A Secure Erasure Code-Based Cloud Storage System with Secure Data Forwarding

Nagarjuna Y[1] , Shiva Kumar Reddy K [2], Siddharth Katariya[3] ,A Adesh T G [4] , Prof. Suresh T [5]

[1,2,3,4,5] Department of CSE,BITM, Bellary Karnataka
[1]nagyalamanchili9@gmail.com , [2] shivakumarreddy707@gmail.com ,
[3]siddharthkatariya1@gmail.com , [4] adesh2424@gmail.com.

Abstract : A cloud storage system, consisting of a collection of storage servers, provides long-term storage services over the Internet. Storing data in a third party's cloud system causes serious concern over data confidentiality. General encryption schemes protect data confidentiality, but also limit the functionality of the storage system because a few operations are supported over encrypted data.
Constructing a secure storage system that supports multiple functions is challenging when the storage system is distributed and has no central authority. We propose a threshold proxy re-encryption scheme and integrate it with a decentralized erasure code such that a secure distributed storage system is formulated. The distributed storage system not only supports secure and robust data storage and retrieval, but also lets a user forward his data in the storage servers to another user without retrieving the data back. The main technical contribution is that the proxy re-encryption scheme supports encoding operations over encrypted messages as well as forwarding operations over encoded and encrypted messages. Our method fully integrates encrypting, encoding, and forwarding. We analyze and suggest suitable parameters for the number of copies of a message dispatched to storage servers and the number of storage servers queried by a key server. These parameters allow more flexible adjustment between the number of storage servers and robustness.

Keywords: Decentralized erasure code, proxy re-encryption, threshold cryptography, secure storage system.

1. INTRODUCTION

As high-speed networks and ubiquitous Internet access become available in recent years, many services are provided on the Internet such that users can use them from anywhere at any time. For example, the email service is probably the most popular one. Cloud computing is a concept that Treats the resources on the Internet as a unified entity, a cloud. Users just use services without being concerned about how computation is done and storage is managed. In this paper, we focus on designing a cloud storage system for robustness, confidentiality, and functionality. A cloud storage system is considered as a large-scale distributed storage system that consists of many independent storage servers.

Data robustness is a major requirement for storage systems. There have been many proposals of storing data over storage servers. One way to provide data robustness is to replicate a message such that each storage server stores a copy of the message. It is very robust because the message can be retrieved as long as one storage server survives. Another way is to encode a message of k symbols into a code word of n symbols by erasure coding. To store a message, each of its code word symbols is stored in a different storage

server. A storage server failure corresponds
to an erasure error of the code word symbol. As long as the number of failure servers is under the tolerance threshold of the erasure code, the message can be recovered from the code word symbols stored in the available storage servers by the decoding process. This provides a tradeoff between the storage size and the tolerance threshold of failure servers. A decentralized erasure code is an erasure code that independently computes each code word symbol for a message. Thus, the encoding process for a message can be split into n parallel tasks of generating code word symbols. A decentralized erasure code is suitable for use in a distributed storage system. After the message symbols are sent to storage servers, each storage server independently computes a code-word symbol for the received message symbols and stores it. This finishes the encoding and storing process. The recovery process is the same.

Storing data in a third party's cloud system causes serious concern on data confidentiality. In order to provide strong confidentiality for messages in storage servers, a user can encrypt messages by a cryptographic method before applying an erasure code method to encode and store messages. When he wants to use a message, he needs to retrieve the code word symbols from storage servers, decode them, and then decrypt them by using cryptographic keys. There are three problems in the above straightforward integration of encryption and encoding. First, the user has to do most computation and the communication traffic between the user and storage servers is high. Second, the user has to manage his cryptographic keys. If the user's device of storing the keys is lost or compromised, the security is broken. Finally, besides data storing and retrieving, it is hard for storage servers to directly support other functions. For example, storage servers cannot directly forward a user's messages to another one. The owner of messages has to retrieve, decode, decrypt and then forward them to another user.

2 RELATED WORKS

We briefly review distributed storage systems, proxy re-encryption schemes, and integrity checking mechanisms.

2.1 Distributed Storage Systems

At the early years, the Network-Attached Storage (NAS) and the Network File System (NFS) provide extra Storage devices over the network such that a user can access the storage devices via network connection. Afterward, many improvements on scalability, robustness, efficiency, and security were proposed.

A decentralized architecture for storage systems offers good scalability, because a storage server can join or leave without control of a central authority. To provide robust-ness against server failures, a simple method is to make replicas of each message and store them in different servers. However, this method is expensive as z replicas result in z times of expansion.

One way to reduce the expansion rate is to use erasure codes to encode messages. A message is encoded as a code word, which is a vector of symbols, and each storage server stores a code word symbol. A storage server failure is modeled as an erasure error of the stored code word symbol. Random linear codes support distributed encoding, that is, each code word symbol is independently computed. To store a message of k blocks, each storage server linearly combines the blocks with randomly chosen coefficients and stores the code word symbol and coefficients. To retrieve the message, a user queries k storage servers for the stored code word symbols and coefficients and solves the linear system. Dimakis et al. considered the case that n ¼ ak for a fixed constant a. They showed that distributing each block of a message to v randomly chosen storage servers is enough to have a probability 1 _ k=p _ oð1Þ of a successful data retrieval, where v ¼ b ln k, b > 5a, and p is the order of the used group. The sparsity parameter v ¼ b ln k is the number of storage servers which a block is sent to. The larger v is, the communication cost is higher and the successful retrieval probability is higher. The system has a light data confidentiality because an attacker can compromise k storage servers to get the message.

Lin and Tzeng addressed robustness and confidenti-ality issues by presenting a secure decentralized erasure code for the networked storage system. In addition to storage servers, their system consists of key servers, which hold cryptographic key shares and work in a distributed way. In their system, stored messages are encrypted and then encoded. To retrieve a message, key servers query storage servers for the user. As long as the number of available key servers is over a threshold t, the message can be successfully retrieved with an overwhelming probability.

One of their results shows that when there are n storage
p p
servers with nk, the parameterk ln k
¼ ak v is b with
b > servers
5a , and eachffiffiffiffiserver for
key queries 2 storage ffiffiffiffi
each retrieval request, the probability of a successful retrieval is at least 1 _ k=p _ oð1Þ.

2.2 Proxy Re-Encryption Schemes

Proxy re-encryption schemes are proposed by Mambo and Okamoto and Blaze et al. In a proxy re-encryption scheme, a proxy server can transfer a ciphertext under a public key PK_A to a new one under another public key PK_B by using the re-encryption key $RK_{A!B}$. The server does not know the plaintext during transformation. Ateniese et al. proposed some proxy re-encryption schemes and applied them to the sharing function of secure storage systems. In their work, messages are first encrypted by the owner and then stored in a storage server.

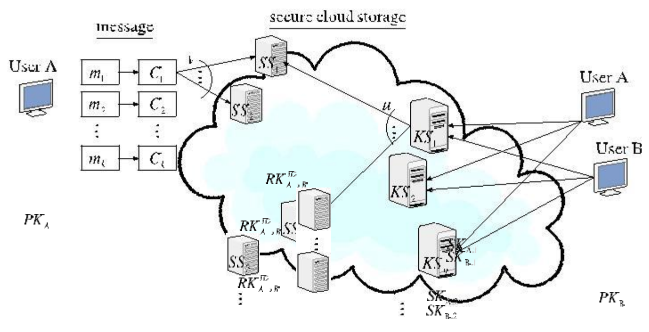


Fig. 1. A general system model of our work.

When a user wants to share his messages, he sends a re-encryption key to the storage server. The storage server re-encrypts the encrypted messages for the authorized user. Thus, their system has data confidentiality and supports the data forwarding function. Our work further integrates encryption, re-encryption, and encoding such that storage robust-ness is strengthened.

Type-based proxy re-encryption schemes proposed by Tang provide a better granularity on the granted right of a re-encryption key. A user can decide which type of messages and with whom he wants to share in this kind of proxy re-encryption schemes. Key-private proxy re-encryption schemes are proposed by Ateniese et al. In a key-private proxy re-encryption scheme, given a re-encryption key, a proxy server cannot determine the identity of the recipient. This kind of proxy re-encryption schemes provides higher privacy guarantee against proxy servers. Although most proxy re-encryption schemes use pairing operations, there exist proxy re-encryption schemes without pairing [19].

2.3 Integrity Checking Functionality

Another important functionality about cloud storage is the function of integrity checking. After a user stores data into the storage system, he no longer possesses the data at hand. The user may want to check whether the data are properly stored in storage servers. The concept of provable data possession and the notion of proof of storage are proposed. Later, public auditability of stored data is addressed in. Nevertheless all of them consider the messages in the clear text form.

3 SCENARIO

We present the scenario of the storage system, the threat model that we consider for the confidentiality issue, and a discussion for a straightforward solution.

3.1 System Model

As shown in Fig. 1, our system model consists of users, n storage servers SS_1; SS_2; . . . ; SS_n, and m key servers KS_1; KS_2; . . . ; KS_m. Storage servers provide storage services and key servers provide key management services. They work independently. Our distributed storage system consists of four phases: system setup, data storage, data forwarding, and data retrieval. These four phases are described as follows.

In the system setup phase, the system manager chooses system parameters and publishes them. Each user A is assigned a public-secret key pair ðPK_A; SK_AÞ. User A distributes his secret key SK_A to key servers such that each
key server KS_i holds a key share $SK_{A;i}$, 1 _ i _ m. The key is shared with a threshold t.

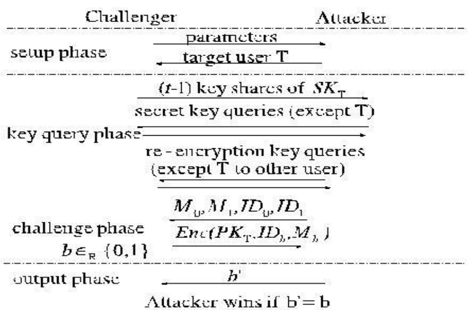
In the data storage phase, user A encrypts his message M and dispatches it to storage servers. A message M is decomposed into k blocks m_1; m_2; . . . ; m_k and has an identifier ID. User A encrypts each block m_i into a ciphertext C_i and sends it to v randomly chosen storage servers. Upon receiving cipher texts from a user, each storage server linearly combines them with randomly chosen coefficients into a code word symbol and stores it. Note that a storage server may receive less than k message blocks and we assume that all storage servers know the value k in advance.

In the data forwarding phase, user A forwards his encrypted message with an identifier ID stored in storage servers to user B such that B can decrypt the forwarded message by his secret key. To do so, A uses his secret key SK_A and B's public key PK_B to compute a re-encryption key $RK^{ID}_{A!B}$ and then sends $RK^{ID}_{A!B}$ to all storage servers. Each storage server uses the re-encryption key to re-encrypt its code word symbol for later retrieval requests by B. The re-encrypted code word symbol is the combination of cipher texts under B's public key. In order to distinguish re-encrypted code word symbols from intact ones, we call them original code word symbols and re-encrypted code word symbols, respectively.

In the data retrieval phase, user A requests to retrieve a message from storage servers. The message is either stored by him or forwarded to him. User A sends a retrieval request to key servers. Upon receiving the retrieval request and executing a proper authentication process with user A, each key server KS_i requests u randomly chosen storage servers to get codeword symbols and does partial decryption on the received codeword symbols by using the key share $SK_{A;i}$. Finally, user A combines the partially decrypted codeword symbols to obtain the original message M.

System recovering. When a storage server fails, a new one is added. The new storage server queries k available storage servers, linearly combines the received code word symbols as a new one and stores it. The system is then recovered.

3.2 Threat Model



We consider data confidentiality for both data storage and data forwarding. In this threat model, an attacker wants to break data confidentiality of a target user. To do so, the attacker colludes with all storage servers, non target users, and up to ðt _ 1Þkey servers. The attacker analyzes stored messages in storage servers, the secret keys of non target users, and the shared keys stored in key servers. Note that the storage servers store all re-encryption keys provided by users. The attacker may try to generate a new re-encryption key from stored re-encryption keys. We formally model this attack by the standard chosen plaintext attack[1] of the proxy

1. Systems against chosen ciphertext attacks are more secure than systems against the chosen plaintext attack. Here, we only consider the chosen plaintext attack because a homomorphic encryption scheme is not secure against chosen ciphertext attacks. Consider a multiplicative homo-morphic encryption scheme, where DðSK; EðPK; m_1Þ _ EðPK; m_2 ÞÞ ¼ m_1_ m_2 for the encryption function E, the decryption function D, a pair of public key PK and secret key SK, an operation _, and two messages m_1 and m_2. Given a challenge ciphertext C, where C ¼ EðPK; m_1Þ, the attacker chooses m_2, computes EðPK; m_2Þ, and computes C^0 ¼ C _ EðPK; m_2Þ. The attacker queries C^0 to the decryption oracle. The response m ¼ m_1_ m_2 from the decryption oracle reveals the plaintext m_1 to the attacker since m_1 ¼ m=m_2.**Fig. 2.** The security game for the chosen plaintext attack.

re-encryption scheme in a threshold version, as shown in Fig. 2.

The challenger C provides the system parameters. After the attacker A chooses a target user T , the challenger gives him ðt _ 1Þ key shares of the secret key SK_T of the target user T to model ðt _ 1Þ compromised key servers. Then, the attacker can query secret keys of other users and all re-encryption keys except those from T to other users. This models compromised non target users and storage servers. In the challenge phase, the attacker chooses two messages M_0 and M_1 with the identifiers ID_0 and ID_1, respectively. The challenger throws a random coin b and encrypts the message M_b with T 's public key PK_T . After getting the ciphertext from the challenger, the attacker outputs a bit b^0 for guessing b. In this game, the attacker wins if and only if b^0 ¼ b. The advantage of the attacker is defined as j1=2 _ Pr½b^0 ¼ b&j.

A cloud storage system modeled in the above is secure if no probabilistic polynomial time attacker wins the game with a non negligible advantage. A secure cloud storage system implies that an unauthorized user or server cannot get the content of stored messages, and a storage server cannot generate re-encryption keys by himself. If a storage server can generate a re-encryption key from the target user to another user B, the attacker can win the security game by re-encrypting the cipher text to B and decrypting the re-encrypted cipher text using the secret key SK_B. Therefore, this model addresses the security of data storage and data forwarding.

3.3 A Straightforward Solution

A straightforward solution to supporting the data forward-ing function in a distributed storage system is as follows: when the owner A wants to forward a message to user B, he downloads the encrypted message and decrypts it by using his secret key. He then encrypts the message by using B’s public key and uploads the new ciphertext. When B wants to retrieve the forwarded message from A, he downloads the ciphertext and decrypts it by his secret key. The whole data forwarding process needs three communication rounds for A’s downloading and uploading and B’s downloading. The communication cost is linear in the length of the forwarded message. The computation cost is the decryption and encryption for the owner A, and the decryption for user B.

Proxy re-encryption schemes can significantly decrease communication and computation cost of the owner. In a proxy re-encryption scheme, the owner sends a re-encryptionkey to storage servers such that storage servers perform the re-encryption operation for him. Thus, the communication cost of the owner is independent of the length of forwarded message and the computation cost of re-encryption is taken care of by storage servers. Proxy re-encryption schemes significantly reduce the overhead of the data forwarding function in a secure storage system.

4 CONSTRUCTION OF SECURE CLOUD STORAGE SYSTEMS

Before presenting our storage system, we briefly introduce the algebraic setting, the hardness assumption, an erasure code over exponents, and our approach.

Bilinear map. Let G_1 and G_2 be cyclic multiplicative groups[2] with a prime order p and g 2 G_1 be a generator. A map e~ : G_1_ G_1 ! G_2 is a bilinear map if it is efficiently computable and has the properties of bilinearity and nondegeneracy: for any x; y 2 Z_p; e~ðg^x; g^{y}Þ ¼ e~ðg; gÞ^{xy} and e~ðg; gÞ is not the identity element in G_2. Let Genð1⁻Þ be an algorithm generating ðg; e;~ G_1; G_2; pÞ, where _ is the length of p. Let x 2_R X denote that x is randomly chosen from the set X.

Decisional bilinear Diffie-Hellman assumption. Thisassumption is that it is computationally infeasible to distinguish the distributions (g, g^x, g^y, g^z, e~ðg; gÞ^{xyz}) and (g,g^x, g^y, g^z, e~ðg; gÞ^r), where x; y; z; r 2_R Z_p. Formally, for any probabilistic polynomial time algorithm A, the following is negligible (in _):j Pr½Aðg; g^x; g^y; g^z; Q_bÞ ¼ b : x; y; z; r 2_R Z_p;Q_0 ¼ e~ðg; gÞ^{xyz}; Q_1 ¼ e~ðg; gÞ^r; b 2_R f0; 1g&_ 1=2j:

Erasure coding over exponents. We consider that the message domain is the cyclic multiplicative group G_2 described above. An encoder generates a generator matrix G ¼ ½$g_{i;j}$& for 1 _ i _ k; 1 _ j _ n as follows: for each row, the encoder randomly selects an entry and randomly sets a value from Z_p to the entry. The encoder repeats this step v times with replacement for each row. An entry of a row can be selected multiple times but only set to one value. The values of the rest entries are set to 0. Let the message be

; ; . . . process is to
m m ; m G^k generate
ð kÞ . The^{g}1;^{g}2
1 2 ^{2}n2 encodingj ;j ^{g}k;j
_ _
ðw$_1$; w$_2$; . . . where w$_j$ ¼m _
; w$_n$Þ 2 G_2 ,m$_1$ $_2$ m$_k$ for

1 _ j _ n.step of theis
The first decoding process to
Compute the inverse of a k _ k submatrix K of G. Let K be
for i; k. Let . The final
g 1 j K^{-1} d step of
$^{i;j\&}1$ ^{d}
i; i ¼ _i;j_ 1;^{d}2
^{1}j$_{i}$ ξ _ _ ½ k i ;i ^{d}k;i
_ _f
the decoding process is to^{w}j _ o
compute m_i ¼ w_{j1} $_2$ w_{jk} r
1 _ i _ k. An example is shown in Fig. 3. User A stores two messages m_1 and m_2 into four storage servers. When the storage servers SS_1 and SS_3 are available and the k _ k submatrix K is invertible, user A can decode m_1 and m_2 from the codeword symbols w_1; w_3 and the coefficients ð$g_{1;1}$; 0Þ; ð0; $g_{2;3}$Þ, which are stored in the storage servers SS_1 and SS_3.

Our approach. We use a threshold proxy re-encryption scheme with multiplicative homomorphic property. An encryption scheme is multiplicative homomorphic if it

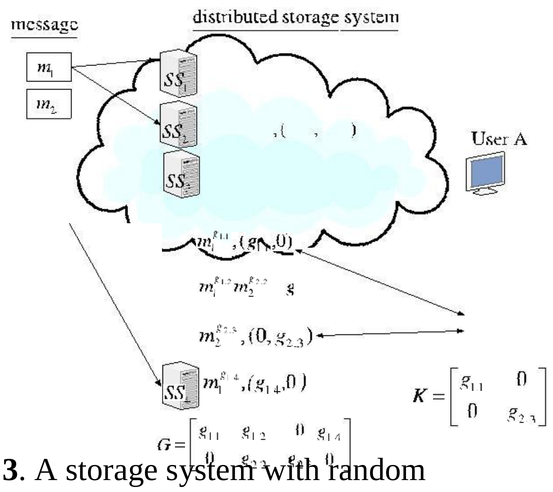

message
distributed storage system
m_1
m_2
SS_1
SS_2
SS_3
User A
$m_1^{g_{1,1}}, (g_{1,1}, 0)$
$m_1^{g_{1,2}} m_2^{g_{2,2}}$
$m_2^{g_{2,3}}, (0, g_{2,3})$
SS_4
$m_1^{g_{1,4}}, (g_{1,4}, 0)$
$K = \begin{bmatrix} g_{1,1} & 0 \\ 0 & g_{2,3} \end{bmatrix}$
$G = \begin{bmatrix} g_{1,1} & g_{1,2} & 0 & g_{1,4} \\ 0 & g_{2,2} & g_{2,3} & 0 \end{bmatrix}$

Fig. 3. A storage system with random linear coding over exponents.

4.1 A Secure Cloud Storage System with Secure Forwarding

As described in Section 3.1, there are four phases of our storage system.

System setup. The algorithm SetUpð1⁻ Þ generates the system parameters _. A user uses KeyGenð_Þ to generate his public and secret key pair and ShareKeyGenð_Þ to share his secret key to a set of m key servers with a threshold t, where k _ t _ m. The user locally stores the third component of his secret key.

. SetUp(1⁻). Run Genð1⁻Þ to obtain ðg; h; e;~ G_1; G_2; pÞ, where e~ : G_1_ G_1 ! G_2 is a bilinear map, g and h are generators of G_1, and both G_1 and G_2 have the prime order p. Set _ ¼ ðg; h; e;~ G_1; G_2; p; fÞ, where f : Z_p_ f0; 1g ! Z_p is a one-way hash function.

. KeyGen(_). For a user A, the algorithm selects a_1; a_2; a_3 2_R Z_p and sets

PK_A ¼ ðg^{a1} ; h^{a2} Þ; SK_A ¼ ða_1; a_2; a_3Þ:

. ShareKeyGen(SK_A, t, m). This algorithm shares the secret key SK_A of a user A to a set of m key servers by using two polynomials $f_{A;1}$ðzÞ and $f_{A;2}$ðzÞ of degree ðt _ 1Þ over the finite field GF(p)
$f_{A;1}$ðzÞ ¼ a_1 þ v_1z þ v_2z^2 þ _ _ _ þ $v_{t_1}z^{t_1}$ðmod pÞ; $f_{A;2}$ðzÞ ¼ a^{-1}_2 þ v_1z þ v_2z^2 þ _ _ _ þ $v_{t_1}z^{t_1}$ðmod pÞ;

where v_1; v_2; . . . ; v_{t_1} 2_R Z_p. The key share of the secret key SK_A to the key server KS_i is $SK_{A;i}$ ¼
ð$f_{A;1}$ðiÞ; $f_{A;2}$ðiÞÞ, where 1 _ i _ m.

Data storage. When user A wants to store a message of k blocks m_1; m_2; . . . ; m_k with the identifier ID, he computes the identity token _ ¼ $h^{fða3;IDÞ}$ and performs the encryption algorithm Encð_Þ on _ and k blocks to get k original ciphertexts C_1; C_2; . . . ; C_k. An original ciphertext is indicated by a leading bit b ¼ 0. User A sends each ciphertext C_i to v randomly chosen storage servers. A storage server receives a set of original ciphertexts with the same identity token _ from A. When a ciphertext C_i is not received, the storage server inserts C_i ¼ ð0; 1; _; 1 Þ to the set. The special format of ð0; 1; _; 1Þ is a mark for the absence of C_i. The storage server performs Encodeð_Þ on the set of k ciphertexts and stores the encoded result (codeword symbol).

. Enc(PK_A; _; m_1; m_2; . . . ; m_k). For 1 _ i _ k, this algo-rithm computes

C_i ¼ ð0; _i; _; _iÞ ¼ ð0; g^{ri} ;_; m_ie~ðg^{a1} ; _^{ri} ÞÞ;

where r_i 2$_R$ Z_p; 1 _ i _ k and 0 is the leading bit indicating an original ciphertext.

. Encode(C_1; C_2; . . . ; C_k). For each ciphertext C_i, the algorithm randomly selects a coefficient g_i. If some ciphertext C_i is ð0; 1; _; 1Þ, the coefficient g_i is set to 0.

Let C_i ¼ ð0; _i; _; _iÞ.encoding
The process is to
compute an original codeword symbol
C^0
k k _i
C^0 ¼ 0; i _i^{gi} ; _; i ^{gi} !
¼1 ¼1
Y_ _ Y_ _
k k k
¼ 0; g^{P}i¼1 ^{giri} ; _; i¼1 m_i^{gi} e~ðg^{a1} ;
_Þ^{P}i¼1 ^{giri} !

Y

¼ ð0; g^{r0} ; _; We~ðg; _Þ^{a1r0} Þ;
where W ¼ i k¼1
m_i^{gi} and r^0 ¼ i k¼1 g_ir_i. The
; g ; g ; . . . ;
is C g .
coded result
^{Q}ð 0 1 2 kÞ P

en
-

Data forwarding. User A wants to forward a message to another user B. He needs the first component a_1 of his secret key. If A does not possess a_1, he queries key servers for key shares. When at least t key servers respond, A recovers the first component a_1 of the secret key SK_A via the KeyRecoverð_Þ algorithm. Let the identifier of the message

be ID. User A computes the re-encryption key RK via

the ReKeyGenð_Þ algorithm and securely sends the re-

encryption key to each storage server. By using RK , a

storage server re-encrypts the original codeword symbol C^0 with the identifier ID into a re-encrypted codeword symbol C^{00} via the ReEncð_Þ algorithm such that C^{00} is decryptable by using B's secret key. A re-encrypted codeword symbol is indicated by the leading bit b ¼ 1. Let the public key PK_B of user B be ðg^{b1} ; h^{b2} Þ.

. **Key Recover**($SK_{A;i1}$; $SK_{A;i2}$; . . . ; $SK_{A;it}$). Let T ¼ fi$_1$; i$_2$; . . . ; i$_t$g.

5 CONCLUSION

In this paper, we consider a cloud storage system consists of storage servers and key servers. We integrate a newly proposed threshold proxy re-encryption scheme and erasure codes over exponents. The threshold proxy re-encryption scheme supports encoding, forwarding, and partial decryption operations in a distributed way. To decrypt a message of k blocks that are encrypted and encoded to n code word symbols, each key server only has to partially decrypt two code word symbols in our system. By using the threshold proxy re-encryption scheme, we present a secure cloud storage system that provides secure data storage and secure data forwarding functionality in a decentralized structure. Moreover, each storage server independently performs encoding and re-encryption and each key server independently performs partial decryption.

Our storage system and some newly proposed content addressable file systems and storage system are highly compatible. Our storage servers act as storage nodes in a content addressable storage system for storing content addressable blocks. Our key servers act as access nodes for providing a front-end layer such as a traditional file system interface. Further study on detailed cooperation is required.

REFERENCES

[1] C. Dubnicki, L. Gryz, L. Heldt, M. Kaczmarczyk, W. Kilian, P. Strzelczak, J. Szczepkowski, C. Ungureanu, and M. Welnicki, “Hydrastor: A Scalable Secondary Storage,” Proc. Seventh Conf. File and Storage Technologies (FAST), pp. 197-210, 2009.

[2] C. Ungureanu, B. Atkin, A. Aranya, S. Gokhale, S. Rago, G. Calkowski, C. Dubnicki, and A. Bohra, “Hydrafs: A High-Throughput File System for the Hydrastor Content-Addressable Storage System,” Proc. Eighth USENIX Conf. File and Storage Technologies (FAST), p. 17, 2010.

[3] W. Dong, F. Douglis, K. Li, H. Patterson, S. Reddy, and P. Shilane, “Tradeoffs in Scalable Data Routing for Deduplication Clusters,” Proc. Ninth USENIX Conf. File and Storage Technologies (FAST), p. 2, 2011.

[4] C. Wang, Q. Wang, K. Ren, and W. Lou, “Privacy-Preserving Public Auditing for Data Storage Security in Cloud Computing,”Proc. IEEE 29th Int’l Conf. Computer Comm. (INFOCOM), pp. 525-533, 2010.