}_{y\in[1,p]})$ called the $\mathcal{H}$-table. The table is initially empty. On receipt of a query $\mathcal{H}(w)$ for $w \in \{0,1\}^\lambda$, $\mathcal{B}$ first searches for a matching tuple entry in the table. If found, it returns the second component of the corresponding entry. Otherwise, it samples $\left(\{g_y\}_{y\in[1,p]}\right) \xleftarrow{R} \mathbb{G}^p$, adds the tuple $(w, \{g_y\}_{y\in[1,p]})$ to the $\mathcal{H}$-table and returns $\left(\{g_y\}_{y\in[1,p]}\right)$ as response. Since each g_y is uniformly random in $\mathbb{G}$, the response is random from $\mathcal{A}$'s view as desired.

Simulating $\mathcal{H}_1$: $\mathcal{B}$ maintains a list of tuples of the form $(w, \{l_j\}_{j\in[1,p]})$ called the $\mathcal{H}_1$-table. The table is initially empty. On receipt of a query $\mathcal{H}_1(w)$ for $w \in \{0,1\}^\lambda$, $\mathcal{B}$ first searches for a matching tuple entry in the table. If found, it returns the second component of the corresponding entry. Otherwise, it samples $\left(\{l_y\}_{y\in[1,p]}\right) \xleftarrow{R} \mathbb{Z}_q^p$, adds the tuple $(w, \{l_y\}_{y\in[1,p]})$ to the $\mathcal{H}_1$-table and returns $\left(\{l_y\}_{y\in[1,p]}\right)$ as response. Since each l_y is uniformly random in $\mathbb{Z}_q$, the response is random from $\mathcal{A}$'s view as desired.

Simulating $\mathcal{H}_2$: $\mathcal{B}$ maintains a list of tuples of the form (id, i) called the $\mathcal{H}_2$-table. The table is initially empty. On receipt of a query $\mathcal{H}_2(id)$ for $id \in \mathcal{ID}$, $\mathcal{B}$ first searches for a matching tuple entry in the table. If found, it returns the second component of the corresponding entry. Otherwise, it sets $i \xleftarrow{R} \mathbb{Z}_q$, adds the tuple (id, i) to the $\mathcal{H}_2$-table and returns i as response. Since i is uniformly random in $\mathbb{Z}_q$, the response is random from $\mathcal{A}$'s view as desired.

Claim C.1 *$\mathcal{B}$ can compute c_0^* in the **Challenge Phase**.*

Proof. Observe that $(x-i)t(x)$ is a polynomial of degree $d+2N+4$. Owing to the constraints on $(x-i)t(x)$ mentioned earlier and the fact that $\hat{g}_2 = g^{a_0 \cdot a_2}$, $\mathcal{B}$ only requires the knowledge of g^{α^j} for $j \in \{0,1\} \cup [d+N, d+2N+4]$, which is provided as part of its input instance. In particular, for $d = q+N$, $d+2N+4$ is certainly less than $2d+N-2$.

Claim C.2 *$\mathcal{B}$ can compute c_1^* in the **Challenge Phase**.*

Proof. Observe that $f(x)t(x)$ is a polynomial of degree $2d+2N$. Owing the constraints on $t(x)$ mentioned earlier and the fact that $g_2 = g^{a_2 \alpha^{d+N}}$, $\mathcal{B}$ only requires the knowledge of g^{α^j} for $j \in [d+N, 2d+N-2] \cup [2d+2N, \lfloor 5d/2 \rfloor + \lfloor 5N/2 \rfloor - 2] \cup \{3d+3N\}$, which is provided as part of its input instance.

Claim C.3 *$\mathcal{B}$ can compute c_3^* in the **Challenge Phase**.*

Proof. Observe that the $\mathcal{B}$ needs to compute $Z^{a_0 . a_1 . a_2}.e(\hat{g}_1^{\alpha^N}, g_2^{\alpha f(\alpha)t(\alpha) - \alpha^{d-1}})$. Since $xf(x)t(x)$ is a polynomial of degree $2d+2N+1$, given the constraints on $f(x)t(x)$ mentioned earlier and the fact that $g_2 = g^{a_2 \alpha^{d+N}}$, the only term in $g_2^{\alpha f(\alpha)t(\alpha)}$ that $\mathcal{B}$ cannot compute from its input instance is the α^{d-1} term as it is not provided with $g^{\alpha^{2d+N-1}}$. However, the constraint that $f(x)t(x) \mid_{d-2} = 1$ implies that $xt(x) \mid_{d-1} = 1$, hence this term cancels out and need not be computed.

Claim C.4 *For $Z = e(g,g)^{\alpha^{5d+5N-1}}$, $\mathcal{A}$'s view of $\mathbf{I}^*$ is exactly as in the real world and $\mathcal{B}$'s simulation is perfect.*

Proof. Let $\gamma = f(\alpha)$ and $t = t(\alpha)$. Then we have:

$$c_0^* = \hat{g}_2^{(\alpha - i).t} \quad , \quad c_1^* = g_2^{\gamma . t}$$
$$c_{2,y}^* = e\left(\mathcal{H}(w,y), c_0^{\nu}\right) \text{ for } y \in [1,p]$$

Further, for $Z = e(g,g)^{\alpha^{5d+5N-1}}$, we have:

$$\begin{aligned} Z^{a_0.a_1.a_2}.e(g_1, \hat{g}_2)^{r(\alpha)} &= \left(e(g,g)^{a_0.a_1.a_2.\alpha^{5d+5N-1}}\right).\left(e(g_1,\hat{g}_2)^{\alpha^{N+1}.f(\alpha)t(\alpha)}\right).\left(e(g_1,\hat{g}_2)^{-\alpha^{d+N-1}}\right) \\ &= \left(e(g,g)^{a_0.a_1.a_2.\alpha^{5d+5N-1}}\right).\left(e(g_1,\hat{g}_2)^{\alpha^{N+1}.f(\alpha)t(\alpha)}\right).\left(e(g^{a_1.\alpha^{4d+4N}}, g^{a_0.a_2})^{-\alpha^{d+N-1}}\right) \\ &= e(g_1^{\alpha^N}, \hat{g}_2^{\gamma.\alpha})^t \end{aligned}$$

On the other hand, when Z is random in $\mathbb{Z}_q$, $\mathbf{I}^*$ is uniformly random from $\mathcal{A}$'s view point. From this, we see that $\mathcal{B}$'s advantage in deciding the $(\mathcal{S}, M)$-BDHES instance is precisely $\mathcal{A}$'s advantage in breaking the adaptive data privacy of our KASE construction. This completes the proof of Theorem 4.1.

D Proof of Theorem 4.2

Let $\mathcal{A}$ be a computationally unbounded adversary that makes at most polynomially many (T,k)-block source real-or-random queries and at most polynomially many trapdoor queries during the trapdoor privacy game. We prove here that $\mathcal{A}$'s view of the trapdoor distribution when the challenger chooses $b = 0$ is statistically indistinguishable from its view of the trapdoor distribution when the challenger chooses $b = 1$.

Injectivity of Hash Functions. As in the proof of data privacy, we assume here that the hash functions $\mathcal{H} : \mathcal{W} \longrightarrow \mathbb{G}^p$ and $\mathcal{H}_1 : \mathcal{H} : \mathcal{W} \longrightarrow \mathbb{Z}_q^p$ are modeled as random oracles. Our choice of $p \geq \frac{2\lambda+\omega \log \lambda}{\log q}$ ensures that the probability that each of these has functions is injective over the domain $\mathcal{W}$ is lower bounded by $1 - \frac{|\mathcal{W}|^2}{q^p} = 1 - 2^{-\omega \log \lambda}$. Hence, we may assume in our forthcoming discussions that both these hash functions are injective, and demonstrate under this assumption that $\mathcal{A}$'s view of the trapdoor distribution when the challenger chooses $b = 0$ is statistically indistinguishable from its view of the trapdoor distribution when the challenger chooses $b = 1$.

Knowledge of Secret Parameters and Keys. Further, since $\mathcal{A}$ is a computationally unbounded adversary, we assume without loss of generality that it has full knowledge of the secret parameters α, β, as well as the master secret key $msk = (\gamma, \nu)$, from the public parameters *params* and the public key PK. Hence, the queries to the trapdoor oracle may be simulated internally by $\mathcal{A}$ itself. Our aim is to show that for any given $(\alpha, \beta, \gamma, \nu)$, $\mathcal{A}$'s view of the trapdoor distribution when the challenger chooses $b = 0$ is statistically indistinguishable from its view of the trapdoor distribution when the challenger chooses $b = 1$.

Single-Shot and Multi-Shot Adversaries. Finally, we argue that a *single-shot* adversary $\mathcal{A}$ that makes only one real-or-random oracle query during the trapdoor privacy game is polynomially equivalent to a *multi-shot* adversary $\mathcal{A}'$ that makes polynomially many such queries. A straightforward approach to prove this is to construct a hybrid argument, where the hybrids are constructed such that only one of the adversary $\mathcal{A}'$ queries is answered by the real-or-random oracle depending on the challenger's choice of b, while the rest are forwarded to the trapdoor generation oracle, and are hence answered independent of b.

View of the Adversary. With the above assumptions in place, we are now in a position to formally define the view of the adversary $\mathcal{A}$ in the trapdoor privacy game. Let $\mathbf{W}$ be a circuit description of a (T, k)-block source over the space $\mathcal{W}$ of all keywords, such that $\mathcal{A}$ makes a real-or-random oracle query with $(\mathbf{W}, \mathcal{S})$ for some $\mathcal{S} \subset \mathcal{ID}$. Recall that the trapdoor $\mathbf{T}_{w,\mathcal{S}}$ for a keyword w and a a subset $\mathcal{S}$ is of the form (T_1, T_2), where:

$$T_1 \leftarrow \prod_{y=1}^{p} \mathcal{H}_1(w, y)$$

$$T_2 \leftarrow \left(g_1^{\gamma \cdot P_{\mathcal{S}}(\alpha)}\right) \cdot \prod_{y=1}^{p} (\mathcal{H}(w, y))^{\nu \cdot r_y}$$

Assuming that the adversary $\mathcal{A}$ has full knowledge of $(\alpha, \beta, \gamma, \nu)$, it can easily strip away the component $\left(g_1^{\gamma \cdot P_{\mathcal{S}}(\alpha)}\right)$ of T_2, as well as the power ν to which the product of the hash functions is raised. Hence, in light of these facts, the adversary's view in the trapdoor privacy game is defined as:

$$\mathsf{View}_b = \left(\left(\prod_{y=1}^{p} \mathcal{H}_1(w_1, y), \prod_{y=1}^{p} \mathcal{H}^{r_y}(w_1, y), r_{1,1}, \cdots, r_{p,1} \right) \cdots, \left(\prod_{y=1}^{p} \mathcal{H}_1(w_T, y), \prod_{j=1}^{p} \mathcal{H}^{r_j}(w_T, y), r_{1,T}, \cdots, r_{p,T} \right) \right)$$

where $(w_1, \cdots, w_T) \xleftarrow{R} \mathbf{W}$ for $b = 0$ and $(w_1, \cdots, w_T) \xleftarrow{R} \mathcal{W}^T$ for $b = 1$, and $r_{i,j} \xleftarrow{R} \mathbb{Z}_q$ for $i \in [1, p]$ and $j \in [1, T]$.

Statistical Indistinguishability of the Adversary's View. It remains to prove the statistical indistinguishability of View_0 and View_1. Note that the collection of functions of the form

$$f_{r_1,\cdots,r_p}\left((h_1, \cdots, h_p), (h'_1, \cdots, h'_p)\right) = \left(\prod_{y=1}^{p} h'_y, \prod_{y=1}^{p} h_y^{r_y}, r_1, \cdots, r_p \right)$$

is universal. Assuming that $\mathbf{W}$ is a (T, k)-block source, and that $\mathcal{H}$ and $\mathcal{H}_1$ are modeled as injective hash functions, we can directly apply Lemma 2.1 to argue that View_0 is statistically indistinguishable from View_1. This completes the proof of Theorem 4.2.