2018 IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems

Efficient Reconstruction Techniques for Disaster Recovery in Secret-Split Datastores

Sinjoni Mukhopadhyay[1,2] Joel C. Frank[3,4] Justin C. King[5,6] Daniel Bittman[2,7]
Darrell D. E. Long[2,8] Ethan L. Miller[2,9,10]
University of California, Santa Cruz[2]
Cat Digital Labs[4], *Michigan Technological University*[6], *Pure Storage*[10]
simukhop@ucsc.edu[1], jcfrank@ucsc.edu[3], dbittman@ucsc.edu[7], darrell@ucsc.edu[8], elm@ucsc.edu[9]
jcking@mtu.edu[5]

***Abstract*—Increasingly, archival systems are relying on authentication-based techniques that leverage secret-splitting rather than encryption to secure data for long-term storage. Secret-splitting data across multiple independent repositories reduces complexities in key management, eliminates the need for updates due to encryption algorithm deprecation over time, and reduces the risk of insider compromise. While reconstruction of stored data objects is straightforward if a user-maintained index is available, the system must also support disaster recovery incase the index is unavailable. Designing a mechanism for efficient index-free reconstruction, that does not increase the risk of attacker compromise, is a challenge. Reconstruction requires the association of chunks that make up an object, which is the kind of information attackers can use to identify chunks they must steal to illicitly obtain data.**

We propose two new techniques, the *set-subset* reconstruction and *secret-split secure hash (S3H)* reconstruction, which allow chunks of data to be correlated and quickly reconstructed without providing useful information to an attacker. Both techniques operate on the entire collections of secret-split chunks in the archive. While they can efficiently rebuild an entire archive, they are inefficient and impractical for rebuilding single objects, making them useless for attackers that do not have access to *all* of the data. These techniques can each be tuned to trade-off between reconstruction performance and security, reducing overall runtime from $O(N^K)$ (for N objects requiring K recombined chunks each to return the original object) to between $O(N)$ and $O(N^2)$. These runtimes are practical for archives containing as many as 10^7 objects for the secret-split secure hash method and 10^9 objects for the set-subset method. Larger archives can run these techniques with manageable runtimes by grouping data into separate smaller collections and running the algorithms on each collection in parallel.

***Index Terms*—Archival Storage, Secret-Splitting, Security, Disaster Recovery**

I. Introduction

In comparison to traditional encryption techniques, authentication-based methods are preferred for storing long-term data. Key management complexities and deprecation of encryption algorithms over time makes traditional encryption techniques unreliable for securing archival storage. Increasingly, approaches that rely on securely splitting data across archives and requiring authentication at *each* archive are becoming more common. Under this approach, data is split into n chunks so that at least $m \leq n$ of them are required to rebuild the original data, often using techniques based on Shamir's secret-splitting [1]. The chunks are then distributed across n administratively-isolated servers, ensuring both the reliability and availability of the data while also protecting it from internal attacks [2], [3].

Secret-splitting techniques effectively hide the relationships between the individual chunks for a single data object, requiring the user to know which chunks need to be retrieved to rebuild a given object. In systems such as POTSHARDS [3] and Cleversafe [4], users maintain an index containing this information, allowing them to request the necessary chunks for a specific object from the servers on which they are stored. To allow reconstruction in the absence of an index, each chunk could explicitly refer to the other chunks from the same object. However, this technique would make it straightforward for an unscrupulous archive operator, with access to one chunk from an object, to identify a small number of chunks to steal from other archives to rebuild the desired object. To avoid this major security risk some secret-split archives avoid storing pointers to chunks for the same object. If the index is unavailable, the user could obtain *all* the chunks from all the servers. Then they can test *all* possible reassembly combinations consisting of one chunk from each server, producing a combinatorial explosion with an increase in the number of data chunks. To address this issue, POTSHARDS proposed a disaster recovery technique called *approximate pointers* [3], where each data chunk points to a subset of chunks in the next server that could potentially belong to the same object.

While more efficient than the naive method, the number of reconstructions in approximate pointers increases exponentially with an increase in the size of the pointer subset of shares and the threshold scheme. It is important to maintain a higher threshold with larger number of servers to improve data privacy, as more servers will need to be compromised to get the data. However the reconstruction of such an archive is combinatorially prohibitive. In the case of approximate pointers, this increases the reconstruction time needed to rebuild the data in the archive. We propose two new techniques, the *set-subset* reconstruction and the *secret-split secure hash* reconstruction, that use hints to correlate between chunks of the same object without giving away any meaningful information to attackers, resulting in faster reconstruction of the datastore. The set-subset method tags each data chunk with a set of numbers. For a group of chunks to be a potential match, the number of unique numbers across all the chunks in the group must be below a threshold. The secret-split secure hash method uses hints derived from the unique ID of object chunks to find chunks that are a potential match. To find matching chunks the

2375-0227/18/$31.00 ©2018 IEEE
DOI 10.1109/MASCOTS.2018.00014

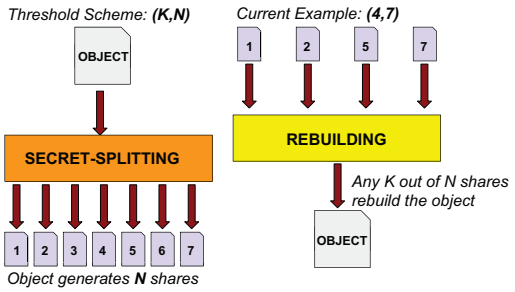

Fig. 1. Shamir's secret-splitting generates N equal-sized object shares out of which any K shares are both necessary and sufficient to rebuild the original object.

hints are recombined and tested against the unique ID. Both our methods are an improvement over approximate pointers and, efficiently reduce the reconstruction space needed to identify sets of object chunks across servers. Both techniques provide *hints* to reassemble data chunks, while not providing enough information for an attacker to identify *exactly* which chunks go together, preventing targeted theft of those chunks.

The main contributions of this research are:

1) Two new disaster recovery algorithms for rebuilding objects in a secret-split archival storage system.
2) Theoretical modeling of reconstruction times of secret-split datastores, with experimental validation for up to 10^7 shares for the *secret-split secure hash* reconstruction and 10^9 shares for the *set-subset* method.
3) Exploration of performance versus security trade-offs in parameters for these new algorithms.

II. Background

To understand how our techniques work and why they are necessary, we discuss several approaches that have been used in the past for information storage and retrieval that have motivated us to develop our disaster recovery methods.

A. Secret-splitting

Secret-splitting is a technique used to store data securely within an archive. It generates N chunks, often called *shares*, from an object [1] that are distributed among N servers within the archive, such that each server only contains a single share of each object. For a single object, $K \leq N$ shares known as *siblings*, must be combined to reconstruct the original object. Such an approach is known as a (K, N) threshold scheme, shown in Figure 1, and may be either information theoretically-secure or just computationally secure. Under information theoretic security, any $K - 1$ shares provide zero information about the object. Non-information theoretically secure schemes provide more information, but usually insufficient information to allow brute-force reconstruction to succeed. Because information theoretic security requires that *each* share be the same size as the original object, it requires N times the storage of the original un-split data. Thus archives may choose not to use information theoretically secure secret-splitting for their data because of the high storage overhead [4], [5]. Data can be split using non-information theoretically secure methods. Our methods work with object identifying metadata information, and therefore always use information theoretically secure techniques.

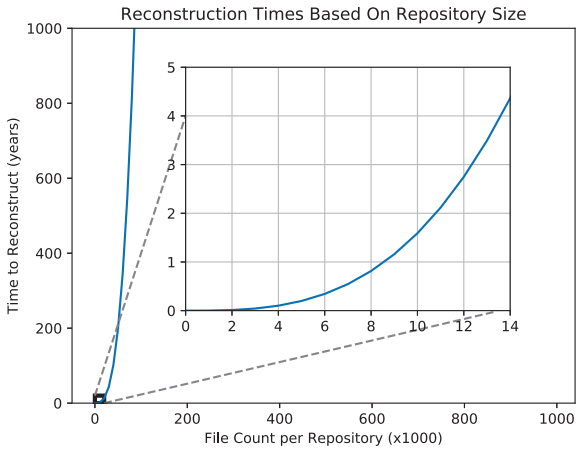

Fig. 2. Our base result shows the combinatorial explosion in reconstruction of a secret-split datastore using the brute force technique without optimizations. For a 1 MB file with threshold three and a reconstruction rate of 20 GB/sec, it would take approximately 1.5 years to reconstruct 10,000 shares.

Reconstruction time in our secret-split datastore is affected by two major factors, the threshold scheme and the number of object shares in each server. Assuming a scenario where we do not know which shares go together, an increase in the number of shares in each server results in a factorial increase in reconstruction performance. Our base results in Figure 2 shows the combinatorial explosion when none of the disaster recovery algorithms are being used. The threshold scheme decides the number of candidates to be tested in each round.

B. Secret-splitting versus Encryption

Traditional encryption techniques have many limitations, including key management and algorithm deprecation over time [6], [7], both making secret-splitting more preferable for archival storage. Other factors such as massive cloud computing power combined with advances in parallelization have made bulk resources readily available. We assume that with enough time, bulk computational resources, and advances in cryptography, many encryption techniques can eventually be broken. Additionally, for archival data, encryption has major confidentiality versus reliability trade-offs. Storing a single copy of an encryption key would produce a single point of maximum security, which could cause breach of data incase that single key was stolen. At the same time storing multiple copies of the key in different locations would provide reliability but sacrifice confidentiality making the system more vulnerable to attacks. Compromising *any* one of the total keys would allow an attacker to steal data. In addition to encryption-related problems, there are many issues specifically associated with long term storage of data such as the need for constant

updates to maintain consistency and availability in systems [8], [9]. Secret-splitting simultaneously achieves high levels of both confidentiality and reliability, and is therefore preferred for archival storage of data.

C. Systems using Information Dispersal

There are many information dispersal techniques apart from secret-splitting that have been used to securely store data. Nirmala describes Rabin's Information Dispersal Algorithm (IDA), which splits the data with a threshold of $(m : n)$ such that m shares are needed to rebuild the object. Unlike secret-splitting schemes, the total size of the resulting data in Rabin's IDA grows by a factor of n/m. To simplify the operations in Rabin's IDA, Mackay introduced IDA using Galois fields [10], [11], which uses an $n \times m$ matrix to encode and disperse the data, providing availability but sacrificing confidentiality. Abbadi preserves both reliability and confidentiality by introducing *Salted IDA* [12], providing more secure information dispersal by maintaining a secret seed and a deterministic function on the client side. PASIS [13], [14] uses erasure-coding to break fault tolerant data into fragments to reduce the space and bandwidth overhead. AONT-RS [15] implements the All-Or-Nothing Transform, which ensures that data can only be known if all of it is present, with Reed-Solomon coding. This approach reduces storage costs while simultaneously increasing security. Shor et al discusses Secure RAID that minimizes the computational overheads of secret sharing, but requires non-negligible storage overhead and random data generation [16]. Other systems using information dispersal are Oceanstore [17], Glacier [5], Mnemosyne [18] and IBM's Cleversafe [4]. Storer, *et al.*, use secret-splitting to securely store objects with POTSHARDS [3]. An index containing information about the location of the shares and the objects they belong to is maintained by the users. Every share stores a header that can be used to reconstruct the data provided all shares of the data are present. Losing the index to an attacker does not compromise data of other users. Percival [2], a query-based system provides us with a subset of relevant shares within a secret-split datastore. While these systems differ in the way that they split data across multiple archives, our techniques for rebuilding data can work with any of them, as well as other systems that distribute data across multiple systems such as SafeStore [19].

D. Disaster Recovery

Correlation between sibling shares without an index is difficult for an information theoretically secure secret-split datastore. In such cases, disaster recovery techniques are necessary to associate between related shares. The naive method to correlate between shares requires testing every possible combination between all shares in the secret-split datastore. As we saw earlier in Figure 2, this is impractical as the number of shares in the datastore increases because of the exponential increase in reconstruction time needed to rebuild the original object.

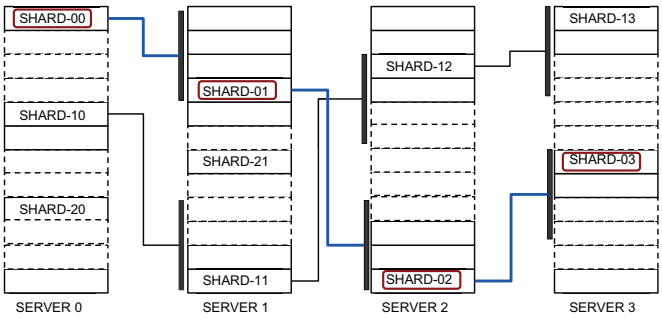

Fig. 3. Approximate Pointers uses a pointer to a set of shares in the next server such that any share from the set could be a sibling of the object. The figure shows a pointer set of size four and the path to finding siblings for object 0. Like object 0 there are pointers that point to a set of approximate siblings from each share on the server. As the threshold increases, the number of reconstructions to be tested increases.

POSTHARDS [3] uses *approximate pointers*, shown in Figure 3, as a disaster recovery method to find sibling shares, where each share points to a subset of possible siblings in the next server. This reduces the number of potential reconstructions by narrowing down sibling options. However, to find the exact position of the sibling this approach still needs to go through combinations of all shares within the pointer set in the following server. The number of reconstructions to rebuild a datastore depends on the size of the pointer set S and the threshold K:

$$S^{(K-1)} \times \text{Total shares in server}$$

The model recommends choosing a larger pointer set and a higher threshold for higher data security. Optimizations were performed on approximate pointers by masking off lower order bits of the next shard's identifier that further reduced improved the reconstruction speed:

$$(S^{(K-1)})/2 \times \text{Total shares in server}$$

Approximate pointers also presents a tradeoff between reconstruction performance and security. The advantage of approximate pointers is that, to identify a sibling, the intruder will need to steal *all* of the shares and test every combination in the pointer set. To rebuild an object, an attacker would need to compromise threshold number of archives, which is more difficult and easily detectable for a larger threshold scheme. In this case, the space of potential siblings increases exponentially with the threshold scheme for the system. However, this method fails when the chain of reference is compromised by a missing server. A missing server potentially causes data loss such that reconstruction of the datastore cannot be performed.

Our methods reduce the number of groupings that must be attempted to rebuild the objects by leveraging the following properties:

- The set-subset method uses similarity in sets attached to each share to determine if a group of shares are siblings.

- The secret-split secure hash method recombines hints derived from share IDs and tests them against the share IDs to identify potential siblings.

Due to the reduced reconstruction space, our methods rebuild a secret-split datastore faster than approximate pointers, with the added benefit that despite missing shares they can rebuild the datastore as long as threshold number of shares are present.

III. Threat Analysis

There are different points of vulnerabilities in secret-split datastores. The user-share index or the hash table being used to correlate between the shares could be stolen or compromised. This would make it easier to steal meaningful shares belonging to single or multiple objects. Despite this vulnerability, an attacker would find it difficult to steal an object, because he could easily be detected while he tries to compromise multiple servers. Secret-split datastores are mainly susceptible to insider attacks. Datastore administrators who can read and manage information about which shares are siblings and the server they belong in have complete access to local shares and can easily steal them. In this scenario, we rely on the administratively-separated property of servers within a secret-split datastore to prevent administrators from stealing shares from *multiple* servers and subsequently running our algorithms to rebuild data. Attempts to steal large quantities of data are easier to detect, making it nearly impossible to attack a secret-split datastore by doing so.

Our methods deal with two types of adversaries: an external intruder that can compromise the archive E and the inside attacker I who has information about shares and their exact locations in the archive servers. In case of I, when the adversary has access to all shares and there information on a single server, he can use this information or pointers from these shares to steal a small number of potential sibling shares from another server without being caught. Stealing a significant number of shares from the second server would be noticeable and the adversary could be intercepted. Additionally, since he does not have access to share information of the second server he cannot steal shares from any other servers. Having a high value of threshold can prevent adversaries from rebuilding the data if more than one and less than threshold number of servers are compromised. In case of E, the adversary will have to steal all shares from the archive that he has compromised in order to rebuild it. Not only will he have to go through all authentication barriers of each archive, additionally the intrusion detection systems will be able to identify access patterns from the adversary and stop him from getting hold of all the shares.

Archival storage adversaries are assumed to have unlimited time and computation power to steal object shares and rebuild the object. Attacks to steal shares over a longer period of time are difficult to detect as they may fall below the threshold that triggers most intrusion detection algorithms. For example in the case of attacker I, who has information about shares belonging to a particular object and their locations, may compromise a single server and steal the share belonging to the object from that server at some point. One missing share may alert administrators of a breach, but due to lack of any particular pattern they will assume that the attacker has gained information that is not useful. The same attacker waits a few years and breaks into another server, stealing another share belonging to the same object, leading to similar assumptions by the administrators of the second server. This scenario if repeated again and again over time may provide the attacker with threshold number of shares that he can then use to rebuild the object. Our algorithms are designed to work with storage mechanisms that prevent long-term attacks on secret-split datastores while still allowing for disaster recovery without storing pointers to object siblings.

Secret-split datastores are also susceptible to targeted theft attacks. During targeted data theft, an attacker identifies a small set of shares based on some common characteristics. They then use that information to exclusively access only those shares, thereby compromising the secure properties of a secret-split datastore. Breaking into sufficient number of servers may allow an attacker to reconstruct objects from the shares stored in those servers. The typical targeted theft attack example consists of an insider at a single location who has full access to the shares at that location, authorized access or not, and is attempting to reconstruct some data of interest by gaining access to the constituent shares on another server without detection [20]. Furthermore, because the original data can be reconstructed if enough of the sibling shares are recombined, targeted theft typically results in the unauthorized release of information.

For our methods we define two types of targeted theft attacks, strict targeted theft and loose targeted theft. Strict targeted theft is when an attacker identifies and accesses a small number of shares without detection using *only* a server's public interface. For example, the potential intruder E could use POTSHARDS' public interface to query for a share based on its ID, but that does not allow him to browse the collection of shares in any way. Building upon this attack vector, loose targeted theft refers to an attacker not only using a server's public interface, but also having the ability to access a small number of shares directly without detection. In this case the intruder would be I with administrative privileges and will not only be able to query for shares on POTSHARDS' public interface but will also access to a bunch of shares that they can then use to rebuild an object.

Since it takes into account insider threat, loose targeted theft is a more realistic attack vector. It is assumed that such an attacker would be limited by standard monitoring practices, and as a result be able to only access small sections of the datastore at a time. The term *small* is subjective and varies with system requirements and design. Neither of these methods allow an attacker to browse a server's full contents since many other attacks can be performed once an attacker has achieved that level of compromise [21]. It is by these two definitions of targeted theft that we acknowledge that compromising a server is not a binary, i.e. an all-or-nothing action.

IV. Efficient Reconstruction Algorithms

We present two novel methods that address the potentially combinatorially prohibitive task of reconstructing a large amount of secret-split data. Both methods can quickly rule out false siblings, thus reducing the possible reconstruction space. The two methods provide differing resistance to strict and loose targeted theft, and exhibit different tradeoffs between security and reconstruction efficiency.

A. Set-Subset Method

The set-subset method uses similarity between sets of numbers attached as identifiers to object shares to determine if they are siblings. In the *set-subset method*, each object i chooses a set of values S_i from a much larger set of values M shared across all objects in the archive. For simplicity, we assume that:

$$M = \{0, 1, 2 \ldots |M| - 1\}$$

The system then decides on $|S|$, which determines how many values are associated with each object's S_i. Each sibling share j generated from object i is then assigned a randomly-chosen proper subset of the object's value set: $P_{ij} \subset S_i$. The set-subset method is thus characterized by these parameters:

- M is a set of allowable values for each object's value set.
- $|S|$ is the size of an object's value set.
- $|P|$ is the size of a sibling share's value set. We use $|P| = \lceil \frac{|S|}{2} \rceil$.
- $|R_{sm}| = \frac{|S|}{|M|}$ determines the fraction of available values that are used by any given object.

First, when an archival datastore is created, the size of the value set M is fixed, along with the other parameters listed above. For each object i that is stored, the system first chooses the object's value set S_i from the integers $\{0, 1, 2 \ldots |M| - 1\}$. S_i need not be unique within the archive, so no check is required to ensure this property. When object i is split into shares, each resulting sibling share is associated with its own value set $P_{ij} \subset S_i$, and the sibling shares are then sent to separate servers for permanent storage. Note that a malicious administrator on a server storing $share_{ij}$ cannot directly access sibling shares $share_{ix}, x \neq j$, since there are *many* non-sibling shares with several values in common with $share_{ij}$. Moreover, the data store API likely does not index shares by their value sets, further increasing the difficulty of targeted theft.

To reconstruct *all* stored objects, the user must first obtain each server's shares, keeping them separate, since a given object will store at most one sibling share on a given server. The system then builds an index for each server's shares, associating a reference to a share with each *pair* of values in its value set. For example, when $|S|$ is odd, if P_{ij} contained $\langle 2, 3, 40, 505 \rangle$, a reference to $share_{ij}$ would be associated with $\langle 2, 3 \rangle$, $\langle 2, 40 \rangle$, $\langle 2, 505 \rangle$, $\langle 3, 40 \rangle$, $\langle 3, 505 \rangle$, and $\langle 40, 505 \rangle$. Because $|P| = \frac{|S|}{2}$, or 4 here, we know that any two sibling shares a and b *must* share at least two values in their value set since, if they did not, $|P_a \cup P_b| > |S|$. These indices thus allow us to quickly identify pairs of shares that may be siblings and quickly rule out those that don't have at least two values in common.

The algorithm then proceeds by building up larger and larger sets of potential sibling shares, with increasingly greater constraints on how many values must match in a potential additional sibling. For example, suppose that $|S| = 17$. If two potential siblings a and b have value sets where $|P_a \cup P_b| = 14$, any potential additional sibling c must have no more than 3 values *not* included in $P_a \cup P_b$. Since each share has 10 values in it, $|(P_a \cup P_b) \cap P_c| \geq 7$. This means that we can now index larger tuples, further increasing the filtering effect and more quickly identifying potential sibling shares. We can also rule out potential sibling groups for which there are *no* suitable shares that can join the group.

Another example is where we consider two potential siblings with an even $|S| = 6$ and $|P| = 4$: $P_a = \langle 2, 3, 5, 7 \rangle$ and $P_b = \langle 2, 5, 7, 9 \rangle$. The resulting union contains $\langle 2, 3, 5, 7, 9 \rangle$, and so has room for only one more value. A potential sibling share with $P_c = \langle 2, 3, 7, 12 \rangle$ may be added to the set, since it produces a value set of $\langle 2, 3, 5, 7, 9, 12 \rangle$ whose size is no greater than 6. However, any additional siblings needed for this group *must* have value sets containing only these values. For the same P_a and P_b, a potential sibling with $P_d = \langle 2, 5, 14, 15 \rangle$ would *not* be suitable for this potential group, even though it shares two values with each of P_a and P_b, since it would increase the total size of the value set to 7. However, it *might* be the case that this new share d could be in a different group with either a or b, but not both.

The ratio between the maximum allowed value, M, and number of values chosen, $|S|$, defines both the efficiency and loose targeted theft resistance of the set-subset method. This ratio is denoted by R_{sm}, and the effects of varying R_{sm} are discussed in the runtime analysis of the method. With a lower value of R_{sm}, there is an improvement in performance at the cost of reduced resistance to loose targeted theft.

The size of P is designed to be as small as possible while minimizing the number of sets that need to be combined in order to make a determination regarding potential siblings. By setting $|P|$ as a function of $|S|$, $|P| = \frac{|S+1|}{2}$, P is as small as it can be while allowing a determination to be made after only two sets are combined. A smaller value of P would result in more sets having to be combined before being able to make a determination since the size of the union of two disjoint P sets would still be less than $|S|$. Larger values of P, along with speedy reconstructions, are possible, albeit at the cost of increasing the likelihood of targeted theft, since sibling shares must now share more than two values in their value sets.

B. Secret-Split Secure Hash Method

The secret-split secure hash method recombines subset of hint bits to reconstruct an ID that is tested against the share ID to determine if two shares are siblings. Like the set-subset method, the main goal behind the secret-split secure hash method is to quickly reduce the reconstruction space required to identify sibling shares in order to perform a reconstruction

of a complete secret-split datastore without apriori detection of siblings.

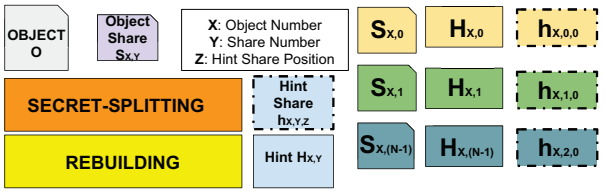

Fig. 4. Notations used in the diagrams for secret-split secure hash method to denote object, object shares, hint, hints shares, secret-split and rebuild functions.

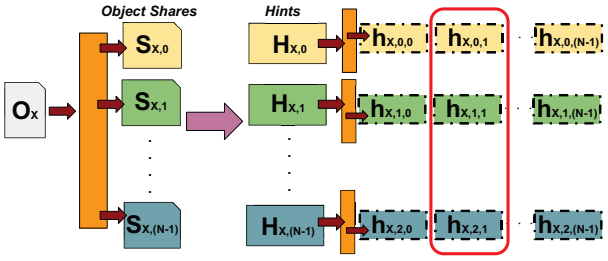

Fig. 5. An object is split into object shares, and every share is assigned a unique ID. Hints are generated from the unique ID. Hints are further secret-split into hint shares.

The secret-split secure hash method recombines hints derived from unique share IDs to identify sibling shares. It can be broadly separated into two steps; the distribution of object shares among servers with appropriate share headers and the correlation between object shares. Given a threshold scheme of (K, N), every object is first split into N siblings, and each sibling is assigned a unique 256-bit ID. This ID can either be a set of random bits or a set of hashed bits.

A subset of bits, called a *hint*, is extracted from the 256-bit unique ID. This subset of bits is further secret-split to generate a set of *hint shares*, which are distributed among sibling shares. Figure 4 shows a key to all the notations used in the diagrams. Step 1 in Figure 5 illustrates the subset of bits, H_{XY}, extracted from 256-bit ID of share S_{XY}, where X is the object number and Y is the sibling number. Step 2 in Figure 6 illustrates the secret-split pieces of the hint, the hint shares, and their distribution among the object siblings.

Hint shares enable us to perform fewer reconstructions to eliminate shares that are not siblings. The distribution is made such that each share contains a piece of its own hint and a piece of hint from all its other sibling shares. Step 3 in Figure 6 shows the distribution of object siblings among servers, where server Y stores the object share S_{XY} with an internal hash table using the hint H_{XY} as a key. Each server has a unique hash table that allows the algorithm to pre-filter out the shares that will never be siblings, all the while not enabling strict targeted theft.

We denote K_h as the hint splitting threshold which is the number of hint shares needed to reconstruct a hint. The value

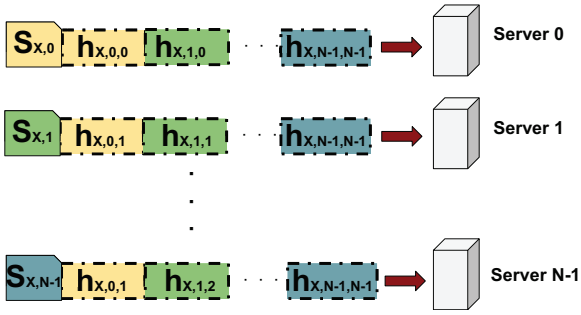

Fig. 6. Hint shares are distributed among sibling shares such that every share contains a piece of it own hint and pieces of hint from its siblings. Finally object shares are distributed among servers such that no server contains more than a single share from an object.

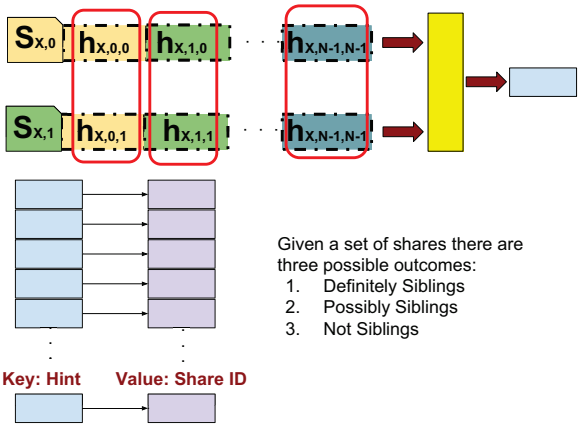

Fig. 7. Recombination of any set of shares gives three possible outcomes If the value of hint generated after recombining hint shares exists in the hash table, then the set of shares may be siblings.

of K_h does not have to be the same as the threshold of the object shares $K \neq K_h$. The splitting threshold of the hint shares does not affect the algorithm in any way as long as the total number of hint shares is the same as the total number of object shares, N in this case.

Reconstruction uses the hint as well as the hint shares from candidate shares to test for siblings. Every server has a hash table with the 256-bit unique ID of the object share as the value and its corresponding subset of hint bits as the key. Step 2 in Figure 7 illustrates reconstruction tests for a given a candidate tuple of shares. There are three possible outcomes that can occur after reconstruction of a candidate tuple of shares, the shares could definitely be siblings, the shares could potentially be siblings or they could definitely not be siblings.

Definite and potential siblings result in positive outcomes which leads to an addition being made to the value tuple of the hash table. For every negative outcome the corresponding share values are removed from the hash table. Recombining servers less than the threshold number of servers results in two possible outcomes, potential siblings and definitely not siblings. Recombination of threshold number of servers

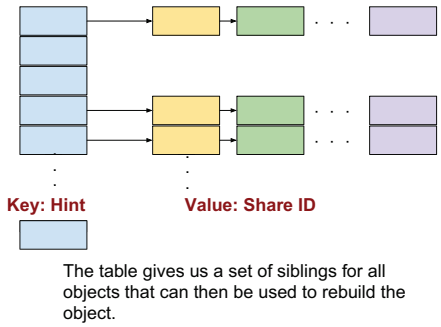

Fig. 8. In the final hash table every hint key points to a set of sibling share IDs.

generates definitely sibling tuples. Duplicate computations are avoided by checking for previously tested combinations in a bit vector, which reduces the total reconstruction time of the secret-split datastore. Once all reconstructions are finished, every hash table is left with a hint value pointing to a set of share IDs. These shares can be recombined using the object threshold scheme to give us the original object.

V. Modeling and Implementation

The first step prior to implementation is to build a model which helps us analyze the theoretical reduction in reconstruction space and ultimately used to validate each practical implementation. We use results from our theoretical model to validate our experimental results.

Table I summarizes the comparisons of Approximate Pointers to our methods in terms of efficiency, overhead, runtime and resistance to targeted thefts. All tests were run on a 4-core, 64 bit Linux machine with 24 GB of RAM. The results have been derived for a datastore up to 10^7 shares for S3HA, 10^9 shares for set-subset method and threshold schemes of $(2,5),(3,5)$ and $(4,5)$. Overall, both set-subset and secret-split secure hash methods perform better than approximate pointers. All the methods have high data availability due to secret-splitting property allowing for an object to be rebuilt as long as threshold number of shares are available. The space overhead for basic splitting and rebuilding a single object depends on the secret-splitting algorithm being used. We built all our experiments on top of the libgfshare, JErasure and Cryptopp secret-splitting libraries. Approximate Pointers allows users to query for share IDs through POTSHARDS' public interface without allowing them get any additional information about the object making it highly resistant to strict targeted theft. Our methods are immune to strict targeted theft as the object can only be rebuilt after all the shares are acquired. In case of broken chain of references, approximate pointers is highly susceptible to loose targeted theft as an intruder could access the shares from the broken server. The runtime of all methods are given in terms of number of reconstructions needed to reconstruct an object. As we discussed earlier approximate pointers has an exponential rise in reconstructions as the threshold is increased, and is thereby less efficient with a higher runtime as compared to S3HA which avoids duplicate reconstructions and has a lower runtime.

A. Set-Subset Method

The set-subset method was modeled by first identifying the factors that probabilistically determine the reduction in the amount of reconstruction space in every step of the algorithm. Leveraging every server's preprocessing, only groups with at least two values in common were tested. Recall that each server maintains groups of shares based on tuples of common shares.

The set-subset method is a series of set union operations, where an incoming set, P_1, is combined with an existing set, P_2, in a way that the existing set P_2' is updated to be the union of the two sets, $P_2' = P_1 \cup P_2$. There is a gradual increase in the number of values in the existing set over the life of the algorithm. However, the number of values in the incoming set, P_1, always remains constant.

For two candidates to be siblings, they must have a minimum number of values, x, in common between their P sets. This minimum number, x, is defined by Equation 1 and states that the minimum number of values required to be in common between the two sets is primarily determined by how many values can be added to P_2 while keeping $|P_2| \leq |S|$.

$$x = |P_1| - (|S| - |P_2|) \tag{1}$$

Looking at an example using Equation 1, given $|S| = 16$ and $|P| = 9$, if at a certain point during the execution of the algorithm an existing P_2 set has 13 values due to previous operations, these two sets need to have at least 6 values in common in order to be potential siblings. We can determine the required number of values in common between any two P sets in order for them to be siblings. Therefore, it is possible to determine the probability of those two sets having x values in common. This probability, $\rho(x)$, is defined in a simplified form in Equation 3:

$$\rho(x) = \binom{|P_1|}{x} \prod_{a=0}^{x-1} \left(\frac{|S|-a}{M-a}\right) \prod_{b=0}^{x-1} \left(\frac{|P_2|-b}{|S|-b}\right) \tag{2}$$

$$= \frac{|P_2|\binom{|P_1|}{x}(|P_2|-1)_{x-1}}{M_{x-1}} \tag{3}$$

To determine the probability that the two sets have *at least* y values in common, we need the sum of required values for y as in Equation 4:

$$\sum_{c=y}^{|P|} \rho(c) \tag{4}$$

This model was then tested by varying both the range M and the size of S, resulting in the probability in Equation 4 being altered. This directly affects the number of sibling tests needed to correctly identify all sets of siblings.

TABLE I
RELATIVE COMPARISON OF THE KEY FEATURES BETWEEN THE THREE METHODS USED TO PREVENT DATA LOSS IN SECRET-SPLIT DATASTORES.

	Approximate Pointers	Set-Subset	Secret-Split Secure Hash
Availability	**High**	**High**	**High**
Space Overhead	**Low**	**Low**	**Low**
Strict Targeted Theft Resistance	**High**	**Immune**	**Immune**
Loose Targeted Theft Resistance	**Low**	Medium	**Low**
Efficiency	**Low**	Medium	**High**
Runtime	**High**	Medium	**Low**

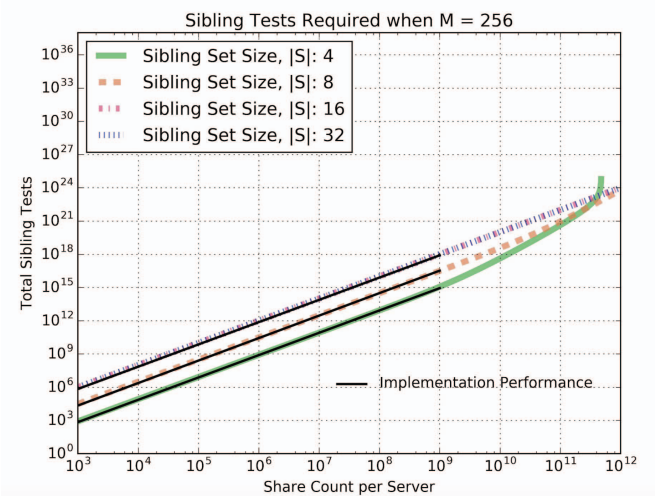

Fig. 9. Practical implementation of Set-Subset Method in comparison to theoretical model.

Figure 9 illustrates a model for $M = 256$ that highlights the number of siblings tests required for varying share counts per server and varying set sizes S. Each sibling test represents a union of each share's P set and then testing for compatibility. For example, in case of compatible shares, $|P_1 \cup P_2| \leq |S|$ whereas for incompatible shares $|P_1 \cup P_2| > |S|$. The increase in required tests when $|S| = 8$ and shares per server $> 10^9$ is due to decreased effect that pre-filtering has at that R_{sm}.

The implemented set-subset method is validated against the theoretical model. The results are illustrated using a black line in Figure 9. Each experiment was run until the sample set's variance fit a student *t*–distribution. In a *t*–distribution, the variance is equal to $d/(d-2)$, where d is the degrees of freedom in the experiment, the number of test runs minus one. It can be seen that the theoretical model accurately predicted how the implementation would perform.

B. Secret-Split Secure Hash Method

Although both the set-subset method and the secret-split secure hash method have the same goals, they have different operations. The set-subset method's efficiency relies on the probability of the set commonality whereas the secret-split secure hash method's efficiency relies on the degrees of freedom with varying hint sizes.

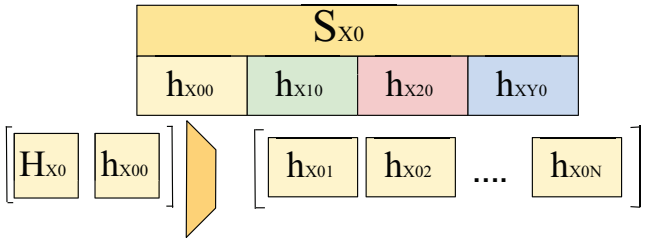

Fig. 10. Object shares can be pre-filtered by reconstructing hint shares for a particular share and the testing candidates specifically with those hint share values.

The first step to reconstruction is pre-filtering. For each share on a server, a complete set of hint shares are reconstructed. This is done by combining the original hint with that share's corresponding hint share, as shown in Figure 10, giving us a set of hint shares. This set contains the possible values for potential siblings in other server hash tables. For example, in order to find the pre-filtered set of shares on server 2, h_{X01} can be used to perform a hash table lookup on server 2.

Once pre-filtering is complete, there are three types of candidate shares that can be tested: shares that appear to be siblings, shares that are definitely siblings and shares that are not siblings. The effects of false positives are discussed in the runtime analysis section of secret-split secure hash method.

The theoretical model of the secret-split secure hash method is mainly based on the space reduction factor, that is the fraction by which the reconstruction space reduces after every round of candidate share tests. This ensures that no candidate tuple is tested more than once. We show the number of hint share combinations, λ, for step number x using a formula. Taking into consideration the splitting threshold K, the number of shares in each server c, the number of shares recombined in each step H, hint bits b and space reduction Δ. H varies from splitting threshold to the total number of shares. Here space reduction is being defined as the reduced computation space in S3HA due to the elimination of duplicate computations.

$$\lambda_x = \sum_{i=(K-1)}^{H-1} \binom{H-1}{i}(i+1)$$

Possible combinations for a set of shares
{ S_{00} S_{01} S_{02} S_{03} } :

Round 1: Contents of two servers

For example; 2 hint combinations between **{ S_{00} S_{01} } :**
(h_{000} # h_{001}) and (h_{010} # h_{001}), *where # stands for reconstruction.*

Round 2: Contents of three servers **{ S_{00} S_{01} S_{02} }**

{ S_{00} S_{01} } { S_{00} S_{02} } { S_{01} S_{02} } { S_{00} S_{01} S_{02} }
Total hint combinations: 2 + 2 + 3 = 7

Round 3: Contents of three servers **{ S_{00} S_{01} S_{02} S_{03} }**

{ S_{00} S_{01} } { S_{00} S_{02} } { S_{00} S_{03} } { S_{01} S_{02} }
{ S_{01} S_{03} } { S_{02} S_{03} } { S_{00} S_{01} S_{02} }
{ S_{00} S_{01} S_{03} } { S_{01} S_{02} S_{03} } { S_{00} S_{02} S_{03} }
{ S_{00} S_{01} S_{02} S_{03} }
Total hint combinations: 2 + 2 + 2 + 3 + 3 + 3 + 4 = 19

Fig. 11. For a threshold of (3,4) we show the number of possible combinations in each round. Duplicate combinations are tested only once. In the practical implementation we have eliminated duplicate combinations by looking them up in a bit vector at the beginning of each round.

For $x = 1$ $\Delta_1 = 2$
For $x > 1$

$$\Delta_x = \sum_{i=1}^{(x-1)} \lambda_i$$

Space reduction= 2^{λ_x}
Stepwise recombination attempts considering space reduction:

$$R = \frac{\lambda_x c^H}{2^{\Delta_x b}}$$

As the number of servers increases, the combinations of hints that can be reconstructed increases. For example calculations for Figure 11, assuming a threshold value of two, when contents of four servers are considered is: In Round 1:

$$\lambda_1 = \sum_{i=1}^{1} \binom{1}{i} (i+1)$$

$\lambda_1 = 2$ and Space Reduction is 2^{2b}
In Round 2:

$$\lambda_2 = \sum_{i=1}^{2} \binom{2}{i} (i+1)$$

$\lambda_2 = 7$ and Space Reduction increases to 2^{7b}.
In Round 3:

$$\lambda_3 = \sum_{i=1}^{3} \binom{3}{i} (i+1)$$

$\lambda_3 = 19$ and Space Reduction increases to 2^{19b}.

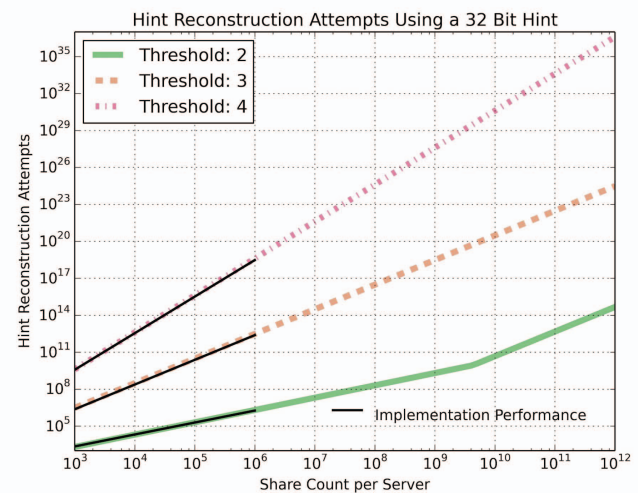

Fig. 12. Implementation of S3HA on a secret-split datastore with a million shares assuming an 8-bit hint size and 2, 3, 4 threshold number of shares.

This model was tested by varying the size of the hint, b, for several hint splitting thresholds for a wide range of number of shares per server. Figure 12 shows the model for $b = 32$ and the resulting number of hint reconstruction tests that were required. For 10^6 shares per server, it will take roughly 10^{10} reconstruction tests when using a hint splitting threshold of two, but will require 10^{18} reconstruction tests for the same size server using a hint splitting threshold of four. This illustrates that increasing the hint splitting threshold increases security. Adding more servers which need to be compromised to rebuild the object affects the performance negatively, further highlighting the speed versus security tradeoff in the secret-split secure hash method.

This method is implemented using JErasure, Cryptopp and the libgfshare secret-splitting libraries. The secret-split secure hash method's implementation was then validated against the theoretical model by running multiple performance tests and the results of which are depicted as the black line in Figure 12. This shows that the theoretical model accurately predicted how the implementation would perform.

VI. Runtime Analysis

The two core problems we address during calculation of runtime are the potential data loss due to full reconstruction of all secret-split data and the infeasible amount of time it takes to do so. Therefore it is critical to understand how the design parameters for each method affect their runtime as well their resistance to loose targeted theft. Figure 13 shows a performance comparison between the mathematical models of approximate pointers, set-subset method and secret-split secure hash method. Set-subset method outperforms both methods. Data security in approximate pointers is based on its pointer set size. A larger pointer set increases the number of share combinations needed to rebuild the archive making reconstruction without an index difficult. In Figure 13, we notice the

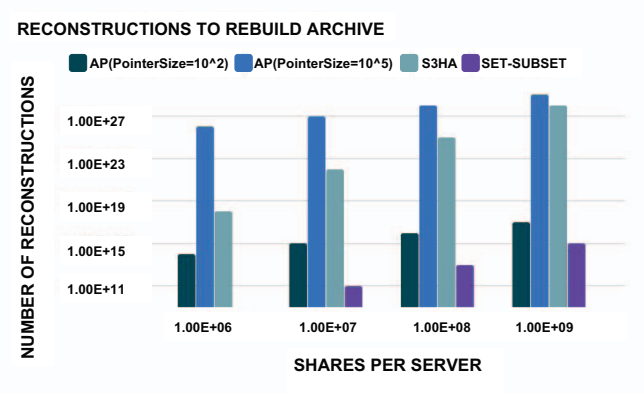

Fig. 13. Each method runs for a threshold of $(4, 5)$. Although the number of reconstructions in the graph are shown for a single threshold, performance of each method variously greatly due to their parameters. Set-subset method uses a $SiblingSetSize = 4$ and $M = 256$, and secret-split secure hash method uses 32-bit hints.

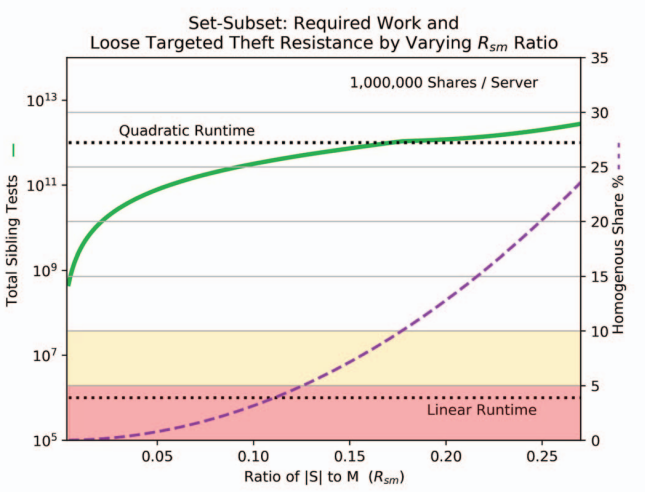

Fig. 14. The set-subset method roughly performs between $O(n)$ and $O(n^2)$ depending on the chosen values of $|S|$ and M. As one moves left on the curve, performance increases at the cost of decreased resistance to loose targeted theft. The shaded areas denote regions where the percentage of homogeneity is potentially low enough to enable targeted theft depending on requirements of the operating environment.

stark difference in the performance of approximate pointers when pointer size is increased from 10^2 to 10^5, highlighting a performance versus security tradeoff. Similar to approximate pointers, S3H also has a performance versus security tradeoff. Security in S3H can be improved by choosing lower number of hint bits. S3H with a hint size of 32-bits performs well for up to 10 million shares. As number of shares increases the performance of S3H goes down. This performance can be improved by choosing a higher number of hint bits that would then reduce the total number of combinations significantly.

A. Set-Subset Method

The runtime graphs show a quadratic and linear growth rate line for the number of operations, the number of sibling or reconstruction tests required given an input size of 10^9 shares per server. Considering a best case scenario in which a full reconstruction of all secret-split data can be performed in a single pass of a single server, we get the linear growth rate line. This occurs in the trivial case where $K = 1$, as well as when only a single pass is required of an *additional* server's contents and required siblings can be retrieved in constant time. For example, when $K = 2$, linear time reconstruction can be achieved when during a single pass of server two, each required sibling on server one can be retrieved in constant time.

The quadratic growth rate line is when $K = 2$ and is the equivalent to the brute-force reconstruction, in which one must attempt to reconstruct all combinations of shares from each server. In general, brute-force reconstruction requires polynomial runtime $O(N^K)$, where K is the minimum number of shares required for reconstruction and N is the total number of objects. The quadratic growth rate line is a special case of this general process.

The solid curve in Figure 14 shows the number of sibling tests required for various R_{sm} ratios. We already know that $|S|$ is the number of values chosen from which each share's P set values are drawn, and M defines the range of those values. Logically, as the R_{sm} decreases, each P set becomes more unique increasing the efficiency of the algorithm. This results in a quicker reduction of the reconstruction space for sets of sibling shares.

The trade-off to this performance increase is the increased susceptibility to loose targeted theft. This is denoted by moving to the left along the dashed line in Figure 14. As each P set becomes more unique, eventually it becomes trivial to identify a share's siblings using their P sets. This effect is visible in Figure 14 as the R_{sm} approaches zero. The red and yellow regions on the graph are a rough guideline of an acceptable amount level of loose targeted theft vulnerability, illustrating the R_{sm} ratios that result in 5% and 10% homogeneity of a server's shares. In this context, homogeneity refers to the similarity of P sets between shares. For example, in Figure 14 it can be seen that when the R_{sm} is approximately 0.125, 5% of the shares on each server will have identical P sets. As the P sets on a server become more homogeneous, loose targeted theft becomes more difficult to perform without detection since a larger percentage of the shares would need be accessed without detection. The shaded regions only serve as guidelines and the systems level of detection would mainly be based on their actual operating environment.

In general it can be seen that the set-subset method performs between $O(n)$ and $O(n^2)$ depending on the chosen values of $|S|$ and M. Ultimately the design choice of what R_{sm} to implement, is up to the system designer, and is based on both the size of the datastore and the environment's acceptable level of susceptibility to loose targeted theft.

Algorithm 1 defines the process of how the set-subset

Algorithm 1 SetSubset Algorithm

```
1:  function SETSUBSET
2:    existingSets = setsByServer[0]
3:    step = 1
4:    repeat
5:      for each set in existingSets do
6:        incomingSets = setsByServer[step]
7:        for each incomingSet in incomingSets do
8:          union = set ∪ incomingSet
9:          if |union| ≤ |S| then
10:           outgoingSets.push(union)
11:     existingSets = outgoingSets
12:     outgoingSets = new set
13:     step = step + 1
14:   until |existingSets| ≤ filesPerServer
```

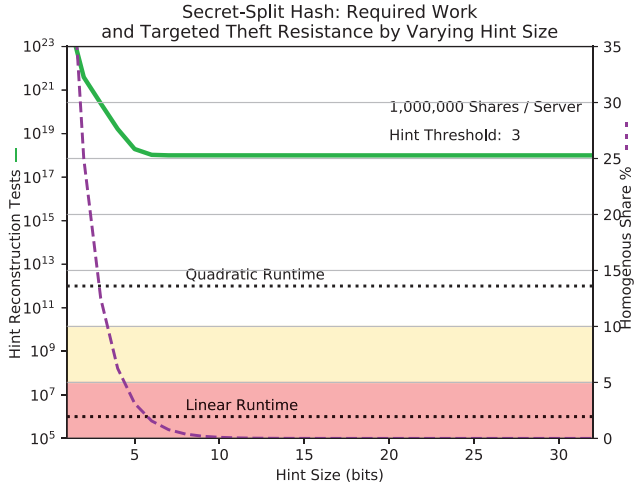

Fig. 15. Change in number of hint reconstruction tests with varying hint sizes for a million shares per server. Performance can be maximized by choosing a hint size for the expected maximum size of the datastore, which will result in the optimally linear runtime. In this example, linear runtime occurs when b=7.

method reduces the reconstruction search space. Line 2 initializes the algorithm to use server 1's P sets as the *existingSets*. Line 7 then tests each of these P sets against each of the P sets from server 2 by testing the size of their union set. If the test passes, the new union set is stored in a list to be used as the *existingSets* during the next iteration. The algorithm exits when the sets of sibling shares have all been identified or fails due to visiting all servers in the system without successfully identifying all sets of sibling shares.

B. Secret-Split Secure Hash Method

The main trade-off setting the secret-split secure hash method apart from the previous method is a large performance increase at the cost of resistance to loose targeted theft, that is if loose targeted theft is not a concern then S3H can run in linear time.

In general, the design choice of hint size is based on the trade-off between performance and resistance to loose targeted theft. If we are concerned about the low resistance to loose targeted theft in this method, then we can alternately choose to use the set-subset method.

Considering all optimizations used in this project, our model performs best for a secret-split datastore up to 10^7 objects. The model can be made to work for a larger datastore with a different grouping of objects. There are two different groupings that can be applied for different system requirements. Objects that have a higher accessibility rate can be recombined using a model that has better performance that is lower threshold and higher hint size, while objects that are highly secure can be recombined using a model with higher threshold and lower hint size. Larger datastores can also be efficiently reconstructed by grouping the shares into smaller subsets based on priority and accessibility.

Algorithm 2 defines the process of how the secret-split hash method reduces the search space required to identify sets of sibling shares. Line 2 sets up the initial candidate tuples such that each tuple consists of c hint shares, where $c = K_h - 1$, consisting of every combination of hint shares from c servers. Each of these tuples is tested against an incoming hint share from the next server in the datastore on line 8. In order to perform this test, the hints for each of the shares in the tuple, as well as the incoming share, need to be reconstructed.

If all of the sibling tests pass, the incoming share is a possible sibling with the other shares in the tuple; as a result, it is added to the tuple as well as adding the tuple to the new candidates list for the following round. Once all shares on a given server are tested against each of the existing candidate tuples, the process continues to the next server, and continues until all sets of sibling shares have been identified.

VII. Conclusion/Future Work

Data analysis and management has become an integral part of all business models. Not only are enterprises generating large quantities of data, but they also want to make the data available for an infinite period of time. Enterprises rely on this stored data to analyze trends, develop future business models and products. For example, the medical industry stores a variety of different history in its archives like patient information, preexisting diseases and their cures, tried and tested results in terms of successes and failures. Shamir's secret-splitting allows such long term data to be securely stored in archives. The information-theoretically secure property of this secret-splitting makes it time-consuming to reconstruct data without information that groups shares of the same object. Our methods provide customizable parameters that can be tuned to tradeoff between speed and performance based on system requirements.

Our implemented disaster recovery methods reconstruct data objects in a secret-split data store when an index explicitly correlating sibling shares is unavailable. These methods both tag shares belonging to an object with limited information that facilitates reassembly if *all* shares are available while

Algorithm 2 Secret-split secure hash algorithm

```
1: function SECRETSPLITHASH(st)  ▷ st: the hint splitting threshold
2:   candidateTuples = INITIALIZE
3:   step = 1
4:   repeat
5:     for each tuple in candidateTuples do
6:       hintShares = server[st + step − 1]
7:       for each hintShare in hintShares do
8:         if SIBLINGTEST(tuple, hintShare, st) then
9:           tuple.push(hintShare)
10:          newCandidates.push(tuple)
11:    candidateTuples.clear()
12:    candidateTuples = newCandidates
13:    newCandidates.clear()
14:    step = step + 1
15:  until |candidateTuples| ≤ filesPerServer
16: function INITIALIZE
17:   tempList
18:   for each repoIndex do
19:     for each hintShare do
20:       if tempList.size() is hintShareIndex then
21:         tempList.push(new Tuple(hintShare))
22:         continue
23:       tuple = server[repoIndex][hintShareIndex]
24:       clone = tuple.clone()
25:       clone.push(hintShare)
26:       tempList.push(clone)
27:   return tempList
28:
29: function SIBLINGTEST(tuple, hintShare, st)
30:   for st - 1 ≤ count ≤ |tuple| do
31:     for choose count shares from tuple do
32:       hint = RECONSTRUCT(shares, hintShare)
33:       if hint ≠ origHint then
34:         return false
35:   return true
```

preventing an attacker from identifying the specific shares needed to rebuild a particular object, even if the attacker has full access to one of the data store's servers. Both the *set-subset* and the *secret-split secure hash methods* are customizable, allowing the system implementer to choose either reconstruction efficiency or added resistance to targeted theft.

Our methods improve on existing reconstruction techniques, like *approximate pointers*, in several ways. First, they are both more resistant to the loss of an entire server, since they both rely on building groups of shares that need not be done in a strict order. Second, our approaches scale better than existing approaches, which take longer as the threshold for the number of required shares increases. Both of our methods have runtimes varying between $O(N)$ to $O(N^2)$, and greatly outperform both the naive method with complexity of $O(N^K)$ and the combinatorially prohibitive approximate pointers with runtime of $O(N \times S^{K-1})$, while providing higher availability and immunity to targeted theft.

By making these reconstruction methods available, system implementers can develop storage systems that leverage multiple independent servers to provide higher data security with increased resistance to theft and insider attack. Both our systems can survive the loss of the index that allows users to "put the pieces back together", providing approaches that can reconstruct entire archives efficiently while still maintaining high levels of security. As part of future work we plan to implement these methods with systems like POTSHARDS, Cleversafe, Percival, SafeStore and other systems that secret-split the data, to compare our system performance with different information dispersal techniques.

In combination with systems such as POTSHARDS or Cleversafe, our techniques pave the way for highly secure authentication-based archival storage that can survive changes in encryption algorithms and insider attacks on single archives while allowing the recovery of an entire data store without the need for user input.

ACKNOWLEDGEMENT

This research was supported by the National Science Foundation grant number IIP-1266400 and by the industrial members of the Center for Research in Storage Systems. The authors additionally thank Ana McTaggart, Staunton Sample and the other members of the Storage Systems Research Center for their support and invaluable feedback.

REFERENCES

[1] 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

[2] J. Frank, S. Frank, L. Thurlow, T. Kroeger, E. L. Miller, and D. D. E. Long, "Percival: A searchable secret split datastore," in *Proceedings of the 31st IEEE Conference on Mass Storage Systems and Technologies*, Jun. 2015. [Online]. Available: http://www.ssrc.ucsc.edu/Papers/frank-msst15.pdf

[3] M. W. Storer, K. M. Greenan, E. L. Miller, and K. Voruganti, "POTSHARDS—a secure, recoverable, long-term archival storage system," *ACM Transactions on Storage*, vol. 5, no. 2, Jun. 2009. [Online]. Available: http://www.ssrc.ucsc.edu/Papers/storer-tos09.pdf

[4] CleverSafe, "Highly secure, highly reliable, open source storage solution," Available from http://www.cleversafe.org/, Jun. 2006. [Online]. Available: http://www.ssrc.ucsc.edu/PaperArchive/cleversafe-whitepaper06.pdf

[5] 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

[6] R. J. McEliece and D. V. Sarwate, "On sharing secrets and Reed-Solomon codes," *Communications of the ACM*, vol. 24, no. 9, pp. 583–584, 1981. [Online]. Available: http://www.ssrc.ucsc.edu/PaperArchive/mceliece-cacm81.pdf

[7] E. Riedel, M. Kallahalla, and R. Swaminathan, "A framework for evaluating storage system security," in *Proceedings of the Conference on File and Storage Technologies (FAST)*, Jan. 2002. [Online]. Available: http://www.ssrc.ucsc.edu/PaperArchive/riedel-fast02.pdf

[8] 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

[9] 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

[10] C. S. Yeh, I. S. Reed, and T. K. Truong, "Systolic multipliers for finite fields (GF)," *IEEE Transactions on Computers*, vol. C-33, no. 4, pp. 357–360, Apr. 1984.

[11] M. C. Davey and D. Mackay, "Low-density parity check codes over GF," *IEEE Communications Letters*, vol. 2, no. 6, pp. 165–167, Jun. 1998.

[12] D. Agrawal and A. El Abbadi, "Integrating security with fault-tolerant distributed databases," *Computer Journal*, vol. 33, pp. 71–78, Jan. 1990.

[13] 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

[14] J. Hendricks, G. R. Ganger, and M. K. Reiter, "Low-overhead Byzantine fault-tolerant storage," in *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP '07)*, Oct. 2007. [Online]. Available: http://www.ssrc.ucsc.edu/PaperArchive/hendricks-sosp07.pdf

[15] 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

[16] W. H. Roman Shor, Gala Yadgar and J. Bruck, "How to best share a big secret," *11th ACM International Systems and Storage Conference*, no. 11, pp. 76–88, Jun. 2018.

[17] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "OceanStore: An architecture for global-scale persistent storage," in *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Cambridge, MA: ACM, Nov. 2000. [Online]. Available: http://www.ssrc.ucsc.edu/PaperArchive/kubiatowicz-asplos00.pdf

[18] S. Hand and T. Roscoe, "Mnemosyne: Peer-to-peer steganographic storage," *Lecture Notes in Computer Science*, vol. 2429, pp. 130–140, Mar. 2002. [Online]. Available: http://www.ssrc.ucsc.edu/PaperArchive/hand-iptps02.pdf

[19] R. Kotla, L. Alvisi, and M. Dahlin, "SafeStore: a durable and practical storage system," in *Proceedings of the 2007 USENIX Annual Technical Conference*, Jun. 2007, pp. 129–142. [Online]. Available: http://www.ssrc.ucsc.edu/PaperArchive/kotla-usenix07.pdf

[20] M. Srivatsa and L.Liu, "Countering targeted file attacks using locationguard," *In Proceedings of the 14th Conference on USENIX Security Symposium*, vol. 14, 2005.

[21] S. C. T. Shibata and K. Taura, "File-access patterns of data-intensive workflow applications and their implications to distributed filesystems," *In Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pp. 746–755, 2010.