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

Hsiao-Ying Lin, *Member*, *IEEE*, and Wen-Guey Tzeng, *Member*, *IEEE*

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.

Index Terms—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 [1], [2], [3], [4], [5]. 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 codeword of n symbols by erasure coding. To store a message, each of its codeword symbols is stored in a different storage server. A storage server failure corresponds to an erasure error of the codeword 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 codeword 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 codeword symbol for a message. Thus, the encoding process for a message can be split into n parallel tasks of generating codeword 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 codeword 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 codeword 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.

In this paper, we address the problem of forwarding data to another user by storage servers directly under the

- *H.-Y. Lin is with the Intelligent Information and Communications Research Center, Department of Computer Science, National Chiao Tung University, No. 1001, University Road, Hsinchu City 30010, Taiwan. E-mail: hsiaoying.lin@gmail.com.*
- *W.-G. Tzeng is with the Department of Computer Science, National Chiao Tung University, No. 1001, University Road, Hsinchu City 30010, Taiwan. E-mail: wgtzeng@cs.nctu.edu.tw.*

Manuscript received 21 Mar. 2011; revised 12 Sept. 2011; accepted 18 Sept. 2011; published online 30 Sept. 2011.
Recommended for acceptance by J. Weissman.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number tpds-2011-03-0162.
Digital Object Identifier no. 10.1109/TPDS.2011.252.

command of the data owner. We consider the system model that consists of distributed storage servers and key servers. Since storing cryptographic keys in a single device is risky, a user distributes his cryptographic key to key servers that shall perform cryptographic functions on behalf of the user. These key servers are highly protected by security mechanisms. To well fit the distributed structure of systems, we require that servers independently perform all operations. With this consideration, we propose a new threshold proxy re-encryption scheme and integrate it with a secure decentralized code to form a secure distributed storage system. The encryption scheme supports encoding operations over encrypted messages and forwarding operations over encrypted and encoded messages. The tight integration of encoding, encryption, and forwarding makes the storage system efficiently meet the requirements of data robustness, data confidentiality, and data forwarding. Accomplishing the integration with consideration of a distributed structure is challenging. Our system meets the requirements that storage servers independently perform encoding and re-encryption and key servers independently perform partial decryption. Moreover, we consider the system in a more general setting than previous works. This setting allows more flexible adjustment between the number of storage servers and robustness.

Our contributions. Assume that there are n distributed storage servers and m key servers in the cloud storage system. A message is divided into k blocks and represented as a vector of k symbols. Our contributions are as follows:

1. We construct a secure cloud storage system that supports the function of secure data forwarding by using a threshold proxy re-encryption scheme. The encryption scheme supports decentralized erasure codes over encrypted messages and forwarding operations over encrypted and encoded messages. Our system is highly distributed where storage servers independently encode and forward messages and key servers independently perform partial decryption.
2. We present a general setting for the parameters of our secure cloud storage system. Our parameter setting of $n = ak^c$ supersedes the previous one of $n = ak\sqrt{k}$, where $c \geq 1.5$ and $a > \sqrt{2}$ [6]. Our result $n = ak^c$ allows the number of storage servers be much greater than the number of blocks of a message. In practical systems, the number of storage servers is much more than k. The sacrifice is to slightly increase the total copies of an encrypted message symbol sent to storage servers. Nevertheless, the storage size in each storage server does not increase because each storage server stores an encoded result (a codeword symbol), which is a combination of encrypted message symbols.

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) [7] and the Network File System (NFS) [8] 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 [1], [2], [9].

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 robustness 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 [10], [11], [12], [13], [5]. A message is encoded as a codeword, which is a vector of symbols, and each storage server stores a codeword symbol. A storage server failure is modeled as an erasure error of the stored codeword symbol. Random linear codes support distributed encoding, that is, each codeword 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 codeword symbol and coefficients. To retrieve the message, a user queries k storage servers for the stored codeword symbols and coefficients and solves the linear system. Dimakis et al. [13] considered the case that $n = ak$ 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 $1 - k/p - o(1)$ of a successful data retrieval, where $v = b \ln k$, $b > 5a$, and p is the order of the used group. The sparsity parameter $v = b \ln k$ 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 [6] addressed robustness and confidentiality 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 servers with $n = ak\sqrt{k}$, the parameter v is $b\sqrt{k}\ln k$ with $b > 5a$, and each key server queries 2 storage servers for each retrieval request, the probability of a successful retrieval is at least $1 - k/p - o(1)$.

2.2 Proxy Re-Encryption Schemes

Proxy re-encryption schemes are proposed by Mambo and Okamoto [14] and Blaze et al. [15]. In a proxy re-encryption scheme, a proxy server can transfer a ciphertext under a public key $\mathrm{PK_A}$ to a new one under another public key $\mathrm{PK_B}$ by using the re-encryption key $\mathrm{RK_{A \to B}}$. The server does not know the plaintext during transformation. Ateniese et al. [16] 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. When a user

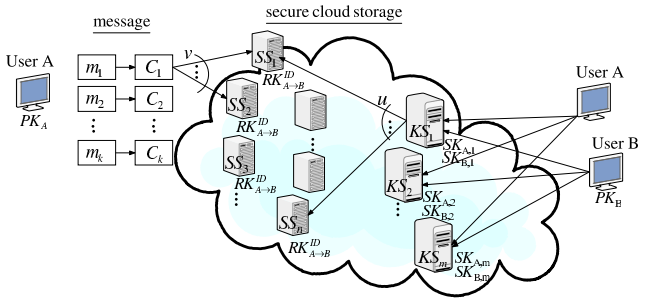

Fig. 1. A general system model of our work.

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 robustness is strengthened.

Type-based proxy re-encryption schemes proposed by Tang [17] 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. [18]. 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 [20], [21] and the notion of proof of storage [22], [23], [24] are proposed. Later, public auditability of stored data is addressed in [25]. Nevertheless all of them consider the messages in the cleartext 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 $\mathrm{SS}_1, \mathrm{SS}_2, \ldots, \mathrm{SS}_n$, and m key servers KS_1, $\mathrm{KS}_2, \ldots, \mathrm{KS}_m$. 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 $(\mathrm{PK_A}, \mathrm{SK_A})$. User A distributes his secret key $\mathrm{SK_A}$ to key servers such that each key server KS_i holds a key share $\mathrm{SK}_{\mathrm{A},i}$, $1 \le i \le m$. 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 $m_1, m_2, \ldots, m_k$ 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 ciphertexts from a user, each storage server linearly combines them with randomly chosen coefficients into a codeword 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 $\mathrm{SK_A}$ and B's public key $\mathrm{PK_B}$ to compute a re-encryption key $\mathrm{RK}^{\mathrm{ID}}_{\mathrm{A}\to\mathrm{B}}$ and then sends $\mathrm{RK}^{\mathrm{ID}}_{\mathrm{A}\to\mathrm{B}}$ to all storage servers. Each storage server uses the re-encryption key to re-encrypt its codeword symbol for later retrieval requests by B. The re-encrypted codeword symbol is the combination of ciphertexts under B's public key. In order to distinguish re-encrypted codeword symbols from intact ones, we call them *original* codeword symbols and *re-encrypted* codeword 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 $\mathrm{SK}_{\mathrm{A},i}$. 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 codeword 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, nontarget users, and up to $(t-1)$ key servers. The attacker analyzes stored messages in storage servers, the secret keys of nontarget 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 homomorphic encryption scheme, where $D(SK, E(PK, m_1) \odot E(PK, m_2)) = m_1 \cdot m_2$ for the encryption function E, the decryption function D, a pair of public key PK and secret key SK, an operation $\odot$, and two messages m_1 and m_2. Given a challenge ciphertext C, where $C = E(PK, m_1)$, the attacker chooses m_2, computes $E(PK, m_2)$, and computes $C' = C \odot E(PK, m_2)$. The attacker queries C' to the decryption oracle. The response $m = m_1 \cdot m_2$ from the decryption oracle reveals the plaintext m_1 to the attacker since $m_1 = m/m_2$.

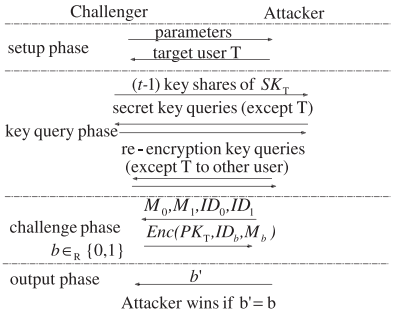

Fig. 2. The security game for the chosen plaintext attack.

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

The challenger $\mathcal{C}$ provides the system parameters. After the attacker $\mathcal{A}$ chooses a target user $\mathcal{T}$, the challenger gives him $(t-1)$ key shares of the secret key $\mathrm{SK}_{\mathcal{T}}$ of the target user $\mathcal{T}$ to model $(t-1)$ compromised key servers. Then, the attacker can query secret keys of other users and all re-encryption keys except those from $\mathcal{T}$ to other users. This models compromised nontarget 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 $\mathcal{T}$'s public key $\mathrm{PK}_{\mathcal{T}}$. After getting the ciphertext from the challenger, the attacker outputs a bit b' for guessing b. In this game, the attacker wins if and only if $b' = b$. The advantage of the attacker is defined as $|1/2 - \Pr[b' = b]|$.

A cloud storage system modeled in the above is secure if no probabilistic polynomial time attacker wins the game with a nonnegligible 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 ciphertext to B and decrypting the re-encrypted ciphertext 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 forwarding 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-encryption key 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 $\mathbb{G}_1$ and $\mathbb{G}_2$ be cyclic multiplicative groups[2] with a prime order p and $g \in \mathbb{G}_1$ be a generator. A map $\tilde{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ is a bilinear map if it is efficiently computable and has the properties of bilinearity and nondegeneracy: for any $x, y \in \mathbb{Z}_p^*$, $\tilde{e}(g^x, g^y) = \tilde{e}(g, g)^{xy}$ and $\tilde{e}(g, g)$ is not the identity element in $\mathbb{G}_2$. Let $\mathrm{Gen}(1^\lambda)$ be an algorithm generating $(g, \tilde{e}, \mathbb{G}_1, \mathbb{G}_2, p)$, where λ is the length of p. Let $x \in_R X$ denote that x is randomly chosen from the set X.

Decisional bilinear Diffie-Hellman assumption. This assumption is that it is computationally infeasible to distinguish the distributions $(g, g^x, g^y, g^z, \tilde{e}(g, g)^{xyz})$ and $(g, g^x, g^y, g^z, \tilde{e}(g, g)^r)$, where $x, y, z, r \in_R \mathbb{Z}_p^*$. Formally, for any probabilistic polynomial time algorithm $\mathcal{A}$, the following is negligible (in λ):

$$|\Pr[\mathcal{A}(g, g^x, g^y, g^z, \mathbb{Q}_b) = b : x, y, z, r \in_R \mathbb{Z}_p^*, \mathbb{Q}_0 = \tilde{e}(g, g)^{xyz}; \mathbb{Q}_1 = \tilde{e}(g, g)^r; b \in_R \{0, 1\}] - 1/2|.$$

Erasure coding over exponents. We consider that the message domain is the cyclic multiplicative group $\mathbb{G}_2$ described above. An encoder generates a generator matrix $G = [g_{i,j}]$ for $1 \le i \le k, 1 \le j \le n$ as follows: for each row, the encoder randomly selects an entry and randomly sets a value from $\mathbb{Z}_p^*$ 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 $(m_1, m_2, \ldots, m_k) \in \mathbb{G}_2^k$. The encoding process is to generate $(w_1, w_2, \ldots, w_n) \in \mathbb{G}_2^n$, where $w_j = m_1^{g_{1,j}} m_2^{g_{2,j}} \cdots m_k^{g_{k,j}}$ for $1 \le j \le n$. The first step of the decoding process is to compute the inverse of a $k \times k$ submatrix K of G. Let K be $[g_{i,j_i}]$ for $1 \le i, j_i \le k$. Let $K^{-1} = [d_{i,j}]_{1 \le i,j \le k}$. The final step of the decoding process is to compute $m_i = w_{j_1}^{d_{1,i}} w_{j_2}^{d_{2,i}} \cdots w_{j_k}^{d_{k,i}}$ for $1 \le i \le k$. 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 $k \times k$ submatrix K is invertible, user A can decode m_1 and m_2 from the codeword symbols w_1, w_3 and the coefficients $(g_{1,1}, 0), (0, g_{2,3})$, 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

2. It can also be described as additive groups over points on an elliptic curve.

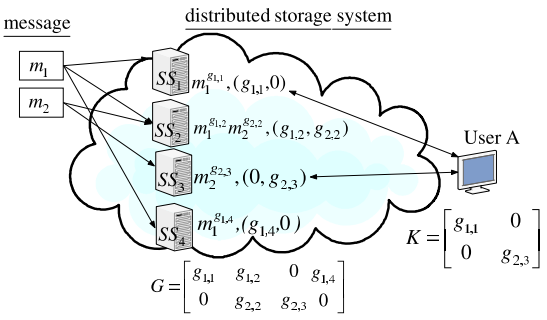

Fig. 3. A storage system with random linear coding over exponents.

supports a group operation $\odot$ on encrypted plaintexts without decryption

$$D(SK, E(PK, m_1) \odot E(PK, m_2)) = m_1 \cdot m_2,$$

where E is the encryption function, D is the decryption function, and (PK, SK) is a pair of public key and secret key. Given two coefficients g_1 and g_2, two message symbols m_1 and m_2 can be encoded to a codeword symbol $m_1^{g_1} m_2^{g_2}$ in the encrypted form

$$C = E(PK, m_1)^{g_1} \odot E(PK, m_2)^{g_2} = E(PK, m_1^{g_1} \cdot m_2^{g_2}).$$

Thus, a multiplicative homomorphic encryption scheme supports the encoding operation over encrypted messages. We then convert a proxy re-encryption scheme with multiplicative homomorphic property into a threshold version. A secret key is shared to key servers with a threshold value t via the Shamir secret sharing scheme [26], where $t \geq k$. In our system, to decrypt for a set of k message symbols, each key server independently queries 2 storage servers and partially decrypts two encrypted codeword symbols. As long as t key servers are available, k codeword symbols are obtained from the partially decrypted ciphertexts.

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 $\mathsf{SetUp}(1^\tau)$ generates the system parameters μ. A user uses $\mathsf{KeyGen}(\mu)$ to generate his public and secret key pair and $\mathsf{ShareKeyGen}(\cdot)$ to share his secret key to a set of m key servers with a threshold t, where $k \leq t \leq m$. The user locally stores the third component of his secret key.

- $\mathsf{SetUp}(1^\lambda)$. Run $\mathrm{Gen}(1^\lambda)$ to obtain $(g, h, \tilde{e}, \mathbb{G}_1, \mathbb{G}_2, p)$, where $\tilde{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ is a bilinear map, g and h are generators of $\mathbb{G}_1$, and both $\mathbb{G}_1$ and $\mathbb{G}_2$ have the prime order p. Set $\mu = (g, h, \tilde{e}, \mathbb{G}_1, \mathbb{G}_2, p, f)$, where $f : \mathbb{Z}_p^* \times \{0,1\}^* \to \mathbb{Z}_p^*$ is a one-way hash function.
- $\mathsf{KeyGen}(\mu)$. For a user A, the algorithm selects $a_1, a_2, a_3 \in_R \mathbb{Z}_p^*$ and sets

$$\mathrm{PK_A} = (g^{a_1}, h^{a_2}), \mathrm{SK_A} = (a_1, a_2, a_3).$$

- $\mathsf{ShareKeyGen}(\mathrm{SK_A}, t, m)$. This algorithm shares the secret key $\mathrm{SK_A}$ of a user A to a set of m key servers by using two polynomials $f_{\mathrm{A},1}(\mathrm{z})$ and $f_{\mathrm{A},2}(\mathrm{z})$ of degree $(t-1)$ over the finite field GF(p)

$$f_{\mathrm{A},1}(\mathrm{z}) = \mathbf{a_1} + \mathrm{v}_1\mathrm{z} + \mathrm{v}_2\mathrm{z}^2 + \cdots + \mathrm{v}_{t-1}\mathrm{z}^{t-1} (\mathrm{mod}\ p),$$
$$f_{\mathrm{A},2}(\mathrm{z}) = \mathrm{a}_2^{-1} + \mathrm{v}_1\mathrm{z} + \mathrm{v}_2\mathrm{z}^2 + \cdots + \mathrm{v}_{t-1}\mathrm{z}^{t-1} (\mathrm{mod}\ p),$$

where $\mathrm{v}_1, \mathrm{v}_2, \ldots, \mathrm{v}_{t-1} \in_R \mathbb{Z}_p^*$. The key share of the secret key $\mathrm{SK_A}$ to the key server KS_i is $\mathrm{SK}_{\mathrm{A},i} = (f_{\mathrm{A},1}(i), f_{\mathrm{A},2}(i))$, where $1 \leq i \leq m$.

Data storage. When user A wants to store a message of k blocks $m_1, m_2, \ldots, m_k$ with the identifier ID, he computes the identity token $\tau = h^{f(a_3, \mathrm{ID})}$ and performs the encryption algorithm $\mathrm{Enc}(\cdot)$ on τ and k blocks to get k original ciphertexts $C_1, C_2, \ldots, C_k$. An original ciphertext is indicated by a leading bit $\mathsf{b} = 0$. 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 τ from A. When a ciphertext C_i is not received, the storage server inserts $C_i = (0, 1, \tau, 1)$ to the set. The special format of $(0, 1, \tau, 1)$ is a mark for the absence of C_i. The storage server performs $\mathrm{Encode}(\cdot)$ on the set of k ciphertexts and stores the encoded result (codeword symbol).

- $\mathsf{Enc}(\mathrm{PK_A}, \tau, m_1, m_2, \ldots, m_k)$. For $1 \leq i \leq k$, this algorithm computes

$$C_i = (0, \alpha_i, \beta, \gamma_i) = (0, g^{r_i}, \tau, m_i \tilde{e}(g^{a_1}, \tau^{r_i})),$$

where $r_i \in_R \mathbb{Z}_p^*, 1 \leq i \leq k$ and 0 is the leading bit indicating an original ciphertext.
- $\mathsf{Encode}(C_1, C_2, \ldots, C_k)$. For each ciphertext C_i, the algorithm randomly selects a coefficient g_i. If some ciphertext C_i is $(0, 1, \tau, 1)$, the coefficient g_i is set to 0. Let $C_i = (0, \alpha_i, \beta, \gamma_i)$. The encoding process is to compute an original codeword symbol C'

$$\begin{aligned} C' &= \left(0, \prod_{i=1}^{k} (\alpha_i^{g_i}), \beta, \prod_{i=1}^{k} (\gamma_i^{g_i})\right) \\ &= \left(0, g^{\sum_{i=1}^{k} g_i r_i}, \tau, \prod_{i=1}^{k} m_i^{g_i} \tilde{e}(g^{a_1}, \tau)^{\sum_{i=1}^{k} g_i r_i}\right) \\ &= (0, g^{r'}, \tau, W\tilde{e}(g, \tau)^{a_1 r'}), \end{aligned}$$

where $W = \prod_{i=1}^{k} m_i^{g_i}$ and $r' = \sum_{i=1}^{k} g_i r_i$. The encoded result is $(C', g_1, g_2, \ldots, g_k)$.

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 $\mathrm{SK_A}$ via the $\mathsf{KeyRecover}(\cdot)$ algorithm. Let the identifier of the message be ID. User A computes the re-encryption key $\mathrm{RK}_{\mathrm{A}\to\mathrm{B}}^{\mathrm{ID}}$ via the $\mathsf{ReKeyGen}(\cdot)$ algorithm and securely sends the re-encryption key to each storage server. By using $\mathrm{RK}_{\mathrm{A}\to\mathrm{B}}^{\mathrm{ID}}$, a storage server re-encrypts the original codeword symbol C' with the identifier ID into a re-encrypted codeword symbol C'' via the $\mathsf{ReEnc}(\cdot)$ algorithm such that C'' is decryptable by using B's secret key. A re-encrypted codeword symbol is indicated by the leading bit $\mathsf{b} = 1$. Let the public key $\mathrm{PK_B}$ of user B be (g^{b_1}, h^{b_2}).

- $\mathsf{KeyRecover}(\mathrm{SK}_{\mathrm{A},i_1}, \mathrm{SK}_{\mathrm{A},i_2}, \ldots, \mathrm{SK}_{\mathrm{A},i_t})$. Let $\mathrm{T} = \{i_1, i_2, \ldots, i_t\}$. This algorithm recovers a_1 via Lagrange interpolation as follows:

$$a_1 = \sum_{s \in \mathrm{T}} \left(f_{\mathrm{A},1}(s) \prod_{s' \in \mathrm{T}/\{s\}} \frac{-s'}{s - s'} \right) \bmod p.$$

- ReKeyGen($\mathrm{PK_A}, \mathrm{SK_A}, \mathrm{ID}, \mathrm{PK_B}$). This algorithm selects $e \in_R \mathbb{Z}_p^*$ and computes

$$\mathrm{RK}_{\mathrm{A}\to\mathrm{B}}^{\mathrm{ID}} = ((h^{b_2})^{a_1(f(a_3,\mathrm{ID})+e)}, h^{a_1 e}).$$

- ReEnc($\mathrm{RK}_{\mathrm{A}\to\mathrm{B}}^{\mathrm{ID}}, C'$). Let $C' = (0, \alpha, \beta, \gamma) = (0, g^{r'}, \tau, W\tilde{e}(g^{a_1}, \tau^{r'}))$ for some r' and some W, and $\mathrm{RK}_{\mathrm{A}\to\mathrm{B}}^{\mathrm{ID}} = (h^{b_2 a_1(f(a_3,\mathrm{ID})+e)}, h^{a_1 e})$ for some e. The re-encrypted codeword symbol is computed as follows:

$$\begin{aligned} C'' &= (1, \alpha, h^{b_2 a_1(f(a_3,\mathrm{ID})+e)}, \gamma \cdot \tilde{e}(\alpha, h^{a_1 e})) \\ &= (1, g^{r'}, h^{b_2 a_1(f(a_3,\mathrm{ID})+e)}, W\tilde{e}(g,h)^{a_1 r'(f(a_3,\mathrm{ID})+e)}). \end{aligned}$$

Note that the leading bit 1 indicates C'' is a re-encrypted ciphertext.

Data retrieval. There are two cases for the data retrieval phase. The first case is that a user A retrieves his own message. When user A wants to retrieve the message with the identifier ID, he informs all key servers with the identity token τ. A key server first retrieves original codeword symbols from u randomly chosen storage servers and then performs partial decryption ShareDec($\cdot$) on every retrieved original codeword symbol C'. The result of partial decryption is called a partially decrypted codeword symbol. The key server sends the partially decrypted codeword symbols ζ and the coefficients to user A. After user A collects replies from at least t key servers and at least k of them are originally from distinct storage servers, he executes Combine($\cdot$) on the t partially decrypted codeword symbols to recover the blocks $m_1, m_2, \ldots, m_k$. The second case is that a user B retrieves a message forwarded to him. User B informs all key servers directly. The collection and combining parts are the same as the first case except that key servers retrieve re-encrypted codeword symbols and perform partial decryption ShareDec($\cdot$) on re-encrypted codeword symbols.

- ShareDec($\mathrm{SK_j}, X_i$). X_i is a codeword symbol, where $X_i = (\mathtt{b}, \alpha, \beta, \gamma)$ and $\mathtt{b}$ is the indicator for original and re-encrypted codeword symbols. $\mathrm{SK_j}$ is a key share, where $\mathrm{SK_j} = (\mathrm{sk}_0, \mathrm{sk}_1)$. By using the key share $\mathrm{SK_j}$, the partially decrypted codeword symbol $\zeta_{i,j}$ of X_i is generated as follows:

$$\zeta_{i,j} = (\mathtt{b}, \alpha, \beta, \beta^{sk_{\mathtt{b}}}, \gamma).$$

- Combine($\zeta_{i_1,j_1}, \zeta_{i_2,j_2}, \ldots, \zeta_{i_t,j_t}$). Let a partially decrypted codeword symbol $\zeta_{i,j}$ be $(\mathtt{b}, \alpha_{i,j}, \beta_{i,j}, \beta'_{i,j}, \gamma_{i,j})$. This algorithm combines t partially decrypted codeword symbols, where $\beta_{i_1,j_1} = \beta_{i_2,j_2} = \cdots = \beta_{i_t,j_t} = \tau$, $j_1 \neq j_2 \neq \ldots \neq j_t$ and there are at least k distinct values in $\{i_1, i_2, \ldots, i_t\}$. Let $\mathrm{S_J} = \{j_1, j_2, \ldots, j_t\}$ and $\mathrm{S} = \{(i_1, j_1), (i_2, j_2), \ldots, (i_t, j_t)\}$. Without loss of generality, let $\mathrm{S_I} = \{i_1, i_2, \ldots, i_k\}$ be k distinct values in $\{i_1, i_2, \ldots, i_t\}$.

  In the first case $\mathtt{b} = 0$ for original codeword symbols, user A wants to retrieve his own message. The algorithm combines the t values $(\beta'_{i_1,j_1}, \beta'_{i_2,j_2}, \ldots, \beta'_{i_t,j_t})$ to obtain $\tau^{a_1} = \tau^{f_{\mathrm{A},1}(0)}$ via the Lagrange interpolation over exponents

$$\tau^{a_1} = \prod_{(i,j)\in \mathrm{S}} \left((\beta'_{i,j})^{\prod_{r \in \mathrm{S_J}, r \neq j} \frac{-j}{r-j}} \right) = \tau^{f_{\mathrm{A},1}(0)}.$$

For each of the partially decrypted codeword symbols $\zeta_{i,j}$, where $i \in \mathrm{S_I}$, the algorithm computes an encoded block

$$w_i = \frac{\gamma_{i,j}}{\tilde{e}(\alpha_{i,j}, \tau^{f_{\mathrm{A},1}(0)})} = \frac{w_i \tilde{e}(g^{a_1}, \tau^{r'})}{\tilde{e}(g^{r'}, \tau^{f_{\mathrm{A},1}(0)})}, \tag{1}$$

for some r', where $f_{\mathrm{A},1}(0) = a_1$.

Observe that $w_i = m_1^{g_{1,i}} m_2^{g_{2,i}} \cdots m_k^{g_{k,i}}$ for $i \in \mathrm{S_I}$, and there are k such equations. Consider the square matrix $K = [g_{i,j}]$, where $1 \leq i \leq k, j \in \mathrm{S_I}$. The decoding process is to compute K^{-1} and output the blocks $m_1, m_2, \ldots, m_k$. The algorithm fails when the square matrix K is noninvertible. We shall analyze the probability of K being noninvertible in Section 4.2.

In the second case $\mathtt{b} = 1$ for re-encrypted codeword symbols, user B wants to retrieve the message forwarded to him. The algorithm does the following computation to obtain

$$\begin{aligned} h^{(f(a_3,\mathrm{ID})+e)a_1} &= \prod_{(i,j)\in \mathrm{S}} \left((\beta'_{i,j})^{\prod_{r \in \mathrm{S_J}, r \neq j} \frac{-j}{r-j}} \right) \\ &= h^{(f(a_3,\mathrm{ID})+e)a_1 b_2 f_{\mathrm{B},2}(0)}, \end{aligned}$$

where $f_{\mathrm{B},2}(0) = b_2^{-1}$. Again, for each of $\zeta_{i,j}$, where $i \in \mathrm{S_I}$, the algorithm computes an encoded block

$$w_i = \frac{\gamma_{i,j}}{\tilde{e}(\alpha_{i,j}, h^{(f(a_3,\mathrm{ID})+e)a_1})} = \frac{w_i \tilde{e}(g,h)^{a_1 r'(f(a_3,\mathrm{ID})+e)}}{\tilde{e}(g^{r'}, h^{(f(a_3,\mathrm{ID})+e)a_1})}, \tag{2}$$

for some e and r'. The rest in the second case is the same as that in the first case.

4.2 Analysis

We analyze storage and computation complexities, correctness, and security of our cloud storage system in this section. Let the bit-length of an element in the group $\mathbb{G}_1$ be l_1 and $\mathbb{G}_2$ be l_2. Let coefficients $g_{i,j}$ be randomly chosen from $\{0,1\}^{l_3}$.

Storage cost. To store a message of k blocks, a storage server SS_j stores a codeword symbol $(\mathtt{b}, \alpha_j, \tau, \gamma_j)$ and the coefficient vector $(g_{1,j}, g_{2,j}, \ldots, g_{k,j})$. They are total of $(1 + 2l_1 + l_2 + kl_3)$ bits, where $\alpha_j, \tau \in \mathbb{G}_1$ and $\gamma_j \in \mathbb{G}_2$. The average cost for a message bit stored in a storage server is $(1 + 2l_1 + l_2 + kl_3)/kl_2$ bits, which is dominated by l_3/l_2 for a sufficiently large k. In practice, small coefficients, i.e., $l_3 \ll l_2$, reduce the storage cost in each storage server.

Computation cost. We measure the computation cost by the number of pairing operations, modular exponentiations in $\mathbb{G}_1$ and $\mathbb{G}_2$, modular multiplications in $\mathbb{G}_1$ and $\mathbb{G}_2$, and arithmetic operations over $GF(p)$. These operations are denoted as Pairing, Exp_1, Exp_2, Mult_1, Mult_2, and F_p, respectively. The cost is summarized in Table 1. Computing an F_p takes much less time than computing a Mult_1 or a

TABLE 1
The Computation Cost of Each Algorithm in Our Secure Cloud Storage System

Operation	Computation cost
Enc	k Pairing + k Exp_1 + k Mult_2
Encode (for each storage server)	k Exp_1 + k Exp_2 + $(k-1)$ Mult_1 + $(k-1)$ Mult_2
KeyRecover	$O(t^2)$ F_p
ReKeyGen	1 Exp_1
ReEnc (for each storage server)	1 Pairing +1 Mult_2
ShareDec (for t key servers)	t Exp_1
Combine	k Pairing + t Mult_1 + $(t-1)$ Exp_1+O(t^2+k^3) F_p + k^2 Exp_2 + $(k+1)k$ Mult_2

- Pairing: a pairing computation of $\tilde{e}$.
- Exp_1 and Exp_2: a modular exponentiation computation in $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively.
- Mult_1 and Mult_2: a modular multiplication computation in $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively.
- F_p: an arithmetic operation in $GF(p)$.

Mult_2. The time of computing an Exp_1 is $1.5\lceil \log p \rceil$ times as much as the time of computing a Mult_1, on average, (by using the square-and-multiply algorithm). Similarly, the time of computing a Exp_2 is $1.5\lceil \log p \rceil$ times as much as the time of computing a Mult_2, on average.

In the data storage phase, a user runs the Enc$(\cdot)$ algorithm and each storage server performs the Encode$(\cdot)$ algorithm. In the Enc$(\cdot)$ algorithm, generating each α_i requires a Exp_1, and generating each γ_i requires a Exp_1, a Pairing, and a Mult_2. Hence, for k blocks of a message, the cost is $(k\,\mathrm{Pairing} + 2k\,\mathrm{Exp}_1 + k\,\mathrm{Mult}_2)$. For the Encode$(\cdot)$ algorithm, each storage server encodes k ciphertexts at most. The cost is $k\,\mathrm{Exp}_1 + (k-1)\,\mathrm{Mult}_1$ for computing α and $k\,\mathrm{Exp}_2 + (k-1)\,\mathrm{Mult}_2$ for computing γ.

In the data forwarding phase, a user runs KeyRecover$(\cdot)$ and ReKeyGen$(\cdot)$ and each storage server performs ReEnc$(\cdot)$. In the KeyRecover$(\cdot)$ algorithm, the computation cost is $O(t^2)$ F_p. In the ReKeyGen$(\cdot)$ algorithm, the computation cost is a Exp_1. In the ReEnc$(\cdot)$ algorithm, the computation cost is a Pairing and a Mult_1.

In the data retrieval phase, each key server runs the ShareDec$(\cdot)$ algorithm and the user performs the Combine$(\cdot)$ algorithm. In the ShareDec$(\cdot)$ algorithm, each key server performs a Exp_1 to get β^{sk_b} for a codeword symbol. For a successful retrieval, t key servers would be sufficient; hence, for this step, the total cost of t key servers is $t\,\mathrm{Exp}_1$. In the Combine$(\cdot)$ algorithm, it needs the computation of the Lagrange interpolation over exponents in $\mathbb{G}_1$, the computation of the encoded blocks w_j's from the partially decrypted codeword symbols $\tilde{\zeta}_{i,j}$'s, and the decoding computation which needs to perform the matrix inversion and recovery of blocks m_i's from the encoded blocks w_j's. The Lagrange interpolation over exponents in $\mathbb{G}_1$ needs $O(t^2)$ F_p, $t\,\mathrm{Exp}_1$, and $(t-1)\,\mathrm{Mult}_1$. Computing an encoded block w_j needs one Pairing and one modular division, which takes 2 Mult_2. As for the decoding computation, the matrix inversion takes $O(k^3)$ arithmetic operations over $GF(p)$, and the decoding for each block takes k Exp_2 and $(k-1)$ Mult_2.

Correctness. There are two cases for correctness. The owner A correctly retrieves his message and user B correctly retrieves a message forwarded to him. The correctness of encryption and decryption for A can be seen in (1). The correctness of re-encryption and decryption for B can be seen in (2). As long as at least k storage servers are available, a user can retrieve data with an overwhelming probability. Thus, our storage system tolerates $n-k$ server failures.

The probability of a successful retrieval. A successful retrieval is an event that a user successfully retrieves all k blocks of a message no matter whether the message is owned by him or forwarded to him. The randomness comes from the random selection of storage servers in the data storage phase, the random coefficients chosen by storage servers, and the random selection of key servers in the data retrieval phase. The probability of a successful retrieval depends on (n,k,u,v) and all randomness.

The methodology of analysis is similar to that in [13] and [6]. However, we consider a different system model from the one in [13] and a more flexible parameter setting for $n=ak^c$ than the settings in [13] and [6]. The difference between our system model and the one in [13] is that our system model has key servers. In [13], a single user queries k distinct storage servers to retrieve the data. On the other hand, each key server in our system independently queries u storage servers. The use of distributed key servers increases the level of key protection but makes the analysis harder.

The ratio n/k is considered as a fixed constant in [13]. In [6], the setting is extended to $n=ak^{3/2}$. Our generalization of parameter setting for $n=ak^c$, where $c \geq 1.5$, allows the number of storage servers be much greater than the number of blocks of a message. It gives a better flexibility for adjustment between the number of storage servers and robustness. This generalization is obtained by observing that $\Pr[\mathrm{E}_1]$ is better bounded by choosing $c \geq 1.5$. The proof of Theorem 1 is given in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.252.

Theorem 1. *Assume that there are k blocks of a message, n storage servers, and m key servers, where $n=ak^c$, $m \geq t \geq k$, $c \geq 1.5$ and a is a constant with $a > \sqrt{2}$. For $v = bk^{c-1}\ln k$ and $u=2$ with $b > 5a$, the probability of a successful retrieval is at least $1 - k/p - o(1)$.*

Security. The data confidentiality of our cloud storage system is guaranteed even if all storage servers, nontarget users, and up to $(t-1)$ key servers are compromised by the attacker. Recall the security game illustrated in Fig. 2. The proof for Theorem 2 is provided in Appendix B, available in the online supplementary material.

Theorem 2. *Our cloud storage system described in Section 4.1 is secure under the threat model in Section 3.2 if the decisional bilinear Diffie-Hellman assumption holds.*

5 Discussion and 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 codeword symbols, each key server only has to partially decrypt two codeword 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 [27], [28], [29] 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.

ACKNOWLEDGMENTS

The authors thank anonymous reviewers for their valuable comments. The research was supported in part by projects ICTL-100-Q707, ATU-100-W958, NSC 98-2221-E-009-068-MY3, NSC 99-2218-E-009-017-, and NSC 99-2218-E-009-020.

REFERENCES

[1] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: An Architecture for Global-Scale Persistent Storage," *Proc. Ninth Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS),* pp. 190-201, 2000.

[2] P. Druschel and A. Rowstron, "PAST: A Large-Scale, Persistent Peer-to-Peer Storage Utility," *Proc. Eighth Workshop Hot Topics in Operating System (HotOS VIII),* pp. 75-80, 2001.

[3] A. Adya, W.J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J.R. Douceur, J. Howell, J.R. Lorch, M. Theimer, and R. Wattenhofer, "Farsite: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment," *Proc. Fifth Symp. Operating System Design and Implementation (OSDI),* pp. 1-14, 2002.

[4] A. Haeberlen, A. Mislove, and P. Druschel, "Glacier: Highly Durable, Decentralized Storage Despite Massive Correlated Failures," *Proc. Second Symp. Networked Systems Design and Implementation (NSDI),* pp. 143-158, 2005.

[5] Z. Wilcox-O'Hearn and B. Warner, "Tahoe: The Least-Authority Filesystem," *Proc. Fourth ACM Int'l Workshop Storage Security and Survivability (StorageSS),* pp. 21-26, 2008.

[6] H.-Y. Lin and W.-G. Tzeng, "A Secure Decentralized Erasure Code for Distributed Network Storage," *IEEE Trans. Parallel and Distributed Systems,* vol. 21, no. 11, pp. 1586-1594, Nov. 2010.

[7] D.R. Brownbridge, L.F. Marshall, and B. Randell, "The Newcastle Connection or Unixes of the World Unite!," *Software Practice and Experience,* vol. 12, no. 12, pp. 1147-1162, 1982.

[8] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and Implementation of the Sun Network Filesystem," *Proc. USENIX Assoc. Conf.,* 1985.

[9] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu, "Plutus: Scalable Secure File Sharing on Untrusted Storage," *Proc. Second USENIX Conf. File and Storage Technologies (FAST),* pp. 29-42, 2003.

[10] S.C. Rhea, P.R. Eaton, D. Geels, H. Weatherspoon, B.Y. Zhao, and J. Kubiatowicz, "Pond: The Oceanstore Prototype," *Proc. Second USENIX Conf. File and Storage Technologies (FAST),* pp. 1-14, 2003.

[11] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G.M. Voelker, "Total Recall: System Support for Automated Availability Management," *Proc. First Symp. Networked Systems Design and Implementation (NSDI),* pp. 337-350, 2004.

[12] A.G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Ubiquitous Access to Distributed Data in Large-Scale Sensor Networks through Decentralized Erasure Codes," *Proc. Fourth Int'l Symp. Information Processing in Sensor Networks (IPSN),* pp. 111-117, 2005.

[13] A.G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Decentralized Erasure Codes for Distributed Networked Storage," *IEEE Trans. Information Theory,* vol. 52, no. 6 pp. 2809-2816, June 2006.

[14] M. Mambo and E. Okamoto, "Proxy Cryptosystems: Delegation of the Power to Decrypt Ciphertexts," *IEICE Trans. Fundamentals of Electronics, Comm. and Computer Sciences,* vol. E80-A, no. 1, pp. 54-63, 1997.

[15] M. Blaze, G. Bleumer, and M. Strauss, "Divertible Protocols and Atomic Proxy Cryptography," *Proc. Int'l Conf. Theory and Application of Cryptographic Techniques (EUROCRYPT),* pp. 127-144, 1998.

[16] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved Proxy Re-Encryption Schemes with Applications to Secure Distributed Storage," *ACM Trans. Information and System Security,* vol. 9, no. 1, pp. 1-30, 2006.

[17] Q. Tang, "Type-Based Proxy Re-Encryption and Its Construction," *Proc. Ninth Int'l Conf. Cryptology in India: Progress in Cryptology (INDOCRYPT),* pp. 130-144, 2008.

[18] G. Ateniese, K. Benson, and S. Hohenberger, "Key-Private Proxy Re-Encryption," *Proc. Topics in Cryptology (CT-RSA),* pp. 279-294, 2009.

[19] J. Shao and Z. Cao, "CCA-Secure Proxy Re-Encryption without Pairings," *Proc. 12th Int'l Conf. Practice and Theory in Public Key Cryptography (PKC),* pp. 357-376, 2009.

[20] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable Data Possession at Untrusted Stores," *Proc. 14th ACM Conf. Computer and Comm. Security (CCS),* pp. 598-609, 2007.

[21] G. Ateniese, R.D. Pietro, L.V. Mancini, and G. Tsudik, "Scalable and Efficient Provable Data Possession," *Proc. Fourth Int'l Conf. Security and Privacy in Comm. Netowrks (SecureComm),* pp. 1-10, 2008.

[22] H. Shacham and B. Waters, "Compact Proofs of Retrievability," *Proc. 14th Int'l Conf. Theory and Application of Cryptology and Information Security (ASIACRYPT),* pp. 90-107, 2008.

[23] G. Ateniese, S. Kamara, and J. Katz, "Proofs of Storage from Homomorphic Identification Protocols," *Proc. 15th Int'l Conf. Theory and Application of Cryptology and Information Security (ASIACRYPT),* pp. 319-333, 2009.

[24] K.D. Bowers, A. Juels, and A. Oprea, "HAIL: A High-Availability and Integrity Layer for Cloud Storage," *Proc. 16th ACM Conf. Computer and Comm. Security (CCS),* pp. 187-198, 2009.

[25] 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.

[26] A. Shamir, "How to Share a Secret," *ACM Comm.,* vol. 22, pp. 612-613, 1979.

[27] 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.

[28] 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.

[29] 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.

Hsiao-Ying Lin received the MS and PhD degrees in computer science from National Chiao Tung University, Taiwan, in 2005 and 2010, respectively. Currently, she is working as an assistant research fellow in Intelligent Information and Communications Research Center. Her current research interests include applied cryptography and information security. She is a member of the IEEE.

Wen-Guey Tzeng received the BS degree in computer science and information engineering from National Taiwan University, in 1985, and MS and PhD degrees in computer science from the State University of New York at Stony Brook, in 1987 and 1991, respectively. He joined the Department of Computer and Information Science (now, Department of Computer Science), National Chiao Tung University, Taiwan, in 1991. He now serves as a chairman of the department. His current research interests include cryptology, information security and network security. He is a member of the IEEE.