Percival: A Searchable Secret-Split Datastore

Joel C. Frank[1], Shayna M. Frank[1], Lincoln A. Thurlow[1]
Thomas M. Kroeger[2], Ethan L. Miller[1], Darrell D. E. Long[1]
[1]*Storage Systems Research Center, University of California, Santa Cruz, CA*
[2]*Sandia National Laboratories, Livermore, CA*

***Abstract*—Maintaining information privacy is challenging when sharing data across a distributed long-term datastore. In such applications, secret splitting the data across independent sites has been shown to be a superior alternative to fixed-key encryption; it improves reliability, reduces the risk of insider threat, and removes the issues surrounding key management. However, the inherent security of such a datastore normally precludes it from being directly searched without reassembling the data; this, however, is neither computationally feasible nor without risk since reassembly introduces a single point of compromise. As a result, the secret-split data must be pre-indexed in some way in order to facilitate searching. Previously, fixed-key encryption has also been used to securely pre-index the data, but in addition to key management issues, it is not well suited for long term applications.**

To meet these needs, we have developed Percival: a novel system that enables searching a secret-split datastore while maintaining information privacy. We leverage salted hashing, performed within hardware security modules, to access pre-recorded queries that have been secret split and stored in a distributed environment; this keeps the bulk of the work on each client, and the data custodians blinded to both the contents of a query as well as its results. Furthermore, Percival does not rely on the datastore's exact implementation. The result is a flexible design that can be applied to both new and existing secret-split datastores. When testing Percival on a corpus of approximately one million files, it was found that the average search operation completed in less than one second.

I. Introduction

Security is often a critical issue for long-term storage, particularly given recent incidents involving insiders releasing large amounts of private or classified information [1]. Much of this risk is due to traditional storage systems having a single point of compromise: the data server. If that one point is compromised at any time during the datastore's lifespan, information can be leaked. This threat is obviously magnified in a distributed environment since, by its nature, the data is stored in multiple locations. In situations where a single location, or site, is not trusted, but the collection of sites as a whole is trusted, secret splitting mitigates this problem. By first dividing a data object into *shares*, and then distributing each share to an independent site in the distributed environment, no single site has enough information to perform reconstruction because a single share reveals *nothing* about the original data.

However, due to the inherent information-theoretic security of a secret-split datastore, searching it is normally not possible without reconstructing the original data from its constituent shares. Reconstruction, however, is not only computationally infeasible, it reintroduces the single point of compromise. As a result, the shares need to be pre-indexed in some way that facilitates searching them. Previously, this has been accomplished using fixed-key encryption, *e.g.* public-key, to minimize, and ideally prevent, information leakage. However, in addition to key management issues, which are undesirable in long-term storage environments, fixed-key encryption typically suffers from a catastrophic release of information upon compromise.

To address the need to maintain information privacy while searching a secret-split datastore, we developed *Percival*: a novel system that accomplishes these tasks without relying on fixed-key encryption. Furthermore, Percival is completely agnostic with regards to the datastore's implementation since whether the datastore is based on POTSHARDS [2] or Cleversafe [3], the user is left with some kind of identifier that can be used to retrieve the user's data. Percival combines this collection of identifiers with each data object's search term(s) in order to produce a set of reverse indexes; each reverse index is in essence a search result since it maps a search term to the set of data objects that should be found using that search term. For our purposes, we define a *search term* as a single word that has been identified to relate to a particular *data object*. For example, to find the data object *Moby Dick*, one might use the search term 'whale' in order to retrieve the object from the datastore.

Once the set of reverse indexes is generated, each individual index is secret split; these resulting shares are each sent to a different query server in the distributed environment. We define a *query server* as a hardware security module backed by one or more machines working together as a single, logical key-value store. A *hardware security module* [4], HSM, is a commercially available, physical device that protects and manages the secure pieces of this design by providing a place to handle sensitive data in a relatively non-secure location. It provides both tamper evidence and resistance by logging intrusion attempts as well as clearing its internal memory if it detects an intrusion attempt. Percival relies on the HSMs to process all secure messages from the client, while not exposing any information to the rest of the query server. In general, query servers and their interaction to clients are discussed in detail in Section IV, but for now they can be viewed as a secure key-value store whose job it is to service search request from authorized clients, and respond with the share of the correct reverse index. Once a client has retrieved the shares

978-1-4673-7619-8/15/$31.00 ©2015 IEEE

to a single reverse index, it is able to reconstruct that index, thereby obtaining the set of data object identifiers that can then be used to retrieve the desired data from the underlying datastore.

Percival also provides mechanisms for clients to add content to the datastore as well as for rotating the secret aspects of the design via the HSM as required. Due to the nature of the environment Percival is intended to operate within, it is designed to operate while compromised. For example, it assumes that at least one query server has been compromised at all times, minimizes the release of information in the event that a client is compromised, and assumes that information is potentially leaked via communication channels despite the security of SSL, *e.g.* message contents remain hidden to an attacker but the size of a message is able to be detected.

In order to test its performance, we implemented Percival and ingested the *Digital Corpora* [5] consisting of approximately one million files of varying types, *e.g.* text, PDF, HTML. Search terms for each file were identified by performing a term-frequency inverse-document-frequency analysis of the word stems contained within each file. Percival's search speed is based upon the time complexity of each query server's key-value store and found to be less than one second. The cost of this performance is the space required at each site to store the secret-split reverse indexes, which is typically on the order of gigabytes.

II. Background: Secret Splitting

Secret splitting is an example of a *threshold scheme* $(N : T)$; it involves the act of splitting a piece of data, D, into N pieces, or *shares*, such that any T shares are required for reassembly, where $0 < T \leq N$. For example, a 10 : 6 splitting scheme generates 10 equally-sized shares, where any 6 shares will enable the reconstruction of the original data. All ten shares are *sibling shares* of one another: a share is a sibling of another share if they are both generated as part of the same set from the same piece of original data, D. A critical property of secret splitting is that, with less than T shares, *no* data can be revealed.

There are several known techniques to accomplish splitting a secret, all of which have varying levels of security. Shamir first introduced the concept of secret sharing [6]; Shamir secret sharing provides provably information-theoretic security, the cost of which being that each share is the size of the original data. AONT-RS [7] combines an all-or-nothing transform with Reed-Solomon coding, resulting in a secret-splitting algorithm that is more efficient with storage space: each share is only d/T bytes, for a total storage of $d \cdot N/T$, where d is the size of the data in bytes. However, AONT-RS is only computationally secure, not information-theoretically secure, since with fewer than T shares an attacker could guess a fixed-size key, thereby making it possible to verify whether those shares correspond to the data of interest.

While both of these approaches provide different levels of security, our design is agnostic with respect to the choice of secret-splitting algorithm, treating the algorithm as a black box. It is worth mentioning, however, that if a Galois field is used to facilitate splitting the data, *e.g.* Shamir's secret sharing, the field prime should not be relied upon to be kept secret as this has been shown to cause information release despite forcing an attacker to reconstruct the data without the field's characteristic [8].

III. Design Overview

It is important to have a clear, high-level, understanding of Percival so that details of its design are framed in the proper context. For this reason, we now present a general overview of the architecture and information flow within Percival.

A. Query Server and Datastore Overview

Percival consists of two main layers: a query layer and a data layer. The query layer represents the collection of query servers: the secure key-value stores responsible for handling search requests from clients. The data layer is not restricted to being a secret-split datastore, *e.g.* POTSHARDS or Cleversafe. However, since it is reasonable to assume that an application that warrants the use of Percival would also warrant a secret-split datastore, we assume that the underlying datastore is also secret-split. Furthermore, Percival does not depend on the exact implementation of the datastore, it can be applied to both new and existing applications. Figures 1 and 2 illustrate two such possible distributed configurations. For brevity, we assume a POTSHARDS type configuration where there exists a collection of data servers to which the client saves and requests data objects. However, the reader should feel free to replace the data layer with any desired secret-split datastore; it will have no impact on the overall design or security of the system.

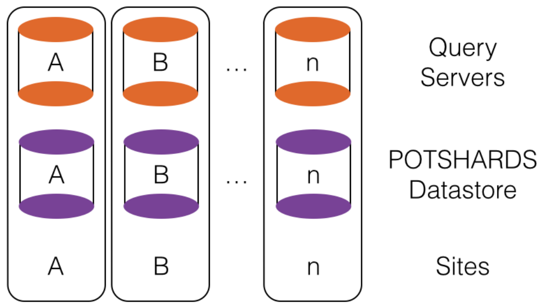

Fig. 1: POTSHARDS example configuration. Each geographically separate site contains two servers: a query server and a data server.

B. Information Flow Overview

During normal operation, clients are able to talk to both query and data servers, but the only in-band communication between any of the servers occurs during disaster recovery. Figure 3 shows the flow of information in the system during a search. In general, a client begins the search process by sending its search request to each query server, who responds

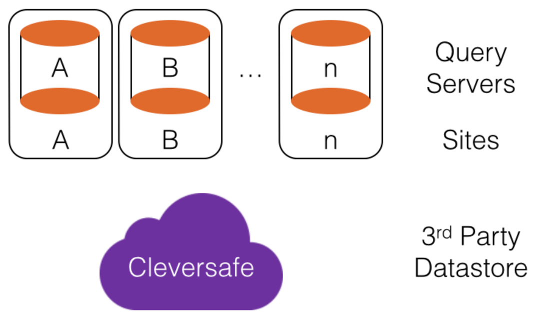

Fig. 2: Cleversafe example configuration. Each geographically separate site contains a single query server. The data layer is handled by a 3rd party, *e.g.* Cleversafe.

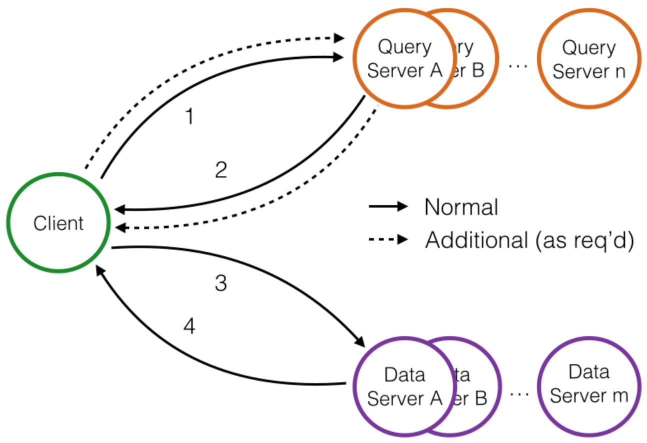

Fig. 3: Overview of information flow during a query. (1) A client starts the process by sending a search request to each of the query servers. (2) Each query server responds with the correct share of a secret-split reverse index, or a non-empty response signifying 'not found'. (3) The client can then either make additional queries, or proceed and request the data of interest from the data servers by sending the appropriate identifier(s). (4) Each data server then responds with its share of the secret-split data object.

with the correct share of a secret-split reverse index, or a non-empty response signifying to the client that there is no data that satisfies the search request. Reconstructing the reverse index reveals the identifiers for data objects that satisfy the search. After which, the client can either make additional queries, or, in the case where it has identified the data of interest, request the data object's shares from the data servers using the appropriate identifier(s).

C. Splitting Scheme Flexibility

Two important design considerations when performing secret splitting are how many shares to generate and setting the reconstruction threshold. Recall that the reconstruction threshold determines the minimum number of shares required in order to reconstruct the original data. The number of shares to generate can be influenced by factors such as performance requirements, *i.e.* more shares results in more processing time, or the number of available sites at which to store the shares. The threshold is typically set based on the tradeoff between security and availability. A high threshold means more sites need to be compromised before data is leaked to an attacker, but if set too high, it increases the risk of a denial of service attack. For example, if the threshold is set to require 9 out of 10 shares to reconstruct the data, it becomes more difficult for an attacker to obtain the required number of shares. However, if just two sites are compromised, an attacker can corrupt or otherwise modify those shares such that it is impossible for anyone to obtain the required number of valid shares, thereby successfully executing a denial of service attack and ultimately data loss. An overly high threshold also increases the risk of a ransomware attack [9]: the attacker encrypts the data, and then ransoms it back for a price.

In contrast to a high threshold, a low threshold improves availability because a denial of service attack is much more difficult to carry out, but the low threshold increases the risk of leaking information due to the few number of sites that need to be compromised in order to reach this lower threshold.

Percival provides a level of flexibility with regard to these design considerations because it separates the reverse indexes from the stored data objects, *i.e.* the query layer from the data layer. As a result, different secret-splitting schemes can be used to address different operational needs or security requirements for each layer.

D. Access Control

Authentication is the linchpin of any security system, the compromise of which results in unauthorized data release and potential data loss. External to Percival, we assume there is a secure framework to which clients authenticate. Percival then uses the provided credentials to segregate the system such that the data loss due to a compromise of said credentials is as localized as possible. Specifically, Percival accomplishes this by creating a unique set of reverse indexes for each authentication credential. Note that access control does not apply to Percival's claim regarding fixed-key encryption. A compromised access control credential can potentially result in a release of information in both Percival and standard fixed-key encryption based systems.

If there are a large number of such credentials, this can potentially result in very large space overhead. To address this issue, we assume that *role based access control*, RBAC, is used as the authentication framework, since it has been shown that on average, organizations have roughly 20 roles defined in their RBAC system [10]. This minimizes the space overhead while allowing the system to be segregated to localize the potential data loss.

IV. Query Server

We now present the design and operational details of a query server, including its role during ingestion, specifics about its hardware security module, and how it performs salted hashing to facilitate search operations. Recall that we define a query server as a hardware security module backed by one or more machines working together as a single, logical key-value store, all contained within a single geographical location. Our use of a key-value store is for clarity and brevity. It is possible, and likely, that a real-world Percival installation would use a suitable replacement. For example, the secure partitioning scheme as defined by Parker-Wood *et al.* [11] can be used as a secure and efficient way to store and quickly access the query server's shares.

A. Ingestion

The first step during ingestion is to identify the *keywords* for each data object. For our purposes, keywords are synonymous with search terms. The set of keywords for a data object define how that data object can be retrieved as a result of a search operation, *i.e.* how clients will search for their data using search terms. Once the set of keywords has been identified for each data object to be ingested, the reverse indexes for the corpus are generated by turning the mapping of data object to keywords to become a mapping of keyword to data object(s). Each of these reverse indexes can be viewed as a successful result to a search operation. This operation is illustrated in Figure 4.

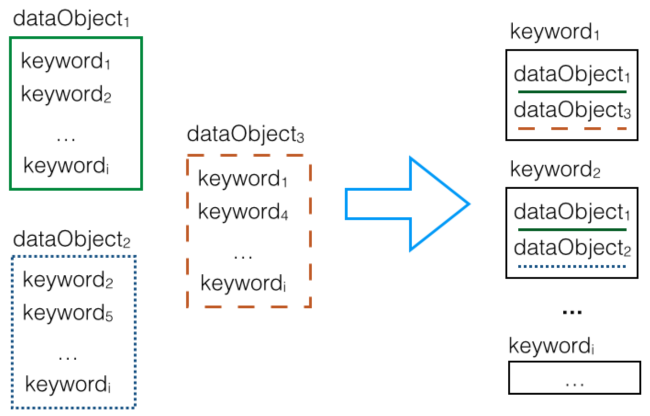

Fig. 4: Generating reverse indexes. Once the keywords for each data object have been identified, they are transformed into reverse indexes mapping keywords to their data object(s).

The next step during ingestion is to secret split each of these reverse indexes, and then distribute the resulting query shares to the query servers such that no two sibling shares are stored on the same query server. Recall that sibling shares are shares that originated from the same piece of data, or reverse index in this context. Figure 5 is an example of sending a single query share and its keyword to be stored on query server n.

The (keyword : RBAC) credential pair is sent directly to the HSM over its own network interface, while the share is sent directly to the key-value store backend of the query server via its separate network interface. This is to ensure that the actual keyword is never exposed to the query server, and is instead pre-processed by the HSM. During this pre-processing,

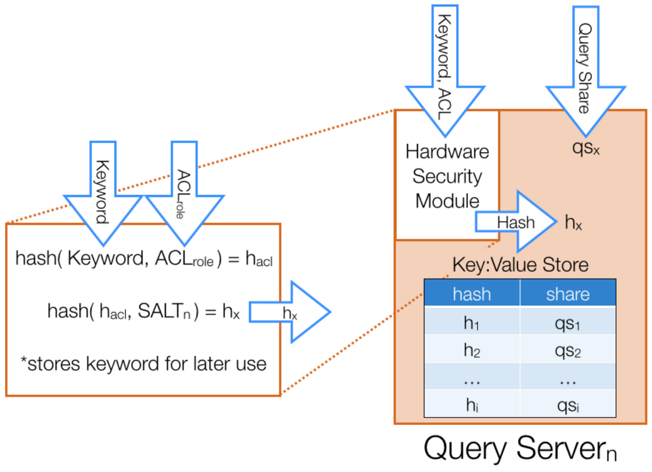

Fig. 5: Overview of ingesting a query share. The keyword and RBAC credential are sent to the HSM, while the query share is sent directly to the key-value store. Once the keyword has been processed into a hash string, the resulting hash and query share are stored in the server's key-value store.

the HSM performs a hash of the keyword and the provided RBAC credential, which is the same credential required by an authorized client in order to retrieve this query share in the future. The HSM then performs one more salted hash using the query server's unique *salt*. Section IV-C discusses the salt in detail. For now, it is sufficient to understand that each salt is kept secret by being generated by, and never leaving, the HSM, and its purpose is to ensure that each query server's contents are unique. Once the HSM has completed its pre-processing, *i.e.* the keyword has been transformed into the final hash string, the resulting hash and query share are stored in the query server's key-value store. The HSM also stores the keyword internally to be used when performing salt rotation.

B. Hardware Security Module

Each HSM is a commercially available, physical piece of rack-mounted hardware that has its own network interface. Typical applications for an HSM include [4]:

1) The key generator and safe key storage facility for a certificate authority,
2) An accelerator for SSL connections,
3) A tool for securely encrypting sensitive data for storage in a relatively non-secure location,
4) A secure key generator for smartcard production.

Percival leverages the HSM to handle all server-side hashing as well as safeguarding the query server's salt. The HSM provides both tamper evidence as well as tamper resistance in the form of logging and clearing its internal memory if it detects a potential intrusion. The price, however, of this secure crypto-processing is limited internal memory and low bandwidth. Typical models have an internal memory of roughly 10 MB and a bandwidth of less than 1 MB/s [12]. However, since Percival separates search processing from the actual data objects, this bandwidth limitation only impacts the rate of incoming search requests from clients, which is easily overcome by placing additional HSMs in parallel on each query server. As a result, Percival easily scales based on the size of the expected client base. Table I lists two examples of commercially available hardware security modules.

It is important to not confuse an HSM with a trusted platform modules, or TPM. A TPM handles similar job types, but is typically mounted directly on the motherboard, and as such, does not have its own network interface. As a result, all communications to and from the TPM must go *through* the motherboard, which would break Percival's security model since Percival relies on the HSM to keep its processing completely isolated from the rest of the query server.

TABLE I: Examples of commercially available hardware security modules [12].

	Infineon SLE 88	IBM 4764
CPU	66 MHz	266 MHz
Memory	16 KB	32 MB
I/O	12 KB/s	9.85 MB/s
SHA-1 1KB	155 KB/s	1.42 MB/s

These modules also feature a means of secure communication between HSMs that provide the same security guarantees as the module itself, which allows multiple HSMs to share the contents of their internal memory without compromising the security guarantees of the HSM. This feature is leveraged by Percival in two ways. First, during initialization and salt rotation, it enables a single HSM to generate the query server's salt and share it with the other HSMs. Second, the HSMs maintain a list of all keywords in the system in order to facilitate hash updates during salt rotation without having to store the keywords externally; this is desirable since storing this list externally is both an unnecessary administrative overhead as well as being a potential vector for information leakage. The space requirement for this list is not large: on the order of megabytes, but it is also non-trivial given the limited available internal memory provided by typical HSMs. By placing multiple HSMs in parallel, less expensive HSMs with less internal memory can be used due to this secure communication channel. For simplicity, the rest of the paper assumes a single HSM is present on each query server, but the reader should keep in mind that they are easily parallelizable.

C. Secret Salt

Each query server has a single, unique salt that it keeps private. The salt is randomly generated by its HSM, and during normal operations, never leaves the HSM. However, since nothing is impossible to break, Percival takes this potential vulnerability into account and minimizes the information released in the event of a salt being exposed. Specifically, without additional elements being compromised, no information is released simply by determining the salt for a single query server. Section V-B1 discusses this threat model in more detail, as well as what other information is required prior to the privacy of the system being compromised.

The purpose of the salt is to ensure that the contents of the query server are unique: the corresponding hashes mapped to sibling shares stored on separate query servers are never the same. This is required since if the hashes were the same, an attacker would be able to identify sibling shares across query servers using only this hash string, and thus enable *targeted theft*: the ability to steal small blocks of data identified by some characteristic, which is the identifying hash string in this case.

The salt is the basis for our claim that Percival is more secure than standard fixed-key encryption. Typically in such systems, if the key used to encrypt the data is compromised, it will result in a catastrophic release of information. Whereas in Percival, a compromised salt alone results in no loss of information or privacy, and does not impact the security of the data or search privacy on the other query servers.

While it is theoretically possible to brute force a query server's salt by performing an exhaustive key search, it is infeasible due to the energy required based on the Landauer limit [13], not to mention the time required to do so. The Landauer limit, L: the theoretical minimum energy required to erase one bit of information, is defined in Equation (1), where k is the Boltzmann constant ($\sim 1.38\times 10^{-23}$ J/K), and T is the operating temperature in kelvins.

$$L = kT \ \mathrm{ln2} \tag{1}$$

For example, in order to simply flip through the bits of a 256 bit salt would require 2^{255} bit flips. Assuming this occurred at room temperature, the Landauer limit states this would require at least 8.9×10^{39} TWh, which is many orders of magnitude greater than all the energy that has ever been produced on the planet [14]. As a result, it is much more feasible to attack the salt via side channels as opposed to attempting to obtain it by brute force.

D. Salt Rotation

In the case that it becomes necessary to change a query server's salt, Percival enables salt rotation such that it only impacts that particular query server for a short period of time. When initiating a salt update operation, the HSM is provided with a list of all available RBAC credentials. This allows the HSM to generate a new salt, and then simply iterate over both the provided list of credentials and the internal list of keywords present in the corpus, recreating the original hash as well as the new hash for each entry in the key-value store. These pairs of hash strings are then passed to the server's key-value store so that it may update its hash table, replacing the old hash with the new one.

E. Performing A Query

To conduct a query, or search request, the client first hashes the search term with the user's RBAC credential. This hash string is then sent to the query servers, via their HSM, which hashes the user's input with the query server's salt. The final hash string is then passed to the key-value store, who finds the query share, if it exists, and sends it back to the client. If no entry is found for the given hash string, the query server responds with a non-empty response signifying to the client that there is no data that satisfies the search request. The size of this 'not found' response is the same size as a reverse index share; this makes all responses, regardless of search outcome, the same size, thus preventing an attacker from differentiating between response types in the encrypted SSL stream.

It is worth noting that while Percival does not explicitly block use of the root RBAC credential, it is not recommended. Since root is, in essence, simply another RBAC credential, it is possible to ingest files and shares using root. However, this will lead to unexpected behavior when performing a query because using the root RBAC credential only grants access to those files that were specifically ingested using that credential, and no other files or shares. It is for that reason that while using the root RBAC credential will not negatively impact the system, its use is discouraged.

Once the client has received the set of sibling query shares, it is able to reconstruct the reverse index in order to obtain the list of data object identifiers, which are then used in accordance with the chosen datastore implementation, *e.g.* POTSHARDS, Cleversafe, etc. Figure 6 illustrates this process.

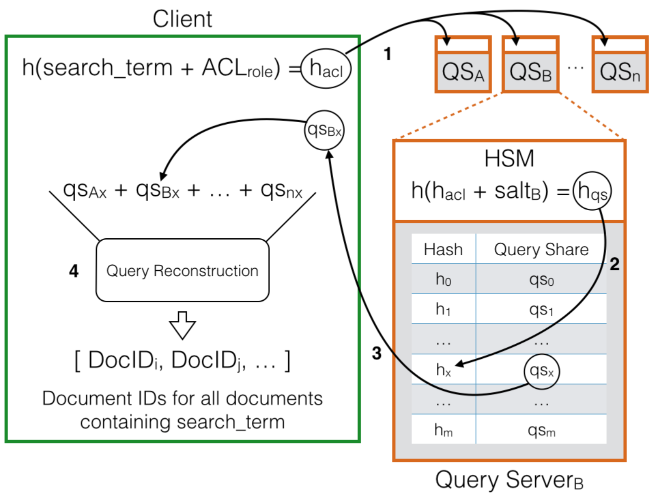

Fig. 6: Simple single search term query. A client starts the process by hashing the search term with its RBAC credential. The resulting hash string is sent to each query server's HSM to be further hashed with that query server's salt, and ultimately used to look up the requested share of the secret-split reverse index. Once the client has received the query shares, it is able to reconstruct the original reverse index: mapping the search term to the appropriate data object identifier(s).

Conjunctive queries are performed in a similar manner, beginning with a client sending a separate search request to the query servers for each search term. The client then reconstructs each of the reverse indexes obtained from the query servers and takes the union of the sets of data object identifiers. The resulting set of identifiers represents data objects that relate to all of the client's search terms.

F. Adding New Content

The process to add content to Percival leverages both the query and ingestion processes. After having identified the keyword(s) for the data object to be ingested, the client performs a query for each of these keywords, reconstructs the corresponding reverse indexes, and adds the data object's identifier to each reverse index. Once the reverse indexes have been updated, they are secret split and sent to the query servers along with the hash of the related keyword and the user's RBAC credential. This hash is sent to each query server's HSM to be subsequently hashed with that query server's salt, the outcome of which is stored in the key-value store along with the newly updated secret-split reverse index.

G. Concurrency

Percival uses *strong strict two-phase locking* [15], SS2PL, managed by a *distributed lock manager*, DLM, to resolve conflicts due to potential concurrent operations. Percival does not rely upon the particular implementation of the DLM, *e.g.* Chubby [16], only that it correctly follows the SS2PL protocol. The three operations in Percival that rely upon the DLM are salt rotation, query execution, and content addition.

During salt rotation, a write lock is obtained from the DLM for all entries in the query server's key-value store. This prevents an *add new content* operation from causing an inconsistent state due to an update collision, and prevents a *query* operation from potentially obtaining an invalid search result.

When performing a query, if a query share has been locked for writing, the query server indicates this state to the client instead of responding with the query share. This can potentially cause a query operation to block if it doesn't receive a number of query shares equal to or greater than the reconstruction threshold.

The last operation that can result in a potential conflict is when adding new content; during such operations, a write lock must be obtained for each entry to be updated. If this was not required, two conflicting, concurrent operations could partially overwrite each other such that a complete set of sibling shares for an entry no longer exists in the database. Specifically, the resulting set of shares for a single entry no longer contains *true* siblings, but instead is comprised of some shares from both operations.

The final outcome of the conflict depends on the nature of the race condition. Recall that in an $(N : T)$ *threshold scheme*, T shares are required for reconstruction. If the resulting set of shares for that entry contained at least T sibling shares, the end result would be diminished reliability and a loss of information from the other operation. Alternatively, if the resulting set of shares does *not* contain at least T sibling shares, the end result would be the corruption of that reverse index due to its inability to be reconstructed; thereby obfuscating the presence of that search term in the datastore.

V. Threat Analysis

Percival's main goal is to maintain information privacy in a distributed, untrusted environment. Furthermore, since its design is intended for long term storage, it is assumed that even though an attacker may have reasonable computing power and storage available, they have potentially unlimited time to carry out an attack. Each of Percival's potential attack vectors are discussed in the following sections.

A. Client

The numerous side channel attacks in which a client may be compromised range from cold boot attacks [17] to social engineering [18] and everything in between. Client security is a large open problem that Percival is not trying to solve. Instead, we assume that one or more clients *will* be compromised and have designed Percival to minimize the damage to the system when that happens.

Compromising a client means that the attacker has access to the user's RBAC credential, and as a result, is able to perform all actions to which that user has access, *e.g.* perform queries, add/modify reverse indexes, and ultimately access the data objects to which the user has access. They do not, however, gain any insight into how the query servers are storing their shares, have access to any salts, nor do they have any access to any information not related to that specific user's role.

In order to recover from a compromised client, the user's role credential must be changed. This can be accomplished by performing an operation similar to salt rotation, in which each HSM is given both the old and new credentials. It can then iterate across its internal list of keywords, generating old and new hash string pairs that the query server can use to update its key-value store.

B. Query Server

Compromising a query server can take several forms, including the system administrator who has an operational need to have access to an entire query server, or even the janitorial staff that needs physical access. It is common to have a single insider at a site, but not for multiple insiders to work in close coordination [19]. That said, it is possible to have insiders unknowingly working in conjunction when guided, or manipulated, by a large external entity, *e.g.* a foreign government [20].

It is also more likely for an insider to control a single query server as opposed to multiple servers [2]. However, we relax this constraint somewhat by assuming that an attacker never controls more than $T-1$ query servers, where T is the threshold number of shares required for reconstruction. If T or more query servers are compromised, there is nothing stopping an attacker from simply reconstructing all reverse indexes, thereby gaining access to the identifiers for all data objects in the datastore.

We assume that an attacker physically controls any compromised query server, and is able to run arbitrary code on them. In general, an attacker is able to read all incoming messages sent to the key-value store, but has no access to those sent to or from the HSM. We also do not consider denial of service attacks, since they can be mitigated at design time by setting the correct reconstruction threshold. See the previous Section III-C for a detailed discussion on the tradeoffs when setting the reconstruction threshold.

One potential attack vector that Percival allows is *targeted theft*: the ability to steal small blocks of data identified by some characteristic. We assume that standard methods for tracking user access patterns and detecting errant behavior are in place on each query server. It is for that reason that we assume that it is not possible to steal large amounts of information from a query server without detection, but that it *is* possible to steal a small amount of data. It is for this reason that steps are taken to ensure that no identifying features of the secret-split reverse indexes exist, *e.g.* ensuring hash strings in the key-value stores differ across servers, all secret-split reverse indexes are of equal size, no correlation exists between the shares' meta-data, etc.

1) Hardware Security Module (HSM): The security guarantees made by hardware security module manufacturers provide a high barrier for attackers to overcome. HSMs are not, however, inviolate. As a result, we acknowledge the potential for an attacker to possibly read the contents of an HSM [21]. If successful, this type of attack would reveal both the static information contained within the HSM, *i.e.* the internal list of keywords for the corpus and the salt for that query server, as well as dynamic search request information, *i.e.* the search terms and user credentials. This would obviously lead to a loss of privacy since an attacker would be able to read the incoming search terms as well as the hash strings in the key-value store in plain text, *i.e.* as keywords. Despite this level of information exposure, it does not expose any information about the underlying data objects, or their potential correlation to keywords because the reverse indexes are unable to be reconstructed without obtaining the correct shares from the other query servers. This is the primary way that Percival's design does not lead to a catastrophic release of information upon compromise.

While the release of all keywords in the corpus is not desirable, it is limited in regards to the amount of information that is released. This can be quantified using the *Shannon entropy*: the average unpredictability in a random variable, which is equivalent to its information content [22]. Shannon entropy, H, can be calculated using Equation (2), where p_i is the probability of character i appearing in the character stream. As a concrete example, the book Moby Dick [23] contains approximately 200,000 words and has a Shannon entropy of 4.55 [24], [25]. In contrast, the book only has approximately 6,800 unique stemmed words, which drops the Shannon entropy to 3.15. This drop in Shannon entropy indicates a significant loss of information, which illustrates that the real data is indeed greater than the sum of its parts.

$$H = -\sum_i p_i \log_2 p_i \tag{2}$$

If it is suspected that a query server's salt has been exposed, rotating the salt will not regain any loss of privacy since the attacker would be able to correlate between the old and new hash strings. To recover from this type of event, the query server would need to be rebuilt by first clearing the key-value store and having the HSM generate a new salt, after which the HSM iterates over its internal list of stored keywords, requesting the appropriate sibling share from each of the other query servers using a process similar to performing a query, *i.e.* send a salted hash of the keyword and the specified RBAC credential, h_{acl}, to the other query servers. Once the HSM has the set of sibling shares for a particular reverse index, it is able to generate a new sibling share without affecting the existing ones [6]. The final step is to generate the salted hash to be stored with the newly created share in this query server's key-value store using the h_{acl} and the new salt.

2) Key-Value Store: Compromising a query server's key-value store is the most basic, and probable attack vector for an inside attacker since it requires no other part of the system to be compromised; it also leads to the least amount of information being leaked. Such an attacker would be able to identify 'hot' shares, *i.e.* those shares that are the accessed at a higher frequency than others. Without the relevant sibling shares, however, this information is of little use. If an attacker was able to identify the hot shares on at least T query servers, this information could be used to perform a targeted theft of those shares. However, this would violate our initial assumption that at most $T - 1$ servers are ever compromised.

Since an attacker is able to read all incoming messages to the key-value store, information about a user's location and search pattern can also be exposed, unless techniques are taken to hide their origin IP address [26].

C. Communication Channels

Percival assumes that secure communication channels, *e.g.* SSL, exist between clients and query servers. It is not that we assume SSL is inviolate, but rather that secure communication, and SSL attacks in particular, are part of a larger open problem that is outside the scope of the project.

As a result, attackers are unable to read the contents of the encrypted data stream, but we assume they are able to perform attacks based on message size. This takes the form of potentially revealing the quantity of search requests for a client as well as whether a search request was successful or not. Depending on where in the system the attacker is listening, the former can likely be mitigated by masking the client's IP address and routing [26]. The latter is not a threat because the query server responds with a meaningful 'not found' message that is the same size as a reverse index share, thus making a negative response indistinguishable from a positive one.

VI. Performance

In order to test Percival's performance, we implemented and tested the system running on five query servers; each server was running 64 bit Linux on four cores using 24 GB of RAM. We ingested the *Digital Corpora* [5] consisting of approximately one million files of varying types, *e.g.* text, PDF, HTML. It is important to keep in mind that Percival's performance does not depend on the size of the data objects ingested, since once ingested, Percival is only concerned with the reverse indexes, and does not interact with the data layer of the system. Keywords for each file were identified by performing a term-frequency inverse-document-frequency [27], TF-IDF, analysis of the word stems contained within each file. The exact method of identifying keywords is not critical since Percival's design does not rely on it, only that they are identified. Once the reverse indexes for the corpus were generated using these keywords, the reverse indexes were secret split using a (5 : 3) splitting scheme, *i.e.* five shares were generated for each reverse index, any three of which were required for reconstruction. These shares were then ingested into a BerkeleyDB database [28] as the query server's key-value store. It took approximately 20 hours to secret split the reverse indexes, and an additional 6 hours to ingest the shares into each query server's key-value store. While these durations are both very corpus and platform specific, they provide the reader with a general sense of the one time, ingestion costs associated with getting Percival online, servicing a corpus of roughly one million pieces of data.

In order to analyze the effect varying the number of keywords per data object has upon both the size and quantity of reverse indexes, we performed the previous steps choosing 10, 20, 30, 50 and 100 keywords per data object. Figure 7 depicts a cumulative density graph for each experiment. As expected, it can be seen that as the number of keywords per data object increases, the total number of keywords present in the corpus also increases. It is also worth noting that $2/3$ of the reverse indexes were found to contain a single data object identifier, and almost all reverse indexes were found to contain fewer than ten identifiers.

While these results are very corpus specific, they illustrate the potential space overhead required to secret split the reverse indexes since all shares must be the same size, *i.e.* the majority of shares must be padded in order to be the size of the largest shares in the corpus. If they are not, then their size becomes a distinguishing characteristic, which would enable targeted theft. For example, if shares are not padded to be of equal size, and there exists only a few shares that are X bytes, then an attacker would be able to identify and steal those shares, simply based upon their size, from non-compromised key-value stores.

With regards to search performance, search requests were found to complete in less than one second; the only variance in performance being due to random network issues causing a small number of resend operations. We measure Percival's search performance using the completion time since standard metrics, *i.e.* precision and recall, are dependent solely upon the number and accuracy of the keywords chosen during ingestion, which are at the implementer's discretion. Percival's search performance is based on both the number of keywords and the size of the corpus. The number of keywords can potentially impact the hash table look up time, *i.e.* from the query server's key-value store, and the size of the corpus

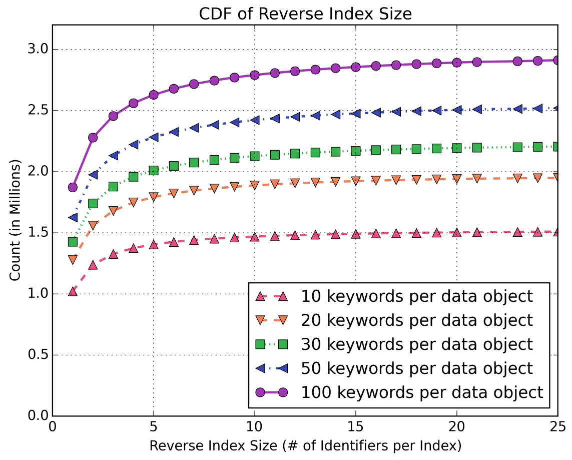

Fig. 7: CDF of Reverse Index Size. For each number of keywords selected per data object, $2/3$ were found to contain a single data object identifier, and almost all reverse indexes were found to contain fewer than ten identifiers. Since all shares must be the same size, these would-be small shares need to be padded in order to make them the same size as the largest shares, otherwise it enables targeted theft of those shares.

affects the reconstruction time since it impacts the size of the query shares.

The bandwidth required to perform a search request is quite low and fundamentally based on the size of the data object identifiers. It can be determined by multiplying the identifier size by the number of keywords per data object times the average number of search terms per search request. For example, assuming an identifier size of 32 bytes and 100 keywords chosen per data object, a search request containing three search terms would require roughly 10 KB, with the majority of that traffic bypassing the low bandwidth HSM.

Since a single HSM is known to be the performance bottleneck, we also tested how long it took to perform the critical, HSM intensive operations in Percival. To complete a salt rotation, it took a single Intel 4764 HSM roughly two minutes to update the entries for approximately three million reverse indexes using a 256 bit salt. By contrast, it took 53 minutes for the same HSM to perform a complete query server rebuild. Both of these benchmark tests were performed for a single credential, or role. Recall that Percival minimizes the impact of an access control credential being compromised by creating an independent set of reverse indexes for each credential. Continuing the previous example, a single set of reverse index shares would require 9.6 GB, which is not trivial, but since it has been shown that most organizations have approximately 20 access control roles defined, the space requirement has a reasonable upper bound, and most would require less than a TB per query server for most applications.

VII. Related Work

Our work on Percival builds on two ares of related work: secret-split datastores and searching encrypted data sets while maintaining information privacy.

Secret-split storage was first developed practically in the PASIS project [29], which also contains a good overview of p-m-n threshold encoding schemes suitable for use in secret-split datastores. In response to several calls for providing long-term storage that can operate through system compromises and provide resilience to insider threat [30], [31], this approach was later adapted for long-term storage by Storer *et al.* in the POTSHARDS system [2].

An alternative to secret-split datastores was developed by Zage *et al.* [32]. In this approach, an algebraic-based encoding solution, Matrix Block Chaining (MBC), is used to "maintain data security and protocol performance when encoding large files. The design of MBC allows for encoding multiple partitions of the original data in parallel as subsequent encoding operations are not dependent on the output of previous encoding steps" [32]. Their technique was developed specifically for cloud storage, however, and as such does not maintain data availability in a compromised environment.

The main threat for Percival is an attacker performing a targeted theft of shares from a repository, thereby being able to reconstruct the original data. There has been a significant amount of work done in recent years regarding encrypted searching [33]–[43], but these works are all based on a single repository storing all of the user's encrypted data. Since they run on a single repository, they do not need to address the vulnerability of search result correlation, particularly across multiple repositories. Our approach on the other hand starts with a fundamental premise that by splitting your data across multiple sites you gain a significant protection that is far stronger that any computationally bound encryption method.

Furthermore, these works rely on the inherent security of the encryption method itself since both the data and the search terms simply use fixed key encryption. However, given enough time and computing power, as well as deliberately poor design, it is possible that fixed key encryption methods will be broken due to the discovery of algorithm short cuts and side channel attacks; thus, these approaches are essentially delayed release. As a result, they are not well suited for applications when data lifetime is measured in decades.

By way of comparison, Octopus [44], does not rely on encryption. It is an anonymous way for P2P nodes to communicate via a distributed hash table that provides a mechanism for individual queries to be sent along "multiple anonymous paths, [while introducing] dummy queries to make it difficult for an adversary to learn the eventual target of a lookup." [44] In contrast to using dummy queries as a means of obfuscation, Percival fundamentally changes the paradigm by ensuring this information does not lead to further information leakage.

Chang *et al.* [41] developed an approach using bit masked dictionaries to enable searching of encrypted remote data without revealing information to the data's custodian. The outcome is similar to using a Bloom filter based system where a single bit is used to represent a term stored in the filter. The main difference is that it does not address conjunctive or disjunctive searches, nor does it address mapping multiple terms to the same bit in the dictionary.

VIII. Future Work

Percival has several open areas of research. The first is to expand its support for additional access control systems. The current design only supports access control for situations in which there are a relatively low number of separate access credentials, *i.e.* less than 100. It would be ideal to expand the current design to fully support a user based access control system, one in which there may be several thousand access credentials. Furthermore, the current design does not support hierarchical access control. Reverse indexes are specific to the access credential under which they were ingested and/or updated. As a result, if multiple roles are authorized to perform a search for the same data object, that object's identifier must be placed in the reverse index for each applicable access credential.

Our next enhancement to Percival's design is to support not only the presence of a keyword within a data object, but its locality as well. This will enhance Percival's search capability beyond conjunctive searches by enabling exact phrase matching. It is theorized this can be accomplished by adding word locality information to each data object's entry within a reverse index, thus allowing the client to test for adjacent locality in addition to taking a simply union of the available reverse indexes. The cost of such functionality is the increased size of each reverse index, which obviously impacts the space overhead of each query server, but also potentially increases a client's reconstruction time due to larger indexes. The danger being that the increase in reconstruction time results in a loss of responsiveness during search operations; therefore extensive testing must be performed to ensure this does not occur.

Another open area of research is to improve query server recovery time in the event a salt is compromised. The time required to do so is not prohibitive, but it would require taking that query server offline for several days, which may impact data availability depending on the secret splitting scheme chosen.

While our performance testing shows that Percival is able to meet the needs of its user base in a timely fashion, it is by no means exhaustive. We look forward to testing Percival using a real world search workload and evaluating its performance.

Our final area of future research in this area is slightly tangential to Percival, and involves improving performance when identifying the correct set of sibling shares is combinatorial prohibitive. This situation does not apply to Percival directly since clients are provided the correct set, or sets, of sibling shares by the query servers. When faced with an extremely large number of shares, *e.g.* millions or more, and no discerning order or association among them, it could potentially result in data loss due to it being an insurmountable combinatorial problem. POTSHARDS made the first steps to address this issue via approximate pointers, but we would like to continue Storer *et al.*'s work [2] by decreasing reconstruction time and improving resilience in the event of missing shares while not enabling targeted theft.

IX. Conclusion

Maintaining information privacy is difficult when sharing data across a distributed, long-term datastore operating in an untrusted environment. To address this need, we have presented Percival, a system that is designed to be applied to new or existing secret-split datastores, operates while compromised, minimizes insider threat, localizes data release upon compromise of an access control credential, and still provides accurate and timely search results; this is all achieved with a space overhead of a few terabytes per query server.

During the course of developing Percival, we made several discoveries, the first being that natural language processing can be quite nebulous as well as time consuming. This discovery reinforced our initial belief that pre-ingestion corpus processing should be left up to, and tailored for, the particular instance to which Percival is being applied.

Another lesson learned was that nothing is inviolate. Every aspect of a system must be assumed to be able to be compromised, regardless of the security guarantees associated with that subsystem. The natural extension of Percival's management of query shares, specifically the distribution of information such that there is a high barrier to cross prior to information release, was to apply the same concept to its handling of the private aspects of the design, *i.e.* the salts. The breakthrough in Percival's design came when we realized that is possible to keep even the salts isolated from each other; this design change converts full data release into a localized loss of privacy, and is what truly sets Percival apart from previous works.

As society needs to store an ever-increasing volume of potentially sensitive data for a long time, we will need to find methods that can maintain data privacy and integrity in spite of security events; this will necessitate distributed approaches such as Percival. The techniques for secure searches developed for this work will help to make such datastores much more usable, ensuring that the long-term data in them will not be simply stored, but rather be available for effective access and use via search queries. By increasing the utility and value of long-term data storage, this approach can make it cost-effective to maintain secure long-term archives.

Acknowledgments

This work was supported in part by Sandia National Laboratories, the National Science Foundation under award IIP-1266400 and industrial members of the Center for Research in Storage Systems. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National

Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[1] "Edward Snowden — Wikipedia, The Free Encyclopedia," 2013, [Online; accessed 26-Sep-2013]. [Online]. Available: http://en.wikipedia.org/wiki/EdwardSnowden

[2] M. W. Storer, K. M. Greenan, E. L. Miller, and K. Voruganti, "POTSHARDS: secure long-term storage without encryption," in *Proceedings of the 2007 USENIX Annual Technical Conference*, Jun. 2007, pp. 143–156. [Online]. Available: http://www.ssrc.ucsc.edu/Papers/storer-usenix07.pdf

[3] "Cleversafe Dispersed Storage Project," 2015, [Online; accessed 1-Feb-2015]. [Online]. Available: http://www.cleversafe.com/

[4] N. I. of Standards and T. (NIST), "Security requirements for cryptographic modules," US Department of Commerce, Tech. Rep. FIPS PUB 140-1, January 1994.

[5] "Digital Corpora – computer forensics research," 2015, [Online; accessed 1-Feb-2015]. [Online]. Available: http://digitalcorpora.org/

[6] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979. [Online]. Available: http://www.ssrc.ucsc.edu/PaperArchive/shamir-cacm79.pdf

[7] J. K. Resch and J. S. Plank, "AONT-RS: Blending security and performance in dispersed storage systems," in *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST)*, Feb. 2011. [Online]. Available: http://www.ssrc.ucsc.edu/PaperArchive/resch-fast11.pdf

[8] J. L. Dautrich and C. V. Ravishankar, "Security limitations of using secret sharing for data outsourcing," in *Proceedings of the 26th Annual IFIP WG 11.3 Conference on Data and Applications Security and Privacy*, 2012. [Online]. Available: http://www.ssrc.ucsc.edu/PaperArchive/dautrich-dbsec12.pdf

[9] A. Young and M. Yung, "Cryptovirology: Extortion-based security threats and countermeasures," in *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, ser. SP '96. Washington, DC, USA: IEEE Computer Society, 1996, pp. 129–.

[10] D. F. Ferraiolo, D. M. Gilbert, and N. Lynch, "An examination of federal and commercial access control policy needs," in *NIST-NCSC National Computer Security Conference*, ser. NIST-NCSC '93, 1993, pp. 107–116.

[11] A. Parker-Wood, C. Strong, E. L. Miller, and D. D. E. Long, "Security aware partitioning for efficient file system search," in *Proceedings of the 26th IEEE Conference on Mass Storage Systems and Technologies*, May 2010. [Online]. Available: http://www.ssrc.ucsc.edu/PaperArchive/parkerwood-msst2010.pdf

[12] J. R. Lorch, J. W. Mickens, B. Parno, M. Raykova, and J. Schiffman, "Toward practical private access to data centers via parallel oram," *IACR Cryptology ePrint Archive*, vol. 2012, p. 133, 2012, informal publication.

[13] R. Landauer, "Irreversibility and heat generation in the computing process," *IBM Journal of Research and Development*, vol. 5, no. 3, pp. 183–191, Jul 1961.

[14] "BP Statistical Review of World Energy June 2014," 2014, [Online; accessed 15-Mar-2015]. [Online]. Available: http://www.bp.com/content/dam/bp/pdf/Energy-economics/statistical-review-2014/BP-statistical-review-of-world-energy-2014-full-report.pdf

[15] Y. Breitbart, D. Georgakopoulos, M. Rusinkiewicz, and A. Silberschatz, "On rigorous transaction scheduling," *IEEE Transactions on Software Engineering*, vol. 17, no. 9, pp. 954–960, Sep 1991.

[16] M. Burrows, "The chubby lock service for loosely-coupled distributed systems," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, ser. OSDI '06, Berkeley, CA, USA, 2006, pp. 335–350.

[17] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest we remember: Cold boot attacks on encryption keys." in *USENIX Security Symposium*. USENIX Association, 2008, pp. 45–60.

[18] C. Hadnagy and P. Wilson, *Social Engineering: The Art of Human Hacking*. Wiley; 1 edition (December 3, 2010), 2010.

[19] M. I. Schwarz, "The Russian-A(merican) Bomb: The Role of Espionage in the Soviet Atomic Bomb Project," *Journal of Undergraduate Studies*, vol. 3, pp. 103–108, 1996.

[20] M. Collins, D. Spooner, D. Cappelli, A. Moore, and R. Trzeciak, "Spotlight on: Insider theft of intellectual property inside the united states involving foreign governments or organizations (2013)," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2013-TN-009, 2013.

[21] M. Bond and R. Anderson, "API-level attacks on embedded systems," *IEEE Computer Magazine*, vol. 34, no. 10, pp. 67–75, Oct 2001.

[22] "Wikipedia, The Free Encyclopedia," 2013, [Online; accessed 26-Sep-2013]. [Online]. Available: http://en.wikipedia.org/wiki/

[23] H. Melville, *Moby-Dick*. Richard Bentley (Britain) and Harper and Brothers (US), 1851.

[24] R. S. Ellis, *Entropy, Large Deviations, and Statistical Mechanics*. Springer; 1st Edition, 1985.

[25] M. S. Stoler, "Re-engineering the enigma cipher," Ph.D. dissertation, University of Louisville, 2011.

[26] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th Conference on USENIX Security Symposium – Volume 13*, ser. SSYM'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 21–21.

[27] A. Rajaraman and J. D. Ullman, *Mining of Massive Datasets*. Cambridge University Press; 1 edition (December 30, 2011), 2011.

[28] M. A. Olson, K. Bostic, and M. Seltzer, "Berkeley db," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ser. ATEC '99. Berkeley, CA, USA: USENIX Association, 1999, pp. 43–43.

[29] J. J. Wylie, M. W. Bigrigg, J. D. Strunk, G. R. Ganger, H. Kiliççöte, and P. K. Khosla, "Survivable storage systems," *IEEE Computer*, pp. 61–68, Aug. 2000. [Online]. Available: http://www.ssrc.ucsc.edu/PaperArchive/wiley-computer00.pdf

[30] M. Baker, M. Shah, D. S. H. Rosenthal, M. Roussopoulos, P. Maniatis, T. Giuli, and P. Bungale, "A fresh look at the reliability of long-term digital storage," in *Proceedings of EuroSys 2006*, Apr. 2006, pp. 221–234. [Online]. Available: http://www.ssrc.ucsc.edu/PaperArchive/baker-eurosys06.pdf

[31] M. W. Storer, K. M. Greenan, and E. L. Miller, "Long-term threats to secure archives," in *Proceedings of the 2006 ACM Workshop on Storage Security and Survivability*, Alexandria, VA, Oct. 2006. [Online]. Available: http://www.ssrc.ucsc.edu/Papers/storer-storagess06.pdf

[32] D. Zage and J. Obert, "Utilizing linear subspaces to improve cloud security," in *Dependable Systems and Networks Workshops (DSN-W)*, June 2012, pp. 1–6.

[33] A. Adya, W. J. Bolosky, M. Castro, R. Chaiken, G. Cermak, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. Wattenhofer, "FARSITE: Federated, available, and reliable storage for an incompletely trusted environment," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*. Boston, MA: USENIX, Dec. 2002. [Online]. Available: http://www.ssrc.ucsc.edu/PaperArchive/adya-osdi02.pdf

[34] H. Weatherspoon, "Design and evaluation of distributed wide-area on-line archival storage systems," University of California, Berkeley, Tech. Rep., 2006.

[35] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, May 2000, pp. 44–55. [Online]. Available: http://www.ssrc.ucsc.edu/PaperArchive/song-ieeesp00.pdf

[36] A. Haeberlen, A. Mislove, and P. Druschel, "Glacier: Highly durable, decentralized storage despite massive correlated failures," in *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI)*. Boston, MA: USENIX, May 2005. [Online]. Available: http://www.ssrc.ucsc.edu/PaperArchive/haeberlen-nsdi05.pdf

[37] Y. Chen, J. Edler, A. Goldberg, A. Gottlieb, S. Sobti, and P. Yianilos, "A prototype implementation of Archival Intermemory," in *Proceedings of DL '99*, 1999, pp. 28–37. [Online]. Available: http://www.ssrc.ucsc.edu/PaperArchive/chen-dl99.pdf

[38] K. M. Greenan, E. L. Miller, T. J. E. Schwarz, and D. D. Long, "Disaster recovery codes: increasing reliability with large-stripe erasure correcting codes," in *StorageSS '07*. New York, NY, USA: ACM, 2007, pp. 31–36.

[39] B.-G. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz, "Tiered fault tolerance for long-term integrity," in *Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST)*, 2009.

[40] S. Bellovin and W. Cheswick, "Privacy-enhanced searches using encrypted bloom filters," in *Technical Report 2004/022, IACR ePrint Cryptography Archive*, 2004.

[41] Y. C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Applied Cryptography and Network Security Conference*, 2005.

[42] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, 2006.

[43] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques*, 2004.

[44] Q. Wang and N. Borisov, "Octopus: A secure and anonymous DHT lookup," in *Proceedings of the 32nd International Conference on Distributed Computing Systems (ICDCS '12)*, Jun. 2012.