SLiCER: Safe Long-term Cloud Event Archival

Adriano Serckumecka Ibéria Medeiros Bernardo Ferreira Alysson Bessani
LASIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal

***Abstract*—Security Information and Event Management (SIEM) systems have been adopted by organizations to enable holistic monitoring of malicious activities in their IT infrastructures. SIEMs receive events from several devices of the organization's IT infrastructure (e.g., servers, firewalls, IDS), correlate these events, and present reports for security analysts. Given the large number of events collected by SIEMs, it is costly to store such data for long periods. Besides, since organizations store a relatively limited time-frame of events, the forensic analysis capabilities severely become reduced. We present SLiCER an archival system for long-term storage that makes use of multi-cloud storage to guarantee data security, low cost and high scalability, and ensures cost-effectiveness by grouping events in blocks and using indexing techniques to recover them. The system was evaluated using a real dataset, and the results show that it is significantly more cost-efficient than competing alternatives.**

***Index Terms*—Security, long-term archival, indexing, multiple clouds, cost-effectiveness**

I. Introduction

Security Information and Event Management (SIEM) systems provide a centralized point of analysis for correlating security-related events generated by a multitude of devices (firewall, IDS, IPS, anti-virus, syslog, etc.). One of the essential benefits of having such systems is the possibility of accessing events in retrospect to investigate and understand why and how security incidents happened.

Modern SIEMs archive collected events in local databases and for short periods, varying from a week to one year [1], [2]. This time limitation is due to storage requirements since large organizations collect thousands of events per second. Accommodating such large datasets is not only a matter of buying more storage space but also of managing the effort of dealing with such data, which is also of critical and private nature. Additionally, such large datasets create a need for efficient data search and retrieval mechanisms, which are not usually supported in traditional SIEM systems.

The consequence is limited support for forensic activities, especially in situations where incidents could only be explained by already deleted events. For instance, some studies show that specific advanced security threats exploring zero-day vulnerabilities take on average 320 days until being discovered [3], and there are cases in which a certain threat was exploring systems for as long as nine years [4]. Furthermore, many countries have data protection laws that require the exact date when a breach of sensitive data was discovered to notify the affected parties promptly. As such, some standardization bodies have started recommending organizations to store events for up to five years [5], [6].

An alternative for increasing the SIEM event storage capacity is the use of public cloud storage services, such as Amazon S3 and Azure Blob Storage. The costs seem promising: one GB costs USD 0.01 per month in S3 [7], which means that a 10TB event database costs USD 1200 per year, with all the backup and security management being performed by the cloud provider. However, many SIEM users still have concerns about moving this sensitive data to a third-party provider that needs to be completely trusted. Such concerns can be alleviated by encrypting data and spreading it to several cloud providers [8], but then the problem is how to search over such encrypted and distributed events. Searchable Symmetric Encryption (SSE) (e.g., [9]–[11]) deals with this problem, allowing encrypted data to be searched, however, it requires a remote server in the cloud to be efficient. The additional cost and management effort associated with this server may not make it the most appropriate technique for a SIEM archival receiving few queries per week.

Instead of exploiting SSE, in this paper, we propose a simpler and more pragmatical approach for implementing such archival. In our solution, this data is organized and indexed before being uploaded to the cloud in such a way that queries can be processed at the client-side, hence removing the need for remote server processing. We employ cloud-of-clouds storage system [12] for dispersing data through multiple cloud providers, ensuring data privacy, integrity, and availability. Events are organized in blocks according to the time periods and devices that generated them. These blocks are stored in the cloud-of-clouds together with their indexes. This organization helps to recover events fast and cheap.

This approach was implemented in a system called SLiCER (*Safe Long-term Cloud Event aRchival*). SLiCER runs together with the SIEM, on the organization premises, collecting events for storage on the cloud, and supporting queries on such archival. Besides security (ensured through encryption and the use of cloud-of-clouds storage), the most important requirement of our proposal is its cost-effectiveness. As mentioned before, cloud storage space is cheap, but reading data and performing requests from the cloud can be very expensive. Therefore, to minimize the costs of using cloud storage services, we propose a domain-specific data and query model that enables high rates of compression and associates small indexes to event blocks, decreasing the amount of data stored and retrieved. We evaluate SLiCER experimentally, using 14.4GB of data obtained from a power utility company, and comparing our system with both a simplistic cloud archival system and an SSE-based solution, showing our approach is substantially more efficient and cost-effective than these alternatives.

In summary, the contributions of this paper are:

1) A cloud-backed archival data model and query strategy for SIEMs, which enable efficient storage and retrieval of events in terms of operation cost and performance. This model, although designed for SIEMs, can be easily extended for other logging systems of critical nature;
2) SLiCER, a system that implements this model, organizes events in data blocks, indexes them, and uses these indexes to recover data stored in the clouds;
3) An experimental evaluation using real SIEM events comparing SLiCER with competing solutions.

II. Related Work

To the best of our knowledge, no previous work tried to solve the problem we address in this paper: efficient and secure archival of SIEM events in public clouds. However, several initiatives broadly related to ours deserve mentioning. Some preliminary ideas that lead to this paper appeared in [13].

a) SIEMs in the cloud: There are many analytics and archival systems currently available that can be used as SIEMs, e.g., Splunk [14], Solr [15], Elastic [16], to cite just a few. These systems aim to provide an analytics platform, offering features to generate alerts and reports, becoming closer to a SIEM. Elastic, for instance, employs Apache Lucene [17] as its background indexing engine, which creates inverted indexes for the stored documents (or events), used to perform keyword searches. Splunk is very similar, aggregating a few different functionalities such as a correlation engine and the use of a proprietary scheme to index the data.

Many of these systems are also offered as a cloud-managed deployment. The companies maintain an online infrastructure in the cloud and charge the user according to the volume of events ingested by the system. These systems could be used for archival but would be an expensive tool as they offer a richer set of features. SLiCER does not aim to offer the full capabilities of a SIEM or data analytics engine. Instead, it provides only long-term event archival in the cloud, with maximum security and minimum operational costs.

b) Efficient forensics in archival data: VAST [18] is a forensic analytic platform to capture and retain network activities, storing and providing queries over the historical network data. It has some similarities to Elasticsearch [19] as a distributed platform with indexing and searching capabilities, however, it uses a different approach to index data, employing bitmap indexes. Although the indexes employed in VAST are quite efficient, they significantly increase the amount of storage: even with compression, VAST indexes require 37% of the size of the raw stored data.

Succint [20] also compresses data using flat files, not requiring any index or decompression before searching. The system can achieve a compression ratio of 1.25-3$\times$ related to original data, but the preprocessing is quite expensive.

As we show in Section VI, SLiCER employs much smaller indexes than VAST and achieves a compression ratio much higher than Succint.

c) Searchable Symmetric Encryption: Searchable Symmetric Encryption (SSE) [9]–[11] deals with the problem of how to securely and efficiently search a remotely-deployed encrypted database. As such, it could also be applied to store and query SIEM events, however, it is a generic approach that is not designed to consider the specific details of SIEMs (i.e., large number of small events, long term archival, and low frequency of queries) and minimize its financial costs when deployed in the cloud.

In general, SSE approaches require a client, a remote processing server, and remote storage (which is usually the low-latency volatile memory of the server). The client preprocesses the data for creating encrypted tokens that either store or enable searching over it. The server processes these tokens and respectively updates or queries an inverted index kept in the storage, returning results to the client. It means that, in a cloud deployment, SSE schemes require not only storage but also computation (e.g., an AWS EC2 instance), which leads to increased operational costs.

In Section VI, we compare SLiCER with state-of-the-art SSE and show that our approach is significantly cheaper for long-term infrequently-accessed data archival.

d) Cloud-of-Clouds archival: There is a plethora of works that advocate the use of multiple cloud storage services for keeping data with security and fault tolerance. Of particular interest are the data-centric solutions such as RACS [21], DepSky [8], or any of their follow ups [12], [22], [23], which do not require any server in the cloud. These systems employ techniques such as erasure codes, secret sharing, and quorum replication to disperse encrypted data on several clouds to ensure stored data survive loss, corruption, leakage, and vendor lock-in, as long as more than two-thirds of the providers are still correct. In the end, the storage overhead can reach up to 50% (when a single fault is tolerated) [8], [12], but the cloud storage costs depend on the price of of the providers used in the cloud-of-clouds. Currently, there are many setups where the cost is similar to or even lower than using a single top-grade provider such as AWS.

SLiCER builds on these systems and complements them by providing a client-side data processing engine that allows content-based querying on stored data.

III. The SLiCER Approach and System

SLiCER aims to overcome and address the storage constraints related to current SIEMs, providing a way to retain security events for long periods by leveraging low-cost cloud storage services in a secure way. In this context, security means preserving the privacy, integrity, and availability of stored events, and the privacy of submitted queries. These properties are ensured by standard encryption and the use of Byzantine replication through multiple cloud providers [8], [12] in the underlying storage.

SLiCER can store events[1] generated from different sources. The core of the system is responsible for organizing the

[1]Which are small *documents* in the information retrieval terminology.

received events for efficient storage and retrieval in the clouds, both in terms of performance and financial costs. Secure storage is fulfilled by using a cloud-of-clouds storage system [12], and secure information retrieval is done by retrieving encrypted indexes and searching over them on site.

In the following, we present the SLiCER architecture, its two core components and their data and query models, and the indexing methods supported in the system.

A. Architecture Overview

The simplest way to design a long-term event archival system would be to store all events in the cloud without local processing, i.e., as they are received and each as a unique file. On the one hand, this solution would allow SIEM operators to retrieve events individually as they are needed. However, it might generate a large number of PUT and GET requests to the cloud, making it an infeasible solution. For example, an infrastructure generating 1000 events per second would spend almost USD 13000 monthly only in PUT requests to S3 [7], and the high number of GET requests to search these events would also make this solution extremely expensive.

Considering these issues, SLiCER organizes events in blocks according to time periods and devices originating them. SLiCER is composed of three main components, namely, *Event Manager*, *Query Manager*, and *Cache*. Figure 1 shows SLiCER's general architecture and its interaction with the SIEM infrastructure and public clouds.

a) Event Manager: This component is responsible for receiving and processing events from the SIEM infrastructure following the data model detailed in Section III-B, which organizes and prepares events before they are stored in the clouds. In a nutshell, events are organized in blocks, and an index is created for each one. Afterwards, block and index are sent encrypted to the clouds.

b) Query Manager: This component allows security analysts to access the events stored in the clouds. It implements our query model (detailed in Section III-C) and permits the execution of queries over the indexes and respective event blocks, resorting for that to the *Cache* module and the clouds. Lastly, it extracts and returns the matched events to the analyst.

c) Cache: This comprises the local storage where recently-fetched indexes and data blocks downloaded from the clouds are stored, as well as those newly generated. This component allows users to execute queries downloading only the missing files, reducing thus the operation and data transfer costs of the system. Using a cache component is quite important for two reasons. First, indexes can be quite small, which means that a large number of them can be kept in the cache (or even all of them) to save accesses to the cloud, and consequently to save time and cost of query execution. Second, it is common in forensics to start with more general queries that are progressively refined, so it is important to have follow-up queries processed locally as often as possible.

SLiCER interacts with the clouds through the Charon system [12], which is an implementation of the cloud-of-clouds concept: Charon distributes data through different clouds to ensure confidentiality, scalability, and security [8], [22]. It works as a mapped directory between the local system and the clouds, where sending data to the different clouds is a transparent step to the analyst.

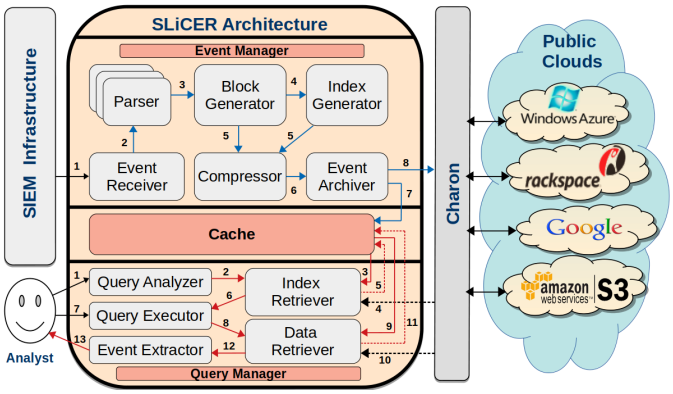

Fig. 1. The SLiCER system architecture.

Lastly, as we stated before, SLiCER was designed having in mind the storage limitations of SIEMs. However, it can also be applied in other scenarios, such as systems that generate logs or events with sensitive information or to extend the forensic analysis time range.

B. Event Manager and Data Model

This section presents the event manager component in detail and the data model it implements.

1) Data Model: The data model we propose aims to strike a balance between financial cost and performance, reaching (1) low cost for data upload and storage, (2) low cost for retrieving data from the cloud, and (3) acceptable query performance.

The data model is based on blocks of events and their indexes. Events are organized in blocks by device and time, which are indexed following a predefined method (see Section III-D). This data model generates fewer files to manage and retrieve than using individual events and, consequently, fewer requests to the cloud (satisfying (1) and (2)). In addition, events generated by the same device are stored in the same block, improving the compression ratio in these blocks (as they tend to be similar), and query performance, since in our query model, queries are executed over events of devices for a given time range (satisfying (3)).

A data block can be defined as a collection of events that belongs to a device in a time window (e.g., 1 day) or having a maximum raw size (e.g., 100MB). Therefore, a block is formed by events received in that time window or that reached the defined size. These two parameters are configured by the user and should be carefully defined because small data blocks affect the compression ratio and lead to an increase in the number of cloud requests. In contrast, large data blocks can negatively impact the cost of executing queries, as more unneeded events may be downloaded with each block retrieval.

An index of a data block is a file that maps event fields to their values in the block and allows queries to be executed more efficiently. Our data model supports two types of indexes: inverted index and Bloom filters. The system can also index

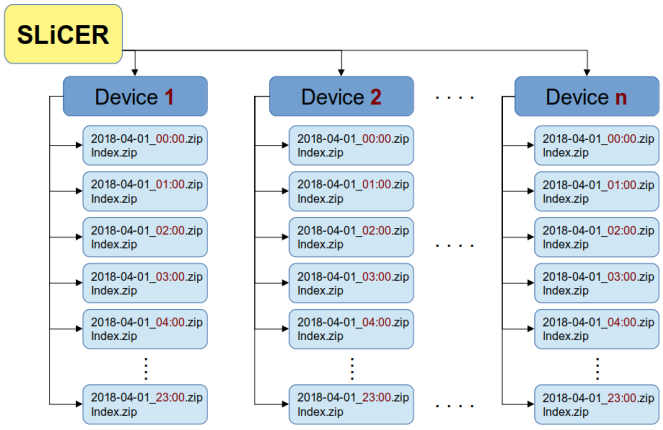

Fig. 2. SLiCER data model.

Algorithm 1: SLiCER query processing.

```
1  Function runQuery(Devices, Interval, Criteria)
2      Matches ← ∅
3      forall d ∈ Devices do
4          /* get indexes from cache or cloud */
5          Indexes ← getIndexes(d, Interval)
6          forall i ∈ Indexes do
7              if search(i, Criteria) ≠ ∅ then
8                  Matches ← Matches ∪ {i.blockId}
9      Events ← ∅
10     forall blockId ∈ Matches do
11         /* get blocks from cache or cloud */
12         block ←getBlock(blockId)
13         Events ← Events ∪ extractEvents(block, Criteria)
14     return Events
```

all fields in the event or just a subset of them - the ones usually employed in queries. All these options lead to different trade-offs in SLiCER operation, as discussed in Section III-D.

Figure 2 shows a representation of the data model. The events generated by devices are organized into blocks, which are accompanied by their respective indexes. In the figure we can observe that blocks are split by hour. However, they could be split by size or even by both parameters.

2) Event Manager: The event manager component comprises six tasks and their interactions. These interactions begin when the events are received from SIEM infrastructure (1) and end when the data is written to the cache (6) and to the clouds through Charon (7), as shown in Figure 1.

The first task is performed by the *Event Receiver*, which monitors the arrival of new events coming from the SIEM infrastructure (devices, SIEM, or Logger[2]). Events are organized by device and forwarded to the *Parser*, which extracts the key/value pairs and the relevant information to create the data blocks and indexes, according to the event format (e.g., Common Event Format - CEF). The *Block Generator* generates a block of events after a specific time or when the amount of data received per device reaches a predefined size. After that, the *Index Generator* creates an index for each data block using a predefined indexing method. The *Compressor* task compresses both block and index (separately) using the predefined compressing algorithm. Lastly, the *Event Archiver* organizes both blocks and indexes following the data model structure (i.e., a directory is created per device and sub-directories per time interval). It sends a copy to the cache, maintaining the recent data locally, and to the clouds through Charon.

C. Query Manager and Query Model

This section details the query model and its implementation in the query manager.

1) Query Model: Our query model ensures that queries typically performed on SIEM archival systems are efficiently supported by SLiCER. A query in SLiCER can be represented by a triple $\langle D, I, K \rangle$, where D is a subset of devices that generate events, $I = (t_s, t_e)$ is an interval of time between t_s and t_e (start time and end time, respectively), and K is a set of keywords to search for in the events. These keywords form the query criteria. All fields in a query support wildcards, e.g., to search for events from all devices, in any time period, and containing any keyword. This query model allows supporting different types of queries since organizations may have several devices with various fields.

Query processing is done in two parts: first, SLiCER executes the query over the indexes to discover which event blocks need to be retrieved from the clouds; then the blocks are processed to find and extract the events, which are delivered to the analyst. Algorithm 1 describes this processing. The query execution starts by looking into the indexes of the event blocks generated by the target devices and that fall in the time interval specified. For each device in the query, the algorithm gets the index files from the cache/cloud based on the device name and time period (lines 3-5). Next, each index file is processed to check if the keywords in the *Criteria* are present in its associated data block, and, if they are found, the block name associated with the index is added to the *Matches* set (lines 6-8). For each matching block name, the corresponding data block is obtained from the cache/cloud, uncompressed, and the events that satisfy the keywords are returned (lines 9-14).

2) Query Manager: The query manager implements Algorithm 1 using a pipeline of threads to execute queries. This component involves 13 interactions, as illustrated in Figure 1. The *Query Analyzer* receives the queries and checks their syntax (1), extracting the device names, the time range, and criteria to select the indexes properly (2). The *Index Retriever* implements the first part of the algorithm, looking for the indexes first in the cache (3) and then in the cloud (4), and updating the cache (5) if they were not found there. The second part of the algorithm is implemented by the *Query Executor*, which uses the selected indexes (6) to look for the block names that match the Criteria, and the *Data Retriever*, which needs the authorization of the analyst (7) to obtain the selected blocks (8) either from the cache (9) or the clouds (10). Just as in (5),

[2]A SIEM archival that stores events for short periods (e.g., six months).

the system updates the cache with the retrieved blocks (11). The resulting blocks (12) are decompressed for searching and extracting the events that satisfy the *Criteria*, and then the query ends (13).

D. Indexing Methods

Indexes are an auxiliary data structure used to locate data efficiently without having to search the whole archival. In our case, the data comprises text events coming from different devices in different formats, i.e., with various fields and sizes. SLiCER supports three indexing methods, as follows:

a) No-Index: As the name suggest, in this method, the data is not indexed at all. This means that the data model proposed in SLiCER only organizes events in blocks by device and time range. Therefore, the queries' criteria (i.e., keywords to search) are applied to all blocks that match the device set and time interval.

b) Inverted index: Maps the content of the events (e.g., keywords) to its locations in a file. The index file contains a list of keyword that appears in the events that compose a data block. This index can be used as *full-event* or *reduced-event*, meaning respectively that it is created based on all distinct keywords existent in a data block or based on a set of selected fields (e.g., srcIP, srcPort) defined previously by the security analyst. This way, our data model can maintain separated file indexes for each block, which are used to organize and perform searches over the inverted index.

c) Bloom filter index: Uses Bloom filters [24] to store keywords appearing in each field of the events in a block. Being probabilistic, it can answer if a keyword may be in the set, or if it definitely is not in the set, meaning that it can generate false positives, but never false negatives. Similarly to the inverted index, this type of index can be used to index all distinct keywords (full-event) within a block or a specified number of fields (reduced-event). The index file always contains the number of filters corresponding to the number of fields being indexed.

IV. Cost Model

This section presents the cost model of SLiCER for storing and retrieving data to/from the cloud. The next two subsections detail all the costs inherent to both processes, taking into account the kind of data (event blocks and indexes), its size and number, and the price of PUT and GET requests.

A. Data Storage Cost

The storage cost comprises the cost of the PUT requests to send data to the cloud and the cost of maintaining the data stored in the cloud (charged monthly). If SLiCER is configured to create indexes, both the event block and its index are sent to the clouds (requiring two PUTs per block); otherwise only the block file is sent (requiring one PUT).

Therefore, the number of PUT requests in a month is given by $N = 2^{index} \times b$, being b the number of blocks generated in a month and $index \in \{0, 1\}$ a flag indicating if indexes are used or not. Similarly, the total size of the data to be sent to the cloud is $S = i_{size} + b_{size}$, with b_{size} and i_{size} representing the block and index average size, respectively ($i_{size} = 0$ if no index is used). Equation 1 shows the storage cost SC for an archival maintained for M months, given the prices $P_{storage}$ (per GB/month) and $P_{request}$ (per PUT request).

$$SC = (S \times M \times P_{storage}) + (N \times P_{request}) \tag{1}$$

B. Data Recovery Cost

The cost of recovering data from the cloud is intrinsically related to the query amplitude (i.e., devices, time range, and criteria), which defines the amount of data to be downloaded. A typical query defines at least the device and time range parameters, and the time range usually does not surpasses a month. This means that the size of the archive does not usually affect the cost of a query, as the amount of accessed data will be narrowed by specific devices and time range. Therefore, the cost is dependent only on the number and size of the data blocks and respective indexes that have to be downloaded, and the prices of GET operations to request and download data from the clouds.

We recall that a block contains multiple events, so the cost of a query is not associated with the number of events found by a query, but the number and size of blocks scanned. For example, considering two queries Q_1 and Q_2 involving blocks with the same size, if Q_1 returns ten events spread by three blocks and Q_2 touches five blocks, Q_2 will be more expensive than Q_1.

Another issue that affects the cost of a query is the size of the data blocks. On the one hand, bigger blocks can force the download of unnecessary data, making the query expensive. On the other hand, smaller blocks can generate more GET requests to retrieve the necessary blocks, which can also result in an expensive query, as mentioned before. Therefore, the challenge is to find the best cost-benefit between the number of blocks and its size. A rule of thumb is the ratio adjustment according to the user needs, i.e., the frequency of queries that are done and their download size (block and index).

Since SLiCER can be configured with or without an index, we envision two ways of getting the blocks and indexes from the cloud to execute a query, namely *No-index* and *Indexed*.

a) No-index: As we stated previously, when SLiCER uses the no-index method, firstly all blocks from devices and time range specified are downloaded from the cloud (if they are not in cache), and then the query is executed over them locally. In this case, there is the possibility of downloading unnecessary blocks not matching the query criteria. Equation 2 defines the index and data block rules for Not-indexed queries, where the resulting set of indexes (I_{NI}) is empty, and the set of blocks (B_{NI}) to be downloaded is formed by all data blocks ($b.device$) belonging to devices (D) and between a time-range $b.ts$ of T specified in the query. After downloading the blocks, SLiCER searches for events that match the query criteria.

$$\begin{aligned} I_{NI} &= \varnothing \\ B_{NI} &= \left\{ b : b.device \in D \wedge b.ts \cap T \neq \varnothing \right\} \end{aligned} \tag{2}$$

b) Indexed: When SLiCER executes queries using an indexing method, it first downloads the index files associated to a time range and devices specified in the query from the cloud (if they are not in cache), next it uses these indexes to find which blocks match the query criteria, and then downloads only the matching blocks from the cloud (again, if they are not in cache). Equation 3 defines the index and data block rules for getting both sets. The I_I set contains all index files that belong to the devices in a given time range (i_b). The B_I set represents all data blocks (b) that match the criteria, which will be downloaded. After the download is completed, SLiCER scans the blocks to find the events.

$$\begin{aligned} I_I &= \{i_b : b.device \in D \wedge b.ts \cap T \neq \varnothing\} \\ B_I &= \{b : \exists i_b \in I_{D,T} : match(i_b, C)\} \end{aligned} \tag{3}$$

c) Query cost: The final query cost (QC) is given by Equation 4, which takes into account the sets I_M and B_M defined for each method M. The equation is formed by two parts: one that considers the number of indexes and blocks that are necessary to be downloaded, i.e., the number of GET requests needed and the associated request price ($P_{request}$); and the other part considers the size of that data and its associated download price ($P_{download}$).

$$\begin{aligned} QC_M = (|I_M| + |B_M|) \times P_{request} + \\ \left[\sum_{i \in I_M} i_{size} + \sum_{b \in B_M} b_{size}\right] \times P_{download} \end{aligned} \tag{4}$$

V. Implementation

We implemented SLiCER as a userspace daemon in Java. The system has two main components: event manager and query manager, as described in Section III. The cache is implemented as a local directory organized in the same fashion as the data model structure. Our implementation resorts to an adjustable thread pool to improve the performance and parallelism of the system in the tasks it performs.

SLiCER uses a no-index and two index algorithms: Lucene v7.7.0 [17] for inverted indexes, and the long fast implementation[3] of Bloom filters [24]. It also employs the XZ compression algorithm[4] to compress event blocks and indexes.

The encryption and interaction with the cloud are done using the Charon cloud-of-clouds storage system [12].

VI. Evaluation

The objective of the experimental evaluation was to answer the following questions:

1) How much does SLiCER reduce the storage space needed for the events?
2) Is SLiCER able to reduce the time to search for data in the cloud when compared with other solutions?
3) Is SLiCER able to reduce the cost of recovering data from the cloud when compared with other solutions?
4) How the SLiCER cost-effectiveness compares with searchable symmetric encryption?

[3]https://github.com/vvcogo/java-longfastbloomfilter

[4]https://git.tukaani.org/

We performed two types of tests: experimental evaluation of our prototype and cost simulations. For the former, we deployed SLiCER on a Dell desktop 7040 with a Intel core i7 CPU 3,4-3,8 GHz, 16 GB RAM, and a HDD 7200rpm 1TB. For the latter, we calculate the costs using Amazon S3 prices as a reference [7]. Thus, we do not consider the prices of a multi-cloud setup in this evaluation, since there is a high variability on such costs depending on the set of providers in use. For instance, a setup with Wasabi, Backblaze, OVH, and Azure storage services will have similar costs to Amazon S3, despite Charon's erasure code storage overhead of 50%. We also do not consider the cache for costs calculation. This way, the results in Figures 4 and 5 represents a worst-case scenario, where all data is downloaded from the clouds.

A. Methods and Algorithms

We evaluated several variants of SLiCER, comparing them with a simplistic "cloud archival" approach, which we called *Strawman*, and SSE (later on Section VI-E).

In the *Strawman* (SM) design, each event block is created for a time range, meaning that it contains all events generated by all devices in a given period. No indexes are created for the blocks. This design reflects a simplistic approach in which collected events are simply sent to the cloud from time to time in blocks, and such blocks are downloaded back in case they need to be queried. Following the cost model we presented in Section IV-B to retrieve data from the cloud, Equation 5 defines the index and data block rules for SM queries. The set of indexes I_{SM} is empty and the set of blocks B_{SM} is equal to all blocks downloaded for the time-range specified in the query. After the download is complete, SM searches locally for events that match the devices and the criteria specified in the query. The query cost is also given by Equation 4.

$$\begin{aligned} I_{SM} &= \varnothing \\ B_{SM} &= \{b : b.ts \cap T \neq \varnothing\} \end{aligned} \tag{5}$$

No-index (NI) is one of the variants of SLiCER in which the blocks are generated following SLiCER's data model but without indexes.

Index represents the main configuration of SLiCER, in which event blocks are indexed. We defined four variants of this design, two using Lucene indexes and two employing Bloom filters. For both algorithms we created indexes for all fields contained in the events, which we call *Lucene full* (LF) and *Bloom filter full* (BF), and indexes considering only the 13 most relevant fields of the events, resulting thus in the *Lucene reduced* (LR) and *Bloom filter reduced* (BR) variants. These 13 fields (e.g., srcIP, dstIP) were chosen due to their relevance for Security Operation Center (SOC) analysts. Depending on the device type, events can contain all or some of these fields.

B. Dataset Characterization and Processing

a) Dataset characterization: The dataset was obtained from the test SIEM environment of a large power utility company, which mimics a subset of the utility production system and is used for training staff. The events were collected

TABLE I
DATASET CHARACTERISTICS

a) Dataset before processing			b) Blocks after processing	
Devices	**Number of events**	**Total Size**	**5 days (real)**	**30 days (estimated)**
10.10.25.1	5 065 666	9 040 679	108	845
10.10.25.2	3 042 827	4 851 493	92	720
10.10.25.6	577 850	917 975	92	720
10.10.25.10	105 766	169 000	92	720
DC_server	62 388	112 021	92	720
collect1	5 581	7 363	205	1 604
DB_server	1 006	1 726	73	571
collect2	417	907	54	423
AV_server	285	675	54	423
10.10.2.10	117	333	45	352
192.168.1.1	96	313	46	360
TOTAL	**8 861 999**	**14,4 GB**	**953**	**7 458**

TABLE II
DATASET SIZES AFTER PROCESSING: 5 AND 30 DAYS

a) 5 days					b) 30 days		
Algorithm	**EPS**	**Data Size**	**Index Size**	**Total (MB)**	**Data Size**	**Index Size**	**Total (MB)**
SM	2382	230	0	**230**	1800	0	**1800**
NI	2339	220	0	**220**	1722	0	**1722**
BR	2314	220	2	**222**	1722	16	**1737**
BF	1294	220	41	**261**	1722	321	**2043**
LR	1843	220	79	**299**	1722	618	**2340**
LF	1635	220	601	**821**	1722	4703	**6425**

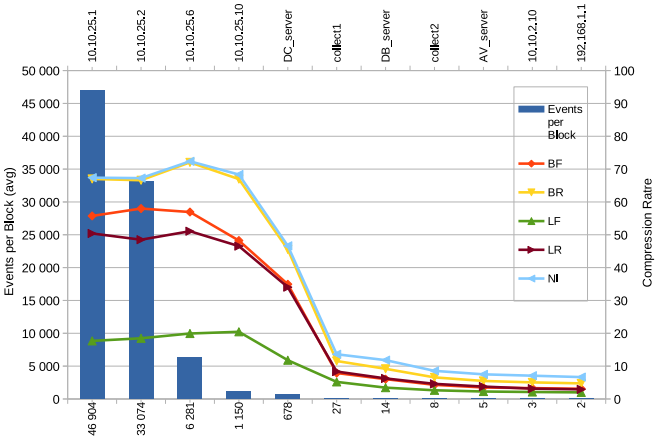

Fig. 3. Number of events/block and device and compression ratio algorithm.

in the CEF format from 11 devices during a period of 5 days, totalling 92 hours. This is a time frame often used for queries, even on larger datasets. The number of events collected each day is similar. About 9 million of events were gathered, each one with an average size of 1.7 kB, leading to a total of 14.4 GB of data. Events have 63 fields on average. This dataset, even being small, allow us to understand the behaviour of SLICER when storing and retrieving real events.

Table I-a describes the number and size of events collected from each device. The volume of events is concentrated mainly on the first two devices, comprising about 8 million events (90% of the dataset).

b) Dataset processing: We processed the dataset using the five variants of SLICER (NI, LF, BF, LR, and BR) and SM, as discussed before.

Table I-b shows the number of blocks generated after processing the dataset (5 days) and an estimate for 30 days. Each block was limited to 100 MB of raw data or a period of one hour, whatever comes first. Table II-a depicts the data and index sizes after processing the data, and the events processed per second (EPS). Table II-a is an estimate, just like Table II-b.

After SLICER and SM organize the events in blocks and compress them, the data was reduced from 14.4 GB to 220 and 230 MB, respectively. SM is approximately 5% bigger than SLICER because it organizes events only by time range, whereas SLICER organizes by device and time range, grouping together similar events and increasing the effectiveness of compression.

As expected, the indexes are bigger in the full variants (LF and BF) than in the reduced variants (LR and BR), while SM and NI do not have indexes at all. LF index size widely contrasts with others as Lucene indexes all keywords of a document, commonly generating an index with up to 30% of the raw data size. In our case, it reached 25% of 14.4 GB and 5% after the index compression. However, it is 3× the size of the compressed data (220 MB). LR indexes, in contrast, reaches little more than 35% of the compressed data size. The fact that LR (and BR) indexes only 13 fields leads to this difference. Finally, Bloom filter indexes were the smallest due to its algorithm characteristics.

The number of events processed per second (EPS) by each method is shown in Table II-a. SM and NI are the most efficient methods as they do not index events. BR follows as it generates at most 13 Bloom filters per block, being thus very efficient. In contrast, BF is the method with worst performance as it creates one Bloom filter for each distinct field a block has. Lucene is not the fastest but different from others. It can store additional data attached to the index, such as the event filename. It permits Lucene to know the exact events that match a query. LF and LR present similar performance, being LR slightly faster since it is smaller.

Figure 3 shows the average number of events contained in a block for each device, as well as the compression ratio achieved for each variant of SLICER when applying XZ compression on both the event block and its index. The figure shows, for example, that blocks from 10.10.25.2 (second device) containing 33,074 events (on average) can reach a compression ratio of 60× with BF indexes. The compression ratio is intrinsically dependent on the method applied to generate the index since the resulting indexes have different sizes. As observed in the figure, the NI and BR methods are the ones that achieve best compression ratios, since NI does not generate indexes and BR creates very small indexes. On the other hand, LF and LR were the methods with the lowest compression ratio due to the size of their indexes.

Figure 3 also shows that the size of event blocks has a significant impact on the effectiveness of compression. For the first device, which has the biggest event blocks, this rate is above 70×. However, blocks with fewer events present a poor compression ratio. Therefore, the block size is an important parameter as it affects the storage size and cost (due to different compression ratio), the time required to collect a

TABLE III
QUERY CHARACTERIZATION.

ID	Query	Description	Matching Blocks
Q1	In [10.10.25.2] where [dhost=SOC4] from [1 to 5 days]	This query aims to represent a search without resulting events. The algorithms based on indexing have advantage compared with the others, since they download only the index blocks, while the others need to download the data to execute the query, i.e., unnecessary blocks for this case.	0%
Q2	In [10.10.25.2] where [dst=10.10.2.12, shost=collect2] from [1 to 5 days]	This query aims to represent a search returning a few data blocks that match the query. The use of indexing algorithms continues to be useful comparatively with algorithms that do not use indexing	40 – 50%
Q3	In [10.10.25.2] where [dhost=SOC1] from [1 to 5 days]	This query aims to represent a search similar to Q2, however, resulting more data blocks. As the resulting number of data blocks increase, the cost benefit of using indexing algorithms can decrease, turning some of them impractical to perform some queries.	80 – 85%

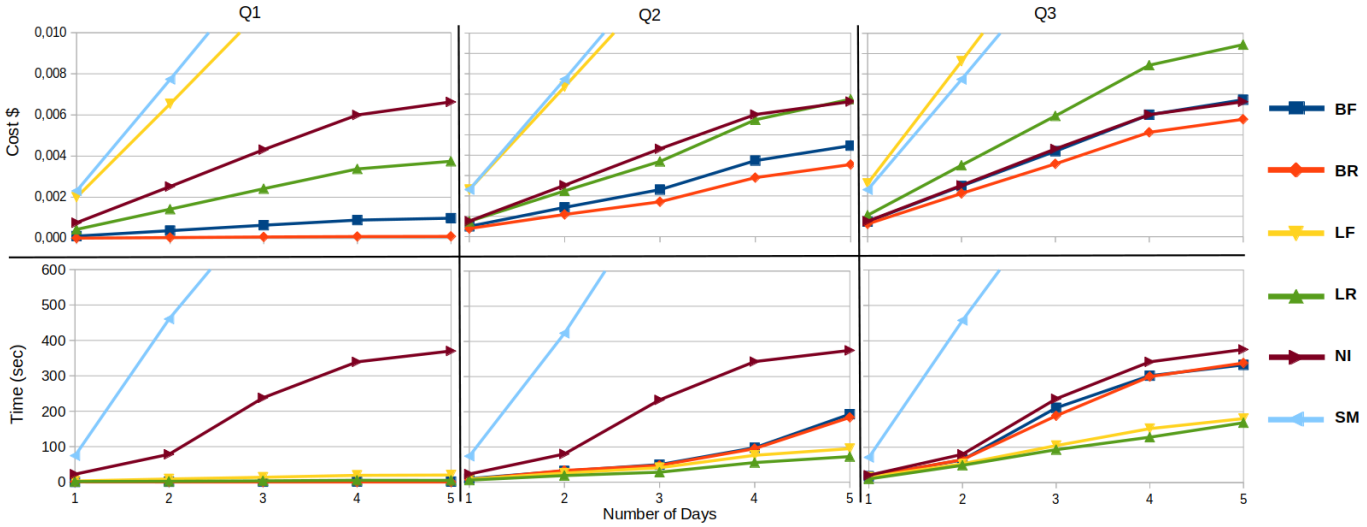

Fig. 4. Cost (top) and execution time (bottom) of the three types of queries considering time ranges of 1 to 5 days of events.

block, and the amount of downloaded data on queries.

This analysis allows us to answer question 1, as our results show that applying the SLICER data model can reduce the size of the SIEM data stored in the cloud by up to 70×.

C. Storage Cost

The storage cost was calculated using the number of requests and the data size stored in a month, and the number of months we want to keep the data in the clouds, as defined in Section IV-A. Since the dataset we used comprises only 5 days of events, we estimate the number of blocks and their size for a month based on an average of the 92 hours collected. The number of blocks shown in Table I-b was 953 and the estimated number of blocks for a month was 7458. The estimated dataset sizes for each algorithm are shown in Table II-b.

Using Equation 1, it is possible to calculate the cost for any M months. For example, if long-term storage was defined to keep the data for 48 months using the AWS (Ireland-S3 Standard) as a basis, and discarding the oldest block after creating a new one, the cost charged by month using BR is:

$$SC = (1.74 \times 48 \times \$0.023) + ((7458 \times 2) \times (0.005 \div 1000))$$

resulting in $2.00 per month. For a real dataset, based on the events generated by the same company in production (around 30 billion events and 750 GB per month), the cost to keep this data for 48 months in the cloud is about $860.00 per month.

D. Query Cost and Execution Time

a) Varying time range: In order to evaluate SLICER's query performance and financial cost, we defined three queries for events with different devices and criteria, and varied their time range from 1 to 5 days, as frequently used for queries in SOCs. Each query was executed 5 times: the first is relative to dataset collected on the first day, the second comprises the first and second days, and so on. Table III describes these three queries. $Q1$ returns no blocks, whereas $Q2$ returns few blocks (40% – 50% of the total in the time range), and $Q3$ returns most of the blocks available for the period (80% – 85%).

Figure 4 shows the *upper bound on the monetary cost* of executing the queries (in USD – upper-half) and the *lower bound in their execution time* (bottom-half), when looking for events in different fractions of our dataset (i.e., time range between 1 to 5 days). To remove the variability of cloud accesses, the upper bound in cost stems from the case in which no data is cached, while the lower bound in latency consider exactly the opposite. The results with different cache miss rates follow exactly the same patterns, albeit with different values.

For all executed queries, the SM and LF were the most expensive because of the former downloads all data blocks for

all time ranges and the latter downloads the largest indexes.

The financial cost of $Q1$ essentially comes from the GET requests and the downloads of the indexes, for the variants that use an index (LR, LF, BR, and BF), or the cost of downloading the data blocks, for SM and NI. In the same way, the query execution time comprises the time necessary to download either the indexes and/or the data blocks. Since $Q1$ does not match any data block, the cost is dictated by the size of the indexes, which are different for each algorithm. However, and differently from the financial cost, the execution times were similar for all index-based variants, while SM and NI execution times were directly proportional to the number of blocks accessed.

Considering $Q2$ and $Q3$, the queries that actually return events, the cost and execution times are similar in each variant. SM and LF provide the most expensive queries, whereas BF and BR are the less expensive ones. Regarding the query execution times, Lucene variants (LF and LR) are the fastest, surpassing the Bloom filter variants (BF and BR), and SM and NI consistently present the worst latency.

The difference in the performance of Lucene and Bloom filter variants is due to the fact that the former stores the ID/name of events from a data block when it generates the index, knowing precisely which are the events that match a given criteria, while the latter only stores membership information, i.e., there are some events that match a specific criteria in the block. Therefore, Lucene knows which events need to be extracted after executing the query, before downloading them from the clouds, while Bloom filter variants need to scan all matching blocks looking for events matching the criteria.

To answer questions 2 and 3, we conclude that the reduced variants of Lucene and Bloom filter have better query execution time and cost, respectively. However, since the size of indexes is a disadvantage of LR, we believe it is preferable to use a method that is less efficient but cheaper. Hence, BR is the best solution in this case.

b) False positives: As we mentioned before, Bloom filters can generate false positives due to its probabilistic approach. Moreover, it can also generate more false positives due to the manner that blocks are indexed. Since each block contains many events, events are indexed together in order to generate one Bloom filter index per field and block. This means that each index is used to check if a keyword exists in the block, not in the event individually. So it is possible for queries with two or more keywords connected by an AND in their criteria to find matching blocks (according to their index) with no matching events. To illustrate this case, we evaluate the query "*In (10.10.25.1, 10.10.25.2) where (dst=10.10.2.12, src=10.10.1.10) from (1 to 2 days)*" in which we look for events with a specific *src* and *dst* fields.

Table IV shows the results of this query when we use the BR and LR algorithms. This query returns zero events for both algorithms, but BR leads to false positives.

The table shows that, although both variants download the same number of indexes, LR indexes are bigger than BR indexes (41MB vs 0,5MB). On the other hand, the number of data blocks returned by this query for LR is smaller than the ones returned by BR (0 against 57). Due to the number of blocks unnecessarily downloaded by BR, the cost of the query for this variant is higher than the cost for LR, even with an index $82\times$ smaller. It is worth to notice that SLICER will filter false positives since it scans blocks locally looking for matches to return, so query accuracy is not affected.

TABLE IV
BLOOM FILTER FALSE POSITIVES.

Alg.	Indices	Blocks	Index (MB)	Blocks (MB)	Download (MB)	Cost
LR	73	0	41,4	0	41,4	0,0037
BR	73	57	0,4	59,2	59,6	0,0053

E. Comparison with Searchable Symmetric Encryption

In this section, we present a brief comparison of SLICER with a functionally-similar system that employs SSE for searching events on the encrypted blocks stored in the cloud. In particular, we used a recent SSE scheme called MuSE [11], which is considered state-of-the-art. MuSE supports data in different media formats, including text, hence for a transparent comparison, we configured it to process our dataset in a similar fashion as SLICER, where SIEM events are grouped in blocks and merged in a single file. In this way, each block is seen as a document and there is a global index that maps all words (i.e., field values) contained in these blocks. Searches are executed using this index, which is stored in the cloud through a server on a virtual machine deployed there. After querying the index, the server finds the matching blocks and replies back.

Our comparison focuses on the financial costs of operating an SSE-based system and SLICER. To do that, we first have to define the cost model for SSE queries. Equation 6 shows that. The set of indexes I_{SSE} is empty since indexes are not downloaded and the set of blocks B_{SSE} is equal to all blocks of the specific device, time range, and criteria specified in the query. The final query cost QC is given by Equation 4, with $P_{request} = 0$, plus the amortized cost of the virtual machine.

$$B_{SSE} = \{b : b.device \in D \wedge b.ts \cap T \wedge match(b, C) \neq \varnothing\} \tag{6}$$

To make our comparison, we consider the cheapest AWS EC2 virtual machine that could support the workloads we are dealing (t3.small) and we chose query $Q2$ of Table III with all 5 days as an example workload.

Figure 5 compares the monthly costs of SSE with SLICER's different variants and Strawman (SM), considering different numbers of *non-cached queries* issued per month. Since SSE requires a virtual machine to operate, its starting cost is much higher than SLICER and SM. The t3.small instance has a base monthly cost of US$ 16.8, independently of how many queries are issued. Moreover, it should be noted that this cost is the minimum to support our workload. As SSE schemes typically keep their index in volatile memory for improved efficiency, in scenarios with larger datasets, more expensive virtual machines with extra memory would need to be requested from the cloud provider. In contrast, SLICER

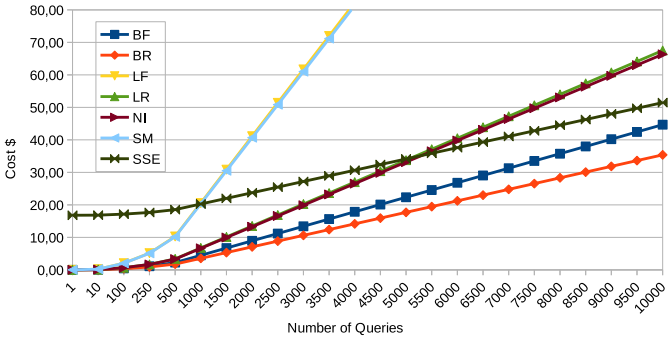

Fig. 5. Query cost comparison between SSE and SLICER.

only requires cloud storage services, which is easier to scale and has a much lower financial cost, especially if only a small number of queries are expected to be issued.

A major source of costs in the cloud is the amount of data downloaded from it, and this cost affects all evaluated approaches. This means that as we scale the number of queries issued, both SLICER and SSE increase their costs. In particular, after 500 queries, all variants start growing in cost by a large fraction that depends on the amount of data transferred by them. Since SSE processes query in the cloud and use a single index that can precisely pinpoint which blocks to return, it minimizes network transfers. Hence, variants that have to transfer more data, such as SM and LF, start to cost more than SSE after 1000 queries. Other variants with average networking needs, like LR and NI, start to cost more than SSE only after 4.000 queries are issued in a month. Bloom filter based variants (BF and BR), which strive to transfer the least amount of data, always cost less than MuSE for up to 10.000 queries. In fact, BR is so cost efficient that only after 230.000 queries per month (not shown in the figure) does it start to cost more than SSE. This amount of non-cached queries seems highly unfeasible even for big organizations, as archival systems are rarely queried.

Based on this comparison, we can answer questions 3 and 4: BR is still the best solution in terms of cost-effectiveness, even when compared with SSE, the standard method for searching over encrypted data in the cloud.

VII. Conclusion

We described SLICER, a system to improve the capacity of SIEMs for long-term event retention for forensic analysis. The approach, on the one hand, leverages public cloud storage, exploiting their low costs and high scalability. On the other hand, it employs a data model that groups events in blocks using different indexing techniques to efficiently recover data, ensuring cost effectiveness in storing and recovering data to/from the clouds. Moreover, security is guaranteed by leveraging a cloud-of-clouds backend system that encrypts data and replicates it through multiple clouds. We evaluated several variants of SLICER with a real dataset provided by a power utility company. The results showed that the system is significantly more cost efficient than competing alternatives, including searchable symmetric encryption.

Acknowledgment

This work is supported by EC through the DiSIEM project (H2020-700692), and by FCT through the IR-CoC (PTDC/EEISCR/6970/2014) and Abyss (PTDC/EEI-SCR/1741/2014) projects and the LASIGE Research Unit (UID/CEC/00408/2019).

References

[1] Alienvault. (2016, Feb) Best practices for configuring your usm installation. [Online]. Available: www.alienvault.com/forums/discussion/6705/q-a-from-webcast-best-practices-for-configuring-your-usm-installationl

[2] B. Walther. (2013, November) Arcsight data retention. [Online]. Available: https://wikis.uit.tufts.edu/confluence/display/exchange2010/ArcSightDataRetention

[3] L. Bilge and T. Dumitras, "Before we knew it: an empirical study of zero-day attacks in the real world," in *Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 833–844.

[4] L. Ablon and A. Bogart, *Zero Days, Thousands of Nights: The Life and Times of Zero-Day Vulnerabilities and Their Exploits*, 2017.

[5] K. Kent and M. Souppaya. (2016, February) Nist-guide to computer security log management. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-92.pdf

[6] S. Gordon. (2010, October) Siem best practices to work. [Online]. Available: www.eslared.org.ve/walc2012/material/track4/Monitoreo/Top_10_SIEM_Best_Practices.pdf

[7] A. W. Services. (2017, July) Amazon s3 pricing. [Online]. Available: https://aws.amazon.com/s3/pricing/

[8] A. Bessani, M. Correia, B. Quaresma, F. Andre, and P. Sousa, "Depsky: dependable and secure storage in a cloud-of-clouds," *ACM Transactions on Storage (TOS)*, vol. 9, no. 4, p. 12, 2013.

[9] C. Orencik, A. Selcuk, E. Savas, and M. Kantarcioglu, "Multi-keyword search over encrypted data with scoring and search pattern obfuscation," *International Journal of Information Security*, pp. 251–269, 2016.

[10] M. Naveed, M. Prabhakaran, and C. A. Gunter, "Dynamic searchable encryption via blind storage," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 639–654.

[11] B. Ferreira, J. Leitao, and H. Domingos, "Muse: Multimodal searchable encryption for cloud applications," in *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*. IEEE, pp. 181–190.

[12] R. Mendes, T. Oliveira, V. Cogo, N. Neves, and A. Bessani, "Charon: A secure cloud-of-clouds system for storing and sharing big data." *IEEE Transactions on Cloud Computing*, 2019.

[13] A. Serckumecka, I. Medeiros, B. Ferreira, and A. Bessani, "A cost-effective cloud event archival for SIEMs," in *Proc. of the 1st Workshop on Distributed and Reliable Storage Systems (SRDS)*. IEEE, Oct. 2019.

[14] Splunk. (2018, January) Splunk siem. [Online]. Available: https://www.splunk.com/en_us/products/quick-start-bundles/siem.html

[15] A. S. Foundation. (2019, March) Apache solr. [Online]. Available: http://lucene.apache.org/solr/

[16] Elastic. (2019, March) Elastic platform. [Online]. Available: elastic.co/

[17] A. S. Foundation. (2017, July) Apache lucene 6.6.0 documentation. [Online]. Available: http://lucene.apache.org/core/6_6_0/

[18] M. Vallentin, V. Paxson, and R. Sommer, "Vast: A unified platform for interactive network forensics," in *13th USENIX Symposium on Networked Systems Design and Implementation, 2016)*, pp. 345–362.

[19] C. Gormley and Z. Tong, *Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine*. O'Reilly, 2015.

[20] R. Agarwal, A. Khandelwal, and I. Stoica, "Succinct: Enabling queries on compressed data," in *12th USENIX Symposium on Networked Systems Design and Implementation*, 2015, pp. 337–350.

[21] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon, "Racs: A case for cloud storage diversity," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, 2010, pp. 229–240.

[22] T. Oliveira, R. Mendes, and A. Bessani, "Exploring key-value stores in multi-writer byzantine-resilient register emulations," in *Proc. of the 20th Int. Conf. On Principles Of DIstributed Systems – OPODIS'16*, 2016.

[23] D. Dobre, P. Viotti, and M. Vukolić, "Hybris: Robust hybrid cloud storage," in *Proc. of the ACM Symposium on Cloud Computing*, 2014.

[24] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.