BigSecret: A Secure Data Management Framework for Key-Value Stores

Erman Pattuk*, Murat Kantarcioglu*, Vaibhav Khadilkar*, Huseyin Ulusoy*, Sharad Mehrotra†
*University of Texas at Dallas, Richardson, Texas USA
{erman.pattuk, muratk, vvk072000, huseyin.ulusoy}@utdallas.edu
†University of California, Irvine, California USA
sharad@ics.uci.edu

***Abstract*—Data storage is one of the most popular cloud services, and is therefore offered by most service providers. Among the various cloud based data storage services, key-value stores has emerged as a popular option for storing and retrieving billions of key-value pairs. Although using such cloud based key-value store services could generate many benefits, companies are reluctant to utilize such services due to security concerns. For example, if keys are used to represent social security numbers of health insurance customers, and values are their medical claim details, then outsourcing such key-value pairs to a public cloud could create significant privacy and security risks. To mitigate such risks, we propose BigSecret, a framework that enables secure outsourcing and processing of encrypted data over public key-value stores. Furthermore, our proposed framework could automatically make use of multiple cloud providers, including existing private clouds, to securely distribute data and workloads for improving efficiency and performance. Our experiments show that efficient and secure processing over outsourced encrypted data residing in key-value stores is possible with a minor overhead in most cases. In addition, we show that BigSecret's data and workload distribution algorithm can lead to major performance gains in a multi-cloud setting.**

I. Introduction

To cope with millions of transactions every day, popular web sites such as Facebook and Twitter have been using Key-Value (KV) stores. Amazon's Dynamo [1]; Memcached used by Facebook, Zynga and Twitter; Apache HBase and Apache Cassandra are some widely-used examples. Furthermore, many cloud service providers offer KV storage services (e.g., Amazon's S3), so that data owners (DO) can now get rid of the burden of database management. However, outsourcing sensitive KV pairs (e.g., personal health records, company financial data) to cloud providers incurs security and privacy risks. To mitigate such risks, one could encrypt data prior to outsourcing. This, however, complicates data processing. Thus, solutions should be offered that intelligently utilize cryptographic techniques, so that the efficiency of query processing is not sacrificed for data security.

Performing secure operations over encrypted data has been investigated previously in several contexts. Searchable symmetric encryption (SSE) (e.g., [2], [3]) is one of the widely investigated problems, where the aim is to provide efficient keyword search capabilities over encrypted data. Performing range queries over outsourced data [4] is another example of secure data processing. Although such solutions are interesting starting points, they need to be adapted for KV stores to enable efficient and secure outsourcing.

One way to balance the security risks versus efficiency when outsourcing to KV stores is to use hybrid cloud architectures [5]. *Hybrid Cloud* architectures leverage a DO's private cloud (i.e., cloud infrastructure already possessed by a DO on its own premises), and offer cost-effective solutions to data outsourcing. It allows a DO to control the amount of sensitive data disclosure risk, by (i) outsourcing encrypted data to a public cloud, and (ii) keeping a subset of data in the private cloud in plaintext form. Another alternative may be to keep parts of the KV store at different cloud providers. This way, data is not vulnerable to a single cloud provider attack, which reduces the disclosure risk. However, it comes at a cost of efficiency specially if there is difference between the computing capabilities of the service providers.

To enable efficient and secure outsourcing of KV stores, we need to develop solutions that take into account various options. For example, a DO may have access to a set of KV storage providers, where each provider has its own pricing policy and processing power. Moreover, a DO may have different levels of trust towards each provider, various monetary and risk constraints, and different private cloud capabilities. Given such options and constraints, DOs need a framework to maximize performance (i.e., minimize the total execution time) by partitioning data and workloads over the providers. Of course, once partitioning is performed, data should be stored in a secure and efficient manner.

In this work, we propose *BigSecret*, a secure data management framework for KV stores. BigSecret offers a solution for securely storing and processing KV data, by using one of three different data storage techniques, which we refer to as *encryption models*. Moreover, it uses a heuristic approach to optimally distribute data and workloads over a cloud setup consisting of multiple providers, (i.e., *Multi-Cloud Setup*), under varying monetary and disclosure risk constraints.

The primary contributions of this work can be listed as:

1) We formalize the data and workload partitioning problem over a multi-cloud setup, with monetary and disclosure risk constraints. To solve instances of this problem, we apply a heuristic approach based on the Hill-Climbing technique.

2) We present encryption models that allow efficient and secure storage and processing of encrypted data on KV stores. Our solution is oblivious to the underlying KV store and can work on any KV store.
3) We provide a formal security analysis of our encryption models.
4) We conduct extensive experiments to compare the performance of our encryption models, and validate the benefits of our partitioning algorithm in an example multi-cloud setup.

The rest of the paper is organized as follows: In Section II, we formally define the data and workload partitioning problem. Section III gives the necessary background information on KV stores and cryptographic tools that we have used. BigSecret is detailed in Section IV, while its security is analyzed in Section V. We then present the results of our experiments in Section VI. Finally, Section VII reviews related work relevant to our paper, followed by the conclusions and future work in Section VIII.

II. Problem Definition

Suppose a DO has access to a number of KV storage providers (which may also include its private clouds), and holds a KV dataset and query workload that needs to be partitioned over these providers. Each provider may have different processing capabilities, pricing methodologies and trustworthiness. For instance, a DO could have access to a *Semi-trusted* provider (i.e., one that offers cloud services, but tries to learn sensitive information) with very low pricing; or a *Trusted* provider (i.e., offers cloud services without any adversarial intent) with weak processing capabilities. In any setup, a DO's aim is to distribute the dataset and workload over all providers, such that the total execution time of the workload is minimized while satisfying (i) a *monetary constraint* (i.e., the total monetary cost is held below a specified threshold), and (ii) a *disclosure constraint* (i.e., the number of sensitive KV entries that may be extracted from outsourced data does not exceed a predefined value). We name the problem of finding a data and workload distribution that satisfies both of the above constraints over a multi-cloud setup as *Multi-Cloud Partitioning Problem* (MCPP).

In this section, we first define the notations in MCPP. It is followed by an explanation of cost metrics, i.e., the way we measure performance, monetary cost, and disclosure. Finally, we define MCPP formally, and discuss how it could potentially create a trade-off between disclosure and cost.

A. Notations and Cost Metrics

Dataset $\mathcal{D}$ is composed of rows, which in turn are composed of KV entries. $\mathcal{D}_q$ represents the set of rows that are needed to answer query q, while $\mathcal{D}_\mathcal{Q}$ represents the set of rows needed to process queries in the workload $\mathcal{Q} = \{q_1, \ldots, q_m\}$. For a query $q \in \mathcal{Q}$, $f(q)$ is the frequency of q in $\mathcal{Q}$. For a provider P_i, $\mathcal{D}_{P_i}$ and $\mathcal{Q}_{P_i}$ denote the data stored and queries executed on P_i respectively. Finally, the expected number of sensitive KV entries stored on P_i is denoted by $sens(\mathcal{D}_{P_i})$.

Metrics in MCPP are investigated in terms of monetary cost, performance, and security.

Monetary Cost: Monetary cost is a DO's expenditure resulting from utilization of KV storage services. Two monetary cost metrics are defined in MCPP: (i) *Storage cost* is the cost of storing a dataset $\mathcal{D}$ on a provider P_i, and is represented as $s(\mathcal{D}, P_i)$. Similarly, the storage cost of a single query q is $s(q, P_i) := s(\mathcal{D}_q, P_i)$; (ii) *Processing cost* is the cost of executing q on P_i, and is denoted as $c(q, P_i)$.

In terms of KV stores, the storage cost may consist of costs arising from a Put operation (defined in Sec. III-B) that entails transferring data to a provider (communication cost). On the other hand, the processing cost is the sum of the costs of issuing a Get or Scan (Sec. III-B) operation, and transferring the results to the client.

The total cost of executing q on P_i is the sum of the storage cost and processing cost: $t(q, P_i) := s(q, P_i) + f(q) \times c(q, P_i)$. Similarly, the total cost of executing a workload $\mathcal{Q}$ on P_i is given by $t(\mathcal{Q}, P_i) := s(\mathcal{D}_\mathcal{Q}, P_i) + \sum_{q \in \mathcal{Q}} f(q) \times c(q, P_i)$. The overall monetary cost of the partitioned workload $\mathcal{Q}_{P_1}, \ldots, \mathcal{Q}_{P_k}$ is the sum of $t(\mathcal{Q}_{P_i}, P_i)$ over all providers, and should be less than the monetary constraint C_{cost}.

Security Metrics: Placing data on a semi-trusted provider comes with a risk of sensitive data disclosure. This risk can be modeled based on many different factors. In this paper, we model it w.r.t. $sens(\mathcal{D}_{P_i})$, and the risk weight assigned to provider P_i (w_{P_i}). Risk weights are user-defined variables, and reflect a DO's perspective on a provider's reliability. For instance, a trusted provider will have $w_{P_i} = 0$, since it will not try to infer any information from the outsourced data. On the other hand, a semi-trusted provider will have $w_{P_i} \in [0, 1]$, due to its adversarial intent. Thus, smaller values of risk weight imply more trust towards a provider. Moreover, a DO may not trust a provider, but may trust the underlying cryptographic mechanism used. For instance, if data is stored using only a semantically secure encryption scheme, then a DO would be able to assign $w_{P_i} = 0$.

The expected number of sensitive KV entries leaked to an adversary on P_i is calculated as $w_{P_i} \times sens(\mathcal{D}_{P_i})$. The total disclosure is the sum of $w_{P_i} \times sens(\mathcal{D}_{P_i})$ over all providers, and should be less than the disclosure constraint C_{risk}.

Execution Time: We assume that the expected execution time of q on P_i is affected by the number of I/O operations needed ($io(q)$), and the processing capabilities of P_i ($pow(P_i)$); and represent it as $r(q, P_i) := \frac{io(q)}{pow(P_i)}$. Moreover, the expected execution time of a workload $\mathcal{Q}$ is the sum of the execution times of each query $q \in \mathcal{Q}$, and is estimated as $r(\mathcal{Q}, P_i) := \sum_{q \in \mathcal{Q}} f(q) \times r(q, P_i)$.

In a KV store, measuring $io(q)$ will naturally depend on the number of rows, the distribution of rows among nodes of the cluster, and the average number of rows per cluster node. Additionally, [6] points out that having more nodes in a cluster results in better performance. Thus, $pow(P_i)$ is proportional to the number of nodes in a cluster, and the processing power of each cluster node.

Once $\mathcal{Q}$ is partitioned into $\mathcal{Q}_{P_1}, \dots, \mathcal{Q}_{P_k}$, the total execution time of the entire workload is calculated as the sum of the execution times of each partitioned workload.

$$OptRun(\mathcal{Q}_{P_1}, \dots, \mathcal{Q}_{P_k}) := \sum_{P_i \in \mathcal{P}} r(\mathcal{Q}_{P_i}, P_i)$$

B. Formal Definition of MCPP

Given a workload $\mathcal{Q}$, a set of providers $\mathcal{P}$, and a dataset $\mathcal{D}$, MCPP aims to optimize the total execution time, by partitioning $\mathcal{Q}$ and $\mathcal{D}$ over $\mathcal{P}$. A solution to MCPP should satisfy both the monetary and disclosure constraints.

$$\begin{aligned} \text{minimize:} \quad & OptRun(\mathcal{Q}_{P_1}, \dots, \mathcal{Q}_{P_k}) \\ \text{subject to:} \quad & \sum t(\mathcal{Q}_{P_i}, P_i) \leq C_{cost} \\ & \sum w_{P_i} \times sens(\mathcal{D}_{P_i}) \leq C_{risk} \\ & \forall P_i \in \mathcal{P}, \forall q \in \mathcal{Q}_{P_i}, \mathcal{D}_q \subseteq \mathcal{D}_{P_i} \end{aligned}$$

By solving an instance of MCPP, a DO can achieve a balance between monetary costs and security. It can be checked if the expenditure can be reduced without exposing too much sensitive data; or if security can be improved with a tolerable increase in expenditure. While doing so, the performance of the overall system is maximized, irrespective of the constraints. An interested reader is referred to our technical report for a more detailed discussion, where we also show a reduction from the 0-1 Knapsack Problem to MCPP, and thus prove that MCPP is NP-Hard [7].

III. Background

In this section, we give descriptions of BigSecret's building blocks. Then, we provide a brief introduction to HBase, the KV store that is currently supported by BigSecret.

A. Preliminaries

Symmetric Encryption: A symmetric encryption scheme is composed of three polynomial-time algorithms, $SKE := (Gen, E, D)$: (i) Gen takes a security parameter k, and returns a secret key K; (ii) E encrypts plaintext message p with secret key K; (iii) D decrypts ciphertext c using the secret key K. The symmetric encryption scheme used in this work is assumed to be semantically secure [2].

Pseudo-Random Functions: A pseudo-random function (PRF) is a polynomial-time computable function, whose output is indistinguishable from a random function by any polynomial-time adversary. A PRF is composed of two functions $PRF := (Gen, H)$: (i) Gen takes a security parameter k, and returns a secret key K; (ii) H takes a secret key K, a message m, and returns a digest h.

Bucketization: Bucketization $B := (Part, Ident, Map)$ is composed of: (i) a partitioning function $Part$ that takes a domain $\mathcal{Z}$, number of partitions n, and returns disjoint partitions $p_1, \dots, p_n$, where $\bigcup_{i=1}^{n} p_i = \mathcal{Z}$; (ii) an identifier function $Ident$ that assigns unique random identifiers to each partition p_i; (iii) a mapping function Map that takes a partitioned domain, a value v from the domain, and returns $Ident(p_i)$, where $v \in p_i$. In an order-preserving partitioning, $Ident(p_i) < Ident(p_j)$ if $i < j$. We will use the term *Bucket Data* for the partition-to-identifier mapping of a bucketizer.

B. HBase

HBase is an open source, distributed KV store based on Google's BigTable [8]. It is a persistent, and strictly consistent storage system, which uses Hadoop Distributed File System (HDFS) for data storage. HBase consists of unique rows, and each row is composed of KV entries. A KV entry has 5 parts: Row-key (row), family (fam), qualifier (qua), timestamp (ts) and value (val). In a KV entry, the key consists of the first four parts, and is denoted as $KEY := row \| fam \| qua \| ts$. Four operations are provided in HBase: (i) *Put* inserts data into an HBase table; (ii) *Get* retrieves data of a specific row; (iii) *Delete* removes a specific row's data from an HBase table; (iv) *Scan* retrieves a range of rows. The last three operations may also be limited to a specific family, qualifiers, or a certain time range. We denote the operations as follows:

$$\begin{aligned} Q_P &:= (row, (fam, qua, val)^+, ts) \\ Q_G &:= (row, ts_{from}, ts_{to}, [(fam)|(fam, qua)]^*) \\ Q_D &:= (row, ts_{from}, ts_{to}, [(fam)|(fam, qua)]^*) \\ Q_S &:= (row_{from}, row_{to}, ts_{from}, ts_{to}, [(fam)|(fam, qua)]^*) \end{aligned}$$

IV. BigSecret

BigSecret is our proposed framework, which provides provable security for outsourced KV data; and enables efficient query execution on encrypted data. To do so, data is transformed prior to outsourcing using *Encryption Models*, which consist of the cryptographic functions from Section III-A. Moreover, when used in a multi-cloud setup containing any number of trusted or semi-trusted providers, BigSecret automatically boosts the overall performance by using a heuristic solution for MCPP.

In this paper, we base the KV entry structure of BigSecret on HBase, i.e., a KV entry in BigSecret consists of five parts as in HBase. Moreover, BigSecret offers the same set of operations on the outsourced data as HBase. However, the methodology used in BigSecret can be easily adapted to the data model of any other KV store.

In this section, we begin by presenting BigSecret's architecture. Then, we explain how data is stored on semi-trusted providers using encryption models. Finally, query translation is explained; followed by a discussion on the heuristic used to solve MCPP.

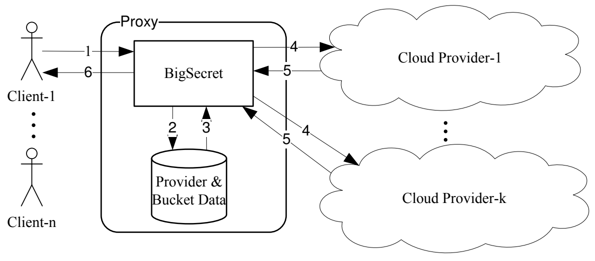

Figure 1. BigSecret Architecture

A. Architecture

Figure 1 shows an outline of the BigSecret architecture. It consists of providers that a DO utilizes, the BigSecret application, and a set of clients that want to perform queries on the outsourced data.

Given a dataset $\mathcal{D}$ and a workload $\mathcal{Q}$, a DO uses BigSecret to distribute $\mathcal{D}$ and $\mathcal{Q}$ over the set of providers. BigSecret solves this partitioning problem (MCPP) using the heuristic approach discussed in Section IV-D. During the partitioning, BigSecret may need to access two different sets of data: (i) *Provider data* that contains information on a provider's parameters (e.g., risk weight, selected encryption model for that provider); (ii) *Bucket data* if bucketization is used. These sets of data are kept on a proxy, which also hosts the BigSecret application. This proxy is placed on a trusted platform, such as a private cloud or a trusted public cloud.

Once partitioning is performed, clients will start interacting with providers via the BigSecret proxy, rather than communicating directly, as shown in Figure 1. When a client wants to issue a query on the outsourced data, the query is first sent to the BigSecret proxy (step 1). BigSecret translates it into its encrypted form using bucket data and provider data (steps 2-3), and sends it to the provider(s) (step 4). BigSecret retrieves a set of results in encrypted form (step 5), decrypts, filters, and returns the final set of results to the client (step 6). The client does not perform any operation, and is oblivious to any security measure taken by BigSecret.

The BigSecret application handles all cryptographic processes and makes sure that the correct set of results are returned to the client. Since all communication passes via the proxy, one may question if this creates a bottleneck. It is obvious that BigSecret is horizontally scalable, i.e., many independent instances of BigSecret can run in parallel to support multiple clients. Moreover in our experiments, we show that increasing concurrent client requests does not incur any more overhead than communicating directly.

B. Encryption Models

In BigSecret, we make use of crypto indices to perform Delete, Get, and Scan operations over encrypted data. We mainly utilize bucketization and PRF's as crypto indices. Primarily, both of them allow us to process a query by checking/retrieving a very small subset of the entire dataset.

Due to its small collusion probability, the use of PRF's as a crypto index results in less false positives, and communication overhead, when compared with bucketization. However, since a PRF's output is distributed (pseudo) randomly over its range, order is not preserved; thus Scan cannot be performed over encrypted data. But as proven in Section V, its (pseudo) randomness enables building provably secure constructions. On the other hand, one must use bucketization to support Scan queries. A bucketizer's limited order preserving property enables Scan queries to be executed over encrypted data. However, a bucketizer's range is not as large as a PRF's range; and results in more false positives compared to PRF's. Moreover, we end up with bucket data that needs to be stored on the proxy.

For any value v, a crypto index is either $Map(v)$, $H(v)$, or a constant value; and is denoted as $I(v)$. Then for a KV entry $e := (row, fam, qua, ts, val)$ in BigSecret, *encryption model* is the definition of the entry's translation to $E(e) := (I(row), I(fam), I(qua) \| E(KEY), I(ts), E(val))$.

	Model-1	**Model-2**	**Model-3**
row	$Map(row)$	$H(row)$	$H(row)$
fam	$Map(fam)$	$H(fam)$	0
qua	$Map(qua) \| E(KEY)$	$H(qua) \| E(KEY)$	$E(KEY)$
ts	$Map(ts)$	$H(ts)$	1
val	$E(val)$	$E(val)$	$E(val)$

Table I
ENCRYPTION MODELS

Encryption models form the core of BigSecret, since they allow (i) securing data on semi-trusted providers, and (ii) performing queries on encrypted data. Table I gives an overview of the models we have used. It shows how a key part is encrypted using the security tools from Section III-A. It should be noted that for each E and H, *a different cryptographic key* is used. For instance, for Model-2, encryption key used in $E(KEY)$ is different from the one used in $E(val)$. Similarly, a *different* partitioning is performed for each key-part's bucketizer.

Model-1 uses bucketization as the crypto index for all key parts. This model should only be used if Scan queries need to be performed over encrypted data. An important factor in this model's performance is the number of buckets in the row bucketizer. Having less buckets results in a large number of rows being mapped to the same bucket, thus increasing false positives. The same is also true with bucketizers for the other key parts. However, the row bucketizer has a larger impact on the performance as compared to the other bucketizers.

Model-2 uses a PRF as the crypto index, and is very suitable for workloads without Scan queries. Utilization of PRF's on all key-parts decreases the number of false positives drastically as compared to Model-1, so a Get or Delete query can be executed very efficiently over encrypted data. As in Model-1, the range of the PRF used for the row-key has the most impact on performance as compared to PRF's for the other key parts.

Model-3 uses a PRF as the crypto index, but only for the row-key. The motivation for doing so is to reduce the sensitivity disclosure arising from multi-dimensional indexing [9]. Fixed values are given to family and timestamp. As before, the range of the PRF on the row-key has a major effect on the performance.

C. Query Translation

Given a query $q \in \{Q_P, Q_G, Q_D, Q_S\}$ and an encryption model $m \in \{m_1, m_2, m_3\}$, we define a translation function $T(m, q) := q^*$.

Put: Given $q \in Q_P$, translation consists of applying the necessary cryptographic operations based on m. For each encryption model m_i, the translated query $q^* \in Q_P$ is:

$$T(m_i, q) := (I(row), (I(fam), I(qua)||E(KEY), E(val))^+, I(ts))$$

Get: For $q \in Q_G$, the translated query $q^* \in Q_G$ changes w.r.t. m_i. For Model-1, we calculate index values for all key parts as follows:

$$T(m_1, q) := (I(row), I(ts_{from}), I(ts_{to}), [I(fam)|(I(fam), I(qua))]^*)$$

For Model-2, index values for timestamp parameters are not translated, since order is not preserved. Instead, q^* is issued over the entire timestamp range, i.e., $[0, LongMax]$:

$$T(m_2, q) := (I(row), 0, LongMax, [I(fam)|(I(fam), I(qua))]^*)$$

Finally for Model-3, we use our only index ($I(row)$) to translate q into q^*:

$$T(m_3, q) := (I(row), 0, LongMax, null)$$

Delete: For $q \in Q_D$, translation is similar to the translation for Get. We need to retrieve a subset of data, decrypt it to see which KV entries need to be actually deleted. Then, matching entries are marked with a deletion request and sent to the providers.

Scan: Given $q \in Q_S$, translation is only possible for Model-1. Moreover, processing q over encrypted data consists of several steps: (i) First q is translated to q^*. (ii) Then q^* is issued on all possible providers, and an initial set of results is retrieved. (iii) The retrieved data is decrypted, and false-positives are discarded. (iv) Finally, the actual set of results is given to the client row-by-row in sorted order. Translation of q to $q^* \in Q_S$ for Model-1 is as follows:

$$T(m_1, q) := (I(row_{from}), I(row_{to}), I(ts_{from}), I(ts_{to}), [I(fam)|(I(fam), I(qua))]^*)$$

D. Heuristic Approach to MCPP

Given a dataset $\mathcal{D}$, a workload $\mathcal{Q}$, a set of providers $\mathcal{P}$, and constraints C_{cost} and C_{risk}, the heuristic approach used in BigSecret aims to minimize the total execution time by using a *Hill-Climbing Technique*. By iterating over each query, we check if a better overall performance can be achieved by moving it to another provider, while satisfying the constraints. This process goes on until no further improvements can be made to the total execution time. An interested reader is directed to our technical report for a more detailed explanation [7].

V. Security Analysis of Encryption Models

A. Model-1

The security of using bucketization as a crypto index is discussed in [4]. Hore et al. state that variance and entropy in a bucket are major factors in limiting sensitive data disclosure. Increased variance and entropy in a bucket decreases sensitive data disclosure by making it harder for an attacker to determine specific values in a bucket.

B. Model-2

PRF's as crypto indices have been investigated in [9] in terms of security and privacy. Damiani et al. point out that using PRF's will flatten the data distribution, thus making it harder for an attacker to gather specific information about the dataset. Decreasing the range of a PRF to a smaller size ensures that many values end up having the same hash value, which increases entropy and variance as in bucketization.

C. Model-3

We adapt the simulation based adaptive security definition described in [2], which tolerates leakage of access and search patterns to an adversary. In our security proof, we assume that each row has only one KV entry. Since only one KV entry will be read at a time for a query, where each KV entry is computationally indistinguishable from one another, access pattern does not reveal any information to the adversary. We assume the dataset $\mathcal{D}$ has n KV entries, and I is the index on the row-key part for Model-3, (i.e., $I := H(row_i)$ for $1 \leq i \leq n$). We now provide some preliminary definitions for our security proof.

History: A q-query history over $\mathcal{D}$ is a tuple $H := (\mathcal{D}, \mathbf{w})$ that includes the dataset $\mathcal{D}$ and the set of queried row-keys $\mathbf{w} := (w_1, \ldots, w_q)$.

Search Pattern: The search pattern induced by the history H is a q-by-q symmetric matrix $\sigma(H)$, such that $\sigma[i, j] := 1$ if $q_i = q_j$, and 0 otherwise.

Trace: The trace induced by the history H is a sequence $\tau(H) := (\sigma(H), |val_1|, |KEY_1|, \ldots, |val_n|, |KEY_n|)$. Trace is the data leaked to an adversary, and consists of the search pattern, and the lengths of each KV entry.

View: View induced by the history H is a sequence $v(H) = (I, I(w_1), \ldots, I(w_q))$. View is the data that is accessible to an adversary, consisting of the index of each row-key, and index values of queried row-keys.

Then, we have the following definition for adaptive security from [2].

Definition 1. *A symmetric searchable encryption scheme is adaptively semantically secure, if there exists a probabilistic polynomial time simulator S that can adaptively simulate an adversary's view of the history from the trace with* *probability negligibly close to 1. More formally, for any polynomial size distinguisher D, for all polynomials $poly$ and a large r:*

$$Pr[D(v(H)) = 1] - Pr[D(S(\sigma(H))) = 1] < \frac{1}{poly(r)}$$

Theorem 1. *Let H be a secure PRF, and E be a semantically secure encryption function, then BigSecret using Model-3 (BS-3) satisfies Definition 1, and is adaptively secure.*

Proof: It suffices to show a polynomial size simulator $\mathcal{S} := (S_0, \ldots, S_q)$ for all adversaries $\mathcal{A} := (A_0, \ldots, A_q)$.

S_0 needs to generate an artificial dataset $\mathcal{D}^*$, such that the number of KV entries is n, and the length of the entries are identical to the entries in $\mathcal{D}$. For each KV entry $e_i := (H(row_i), 0, E(KEY_i), 1, E(val_i))$ in $\mathcal{D}$, S_0 creates three unique random strings r_i, y_i, and z_i, such that $|r_i| = |H(row_i)|$, $|y_i| = |KEY_i|$, and $|z_i| = |val_i|$. Furthermore, it is important that r's are also unique amongst each other. Finally, S_0 sets the i^{th} KV entry for the artificial dataset as $\mathcal{D}^*[i] := (r_i, 0, E(y_i), 1, E(z_i))$.

Since H is a secure PRF, and E is a semantically secure encryption function, $\mathcal{D}$ and $\mathcal{D}^*$ are indistinguishable from one another. Otherwise, one could prove that E or H is not secure, which contradicts our assumption.

Then, an adversary starts querying the simulator, by keeping its current state in a binary string st_A. When A_i asks w_i to S_i, S_i checks if that row-key was asked before. If not, it randomly picks a KV entry $\mathcal{D}^*[k]$; returns y_k, z_k to A_i, and updates st_A for the corresponding $(\mathcal{D}^*[k], w_i)$ tuple. Otherwise, it checks st_A and returns the same information that was returned before.

Since y_i and z_i are distinguishable from the real world KV entry with negligible probability, the adversary cannot distinguish simulated responses from real world responses. Otherwise, our assumption would be wrong as mentioned previously. Thus, $BS-3$ satisfies conditions in Definition 1, and is semantically secure against adaptive adversaries. □

VI. Experiments

In this section, we start by describing our experimental setup, followed by experiments on (i) the performance of encryption models, (ii) varying number of concurrent clients, (ii) varying number of buckets for Model-1, and (iv) a multi-cloud setup with different C_{cost} and C_{risk} values.

Experimental Setup: We conducted experiments on a cluster of 11 nodes, where a node consists of a Pentium IV processor with ≈ 290GB-320GB disk space and 4GB of main memory. To perform experiments, we used Hadoop v1.0.4, HBase v0.94.2, and Yahoo! Cloud Serving Benchmark (YCSB) v0.1.4 [6]. Each experiment is executed five times, and the average is taken as the final result.

For the multi-cloud setup, we leveraged the same cluster twice by storing data in two different forms. P_1 is the provider, where data is stored in plaintext format (without any encryption); P_2 is the provider, where data is stored using Model-1. The choice of using Model-1 and plaintext storage is based on our observations from the performance of the encryption models. The results in Section VI-A show that Model-2 and 3 perform with at most 10% overhead when compared with plaintext storage, while Model-1 always has the worst performance. In such a situation, we check if the overall performance can be improved by using Model-1 along with plaintext storage in a multi-cloud setup.

Dataset and Workload: To measure performance with different table sizes, we created six tables having 1, 2, 4, 8, 16, and 32 million rows. Each row consists of 10 KV entries, where each entry is 100B. Each table has 4 different copies: one for each encryption model, and one in plaintext form.

We defined three workloads with different query frequencies, where each workload consists of 100K queries for the multi-cloud experiments, and 1K queries for all other experiments: (i) Workload-1 contains 5% Put and 95% Get queries; (ii) Workload-2 is 95% Put and 5% Get; (iii) Workload-3 consists of 25% Put, 25% Get and 50% Scan queries, where each Scan query's range is 100 rows.

Monetary Cost: Cost metrics used in our experiments are calculated based on Amazon S3, EC2 and EMR pricing. The price for a Put and Get operation is $\$0.01/1000 requests$ and $\$0.01/10000 requests$ respectively. The storage, communication, and processing cost is $\$0.14/GB$+PUT, $\$0.12/GB$+GET and $\$0.1/hour$ respectively. We calculated the total expected monetary cost, when the entire dataset and workload is given to P_1 as ≈ \$700, and when they are given to P_2 as ≈ \$3700. The difference in costs is due to the overhead associated with Model-1. For a given query, more data will be transferred each time from P_2 to BigSecret, which results in a larger processing cost.

We varied C_{cost} between \$700 and \$3700 as a fraction of \$3000. We set $C_{cost} := 700 + rate * (3700 - 700)$, where $rate \in \{0\%, 20\%, 40\%, 60\%, 80\%, 100\%\}$.

Sensitivity Disclosure: We performed the multi-cloud experiments on the dataset with 32M rows, and assumed that all entries are sensitive. We then defined C_{risk} as a fraction $(0\%, 20\%, 40\%, 60\%, 80\%, 100\%)$ of the $320M$ KV entries.

For the risk weights, we assigned $w_{P_1} := 1$, since data will be in plaintext mode. Any sensitive data placed on P_1 will be captured by an adversary. We estimated $w_{P_2} := 0.7$ based on the results obtained in [9]. When indexing is performed on all attributes of a table (four key parts in our case), the ratio of disclosed data converges to 0.7 as the number of entries increases. Since risk weights are user-defined values, $w_{P_2} := 0.7$ is an acceptable value, and reflects a DO's concerns about any possible sensitivity disclosure from the bucketizers on P_2.

Other Parameters: AES256 in CTR mode is used as SKE, while HMAC-SHA-256 is used as a PRF. For each key part, we created bucketizers with 2^{16} buckets. Moreover, for the row-key part, we created three other bucketizers having

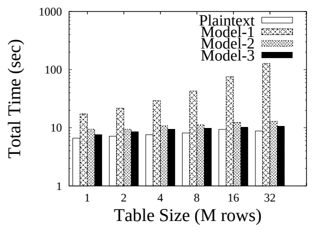

Figure 2. Workload-1

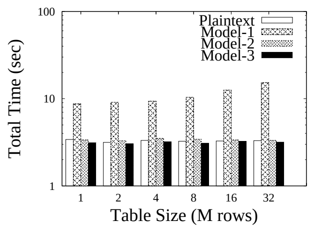

Figure 3. Workload-2

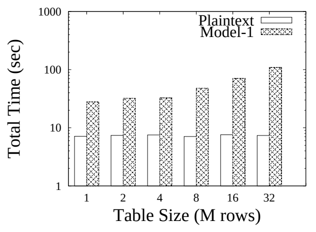

Figure 4. Workload-3

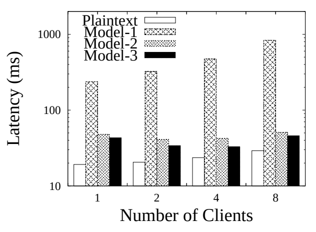

Figure 5. Workload-1

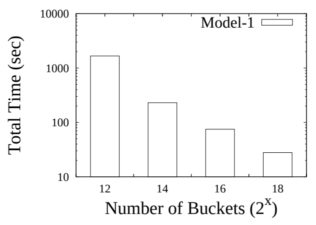

Figure 6. Workload-1

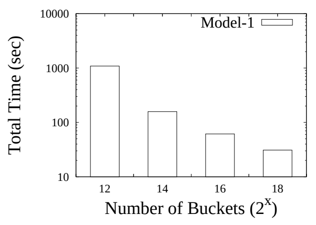

Figure 7. Workload-3

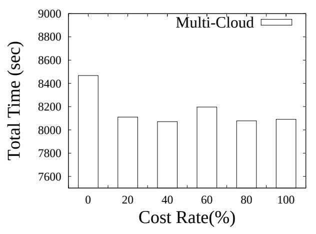

Figure 8. Workload-3

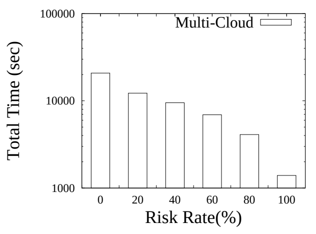

Figure 9. Workload-3

$2^{12}, 2^{14}$, and 2^{18} buckets each.

To calculate $pow(P_1)$ and $pow(P_2)$, we ran Workload-3 with 10K queries over the 32M-row table, and calculated the total I/O operations performed. For each operation, the input size is the table size, while the output size depends on the query type. Finally, we calculated $pow(P_1) = 4.224 \times 10^8$ io/sec and $pow(P_2) = 3.12 \times 10^8$ io/sec.

A. *Workload Experiments*

For all experiments in this subsection, we used 4 clients to send queries to BigSecret concurrently. We varied the number of rows from $1M$ to $32M$ to observe the effect of data size on each model and plaintext storage performance.

Figures 2, 3 and 4 show that Model-2 and 3 perform very similar to plaintext storage, with at most 10% overhead. This is due to these models' low false positive ratios. PRF as a crypto index gives results that are almost as efficient as plaintext storage. On the other hand, Model-1 performs the worst due to its high false positive ratio, and has a linear growth in total execution time as the number of rows increases linearly for all workloads.

B. *Number of Clients*

The aim of this set of experiments is to show that even an increased number of concurrent clients results in latency values that are similar to plaintext storage. We show results only for Workload-1, since it can be executed on all encryption models, and has a high Get query ratio. Figure 5 shows that latency values increase as the number of concurrent clients doubles. Model-1's latency is higher compared to Model-2 and 3, while Model-3 has a lower latency than Model-2 due to its use of a lesser number of cryptographic operations.

C. *Number of Buckets*

Experiments are performed on the 32 million rows table, with 4 concurrent clients. We varied the number of buckets only for the row-key bucketizer, since the number of rows (incurred by the row-key bucketizer) has the most impact on performance. Figures 6 and 7 show results for Workload-1 and 3 respectively. We observe that increasing the number of buckets decreases the total execution time exponentially. The reason is that as the number of buckets increases, a lesser number of actual rows end up having the same bucket identifier. Then, for any Get or Scan query, a lesser number of KV entries are requested by BigSecret.

D. *Multi-Cloud Experiments*

The multi-cloud setup experiments are performed with a single client for the 32M-rows table. Figures 8 and 9 show the results of our experiments on Workload-3. Queries are partitioned over P_1 and P_2 using the heuristic approach based on varying cost and risk values.

Figure 8 shows that for our multi-cloud setup, C_{cost} does not have a major impact on the total execution time. As it changes from 0% to 100%, the total execution time fluctuates between $8450sec$ and $8070sec$. The reason is the way we constructed our multi-cloud setup. For a constant C_{risk} value, increasing C_{cost} means that more queries can be executed on P_2. However, the algorithm will not assign more queries to P_2, since this will reduce the overall performance. On the other hand, we observe from Figure 9 that the total execution time decreases rapidly as we allow more sensitive data disclosure. This is an expected situation, since as C_{risk} increases, more queries will be executed on P_1, which improves the total performance by decreasing the total execution time.

E. Discussion

We observe that Model-2 and 3 both perform similar to plaintext storage, with an overhead of at most 10% in most cases. Thus, a DO should use one of these models for a workload without any Scan query. On the other hand, if a DO wants to perform Scan queries over outsourced data, Model-1 has to be used, which does not perform well as the table size gets larger. In such a scenario, it would be better to use a multi-cloud setup with an allowed data disclosure between $40 - 80\%$ and a monetary constraint of 50%. To further decrease the total execution time, Model-1 can be used with an increased number of buckets for the row-key.

Finally, an interested reader is referred to our technical report for experiments with additional workloads [7].

VII. Related Work

The problem of data and workload distribution among trusted and untrusted servers was previously investigated in [10], [11]. Their aim was to optimize performance by finding the maximum workload that can be executed on untrusted servers. Unlike our work, they do not consider sensitive data disclosure as a part of their data and workload distribution.

[12] uses a risk based approach, where sensitive data disclosure is examined for single machine architectures under memory attacks. However, this work considered relational databases; thus it is not directly applicable to a multi-cloud setup with KV stores.

[13] performs optimization for query and data partitioning, with sensitive data disclosure and monetary cost constraints. This work concentrates on a hybrid cloud setup, and aims to optimize the performance over relational databases. Our work differs from [13] in terms of data model and cloud setup. We consider a more general problem, a multi-cloud setup, while their work is limited to a hybrid setup.

Data partitioning has been investigated in [14], [15], in terms of relational databases, and Map-Reduce. However, they are not directly applicable in our scenario due to differences in data model and cloud setup.

VIII. Conclusions and Future Work

In this paper, we proposed *BigSecret*, a secure data management framework for KV stores. We showed three models that allow outsourcing of encrypted data with efficient processing capabilities, and provided a proof of security for one of these models. We formalized the data and workload partitioning problem over a multi-cloud setup with sensitive data disclosure and monetary cost constraints, and integrated a heuristic solution to this problem within BigSecret. By empirical evaluations, we showed the benefits of our approach, and the performance of our models.

In our experiments, we used a static dataset. Furthermore, we assumed that the query workload is given to us before partitioning is performed. An interesting future work would be to manage a dynamic dataset, and to overcome a dynamic workload scenario.

Finally, in our description of BigSecret, we say that it can be transformed easily to operate over a different KV implementation. In the future, we plan to support additional KV stores other than HBase in BigSecret.

IX. Acknowledgments

This work was partially supported by Air Force Office of Scientific Research MURI Grant FA9550-08-1-0265 and FA9550-12-1-0082, National Institutes of Health Grants 1R01LM009989 and 1R01HG006844, National Science Foundation (NSF) Grants Career-CNS-0845803, CNS-0964350, CNS-1016343, CNS-1111529, CNS-1228198 and Army Research Office Grant W911NF-12-1-0558.

References

[1] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, “Dynamo: amazon’s highly available key-value store,” in *SIGOPS*, 2007.

[2] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, “Searchable symmetric encryption: improved definitions and efficient constructions,” in *CCS*, 2006.

[3] M. Kuzu, M. S. Islam, and M. Kantarcioglu, “Efficient similarity search over encrypted data,” in *ICDE*, 2012.

[4] B. Hore, S. Mehrotra, and G. Tsudik, “A privacy-preserving index for range queries,” in *VLDB*, 2004.

[5] V. Khadilkar, K. Y. Oktay, M. Kantarcioglu, and S. Mehrotra, “Secure data processing over hybrid clouds,” *IEEE Data Eng. Bull.*, vol. 35, no. 4, pp. 46–54, 2012.

[6] B. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, “Benchmarking cloud serving systems with ycsb,” in *SoCC*, 2010.

[7] E. Pattuk, M. Kantarcioglu, V. Khadilkar, and H. Ulusoy, “Bigsecret: A secure data management framework for key-value stores,” Tech. Rep., 2013. [Online]. Available: http://www.utdallas.edu/~exp111430/techReport.pdf

[8] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, “Bigtable: A distributed storage system for structured data,” *TOCS*, 2008.

[9] E. Damiani, S. Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati, “Balancing confidentiality and efficiency in untrusted relational dbmss,” in *CCS*, 2003.

[10] H. Hacigümüs, B. Hore, and S. Mehrotra, “Privacy of outsourced data,” *Encyclopedia of Cryptography and Security*, pp. 965–969, 2011.

[11] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra, “Executing sql over encrypted data in the database-service-provider model,” in *SIGMOD*, 2002.

[12] M. Canim, M. Kantarcioglu, B. Hore, and S. Mehrotra, “Building disclosure risk aware query optimizers for relational databases,” *VLDB*, 2010.

[13] K. Oktay, V. Khadilkar, B. Hore, M. Kantarcioglu, S. Mehrotra, and B. Thuraisingham, “Risk-aware workload distribution in hybrid clouds,” in *IEEE CLOUD*, 2012.

[14] F. Afrati, V. Borkar, M. Carey, and N. Polyzotis, “Map-reduce extensions and recursive queries,” in *EDBT*, 2011.

[15] T. Nykiel, M. Potamias, C. Mishra, G. Kollios, and N. Koudas, “Mrshare: Sharing across multiple queries in mapreduce,” *VLDB*, 2010.