SCUBE: A DoS-Resistant Distributed Search Protocol

Souvik Ray and Zhao Zhang
Department of Electrical and Computer Engineering
Iowa State University
Ames, Iowa 50011, USA
Email:{rsouvik,zzhang}@iastate.edu

Abstract– Many P2P-based storage systems use distributed indexing service for searching documents. There are two security issues when the nodes providing the index service are compromised by adversaries. First, the adversaries may delete the indexes or stop the program of indexing service, making the affected documents disappear in the search infrastructure. Second, the adversaries may leak the locations of the storage nodes hosting certain documents, making those nodes the target of DOS attacks. We propose a protocol called SCUBE which addresses these attacks by using secret-sharing based threshold cryptography and the concept of virtual addresses. Our results show that SCUBE performs appreciably well under different attack scenarios and incurs nominal overhead. A working prototype of SCUBE has also been implemented and tested on the Planetlab testbed.

1 Introduction

Over the past few years, peer-to-peer (P2P) applications have become very popular with the most widespread application being file sharing. P2P based distributed storage systems use indexes to improve search performance [15, 10, 25]. While indexes improve search performance, they may make the distributed storage system more vulnerable to DoS attacks [22, 13]. This is because in a highly dynamic and open P2P system, search indexes may be stored at untrusted nodes which can be compromised by the adversary. Moreover, the location of these index nodes cannot be hidden from the adversary. Index-based search systems are therefore vulnerable to the following types of attacks:

1. File-server attacks: An index server node can be compromised by a malicious adversary and the adversary can then target specific file-servers because it knows the mapping between a file and it's location. This is also called Targeted File Attacks [22].

2. Search-infrastructure attacks: In such an attack, the malicious adversary tampers or destroys the indexes stored at the compromised node thereby decreasing the reliability of the search scheme.

File-server attacks may be partially prevented using an access-control mechanism where the indexes are encrypted using a cryptographic key and the key is shared with legal users. However, encryption alone cannot prevent Search-infrastructure attacks. The reason is that in distributed index-based search systems (for example in DHT based systems [24, 17]), there is a direct mapping between an index and its location (a distributed hash table allows a group of distributed hosts to collectively manage a mapping between keys and values through the use of distributed hashing). Therefore, the location of index nodes cannot be hidden from the adversary. Again cryptography can be used to enforce access control such that only legal users can generate the location of the indexes. This however does not ensure the availability of indexes in the face of random attacks by the adversary. Traditional approaches like replication can enhance availability. However the problem with replication in a security context is that even if a single replica is broken, confidentiality can be compromised.

We propose a **S**ecret **S**haring based **S**earch scheme which we call **SCUBE**. **SCUBE** uses Shamir's secret-sharing to split the location of a file (secret) into shares and generates file-identifier shares using obfuscation techniques. A file-identifier share and a location-share are then combined to create an index-share. A (t, s) secret-sharing scheme then requires atleast t of s shares to regenerate the index (location of a file). In comparison to traditional approaches like replication, secret sharing has better security guarantees and offers more flexibility. First, it makes it difficult for an adversary to correlate shares of the same index. Second, it offers the flexibility of selecting appropriate values of t and s, achieving a tradeoff between security strength and search performance.

The contributions of this paper are as follows. First, we propose a secret sharing based search protocol that uses a DHT-based index structure [24] to speed up search. We have analyzed the resistance of SCUBE to two broad categories of DoS attacks, search infrastructure attacks and file server attacks, under different adversarial situations. We have also considered the effect of churn on the robustness of the protocol and suggested ways to improve the performance of SCUBE. Finally, our prototype implementation and performance measurements have shown that SCUBE delivers an acceptable level of QoS. Note that DoS attacks which send large traffic volumes to exhaust servers is beyond the scope of this paper.

The rest of the paper is organized as follows. We present related work in the next section and then present some background

information on secret-sharing and describe the adversary model in section 3. In sections 4 and 5, we present a detailed description of the protocol including the system model and the different cryptographic operations that are involved. The different types of attacks and how SCUBE addresses these attacks are described in section 6. This is followed by a detailed evaluation of the protocol through simulations and a prototype implementation in section 7. We discuss protocol overhead and group key management in section 8 and finally conclude in section 9.

2 Related Work

The two main components of our work are location-privacy and the use of secret-sharing to increase reliability of index based search systems. Location privacy for index based search infrastructures has been studied in [2]. The focus of this work is the use of randomization to achieve privacy of the content providers. The main inspiration for our work is LocationGuard [22]. While the focus of this work is preventing targeted file attacks on file replicas and hosts by location hiding, our framework is mitigating the effects of those DoS attacks on P2P storage systems, which target both file indexes and the files themselves. A Secure distributed Search system called Mingle was proposed in [29]. The focus of this work is an efficient access control mechanism for searching access controlled data in a small cluster of computers and priority is given to user convenience and keyword searches. Different forms of secret-sharing have been used to increase security and reliability of distributed systems. Publius [27] uses secret-sharing to generate shares of a key which is used to encrypt a file and therefore enforces tamper resistance for file data. as a tool for anonymous communications. Anonymous storage based on Rabin's information dispersal algorithm was proposed in [6], in which secret sharing is used to break a file into shares. A secret-sharing based routing approach called Split-routing was proposed in [3]. Secret sharing has also been used in software engineering [30]. This work proposed a secret sharing based compiler solution to achieve confidentiality, integrity and availability of critical data. Address obfuscation has been used in [19]. MUTE is a protocol for anonymous file sharing and uses the concept of virtual addresses to hide the identity of peers in a MUTE network. We use a similar idea for generating virtual addresses, but only to achieve location privacy.

3 Background and Assumptions

3.1 Shamir's Secret-sharing Scheme

The secret-sharing proposed by Shamir [20] is used to share a secret among a set of participants. A (t,s)-threshold scheme is a method of sharing a message M among a set of s participants such that any subset containing atleast t participants can construct the message. In the context of our fault-tolerant search scheme, the file location is a secret which is split into s shares such that any subset of atleast t shares is required to regenerate the location. Shamir's scheme uses Lagrange's polynomial interpolation on a field Z_p, where p is a prime. The dealer (content-provider in our case) generates a random polynomial of degree

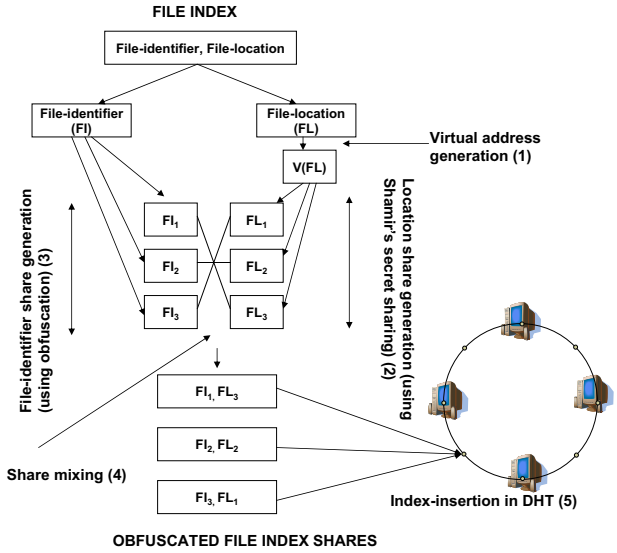

Figure 1. Protocol Steps: **Step1 –** The content provider generates the virtual address, **Step2 –** The virtual file-location is split into shares using secret-sharing, **Step3 –** File-identifier shares are generated using an obfuscation technique, **Step4 –** Random mixing of the file-identifier and file-location shares and generation of obfuscated file-index shares and **Step5 –** Insertion of the index-shares into the Chord DHT

$t-1$:

$$f(x) \quad = \quad a_0 + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1} \qquad (1)$$

This polynomial is constructed over a finite field Z_p and the coefficient is the secret (file-location). The value of p is public. The other coefficients are randomly selected by the content-provider and the location-shares are calculated as follows: $share_i = (x_i, f(x_i)), i = 1..s$. The secret can then be reconstructed as follows:

$$a_0 = f(0) = \sum_{i=1}^{t} y_i \prod_{j=1, j\neq i}^{t} \frac{-x_j}{x_i - x_j} (mod\ p) \qquad (2)$$

The complexity of Lagrange interpolation is $O(t \log_2 t)$.

3.2 Adversary Model

We assume a threat model in which the adversary controls the actions of several adversary agents. An adversary agent controls the actions of a compromised node. Thus the adversary set consists of colluding adversary agents. We use terminology from [18] to describe the threat model. The adversary agents at the compromised nodes can tamper or destroy the file index shares hosted at these nodes (**Active/Internal adversary**). We also assume that the adversary can only compromise the index hosting nodes but cannot compromise communication mediums. On the other hand, the adversary can use packet sniffing on traffic originating at a good node and can therefore observe query

packets (**Passive/External adversary**). We assume that the underlying network layer is secure and the adversary can only observe the query request packets that are addressed to it during a query lookup. Moreover, the adversary follows the Chord protocols correctly (Routing, Stabilization, Finger table updates and so on). The adversary can initiate two types of Denial of Service attacks. The adversary agents at different compromised nodes can tamper the file-index shares hosted at those nodes thereby breaking the search infrastructure. Moreover, by observing search queries and issuing bogus queries, an adversary can possibly reduce the size of the set of possible content providers for a particular file. This can result in targeted file attacks. We assume that a fraction f of a total of N nodes are compromised and therefore act as malicious nodes.

4 Protocol Overview

We first present a general overview of SCUBE. Each file in the storage overlay is associated with an unique identifier. Search protocols use keywords to search for documents. We assume that each file's identifier/keyword is unique which may be semantically attached to the content of the file. The index of each file is a tuple - {file-identifier, file-location}. The content-provider for a particular file first generates a **virtual address** using it's group key. The concept of virtual address has been used in MUTE [19] and is used for obfuscating the actual IP address. This virtual address is then split into s shares using Shamir's (t, s) secret-sharing scheme. Thus, even if an adversary manages to acquire atleast t shares, all it can generate is the virtual address. The virtual address concept therefore provides a second level of security. To generate the corresponding file-identifier shares, a keyed-hash function based **obfuscation technique** is used. This obfuscation of the file-identifier ensures that it is very difficult for an adversary or group of adversaries to correlate shares for the same file during a normal search operation. This is applicable to search schemes in which file identifiers are semantically attached to the file content. Each file-index share is then generated as {file-identifier share, file-location share} and inserted into the DHT (See figure 1 for the protocol steps). A searcher then generates the locations of atleast t untampered shares and generates the virtual address of the content provider. Finally the searcher generates the correct IP address of the content-provider by decrypting the virtual address using the group key. One of the basic requirements of SCUBE is that a searcher must possess the correct group key and must know the correct file-identifier. This aspect of SCUBE is similar to the concept of location-keys in LocationGuard [22]. We also assume that it is possible for a searcher to generate the locations of file replicas once the actual physical address of the file server can be generated.

Thus SCUBE has two levels of defense against malicious entities. First, by obfuscating the file identifiers and then distributing the shares, it makes it difficult for an adversary to correlate the shares. Thus it is very difficult for an adversary to regenerate the virtual location of a file. Moreover, by distributing the shares, the availability of indexes is improved. Second, even if the virtual address is generated by an adversary, it is very difficult to generate the physical address of the file server without knowing the correct group key. Table 3 shows the cryptographic operations used in SCUBE. The obfuscated file identifiers are 20 bytes in length.

K	Symmetric group key
N	Total number of nodes
f	Fraction of malicious nodes
(t,s)	Share scheme used
Z_p	Field used in secret-sharing
$E_K(x)$	Pseudo-random function with input x

Table 1. Protocol Notations

Protocol step	Overhead
Share generation	O(st^2) {Computational} O(s) {Cryptographic}
Share distribution	O(slogN) {Traffic}
Share gathering	O(tlogN) {Traffic} O(t) {Cryptographic}
File location generation	O(1) { Cryptographic } $O(tlogt)$ {Computational}

Table 2. Protocol overhead

5 System Architecture and Implementation

We assume an unstructured P2P based storage overlay with a total of N nodes. To enable search for files hosted at a node, a DHT based search structure is used. Each file is associated with a file-identifier which can be a keyword and a file-index is represented by the tuple {file-identifier,location}. To ensure location privacy, shares of a file-index are generated using the secret-sharing scheme and different shares are hashed to different nodes in the DHT. Each node has a symmetric group key and nodes in the same group have search-access to content hosted in that group. We leave the discussion on group key management to section 8. SCUBE uses the notations shown in Table 1.

5.1 Generation of File-index Shares

To ensure location-privacy, each file index is broken into s index-shares of which a minimum of t shares are required to regenerate the index. Each index share consists of two components: {fileid-share, location-share}. Figures 2 and 3 describe the steps of the search scheme.

Generation of Location Shares: The location shares are generated using shamir's secret-sharing scheme (see Section 3). The location of a file (32-bit ip address of the content-provider) is encrypted using K to generate a virtual address. Different shares of this virtual address (**location-shares**) are then generated using shamir's scheme.

Generation of File-identifier and Index Shares: The following procedure is used to generate file-identifier shares. Let us consider a file with identifier I. Then the j^{th} share of I is represented as

$$I_j = E_K(I||j), j = 1 \cdots s$$

Cryptographic op.	Key used
Virtual address generation	192 bit AES key
Generation of obfuscated file identifiers	HMAC-SHA1

Table 3. Cryptographic operations

where || can be a concatenation or any other operation. Thus each share of I is obfuscated using the pseudo-random function (keyed hash function in SCUBE) with group-key as input. The index-share is finally generated by a random mix-and-match of the file id and location shares.

Note that each location share generated in the previous step could have been combined with the file-identifier to generate the index share. This approach is vulnerable to leak of privacy. A group of colluding adversaries which host different index-shares of the same file can correlate the shares and gather enough shares to regenerate the virtual address of the file. Therefore the protocol requires the generation of file-identifier shares also.

5.2 Distribution of File-Index Shares

Each share generated in the previous step is then inserted into the DHT using chord routing protocol. We assume that the file servers are part of the chord ring and they insert the indexes of their files through trusted proxies or forwarders. To ensure byzantine fault tolerance, there should be a very low probability of two or more shares getting hashed to the same node. For a uniformly distributed chord ring, the **Birthday Paradox** [26] gives the relationship between N and s for which the aforementioned probability is low. Birthday Paradox states that if s keys are randomly hashed to N nodes and $s = \Omega(\sqrt{N})$, then atleast one of the nodes is likely to store more than one key. Thus, for a 1000 node network (N=1000), a value of $(s < 31)$ should be selected.

5.3 Searching File-Index Shares

Secret-sharing allows any searcher or a group of index-hosting nodes to generate the location of a file by collecting t or more shares. Thus the IP address of each file is encrypted into a virtual address. This ensures that only a legal searcher can generate the physical location of a file by collecting enough shares.

A legal searcher can generate the file-identifier shares using the method described in section 5.1. After generating s shares, it can initiate search for t or more shares. The search for each share uses the Chord routing protocol. Thus the average number of hops required for searching a single file is $\frac{t}{2}logN$.

5.4 Generating File-Location

A legal searcher can determine the file location from the virtual address and atleast t untampered shares are needed to generate the virtual address. Thus a searcher collects a set of t shares and generates the virtual address. If the address generated is bogus (either the IP address does not exist or the node with the IP address does not possess the file), the searcher repeats the process until the correct virtual address is generated.

/* Generate virtual address */
Virtual-address = E_K(ip-address)
/* Generate location shares using secret-sharing */
Z_p: public, (t,s): public
1. Select coefficients a_1,a_2...a_{t-1}
2. Generate $(x_i, f(x_i))$ $\forall i = 1..s$
/* Generate file identifier shares */
$I_j = E_K(I||j)$ $\forall j = 1...s$
/* Generate index shares */
Randomly mix and match locations shares and file-identifier shares to generate file-index shares $F_1, F_2 \cdots F_s$.
/* Distribute index shares */
Distribute the index shares on the chord ring using the following API:
for each F_j, j=1...s
PUT(I_j, (x_j, $f(x_j)$))

Figure 2. Share generation and distribution

/* Collect atleast t index shares */
Searcher generates the file identifier shares (same procedure as during generation) and initiates search using the following API:
for each I_j, j=1...t
GET(I_j)
After collecting atleast t untampered shares, searcher generates the virtual address as follows:
Virtual address = a_0 = f(0) {From eqn 2}
/* Generate file location */
Ip-address = $D_K(a_0)$

Figure 3. Share gathering and location generation

6 Analyses of Attack Scenarios

6.1 Search Infrastructure Attacks

A group of colluding adversaries can tamper the file-index shares stored at the compromised nodes which can break the search infrastructure. The adversary strategy would be to destroy atleast $(s - t + 1)$ shares of a single file index to make the file index unavailable to a legal searcher. This is because atleast t shares are needed to regenerate the virtual address of the file server.

The group of colluding adversaries can randomly compromise a set of nodes and tamper the file-index shares stored at those nodes. A second approach that can be used by adversaries is to observe queries routed through them and log information about the index providers. Due to Chord routing properties, an adversary can easily know the identity of the index provider if it is the predecessor node on the chord ring. Assuming that a group of adversaries gather information about index providers for queries routed through them, they first need to correlate index shares of the same file and then compromise the index provider nodes. Note that it is very difficult for an adversary or group of adversaries to correlate two obfuscated file identifiers as belonging to the same file. The keyed hash function (HMAC-SHA1) is used to generate the obfuscated file identifiers. Thus the obfuscated file identifier is a random 160 bit string and it is very difficult to correlate two random 160 bit strings as transformations of the

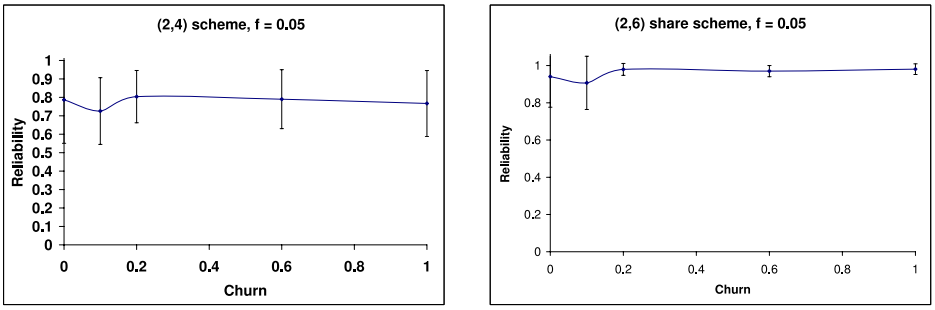

Figure 4. Variation of reliability with churn rates

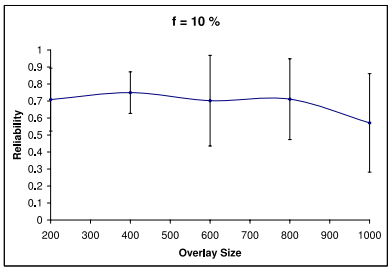

Figure 5. Variation of reliability with overlay size

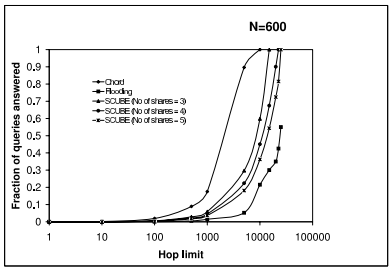

Figure 6. Search success rate (hop-limit in logarithmic scale)

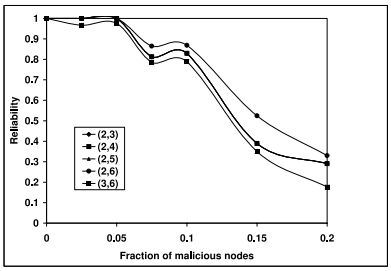

Figure 7. Reliability vs. share scheme

same file identifier.

Sybil Attacks In Sybil Attacks [7], an adversary introduces a large number of corrupt nodes to control certain regions of the chord ring. Targeted node attack [21] is a sybil attack in which the adversary corrupts certain areas of the identifier circle and this degrades the search performance. The effect of such localized attacks can be mitigated by placing different shares of the same file identifier in disparate regions of the chord identifier circle as has been observed in [21].

6.2 File-server Attacks

In File-server attacks, the adversaries employ different strategies to detect the location of a file and attack the file-server nodes. The adversaries can observe and log the queries routed through them. By observing the query traffic, the adversaries can then try to narrow down the set of possible initiators of the query and correlate the shares which originate from the same initiator. Let us assume that a group of adversaries can correlate a set of shares that they have observed and let p be the probability that an adversary lies on the search path. Then for a (t, s) share scheme, the probability that the virtual address can be generated (p_a) is $\frac{1}{p^t}$. When a large fraction of nodes are compromised, a higher number of shares is favorable.

Traffic Analysis Attacks In a Traffic Analysis attack, an adversary observes search traffic generated in the overlay and a group of colluding adversaries then use this information to correlate shares of the same query. Since for each file that is searched, atleast t queries are generated at the initiator, an adversary can use the traffic pattern to correlate shares. Note that SCUBE generates additional traffic due to the additional number of shares that have to be collected for integrity checking. One solution to thwart a Traffic Analysis Attack is to flatten the traffic pattern. Thus instead of generating a burst of traffic, the initiator spreads out the request for the different shares. A second solution is obfuscating the traffic pattern by **Share Mixing**. In **Share Mixing**, shares from different queries are mixed together in a single burst of traffic from the query initiator. We refer the reader to [9] for details.

File frequency Attacks File frequency attacks was studied in [22] in the context of read access on file replicas and an optimum value for the number of replicas was suggested. These attacks use the property that if file popularity is known to the adversaries, then by observing query traffic the file location can be deciphered. If file popularity follows a zipf distribution [4], then the adversaries can try to relate the set of observed queries to a set of files. Note that in the context of SCUBE, this can help in deciphering the relationship between obfuscated file identifiers for a particular file and the file itself. We observe that for a (t, s) scheme, as s/t increases, the probability of a file frequency attack decreases (We refer the reader to [9] for a detailed proof). Other types of attacks and how SCUBE handles those attacks is also discussed in [9].

7 Experimental Evaluation

We carried out some simulation based experiments to evaluate SCUBE using our discrete event simulator. A maximum of 1000 nodes were used in our simulations and the BRITE topology generator [5] was used to generate the physical topology. The topology was generated using the AS model. We use a chord identifier

length of 16 bits. Besides the finger table and other maintenance structures, each node also maintains a share cache. We used a maximum of 5000 file-index insertions in our experiments. A search query was randomly generated every second from a node which does not possess the corresponding file. A random fraction of the nodes in the overlay were selected as malicious nodes with the maximum value being 0.2. To consider the effect of different sets of adversary nodes in the overlay, we generated different initial overlay configurations for each experiment.

We first consider the resilience of SCUBE to infrastructure attacks. The primary metric used in evaluating the DoS resilience of the protocol is **Reliability**.

$$\text{Reliability} = \frac{\text{No of successful searches}}{\text{Total number of searches}}$$

We vary the fraction of malicious nodes and consider different threshold schemes. In our experiments, we vary the values of t and s (Our choice of values for s and t satisfies the condition derived in section 5.2). An increase in the value of t increases the computational overhead at a legal searcher. Moreover, an increase in s and t generates additional traffic during share distribution and share gathering respectively. Since an adversary needs to tamper atleast $(s-t+1)$ shares of an index to make it unavailable, an increase in t for given s decreases the search reliability. On the other hand, a small value of t increases the probability that the adversary can correlate shares and therefore launch file server attacks. Figure 7 shows the variation of reliability with the fraction of malicious nodes for different share schemes. We observe that the reliability is quite high (about 0.8) even for a large fraction of malicious nodes (f=0.1). With an increase in the value of t, reliability drops marginally. This is because the probability of the shares getting tampered increases. We also study the effect of overlay size on the DoS resistance of the protocol. Each experiment consisted of several runs to include the effect of different random distributions of malicious nodes. We assume a linear increase in the number of nodes that are compromised. Therefore, the fraction of malicious nodes is kept constant for each overlay size. If we consider the average case, the fraction of the identifier circle covered by the malicious nodes remains the same and hence there is not much of a variation in the reliability. Figure 5 shows reliability vs. overlay size plot for $f = 0.1$ when (2,4) share scheme is used. As the overlay size increases, the density of nodes on the identifier circle increases and each node then hosts a smaller fraction of indexes (assuming a constant number of searches). This decrease in the fraction of indexes hosted by malicious nodes is compensated by an increase in f and hence the number of tampered shares remains more or less constant.

P2P networks are characterized by dynamic membership because nodes join and leave rapidly. This results in **Churn**. Previous studies [12] have shown that a high churn has a significant negative impact on the performance of protocols. We use simulations to observe how realistic node join and leave scenarios affect search performance. Our churn model is based on the model proposed in [14]. We vary the join and leave rates of the nodes in our simulations to achieve varying degrees of churn. Figure 4 shows the effect of churn on the search reliability for different sets of parameters. A churn value of 0 means a static network and a value of 1 means a node join and leave rate of 1 per second. A 95% confidence interval is used in our plots. We observe that a there is no significant impact on the reliability at high churn rates. A (2,6) scheme shows a reliability of almost 1 for f = 0.05 and the reliability marginally drops for a higher fraction of malicious nodes but does not show significant changes with an increase in churn rate (frequency of node joins and leaves). We see a similar pattern for a (2,4) scheme.

7.1 Search Performance

In Chord, the average number of hops required for searching a key is $\frac{1}{2}logN$. Thus a (t, s) scheme would require $\frac{t}{2}logN$ hops on an average. Besides Data-caching, SCUBE also uses Share-caching to improve the search performance. Each node maintains a share and data cache. A share cache consists of the most recently downloaded shares and each share is tagged with the file identifier. We compare the search performance of SCUBE (with share-caching) with Random scoped flooding [11] which is similar to Gnutella [8]. The number of neighbors for each node in the overlay is in the range [3,8], according to the original Gnutella protocol. For a given hop-limit, we plot the number of successful queries that are answered. Figure 6 shows the query success rate for an overlay size of 600. In the simulations the MAX-NEIGHBOR parameter is set to 8 and the initial SCOPE for each query is set to 5. For small networks, the search performance of SCUBE is slightly better than flooding and performance improves with a smaller number of shares. On the other hand, for a large network, SCUBE performs appreciably better than flooding. For a network with N=600, even a (2,6) share scheme shows an improvement of about 90% over random flooding. This is because in flooding, search time is $O(N)$ and therefore an increase in the network size has a linear increase in search time on an average. On the other hand, even with an increase in the number of shares, search time in SCUBE has a logarithmic increase. We also plot the search success rate for Chord as a reference. Observe that the use of a single index (as opposed to shares) would improve the search time but such an approach would not be fault tolerant.

8 Discussions

We have also implemented a prototype of SCUBE and tested it on the PlanetLab [16] testbed. It runs as an application and consists of about 6000 lines of java code. For lack of space, we refer the reader to the technical report [9] for a detailed discussion of the implementation. SCUBE incurs acceptable cryptographic, storage and bandwidth overhead. Figure 2 shows the number of cryptographic operations during each step of the protocol. To give a quantitative estimate, a successful search for a (2,6) scheme requires about 0.04 ms for generation of obfuscated identifiers and 0.04 ms for decryption of virtual address (assuming an average AES decryption rate of 3.2 Mbps). We again refer the reader to [9] for a detailed discussion on the overhead. Finally, we would like to touch on the issue of Group key man-

agement for enforcing access control in distributed systems. Efficient group key management and key distribution schemes have been widely studied in literature [1, 28]. Group key management can be either centralized (existence of a centralized server) or based on distributed group key agreement protocols like Diffie-Hellman [23]. We do not specify any particular group key management scheme for SCUBE since it is beyond the scope of this paper. We assume that a group of users can share a group key and use this key to search for files within that group. Moreover, a single file owner can give different access rights to different users for the same set of files. In such a case, it is the responsibility of the file owner to distribute the keys securely to the user. Finally, the user needs to store the group key securely using some hardware or software encryption mechanism.

9 Conclusion

We have proposed a secret sharing based search protocol that uses a DHT-based index structure to speed up search. We have analysed the resistance of SCUBE to two broad categories of DoS attacks, search infrastructure attacks and file server attacks, under different adversarial situations. We have also considered the effect of churn on the robustness of the protocol and suggested ways in which the performance of SCUBE can be improved. Our prototype implementation and performance measurements have shown that SCUBE delivers an acceptable level of search performance. As part of our future work, we plan to study the robustness of SCUBE under more complicated attack models and to incorporate a mechanism to detect malicious nodes within the network. We would also like to study the effect of share re-generation and relocation on search performance.

References

[1] S. Banerjee and B. Bhattacharjee. Scalable secure group communication over ip multicast. In *JSAC Special Issue on Network Support for Group Communication*, volume 20, pages 1511–1527, 2002.

[2] M. Bawa, R. B. Jr., and R. Agrawal. Privacy-preserving indexing of documents on the network. In *Proceedings of the 29th Very Large Databases(VLDB) Conference*, Berlin, Germany, 2003.

[3] N. Borisov and J. Waddle. Anonymity in structured peer-to-peer networks. Technical Report UCB/CSD-05-1390, EECS Department, University of California, Berkeley, 2005.

[4] L. Breslau, P. Cao, and L. Fan. Web caching and zipf-like distributions: Evidence and implications. In *Proceedings of the 18th IEEE Infocom*, 1999.

[5] BRITE. http://www.cs.bu.edu/brite/.

[6] R. Dingledine, M. J. Freedman, and D. Molnar. The free haven project: Distributed anonymous storage service. In *Workshop on Design Issues in Anonymity and Unobservability*, pages 67–95, 2000.

[7] J. R. Douceur. The sybil attack. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems(IPTPS)*, 2002.

[8] Gnutella. http://gnutella.wego.com.

[9] http://archives.ece.iastate.edu/secure/00000247/01/tr01.pdf. Scube: A dos resistant distributed search protocol. Technical Report TR-2006-04-23, Computer Engineering, Iowa State University, Ames, 2006.

[10] KaZaA. http://www.kazaa.com.

[11] J. Li, J. Stribling, R. Morris, M. F. Kaashoek, and T. M. Gil. Random walks in peer-to-peer networks. In *Proceedings of the 23rd IEEE Infocom*, 2004.

[12] J. Li, J. Stribling, R. Morris, M. F. Kaashoek, and T. M. Gil. A performance vs. cost framework for evaluating dht design tradeoffs under churn. In *Proceedings of the IEEE Infocom*, 2005.

[13] J. Liang, N. Naoumov, and K. W. Ross. The index poisoning attack in p2p file sharing systems. In *Proceedings of the 25th IEEE Infocom*, 2006.

[14] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the evolution of peer-to-peer systems. In *Proceedings of the 21st Annual Symposium on Principles of Distributed Computing*, 2002.

[15] Napster. http://www.napster.com.

[16] PlanetLab. http://www.planet-lab.org.

[17] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of the 2001 ACM SIGCOMM*, San Diego, CA, 2001.

[18] J.-F. Raymond. Traffic analysis: Protocols, attacks, design issues and open problems. In H. Federrath, editor, *Designing Privacy Enhancing Technologies: Proceedings of International Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *LNCS*, pages 10–29. 2001.

[19] J. Rohrer. Mute: Simple, anonymous file sharing,http://mutenet.sourceforge.net,2004.

[20] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613??, 1979.

[21] E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash tables. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems(IPTPS)*, 2002.

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

[23] M. Steiner, G. Tsudik, and M. Waidner. Diffie-hellman key distribution extended to group communication. In *Proceedings of the 3rd ACM Conference on Computer and Communications Security*, 1996.

[24] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM*, San Diego, CA, 2001.

[25] C. Tang, Z. Xu, and M. Mahalingam. Peersearch: Efficient information retrieval in peer-to-peer networks. In *Proceedings of the HotNets -I*, 2002.

[26] W. Trappe and L. C. Washington. Introduction to cryptography with coding theory. Technical report, 2001.

[27] M. Waldman, A. Rubin, and L. Cranor. Publius: A robust, tamper-evident, censorship-resistant and source-anonymous web publishing system. In *Proceedings of the 9th USENIX Security Symposium*, pages 59–72, 2000.

[28] C. Wong, M. Gouda, and S. Lam. Secure group communications using key graphs. *IEEE/ACM Transactions on Networking*, 8:16–30, 2000.

[29] Y. Xie, D. O'Hallaron, and M. K. Reiter. A secure distributed search system. In *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, 2002.

[30] T. Zhang, X. Zhuang, and S. Pande. Building intrusion-tolerant secure software. In *Proceedings of the 3rd International Symposium on Code generation and optimization*, 2005.