Private and Anonymous Data Storage and Distribution in Cloud

Qi Duan, Yongge Wang, Fadi Mohsen, and Ehab Al-Shaer
Department of Software and Information Systems,
The University of North Carolina at Charlotte
Email: {qduan, yongge.wang, fmohsen, ealshaer}@uncc.edu

***Abstract*—Cloud storage systems have received extensive attention in recent years. Many individuals and business organizations are beginning to move their data to cloud environments. It becomes increasingly important to investigate secure file storage in cloud environments. In this paper, we present a secure distributed file distribution system in which the customers can directly choose appropriate design parameters and service providers. In our scheme, we use novel coding techniques that almost achieve the Shannon information bound with very efficient coding and decoding process. Our evaluations show the correctness and efficiency of the coding scheme. We show that the problem to find the satisfying file distribution under certain cost and security constraints is NP-hard, and present the Satisfiability Modulo Theories (SMT) formalization to find the satisfying data share distribution with cost and security constraints. The SMT formalization is flexible to be applied to other threshold based cloud file distribution system and can accommodate other constraints. We also analyse the security of the scheme by defining the security metric (compromising probability) for both the eavesdropping and DoS attackers and show that one must carefully choose design parameters to achieve the required security.**

I. Introduction

In recent years cloud storage systems are becoming increasingly popular for individuals and organizations to store their data. In cloud storage the data is stored in multiple virtualized pools, which are generally maintained by third parties that operate large data centers. Individuals or organizations that require their data to be stored can buy or lease storage capacity from hosting companies. The resources in cloud storage are virtualized and the contents from a single customer may span across multiple servers.

There have been extensive researches and systems on storing data in different drives or locations to achieve high reliability and security. For example, RAID [14] array has been extensively used and is the de factor standard for reliable computer system design. Though RAID array is normally used within a single computer system, data could also be split into fragments and stored in distributed locations (or nodes). For example, in the seminal paper, Rabin [15] proposed the Information Dispersal Algorithm (IDA) to coding information into n pieces that will be stored among n servers such that the recovery of the information is possible when there are at most $t = n - k$ failed servers (inactive but not malicious Byzantine style servers). It should be noted that Rabin's IDA scheme is essentially an application of Reed-Solomon error code [16] in storage systems. In recent years, there have been many researches in distributed storage systems based on advanced coding techniques. For example, the work in [9] and [8] used information flow graphs and random linear coding to achieve information theoretic minimum functional repair bandwidth.

On the other hand, we have seen many researches on designing efficient coding techniques for RAID array systems and for distributed storage systems. Most of these researches focus on designing codes based pure exclusive or (XOR) operations. These codes include the EVENODD codes [5], which can tolerate two disk faults and correcting one disk errors, and Huang's [10] extension of EVENODD code for tolerating three disk faults, etc.

Most previous researches of coding schemes in cloud storage focus on the coding scheme used by the service providers. Service providers can provide data availability but not confidentiality since the service providers cannot be trusted. Even if the service providers can be trusted, the data confidentiality can never be guaranteed in cloud storage. It will be more secure if the customer can do the coding and choose the appropriate service providers to distribute the file. We also believe that, in the future, the number of cloud storage service providers will continue to increase and it is possible for the customer to outsource data storage to a significantly large number of service providers.

In this paper we present a novel framework of secure cloud data storage from the customer perspective. In our framework, the customer encodes the data with an efficient coding method and sends one or more shares of the encoded data to each of the n service providers. The original data can be retrieved from at lease k ($k < n$) shares. Less than k shares of data can retrieve nothing of the original data.

Our main contributions of paper include:

1) We present two novel cloud file storage coding and distribution schemes. The first scheme is IDA based, and the second one is XOR based. In our schemes, the encoding and decoding is directly done by the customer and the privacy of the stored data is guaranteed by the property of the threshold based encodings.
2) We show that the problem to find the satisfying cloud file distribution scheme under certain cost and security constraints is NP-hard, and present the Satisfiability Modulo Theories (SMT) formalization to find the sat-

isfying data share distribution with these constraints. The formalization can be applied to different types of threshold based cloud file distribution schemes and is flexible to include other constraints.

3) We implement the scheme and evaluate the performance. Our evaluation shows that it is feasible and efficient to do the segmentation, encoding and decoding in customers' machines.
4) We analyse the security of the scheme by defining the security metric (compromising probability) for both the eavesdropping and DoS attackers. We show that one must carefully choose the n and k values to achieve the required security.

The rest of the paper is organized as follows. Section II discusses the design and attack model used in our work. Section III presents the coding schemes. Section IV presents the hardness and SMT formalization of the constrained file distribution problem. Section V presents the security analysis, implementation and evaluation. Section VI presents the related works. Section VII concludes the paper.

II. Design and Attack Model

A. Design Model

In our cloud storage service model, the customer uses her machine to communicate with the service provider. The service provider most likely uses cloud infrastructure of other company or utilizes her own. A customer may use many service providers, and a service provider may be used by many customers. The service to infrastructure relation is also a many to many relation, similar to the customer-service relation. For instance, Google drive is a service on Google cloud but Google cloud is used by other service providers as well. The reasons why we mention this are performance and security. A customer who is interested in using more than one service provider needs to know the underlining infrastructure or authority domain. If two service providers are running on top of the same infrastructure or authority domain then there is no point of diversifying the service to these two providers. An attacker needs just to hack one infrastructure or authority domain to get into customers' files stored in different service providers. A good strategy then is for customers to diversify their service providers in regard to their underline infrastructure or authority domain to assure performance and security.

Fig. 1 and Fig. 2 show an example of the procedure to distribute and retrieve files. In the distribution stage, we partition the file into N segments, then we do XORing between those segments according to our chosen distribution scheme (or other operations, based on the coding scheme). Here 'P' means its an original segment, whereas 'X' means an XORed segment (resulted from XORing two original segments). Then we distribute them among the providers. In the retrieving stage, we choose a number of providers, get all segments ('P' and 'X') and use a sequence of XOR operations between an 'X' and a 'P' to get another original segment. The process continues until all original segments are retrieved.

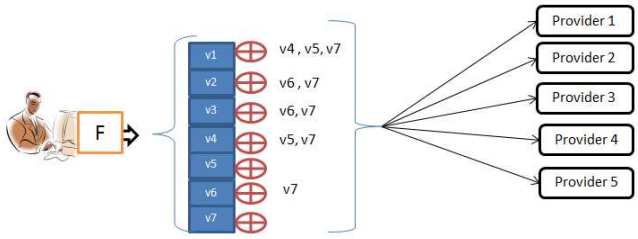

Fig. 1. The procedure to distribute files by the customer

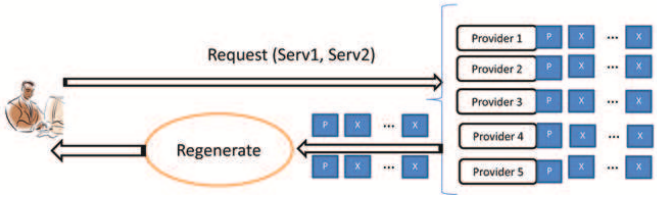

Fig. 2. The procedure to retrieve files by the customer

We could assign the fragmentation, storing the fragmentation information, and reassembling the fragments functionalities to a middleware (or proxy) component that resides between the customer and service provider. However, we decided not to do this for three main reasons: trust, cost, and scalability. Trusting the middleware requires adding more techniques to our model which we tend to keep as simple as possible. The middleware also suffers from being a bottleneck, which affects the overall performance and security. Most customers want to use our technique to store their personal files on set of free accounts on different service providers. Those kinds of customers are looking for zero cost, so adding a middleware component, which most likely will be offered with some prices, would be inappropriate. Industry customers on the other hand can afford the extra cost but would surely consider our technique to reduce the cost, and to avoid any performance deprivation.

B. Adversary Model

In the cloud file distribution systems, the general adversary model is "curious-and-vulnerable", which means that the cloud servers are vulnerable to Byzantine failures and external attacks. Byzantine failures are caused by hardware errors, software bugs, or system misconfiguration. External attacks are caused by natural disasters or malicious behaviors. The cloud server may provide the storage service honestly and correctly but it may try to figure out information from the stored data.

We consider two types of attackers: eavesdropping attackers and DoS attackers. For eavesdropping attackers, we assume that the adversary does not interfere with the normal functioning of the network but only sniff on a limited number of of the network links. For DoS attackers, we also assume that the adversary can block a limited number of the links or nodes to disrupt the communication between certain service providers and the user.

III. Coding techniques

In our system, the customer needs to register to a couple of cloud storage service providers. For example, we may assume that Google Drive, Apple iCloud, Microsoft Sky drive, and Amazon Simple Storage Service (Amazon S3). The customer may register one or few accounts under providers. As an example, in our experiment, we registered an account for each of the five providers: one at Google Drive, one at Apple iCloud, one at Microsoft Sky Drive, one at Amazon S3, and one at Dropbox. Our goal is to achieve reliability and privacy at the same time. In particular, if a customer uses our application to store a data file in the cloud, we want the following guarantees:

1) Less than k providers can get nothing of the customer data file.
2) Customer data file could be recovered from any k of the n providers. In another word, if storage services within $n-k$ providers are not available, the customer will still be able to recover her data files from the fragments stored at the remaining storage service providers.

We designed two techniques for the file fragmentation and coding. The first one is based on Rabin's IDA scheme and the second one is based on XOR code.

The IDA code construction in this section is a 3 out of 4 scheme and the XOR code construction is a 2 out of 6 scheme. One can easily extend the design to k out of n schemes. In another word, the customer could register for n different cloud storage service providers and store her data within these providers. The customer should be able to recover her data file from every k storage providers.

A. IDA based coding

As we have mentioned, though our techniques could be easily extended to general k out of n schemes, we concentrate our discussion on 3 out of 4 schemes in this section. In our system, each byte is considered as elements in $GF(2^8)$ and we use the irreducible polynomial $x^8+x^6+x^5+x^4+1$ over $GF(2)$ for the field modulo operations.

When a customer initiates the reliable cloud storage service with our application, the application will first generate four vectors α_1, α_2, α_3, $\alpha_4 \in GF(2^8)^3$ such that every subset of 3 different vectors are linearly independent. In our design, this process is done only at the beginning when the customer installs the application. In order to achieve data privacy, vectors should be carefully chosen such that each vector will contain at least two non-identity element from $GF(2^8)$. For example, the following choice is not acceptable.

$$\alpha_1 = (00000001, 00000000, 00000000)$$
$$\alpha_2 = (00000000, 00000001, 00000000)$$
$$\alpha_3 = (00000000, 00000000, 00000001)$$
$$\alpha_4 = (00000001, 00000001, 00000001)$$

Now assume that the customer wants to store a file F in the cloud. We divide the file F in blocks $F = F_1, F_2, \cdots$, and each block consists of 3 bytes. In another word, assume that $F_i = (b_{i1}, b_{i2}, b_{i3})$ where b_{ij} are bytes. The fragments that will be stored at storage provider i ($i = 1, 2, 3, 4$) will be:

$$S_j = F_1\alpha_j^T, F_2\alpha_j^T, \ldots$$

When the customer has two or more accounts within one storage provider, the shares for that storage provider will be divided into different groups and each account server will receive one group of data for storage. The split of the data into groups for each provider and the data group size depends on the account quota etc.

Note that each storage provider stores data of length $|F|/3$ and the entire data stored at these storage is of length $(4/3) \cdot |F|$. This scheme is optimal since the data stored at 3 providers has the length of $3 \cdot |F|/3 = |F|$ which is the minimal information required for the recovery of file F.

Given the information at 3 providers j_1, j_2, j_3, let A be the 3×3 matrix $(\alpha_{j_1}^T, \alpha_{j_2}^T, \alpha_{j_3}^T)$ and A^{-1} be the inverse of A. Then the original file F could be recovered as $S_j A^{-1}$.

In our application, there is no data stored at the client machine. The meta-data that are needed for the application to store and recover data files include

1) Four generator vectors α_1, α_2, α_3, α_4.
2) Four decoding matrix A_1^{-1}, A_2^{-1},A_3^{-1},A_4^{-1}, where A_1^{-1} is the inverse of the matrix $(\alpha_2^T, \alpha_3^T, \alpha_4^T)$, A_2^{-1} is the inverse of the matrix $(\alpha_1^T, \alpha_3^T, \alpha_4^T)$, etc.

These meta-data will only need to be processed and stored at the system set up time. We encode the entire meta-data using 3-out-of-4 Shamir Secret Sharing scheme [17] and store the four shares at the four providers respectively. Each time the customer needs to get three shares from three providers to recover the meta-data for file storage and retrieval. Note that it is very inefficient to store the whole file using the Shamir Secret Sharing scheme.

It is straightforward to see that the above schemes have achieved our goals: (1) any two providers will not be able to recover the the customer data file content, and (2) the customer could recover her data from any three providers.

B. XOR based coding

As we have mentioned in the Section I, there has been extensive interests in designing XOR based coding techniques. For reliable storage system design, people normally prefers systematic code. A systematic code is an error-correcting code where the input data is embedded in the encoded output. In another word, the parity data is simply appended to the source block in a systematic code and the data retriever does not need to recover the original source symbols if all of them are received correctly. However, for our goal, trivial application of systematic code is not preferred since we do not want any single server learn the information of the original data file.

In previous section, we implemented a 3-out-of-4 scheme using IDA scheme. In this section, we show an example of a 2-out-of-n scheme with $n = 7$ using array BP-XOR codes which were introduced by Wang [19], [20] recently. These codes were based on colored edge graph models [21]. As an example, assume that the customer has seven accounts

$S_1, S_2, S_3, S_4, S_5, S_6, S_7$. The data file could be recovered from any two accounts. For a data file F, it is split into 6 fragments of equal size (when the size of F is not a multiple of 6, we append empty strings at the end of the file to make it a multiple of 6). A file could also be split into several parts and each part is split into 6 fragments. In another word, $F = v_1v_2v_3v_4v_5v_6$ and $|v_i| = |F|/6$ for $i = 1, \cdots, 6$. The fragments that each storage server will receive is shown in Table I. Note that the original file F can either be encrypted or not encrypted, depending on the security requirement of the customer. If the original file is encrypted, the customer can encode the encryption key using the 2-out-of-7 Shamir Secret Sharing scheme [17] and store the seven shares at the seven providers respectively. The customer may also choose to store the encryption key in his/her local machine or other devices, but the security of the key will be a big problem.

It is straightforward to check that the original data file could always be recovered from any two storage providers. Note the total size of data stored in one provider is half of the size of the original file. This complexity is much smaller than typical binary linear coding and other encoding techniques for storage systems such as EVENODD.

IV. Hardness and Formalization of Constrained File Distribution

A. Problem Definition and Hardness

The cloud coding and distribution schemes we presented in this paper distributes the same number of segments to every provider. Other threshold based file storage schemes may not use this method of distribution. Here we consider the general problem of distributing n shares of a file into m storage providers $p_1, \ldots, p_m$, where k ($k < n$) out of the n shares can retrieve the original file and less than k shares leak no information for the original file. There is an associated cost c_i to store one share into provider p_i ($1 \leq i \leq m$). There are h authorities $a_1, \ldots, a_h$, and every authority a_i ($1 \leq i \leq h$) is associated with a subset P_i of storage providers. Note that a storage provider can be associated with multiple authorities.

We have the following requirement for file distribution:

- The total cost for file distribution should be no more than budget B (**Cost Constraint).**
- One should avoid to distribute k or more than k shares to providers that belong to the same authority (**Security Constraint).**

We denote the problem as CFD (Constrained File Distribution).

Next we show that the problem is NP-Complete.

Theorem 4.1: CFD is NP-Complete.

Proof: We can reduce 3SAT to CFD. Given a 3SAT instance S with r variables and u clauses, we can construct a CFD instance C as follows. For every variable γ_i ($1 \leq i \leq r$) and its negation $\overline{\gamma_i}$, we create a provider p_{2i-1} and p_{2i} ($1 \leq i \leq r$). We also have an additional provider p_0. In total we have $2r + 1$ providers. For every clause l_i ($1 \leq i \leq u$) , we create an authority a_i that associates the three providers that correspond to the literals in the clause. We also create r additional authorities $a_{u+1}, a_{u+2}, \ldots, a_{u+r}$ where a_{u+i} ($1 \leq i \leq r$) is associated with providers p_{2i-1}, p_{2i} and p_0. The unit share storage cost for providers p_i ($1 \leq i \leq 2r$) is 1 and the cost for provider p_0 is 0. We also set $n = r+1$, $k = 3$, and $B = r$. If S is satisfying, we can create a satisfying distribution for C. We just distribute one share of the file to provider p_{2i} if variable γ_i is true in the satisfying assignment of S and to provider p_{2i-1} if variable γ_i is false in the assignment. We also distribute one share to p_0. Now we can see that the total cost for the distribution is B. We can also see that a clause is not satisfied if and only if all the three providers correspond to the three variables in the clause are all distributed with one share. This means that clause l_i ($1 \leq i \leq u$) is satisfied if and only if the security constraint for authority a_i is satisfied. In addition authorities a_{u+i} ($1 \leq i \leq r$) have exactly two associated providers that have distributed share. This means the distribution scheme satisfies the constraints. One the other hand, if there exists a satisfying distribution for C, then we can find a satisfying assignment for S. Because the budget bound is r, we must distribute one share to p_0 and the other r shares to $p_1, p_2, \ldots, p_{2r}$. We have the following observation:

- Every provider p_i ($1 \leq i \leq 2r$) can have at most one share otherwise authority a_{u+i} will have three shares in its associated providers.
- Only one of the providers p_{2i-1} and p_{2i} can be distributed with one share otherwise authority a_{u+i} will have three shares in its associated providers.

Now we can set variable γ_i to be true if p_{2i} is distributed with one share and set γ_i to be false if p_{2i-1} is distributed with one share. We also have that clause l_i ($1 \leq i \leq u$) is satisfied if and only if the security constraint for authority a_i is satisfied. This means that one can find a satisfying assignment for S if there is a satisfying distribution for C, which completes the proof.

■

B. SMT Formalization for Constrained File Distribution

We can use the following SMT formalization for file distribution. SMT is a powerful tool to solve constraint satisfaction problems arise in many diverse areas including software and hardware verification, type inference, extended static checking, test-case generation, scheduling, planning, graph problems, etc. [4]. An SMT instance is a formula in first-order logic with equalities involving uninterpreted functions [7].

Suppose we want to distribute n shares to m providers, the cost constraint for share distribution can be formalized as

$$\sum_{1 \leq i \leq m} w_i c_i \leq B$$

$$\sum_{1 \leq i \leq m} w_i \geq n$$

$$w_i \geq 0,\ 1 \leq i \leq m$$

Here w_i is the integer variable that denotes the number of shares distributed to provider p_i ($1 \leq i \leq m$).

TABLE I
DATA SHARES EACH STORAGE SERVER WILL RECEIVE

$v_1 \oplus v_6$	v_2	$v_3 \oplus v_1$	$v_4 \oplus v_2$	$v_5 \oplus v_3$	$v_6 \oplus v_4$	v_5
$v_2 \oplus v_5$	$v_3 \oplus v_6$	v_4	$v_5 \oplus v_1$	$v_6 \oplus v_2$	v_3	$v_1 \oplus v_4$
$v_3 \oplus v_4$	$v_4 \oplus v_5$	$v_5 \oplus v_6$	v_6	v_1	$v_1 \oplus v_2$	$v_2 \oplus v_3$

The security constraint can be formalized as

$$\sum_{a_j \in P_i} w_j < k, \ 1 \leq i \leq h$$

Note that other constraints can also be formalized conveniently by SMT. The formalization can also be applied (with minor adjustment) to other threshold based coding schemes.

Note that the problem may also be solved by other heuristic algorithms. However, when there are multiple constraints it is usually difficult to design the algorithm that satisfies all the constraints. Our SMT formalization can be easily extended to incorporate various constraints.

V. SECURITY ANALYSIS, IMPLEMENTATION AND EVALUATION

A. Security Analysis and Evaluation

For eavesdropping attackers, we define the compromising probability Γ to be the probability that at least k of the n coding shares is obtained by the eavesdropper. If we assume the probability that one share is eavesdropped is ζ and the probability to eavesdrop different shares is independent, we have

$$\Gamma = 1 - \sum_{i=0}^{k-1} \binom{n}{i} \zeta^i (1-\zeta)^{n-i} \quad (1)$$

Note that the customer can use different user names for different providers to store the shares of the same file. However, the attacker can easily correlate different shares of the same file by the IP address and timestamp of the data packets.

For DoS attackers, we define the compromising probability Γ to be the probability that the at least $n-k+1$ shares of the n coding shares are blocked or corrupted (irretrievable) by the attacker. If we also assume the probability that one share is blocked or corrupted is ζ and the probability to block different shares is independent, we can have a similar equation.

Impact of k and n on Γ: Fig. 3 and Fig. 4 shows the effect of n and k on compromising probability Γ. Here we define the compromising tolerance η to be k/n. In Fig. 3, $\zeta = 0.2$, and in Fig. 4, $\zeta = 0.3$.

- When $\eta > \zeta$, increasing n helps the security of the scheme (decreasing Γ). When $\eta < \zeta$, increasing n has inverse impact on the security (increasing Γ).
- When n increases, Γ converges to some value. When $\eta > \zeta$, Γ converges to 0. When $\eta < \zeta$, Γ converges to 1. When $\eta = \zeta$, Γ converges to 1/2. The impact on Γ diminishes quickly after n reaches some value. This means that after some threshold (which depends on k), increasing n has little impact.

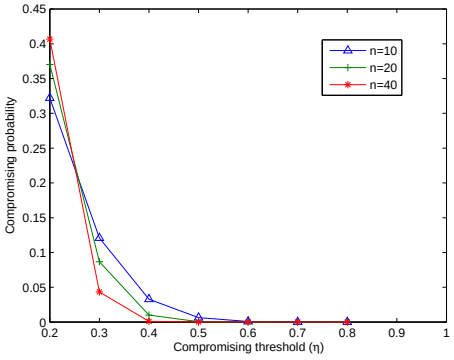

Fig. 3. Impact of η and n on Γ with $\zeta = 0.2$

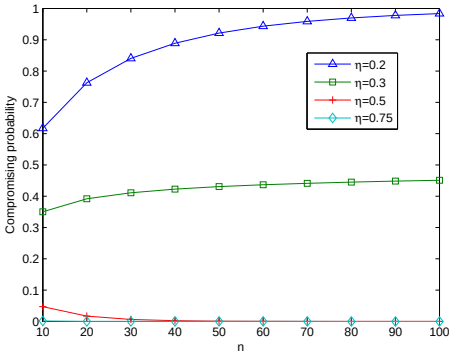

Fig. 4. Impact of η and n on Γ with $\zeta = 0.3$

Impact of k and n on Γ in Random Topology: In a practical network, the probability to eavesdrop different shares may not be independent. Even if we assume that the probability to eavesdrop different link in the network is uniform, the probability to eavesdrop different shares may not be uniform since different shares may go through overlapping links. In this case Equation 1 cannot be used to calculate the compromising probability. We evaluate the compromising probability in random network through simulation. We use BRITE [13] as the random network generator and apply the Waxman model [13] to generate random topologies with the two parameters of Waxman model as $\alpha = 0.2$ and $\beta = 0.15$. The network growth type is set to be incremental. We randomly generate networks with 300 nodes and the customer and n service providers are randomly chosen from the 300 nodes. The communication path between the customer and any service provider is chosen to be the shortest path. We assume that the attacker will eavesdrop any link with probability δ.

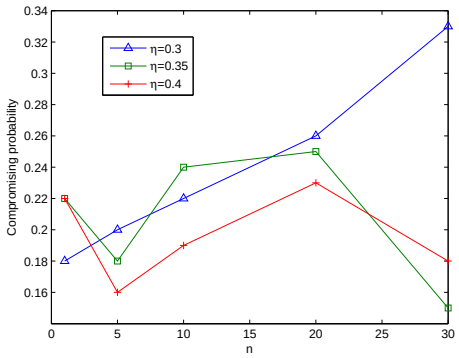

Fig. 5. Impact of η and n on Γ in random topology with $\delta = 0.05$

Fig. 5 shows the effect of n and k on compromising probability Γ in random topology with $\delta = 0.05$. All the values in the figure are the average of 100 runs. From this figure, we can see that when the compromising tolerance η is 0.3, the compromising probability increases when n increases, which means that increasing n has inverse impact on the security. When η is increased to be 0.35, the compromising probability decreases for large n. This means that there exists some value of η which is the turning point between positive impact and inverse impact.

Note that in the above figure we assume that the attacker does not know the topology and the probability for any link to be eavesdropped is uniform. If the attacker knows the paths for the data file, he/she can choose those links in the paths to eavesdrop. To find the minimum subset of links to eavesdrop to get the required information is a minimum partial set cover problem [18].

B. Implementation

We implemented our prototype using Microsoft Visual Studio 2010 Ultimate version and we used the C Sharp language. The underlying operating system is Windows7 Home Premium running on a Dell Optiplex990 Machine. The machine specifications are as follows: Intel(R) Core(TM)i5-2400 CPU @ 3.10GHz and 4.00 GB of RAM. The C Sharp implementation provides the customer with a Graphical customer Interface allowing customers to select the file to be securely saved on the cloud. Our tool divides the file into several segments and then generates new segments by XORing different pairs of the original ones. The number of segments can be left to the customer to specify. Our implementation considers seven segments. In the evaluation we will show how the number of segments and the cost of storage are related. The tool then allocates all segments among the different cloud providers. We consider five providers: Dropbox, Google Drive, Sky Drive, Syncplicity and University of North Carolina Charlotte (UNCC) H: drive. UNCC H: drive gives 200MB of file storage space to every student for data backup at the end of each semester. The other four providers are different brands for easy file management in the cloud to allow the customers to sync, access, share and automatically backup files online. They all provide free edition but with different features and specifications (e.g. Dropbox: 2 GB and Up to 18 GB, Sky Drive: 7 GB, Google Drive: 5 GB, Syncplicity: 2 GB). Providers vary in sharing policy, file types, application support and version control. The most important feature for the providers are safety, security and performance. We excluded the price since we are using the free versions of those providers. We installed and configured the client softwares provided by those companies. Those client softwares allow customers to deal with a local drive. A customer can just drag and drop the files into the client softwares and they will be synchronized automatically to the cloud. For the recovery, given that we have five providers and with the toleration of three providers' failures, the tool can retrieve any file in ten different ways. The tool allows the customer to choose any two providers to retrieve the file.

C. Overhead Evaluation

We conducted an experimental evaluation to demonstrate the feasibility and performance of our approach. The evaluation study consists of three main domains: segmentation, recovery and storage cost. In the segmentation, we evaluated our scheme implementation on different file types and sizes. For the recovery, we compared the performance of the ten recovery options (pairs of providers). In the storage cost, we study the relation between the number of segments, number of providers and the extra space required to save the XORed segments.

Evaluating the Segmentation Process The segmentation process includes reading the file, partitioning it into certain number of segments (specified by the customer) and then placing the overall segments among the providers. The segmentation process may require padding empty bytes at the end of the last segment. Depending on the size of the file (denoted as S) and the number of segments (denoted as N), one can calculate the number of padded bytes (denoted as A). Distributing the file segments (original, XORed, padded) over providers is done based on the 2 of 5 array BP-XOR codes described in [19], [20].

To evaluate the segmentation complexity in terms of processing overhead, we tested it on Winrar files with different sizes. We run the program on every file ten times to get the average results. Fig. 6 shows the processing overhead for these files. From the figure we can see that the processing overhead (measured in milliseconds) increases linearly with the file size.

Evaluating the Retrieving Process The recovery evaluation shows the time to retrieve the original file from any two pairs of providers. A customer who chooses to securely save her files on the five providers will have ten different ways to retrieve the file. In Fig. 7, we measure the time needed to recover a file using all possible combination. We draw the results for every server. The results show that the four providers (Google, Dropbox, Syncplicity, Sky Drive) achieve very close retrieving times, but the UNCC drive shows the worst performance, even we are reading and writing in local drives. This is because that the file retrieval process is managed by different client software provided by each provider, which may affect the response time and the overall recovery time.

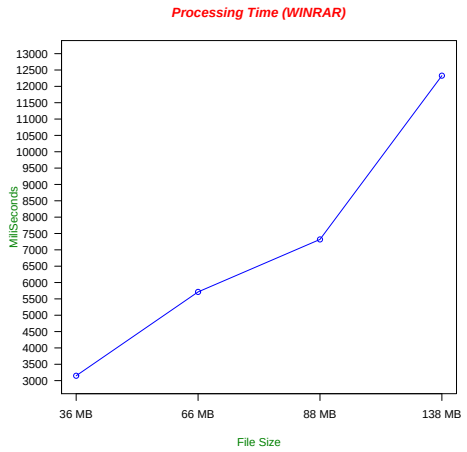

Fig. 6. Processing time for the Winrar file

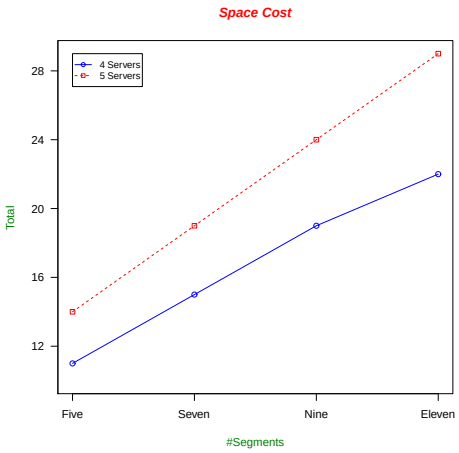

Fig. 8. Cost of storage for odd number of segments

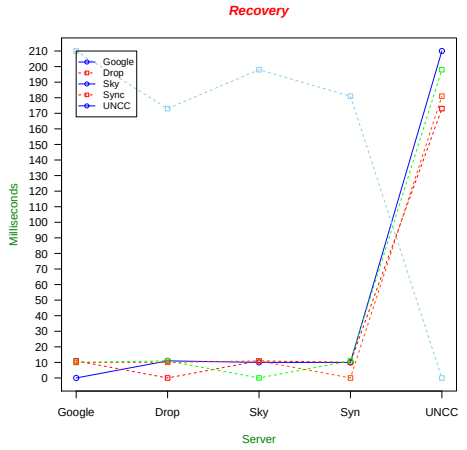

Fig. 7. Time to recover a file

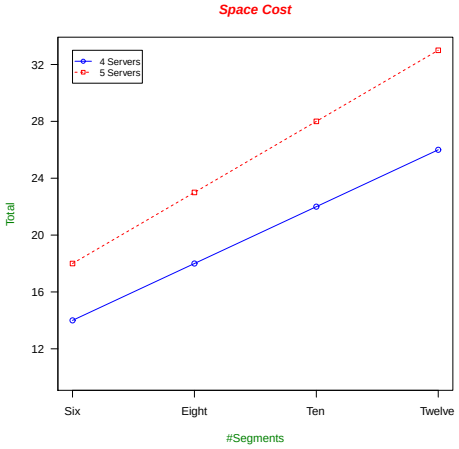

Fig. 9. Cost of storage for even number of segments

Another factor might be that the segments that are distributed to each server play a factor in reported recovery time. Whereas the schemes we implemented in this paper distribute the same number of segments to every provider, other schemes may not. In this case, distributing a small number of shares (segments) to those slow providers is a good practice.

Evaluating the Storage Cost We also evaluated the storage cost of the XOR based scheme with different scenarios. Those schemes are different in the number of providers and the number of failures to tolerate. For the purpose of evaluation, we choose two scenarios: (1) four providers and two failures (2 out of 4 scheme), (2) five providers and three failures (2 out of 5 scheme). The coding for these scenarios are the array BP-XOR codes described in [19], [20]. Fig. 8 shows the cost of storage (the number of duplicates in terms of the original segments) for odd N and Fig. 9 shows the cost for even N. These figures show that the number of extra space can be double of the number of original segments in most cases or triple in the worst cases.

Evaluating the Share Distribution Fig. 10 shows the SMT solving time for the share distribution formalization. The formalization is solved by the Yices SMT solver [1]. We assume that there are 15 providers and 5 authority domains. Every provider has a probability of 1/3 to be associated with an arbitrary authority domain. The file encoding contains 30 shares and at least 10 shares are required to recover the file. The cost to store a share in a provider is a random number between 1 and 3. We use the formalization in Section IV to find the satisfying distribution scheme. From Fig. 10 we can see that the solving time decreases when the cost upper bound increases, and remains stable after certain point. This is because the number of satisfying solutions increases rapidly when the upper bound increases. When the upper bound reaches some threshold, it has no effect on SMT solving time anymore.

VI. Related Works

MDS codes such as Reed-Solomon code [16] are widely used in many applications though it is not efficient both in

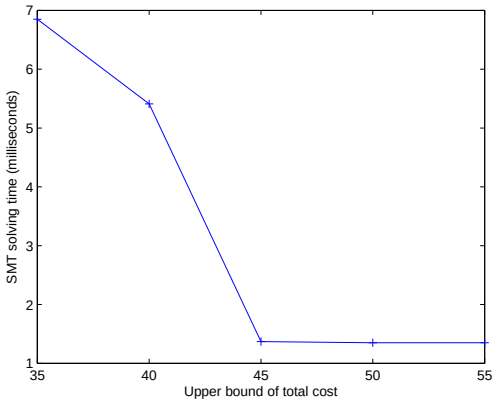

Fig. 10. SMT solving time for the share distribution formalization

coding and decoding process for cloud storage. LT codes [12] proposed by Luby is the first rateless erasure codes that are very efficient as the data length grows.

The OceanStore [11] is a utility infrastructure designed to span the globe and provide continuous access to persistent information, where data is protected through redundancy and cryptographic techniques. B. Bindu et al. present an effectual and adaptable cloud data storage scheme in [3]. The scheme achieves the data storage correctness, allows the authenticated customer to access the data and data error localization, i.e., the identification of misbehaving servers. In [2], Abu-Libdeh et al. introduces RACS, a proxy that transparently spreads the storage load over many providers using RAID. However, the proxy may become the bottleneck of the system and the clients may not have the full trust over the proxy. The work in [6] presents a secure LT codes-based cloud storage service scheme. The scheme has comparable storage and communication cost, but much less computational cost during data retrieval than erasure codes-based storage solutions. It also introduces less storage cost, much faster data retrieval, and comparable communication cost comparing to network coding-based distributed storage systems. However, the paper does not provide a coding scheme with general parameters, and the validity of the generated code is verified in a brute-force way, which is not scalable when the parameters are large.

VII. Conclusion

In this paper we present a novel framework of secure cloud data storage from the customer perspective. In our framework, the customer encode the data with an efficient coding method and send one share of the encoded data to each of the n service providers. The original data can be retrieved from at lease k ($k < n$) shares. Less than k shares of data can retrieve nothing of the original data. The customer only needs the encoding and decoding program. The correctness and efficiency of the scheme are verified by comprehensive evaluation. We also show that the problem to find the satisfying file distribution under certain cost and security constraints are NP-hard, and present the (SMT) formalization to find the satisfying data share distribution with cost and security constraints.

References

[1] Yices: An SMT solver. http://yices.csl.sri.com. Online, Accessed November-11-2011.

[2] Hussam Abu-Libdeh, Lonnie Princehouse, and Hakim Weatherspoon. Racs: a case for cloud storage diversity. In *SoCC*, pages 229–240. ACM, July 2010.

[3] B. Shwetha Bindu and B. Yadaiah. Secure data storage in cloud computing. *International Journal of Research in Computer Science*, 1:63–72, 2011.

[4] N. Bjørner and L. de Moura. $z3^{10}$: Applications, enablers, challenges and directions. In *CFV '09 Sixth International Workshop on Constraints in Formal Verification*, 2009.

[5] M. Blaum, J. Brady, J. Bruck, and J. Menon. EVENODD: An efficient scheme for tolerating double disk failures in raid architectures. *IEEE Trans. Computers*, 44(2):192–202, 1995.

[6] Ning Cao, Shucheng Yu, Zhenyu Yang, Wenjing Lou, and Y. Thomas Hou. Lt codes-based secure and reliable cloud storage service. In *INFOCOM'12*, pages 693–701, 2012.

[7] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7:201–215, July 1960.

[8] A. Dimakis, P. Godfrey, Y. Wu, M.. Wainwright, and K. Ramchandran. Network coding for distributed storage systems. *IEEE Trans. Inf. Theor.*, 56(9):4539–4551, 2010.

[9] A. Dimakis, K. Ramchandran, Y. Wu, and C. Suh. A survey on network codes for distributed storage. *CoRR*, abs/1004.4438, 2010.

[10] C. Huang and L. Xu. STAR: an efficient coding scheme for correcting triple storage node failures. In *FAST*, pages 197–210, 2005.

[11] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000),*, pages 190–201, 2000.

[12] M. Luby. Lt codes. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 271–280. IEEE, 2002.

[13] Alberto Medina, Ibrahim Matta, and John Byers. Brite: A flexible generator of internet topologies. Technical report, Boston, MA, USA, 2000.

[14] David A. Patterson, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (raid). In *Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, SIGMOD '88, pages 109–116, New York, NY, USA, 1988. ACM.

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

[16] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8:300–304, June 1960.

[17] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[18] Vijay V. Vazirani. *approximation algorithms*. Springer, 2004.

[19] Yongge Wang. Efficient LDPC code based secret sharing schemes and private data storage in cloud without encryption. Technical report, UNC Charlotte, 2012.

[20] Yongge Wang. LT codes for efficient and reliable distributed storage systems revisited. Technical report, UNC Charlotte, *submitted for publication*, 2012.

[21] Yongge Wang and Yvo Desmedt. Edge-colored graphs with applications to homogeneous faults. *Inf. Process. Lett.*, 111(13):634–641, July 2011.