Increasing DHT Data Security by Scattering Data

(Invited Paper)

Bryan Mills[†], Taieb Znati[†] [‡]
[†]Department of Computer Science,
[‡]Telecommunications Program
University of Pittsburgh Pittsburgh, PA 15260
(bmills, znati)@cs.pitt.edu

***Abstract*—**

This paper describes methods for increasing the security of data being stored in a distributed hash table (DHT) which leverages the inherent properties of the DHT to provide a secure storage substrate. The methods presented are based upon a framework referred to as "Scatter, Conceal, and Recover" (SCAR). The standard method of securing data in a DHT is to encrypt the data using symmetrical encryption before storing it in the network. SCAR provides this level of security, but also prevents any known cryptoanalisys from being performed. It does this by dividing data into multiple blocks and scattering these blocks within the DHT. The security of SCAR is provided by the property that an attacker is unable to obtain and reassemble the data blocks correctly. However, if the attacker has access to the network communication, the likelihood of a successful attack is significantly increased. This paper defines how such attacks can be executed and provides methods for ensuring data security in spite of such attacks.

I. Introduction

As the use of computers continue to expand, users are faced with a problem of finding highly available and secure storage. Documents, such as medical files, family photos, and financial records, have become digitized and are now stored on desktop computers. This places these documents at risk of loss as the storage location is not resilient to disasters. A small fire or simple hard disk failure could cause a catastrophic data loss. Equally important is the risk of data theft which might be as sophisticated as packet sniffing or as simple as stealing a compact flash drive. This motivates the need for a fully distributed secure storage system that is both highly reliable and retains the privacy of the stored data.

Peer-to-peer distributed hash tables (DHTs) provide an attractive solution to this problem. DHTs provide a distributed storage system that is decentralized yet provides a logically centralized hash table abstraction[10], [5], [8]. These structures have been implemented in a variety of systems, each with different purposes. Most of these applications are primarily concerned with the storage of public data, such as file sharing, web caches, and content distribution. However, the need for private data storage requires that mechanisms for storing and locating data within a DHT be modified.

SCAR[3] introduced a framework for providing a fully distributed DHT-based scheme for providing highly available privacy preserving storage. The main tenet of SCAR is based on the observation that it is harder to break encrypted information if the attacker can not obtain the encrypted data. SCAR achieves this by using a simple, yet powerful concept of concealment through random distribution. The goal is to break data into pieces and randomly distribute the data throughout the network so that only authorized users can locate the pieces and recover the original data.

SCAR efficiently combines erasure coding and hash chaining to increase both availability and security of the data being stored. This combination creates an apparent random distribution of the data while providing an authorized user a deterministic way of gathering the data. Although SCAR makes it computationally infeasible for an attacker to locate the stored data, it did not address the problem of an attacker discovering the data locations by monitoring network traffic. Due to the distributed routing algorithms used in peer-to-peer networks, it is very likely that an attacker would have access to such network traffic. In this paper we introduce several algorithms that mitigate the risk of such an attack.

In Section II, we provide an overview of SCAR's framework. The data availability provided by SCAR is discussed in Section III. We then define an attack model which can be used to compromise SCAR in Section IV. Section V provides algorithms that combat such attacks. In Section VI, we provide an overview of the implementation and present simulation results. This work is put in context of related work in Section VII. We then conclude in Section VIII.

II. SCAR Design

SCAR assumes a structured peer-to-peer network and uses the associated DHT to provide distributed data storage. Given this storage substrate, SCAR builds a framework for providing secure, fully distributed and highly available storage of private data. The framework does not depend upon any particular DHT implementation, but assumes that a *put/get* interface is available.

The process of storing data starts with the user providing three pieces of information: the data to be stored, a name for that data, and a password that will be used to grant access to that data. Given these pieces of information, SCAR then stores the data ensuring that the availability and security requirements are met. The storage process can be broken into 3 logical steps: pre-processing the data, splitting the data into pieces, and storing the pieces. These steps are represented in Figure 1.

As with most storage systems, there are no assumptions made about the data being stored. The first step in SCAR is to pre-process the data making it ready for storage. This pre-processing is used to ensure that the data is ready for storage and provide mechanisms for data verification after retrieving the data from storage.

Once the data is ready for storage, SCAR splits the data into multiple *storage units*. Because storage nodes join and leave the network, SCAR can not guarantee the availability of all

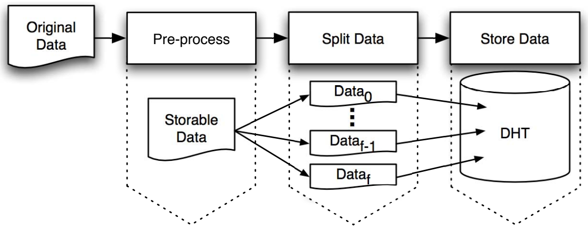

Fig. 1: Overview of SCAR's Encoding Process

of the data pieces. SCAR addresses this problem by encoding data using erasure codes during the splitting process. SCAR implements Rabin's Information Dispersal Algorithm (IDA) [4], which splits data into multiple pieces such that only a fraction of the total pieces are required for data retrieval. After the data is split, a header is added to each storage unit in order to verify the data during the retrieval process. This header is carefully constructed such that it reveals no useful information to an attacker.

SCAR relies upon a hash chain seeded with the user's password to determine where the storage units will be placed within the DHT. The hash chain then produces multiple storage locations, depicted in Figure 2.

$$
\begin{aligned}
L_0 &= \text{hash(password, data name)} \\
L_1 &= \text{hash}(L_0\text{, password, data name)} \\
L_2 &= \text{hash}(L_1\text{, password, data name)} \\
&\vdots \\
L_n &= \text{hash}(L_{n-1}\text{, password, data name)}
\end{aligned}
$$

Fig. 2: Storage Location Generation. Each argument to hash is to be concatenated together.

To retrieve data in SCAR, the user must provide the name of the requested data and a password. SCAR then determines the storage locations for each storage unit and issues *get* requests against the DHT to start retrieving the storage units. Each retrieved storage unit is verified. Once k units have been retrieved and verified, the data is reconstructed using IDA. Finally the retrieved data is verified and the original data is provided to the user. This process is depicted in Figure 3.

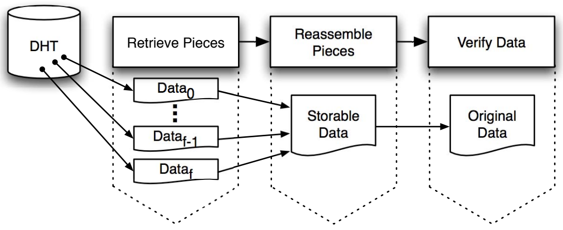

Fig. 3: Overview of SCAR's Decoding Process

III. SCAR Availablity

Availability of data within the DHT is expressed as a probability that the data stored in the DHT will be available when requested. This is dependent upon the availability of at least k out of the f nodes storing the data. This model assumes that the availability of any node is independent of the availability of any other node. Given the average network node availability ($\bar{A}$), the data availability in SCAR is described by the following formula [7]:

$$P_a(k, f, \bar{A}) = \sum_{i=k}^{f} \binom{f}{i} (\bar{A})^i (1 - \bar{A})^{f-i}$$

Looking at Figure 4, it is apparent that if the node's availability is less than 80%, then replication exhibits higher data availability than erasure coding based schemes. This is an important observation because it provides operational bounds upon the SCAR framework. The reason this bound exists is that erasure coding relies upon a quorum of nodes to be available, and as the nodes in the network become unavailable, achieving the necessary quorum becomes less likely. The advantages of erasure coding, however, is that it uses less storage and increases security; but if the node's availability is low, its effectiveness at enhancing data availability is significantly decreased.

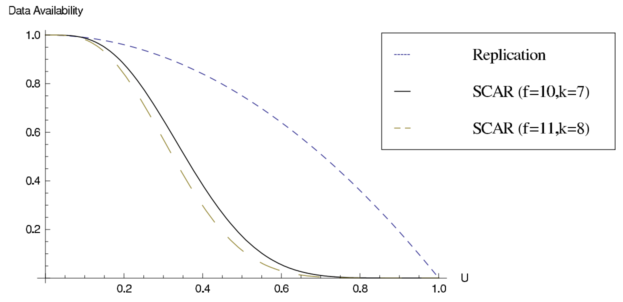

Fig. 4: Comparison between replication and SCAR as the network nodes become increasingly unavailable.

IV. SCAR Security

In this section we define attack models that SCAR will encounter. The first of these models is an uninformed attack model in which the attacker has no previous information about the data they are trying to obtain. Next, we consider a network informed attack where the attacker has access to the network traffic of users of the system.

The fundamental difference between these models is the attackers *perceived capacity* of the network. The perceived capacity represents how large the storage capacity of the network appears to the attacker. For example, if the attacker has no information about where the data pieces are scattered, the perceived capacity of the DHT is the size of the entire DHT. Clearly searching the entire DHT for a small subset of pieces requires that the attacker explore a large number of combinations. On the other hand, if the attacker can monitor the network traffic, he has a smaller perceived capacity. Therefore the attacker can refine his search space to a smaller subset of data within the DHT.

A. Attack Models

An uninformed attacker has no information about the data they are trying to recover except that the information is within the DHT. Under this model, we assume the attacker has access to all of the data being stored in the DHT, as well as implementation details of SCAR. In this case, the attackers perceived capacity is the capacity of the entire DHT.

SCAR's security is based upon the way data is split and scattered throughout the network. Even if the attacker has access to all the data in the DHT they are unable to determine which pieces are for what data object nor can they determine the proper order of these pieces. This is because the header information located in each storage unit does not reveal this information.

On the other hand, a network informed attacker has access to the network traffic being routed within the peer-to-peer network. An attacker under this model has the ability to observe network traffic generated by a particular user. This traffic would reveal which DHT locations the user is accessing.

Using SCAR alone would not protect data under this attack model. Because once an attacker knows what DHT locations are being used by a particular user, their perceived capacity is drastically reduced. In order to execute a successful attack, the attacker simply determines the proper ordering of the data locations they have observed.

V. Algorithms

As described in the previous section, SCAR does not protect data against a network informed attacker. To defend against such an attack, the attackers perceived capacity must be increased, therefore increasing his search space. In this section we will introduce two different methods which provide additional security to SCAR.

A. Batching Data Accesses

This method relies upon obfuscating which storage locations correspond to a single data object by interlacing multiple data accesses over time. This process causes the data access of object A to overlap with the data access of object B. Batching data accesses together increases the attackers perceived capacity. This is because even though the attacker would observe storage locations, they would not know which locations belonged to a single data object. This could be easily implemented in SCAR by caching data writes and requests until multiple data objects were ready to be inserted or retrieved via the DHT.

The number of data objects required for data access would be a configuration variable, min_n. The larger this value, the larger the attackers perceived capacity, thus increasing data security. However, the larger this value, the higher the latency becomes, which introduces the tradeoff between security and latency. The put/get methods are depicted in Algorithm 1 and 2. Details of SCAR's hash collision detection have been removed from the pseudocode.

```
Input: data_object, passwd
Result: data object stored in DHT
load global put_queue;
put_queue.append(data_object, passwd);
if put_queue.length > min_n then
    initialize units array;
    for i=0; i < min_n; i++ do
        data = put_queue[i].data_object;
        passwd = put_queue[i].passwd;
        data_units = scarsplit(data, passwd);
        units.extend(data_units);
        remove put_queue element i;
    end
    foreach units do randomly choose a unit
        storeunit(unit,passwd);
    end
end
```

Algorithm 1: Batching Put Method

```
Input: data_name, passwd
Output: data stored in DHT with data_name and passwd
load global get_cache;
if data_name in get_cache then
    return data_name entry in get_cache;
end
load global request_queue;
if data_name not in request_queue then
    request_queue.append(data_name, passwd);
end
if request_queue.length > min_n then
    initialize locations array;
    for i=0; i < min_n; i++ do
        name = request_queue[i].name;
        pass = request_queue[i].password;
        data_locations = scarlocations(name, pass);
        locations.extend(data_locations, name);
        remove request_queue element i;
    end
    foreach locations as location do randomly choose at
    least k locations
        data = retrieveunit(location);
        units.append(data, location.name);
    end
    reconstruct actual data from retrieved units;
    store actual data in get_cache;
end
else
    wait some period of time;
    call get(data_name, passwd);
end
```

Algorithm 2: Batching Get Method

B. Data Injection

Instead of batching concurrent data accesses, another solution is to introduce fake data when the real data is inserted into the DHT. Similar to batching, the goal is to hide storage locations for a single data object by introducing data accesses not relevant to this data object. In contrast to batching, this method reduces latency but increases storage in order to achieve greater security.

Injecting fake data into the network is trivial, however this only conceals locations when data is stored, and the retrieval process would still expose actual locations. One option would be to retrieve random storage locations during the Get process. This would require the retrieving process to know random locations that actually contain data. Because such information is not known another solution is required.

The solution is to leverage the hash collision protocol defined in SCAR. SCAR was designed to handle hash collisions at any of the storage locations provided by the hash chaining function. During SCAR's storage process, a header is added to each storage unit. One of the items in the header is the *unit signature*. The *unit signature* is a hash value of the previous and next storage locations, a sequence number, and the user's password. By hashing the password into the unit signature, it is assured that an attacker could not determine the contents of the signature, yet it is easily verified by an authorized user. During the retrieval process the *unit signature* of each storage unit is checked to be sure it matches the expected sequence. If a collision is detected, the retrieval process skips to the next location assuming a collision occurred.

Therefore, if fake data is randomly placed at storage locations specified by the data objects hash chain, then the recovery process will automatically request fake data and also identify it as fake. Assuming the fake data is made to look like real data, the attacker would not be able to determine which accesses are for fake data. Implementing this requires modification to SCAR's put function, however no changes would be required to the get function.

VI. Implementation and Simulation

SCAR was implemented as an application outside the DHT, and therefore is not bound to any specific DHT implementation. To accomplish this, there is an interface layer between the generation of the storage bins and the actual data accesses. Currently there are only two storage interfaces defined. One makes use of OpenDHT's web services API [6] and the other is a file system interface used for development.

SCAR was implemented using the Python programming language, and contains a command line interface. This command line utility allows the user to specify a file to store in the DHT. Here, the user is prompted for a password, which is used to store the data in the DHT. In addition, values for f and k can be specified on the command line, the default values are $k = 8$ and $f = 11$. The same command line can also be used to retrieve data from the DHT by specifying a filename, again prompting for a password.

The simulator was built to allow stochastic evaluation of the SCAR system. It simulates the behavior of a DHT and executes SCAR's storage and retrieval processes. To simulate nodes ON and OFF behavior, nodes alternate between exiting and entering the network based on their node type. This ON or OFF probability is based upon a specified distribution, or can be set to fixed probability. At each simulation step, all nodes are given the opportunity to change their current state.

A. Data Availability

The goal of this experiment was to validate the availability model discussed in section III. Using the simulator, we created a network with homogeneous nodes, all with the same fixed node availability. Data was inserted into the network using various values of f, k, n, and $\bar{A}$. We compared the simulation results with the results obtained using the analytical models. As can be seen in Figures 5 and 6, the simulation results show that the analytical models offer accurate estimates of SCAR's behavior.

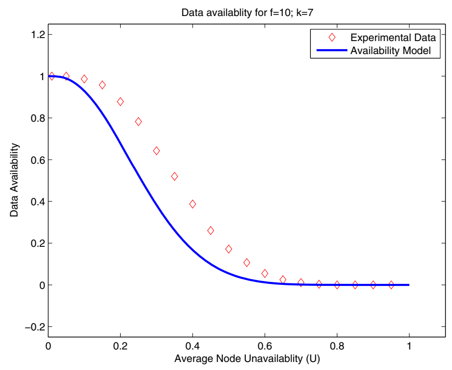

Fig. 5: Comparing Model and Experimental Results varying $\bar{U}$ using $f = 10$, $k = 7$, $n = 1000$. Each experimental point represents mean recovery rate of 1000 samples.

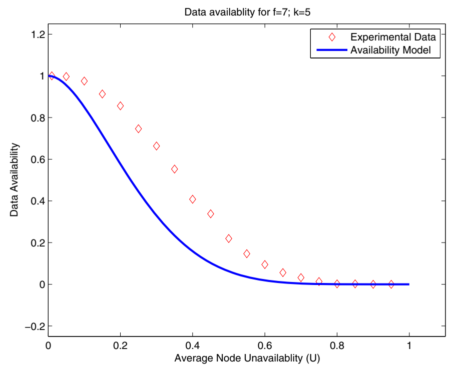

Fig. 6: Comparing Model and Experimental Results varying $\bar{U}$ using $f = 7$, $k = 10$, $n = 1000$. Each experimental point represents mean recovery rate of 1000 samples.

B. Network Sensitivity Analysis

In this section, we focus on SCAR's sensitivity to network changes. In particular, we focus on the data availability as the nodes in the network become less available. As mentioned in section III, erasure coding is only effective when the nodes themselves are fairly reliable ($\sim 80\%$). To validate this assumption, we first define probabilities of nodes leaving and rejoining the network for various node types. The simulation was executed using an exponential distribution with mean probabilities set to the values listed in Table I. The column labeled average node availability is the measured average percentage of time the nodes of that type were available across the lifetime of the network.

The purpose of this experiment is to determine how SCAR performs as the network becomes unstable. To simulate this behavior, we start with a network of only *Infrastructure* nodes and begin adding *Peepers* to the network, thus decreasing the average available of the DHT nodes. As can be observed in Figure 7, when the average node availability is above 80%, SCAR's data availability is comparable with replication; however when availability is below 80%, SCAR's data availability becomes un-acceptable. This is identical to the prediction made by our analytical models, and confirms that SCAR is only effective when the network is made of mostly available nodes.

type	p_{off}	p_{on}	node avail.
Infrastructure	1.0%	95.0%	98.0%
Power Users	20.0%	40.0%	65.0%
Peepers	80.0%	10.0%	15.0%

TABLE I: Network Sensitivity Analysis Settings.

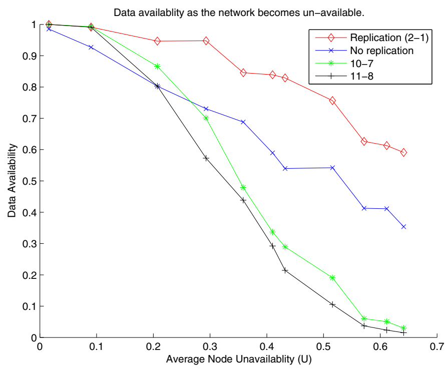

Fig. 7: SCAR Simulation as network becomes unstable. Each data point represents the mean data availability of 10 objects over 200 simulation runs.

VII. Related Work

Research on DHT security has investigated both the maintenance of the DHT structure itself and the data storage system. A survey of the various routing attacks and other DHT infrastructure security concerns can be found in [1], [9]. At the storage level there are two security concerns, privacy of the user and the privacy of the users data. Privacy of the user is concerned with anonymity for both what a user is viewing and posting within the network, as addressed in systems such as Freenet [2]. Data privacy and security is concerned with access control and authorization of the data being stored within the DHT. The typical solution to data privacy is to encrypt the data before inserting it into the DHT. While this does secure the data, it is not adequate because data encryption could be broken. Other solutions rely upon a centralized authority to grant access to the stored data. SCAR addresses this problem in a unique way that is completely distributed and does not rely upon a cryptosystem or central authorities to provide data privacy.

Research in distributed storage has also addressed data privacy concerns. The work in POTSHARDS [11] makes use of secret sharing techniques to divide data, but unlike SCAR, the locations of these pieces are known, or approximately known. Other distributed storage system rely upon the nodes themselves to enforce data access, which cannot be assumed in peer-to-peer networks. On the other hand, SCAR relies only upon the nodes to store the data, and data privacy is provided by the protocol itself.

VIII. Conclusion

Peer-to-peer technologies promise to be a solution to the distributed storage problem, but current research has failed to fully address the problem of distributed data security. SCAR offers a novel and elegant solution that provides both data security and availability using peer-to-peer networks. SCAR efficiently combines erasure coding and hash chaining to increase both availability and security. Erasure coding enables SCAR to provide high availability by splitting data into multiple pieces such that only a fraction of these pieces are required to assemble the original data. Hash chaining provides a way to deterministically conceal the data, by scattering the pieces over randomly selected peers. This creates an apparent random distribution of the data while providing an authorized user a deterministic way of gathering the data. Concealing the location of the data pieces, makes it extremely difficult for an attacker to gain access to enough information to reassemble the original data. As such, cryptoanalisys is no longer achievable, thereby increasing the security of the stored data.

As shown, SCAR provides data security against certain attack models, however if an attacker has access to the network traffic, SCAR's security is compromised. Based upon these attack models methods were developed for augmenting the SCAR framework to provide security in the face of a network informed attacker. These methods combined with the SCAR framework have been shown to provide security beyond data encryption.

References

[1] B. Awerbuch and C. Scheideler. Towards a scalable and robust dht. In *SPAA '06: Proceedings of the eighteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 318–327, New York, NY, USA, 2006. ACM Press.

[2] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009, 2001.

[3] B. Mills and T. Znati. Scar - scattering, concealing and recovering data within a dht. *ANSS*, pages 35–42, 2008.

[4] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. ACM*, 36(2):335–348, 1989.

[5] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM '01*, pages 161–172, New York, NY, USA, 2001. ACM Press.

[6] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. Opendht: a public dht service and its uses. In *SIGCOMM '05*, pages 73–84, New York, NY, USA, 2005. ACM Press.

[7] R. Rodrigues and B. Liskov. High availability in dhts: Erasure coding vs. replication. In *IPTPS '05: 4th International Workshop on Peer-To-Peer Systems.*, February 2005.

[8] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218, 2001.

[9] E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash table. In *IPTPS '02*, 2002.

[10] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01*, pages 149–160, New York, NY, USA, 2001. ACM Press.

[11] M. W. Storer, K. M. Greenan, E. L. Miller, and K. Voruganti. Secure, archival storage with potshards. In *FAST'07: Proceedings of the 5th conference on USENIX Conference on File and Storage Technologies*, pages 11–11, Berkeley, CA, USA, 2007. USENIX Association.