IEEE Cloud Computing,Volume:3,Issue:2,Mar.-Apr.2016.

Encrypted Data Management with Deduplication in Cloud Computing

Zheng Yan, Mingjun Wang, and Yuxiang Li, Xidian University, China
Athanasios V. Vasilakos, Lulea University of Technology, Sweden

This article proposes a scheme based on attribute-based encryption (ABE) to deduplicate encrypted data stored in the cloud while at the same time supporting secure data access control.

Cloud computing offers a new way to deliver services by rearranging resources over the Internet and providing them to users on demand. It plays an important role in supporting data storage, process, and management in the Internet of Things (IoT). Various cloud service providers (CSPs) offer huge volumes of storage to maintain and manage IoT data, which can include videos, photos, and personal health records. These

IEEE Cloud Computing,Volume:3,Issue:2,Mar.-Apr.2016.

CSPs provide desirable service properties, such as scalability, elasticity, fault tolerance, and pay per use. Thus, cloud computing has become a promising service paradigm to support IoT applications and IoT system deployment.

To ensure data privacy, existing research proposes to outsource only encrypted data to CSPs. However, the same or different users could save duplicated data under different encryption schemes at the cloud. Although cloud storage space is huge, this kind of duplication wastes networking resources, consumes excess power, and complicates data management. Thus, saving storage is becoming a crucial task for CSPs. Deduplication can achieve high space and cost savings, reducing up to 90 to 95 percent of storage needs for backup applications (http://opendedup.org) and up to 68 percent in standard file systems.[1] Obviously, the savings, which can be passed back directly or indirectly to cloud users, are significant to the economics of cloud business.

At the same time, data owners want CSPs to protect their personal data from unauthorized access. CSPs should therefore perform access control based on the data owner's expectations. In addition, data owners want to control not only data access but also its storage and usage. From a flexibility viewpoint, data deduplication should cooperate with data access control mechanisms. That is, the same data, although in an encrypted form, is only saved once at the cloud but can be accessed by different users based on the data owners' policies.

However, current industrial deduplication solutions can't handle encrypted data. Existing solutions for deduplication are vulnerable to brute-force attacks[2] and can't flexibly support data access control and revocation (see the "Related Work in Data Deduplication" sidebar for a discussion of some other work in this area).[3] Few existing schemes for cloud data access control support data deduplication simultaneously,[4] and few can ensure flexibility and security with sound performance for cloud data deduplication that data owners control directly.[5–7]

We propose a scheme based on attribute-based encryption (ABE) to deduplicate encrypted data stored in the cloud and support secure data access control at the same time. Analysis and implementation demonstrate that our scheme is secure, effective, and efficient.

System Design and Algorithms

Our scheme works in a system containing three types of entities:

- a CSP that offers a storage service and performs honestly on data storage and management to gain commercial profit but can't be fully trusted since it's curious about the contents of stored data;
- a data owner that stores its data at the CSP (we assume there's only one data owner for one data M); and
- data holders (u_i, $i = 1, \ldots, n$) that are eligible data users and could save the same data as the data owner at the CSP.

We assume that the system uses a data owner or holder device (such as a smart mobile device) to collect and preprocess data collected by IoT devices if they aren't capable of cryptographic operations or networking.

Additional assumptions include the data's hash code $M(H(M))$ being applied as its indicator, which is used to check the duplication of data during data storage. We assume the data holder signs the right hash code honestly for ownership verification at the CSP. This hash code is protected and can't be obtained by attackers. We further assume that the data owner has the highest priority for data storage management. A data holder should provide valid proof of ownership to request special treatment. The CSP, data owners, and data holders communicate with each other through a secure channel.

System Setup and Required Keys

In our scheme, we manage the keys of different authorized data users for deduplication by treating their identities (IDs) as valid attributes for accessing encrypted data stored at the CSP. Our scheme is applicable to scenarios in which the data owner wants to control data storage and access and track its own data location and usage status.

During system setup, every data owner or holder, CSP user u, maintains a public key PK_u, which other users employ to generate personalized secret attribute keys, and a secret key SK_u, which is used in the decryption operation related to PK_u. The

IEEE Cloud Computing,Volume:3,Issue:2,Mar.-Apr.2016.

RELATED WORK IN DATA DEDUPLICATION

Reconciling deduplication is an active research topic. Cloud storage service providers, such as Dropbox (www.dropbox.com), Google Drive (http://drive.google.com), and Mozy (http://mozy.com), perform deduplication to save space by storing only one copy of each file uploaded. Should clients conventionally encrypt their data, however, savings are lost. Existing industrial solutions fail in encrypted data deduplication.

Message-locked encryption (MLE) aims to solve this problem.[1,2] The most prominent manifestation of MLE is convergent encryption,[2] which is used in a wide variety of commercial and research storage service systems. Let M be a file's contents. A client A first computes a key $K \leftarrow H(M)$ by applying a cryptographic hash function $H()$ to M, and then computes the ciphertext $C \leftarrow E(K, M)$ via a deterministic symmetric encryption scheme. A second client B encrypting the same file M will produce the same C, enabling deduplication. However, because convergent encryption is deterministic and keyless, it's subject to an inherent security limitation—namely, susceptibility to offline brute-force dictionary attacks.[1] Secure convergent encryption is only possible when the target M is drawn from a space too large to exhaust.

Mihir Bellare and his colleagues proposed DupLESS, which provides secure deduplicated storage to resist brute-force attacks.[3] In DupLESS, a group of affiliated clients encrypt their data using a key server that's separate from a storage service. Clients authenticate themselves to the key server but don't leak any information about their data to it. As long as the key server remains inaccessible to attackers, DupLESS can ensure high security.

Convergent encryption also suffers from another access control problem. It isn't flexible to control data access, especially for data revocation, since it's hard for data holders to generate the same new key for data reencryption. An image deduplication scheme adopts two servers to achieve verifiability of deduplication.[4] A convergent encryption-based scheme described elsewhere combines file content and user privilege to obtain a file token with unforgeability.[5] However, the schemes we've described directly encrypt data with a convergent encryption key, so suffer from the access control problem we've described.

data owner or holder uses the SK_u to issue secret attribute keys to other users and to generate its own public key $PKID_u$ of identity attribute ID. CSP user u generates $PKID_u$ to encrypt a symmetric key DEK_u, randomly selected for encrypting the data of u, aiming to control data access and deduplication. The corresponding secret attribute keys for decrypting the cipher key encrypted by $PKID_u$ are personalized for eligible data holders and issued by data owner u. To prevent collusion, every holder gets a different secret attribute key that only it can use. A secret attribute key of the attribute ID, issued for an eligible data holder u' by u is denoted as $SKID_{(u,u')}$. Meanwhile, user u also generates a key pair pk_u and sk_u for Public-Key Cryptosystem (PKC), for example, signature generation and verification. The keys (PK_u, SK_u), (pk_u, sk_u) are bound to the unique identity of u, which can be a unique anonymous identifier. This binding is crucial for user identity verification. At system setup, PK_u and pk_u are certified by an authorized third party as $Cert(PK_u)$, $Cert(pk_u)$ which the CSP and any CSP users can verify.

Algorithms

Our scheme consists of several fundamental algorithms. We can adopt either cyphertext policy ABE (CP-ABE) or key policy ABE (KP-ABE) to implement related algorithms.

InitiateNode. The InitiateNode algorithm takes as input the node identifier u. It outputs several user credentials including (PK_u, SK_u) and (pk_u, sk_u). This process is conducted at user u.

CreateIDPK. The CreateIDPK algorithm is executed by u whenever u wants to control its data storage and access in the cloud. The algorithm checks the

To resist attacks involving manipulation of a data identifier, Pierre Meye and his colleagues proposed adopting two servers for intra-user deduplication and interdeduplication.[6] In their scheme, the ciphertext *C* of convergent encryption is further encrypted with a user key and transferred to the servers. However, it doesn't deal with data sharing after deduplication among different users.

ClouDedup also aims to cope with the inherent security exposure of convergent encryption, but it can't solve the problem caused by data deletion.[7] A data holder that removes the data from the cloud can still access that data if it isn't completely removed from the cloud since it still knows the data encryption key. Other work proposed using proxy reencryption (PRE) to achieve encrypted data deduplication by applying an authorized party.[8] In many situations, data owners want to control data storage and access by themselves and track their own data's usage status. However, the solutions described here can't support this requirement in a flexible way.

References

1. M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-Locked Encryption and Secure Deduplication," *Advances in Cryptology* (EUROCRYPT 13), LNCS 7881, 2013, pp. 296–312.
2. J.R. Douceur et al., "Reclaiming Space from Duplicate Files in a Serverless Distributed File System," *Proc. 22nd Int'l Conf. Distributed Computing Systems*, 2002, pp. 617–624.
3. M. Bellare, S. Keelveedhi, and T. Ristenpart, "DupLESS: Server-Aided Encryption for Deduplicated Storage," *Proc. 22nd Usenix Conf. Security*, 2013, pp. 179–194.
4. Z.C. Wen et al., "A Verifiable Data Deduplication Scheme in Cloud Computing," *Proc. Int'l Conf. Intelligent Networking and Collaborative Systems*, 2014, pp. 85–90.
5. J. Li et al., "A Hybrid Cloud Approach for Secure Authorized Deduplication," *IEEE Trans. Parallel Distributed Systems*, vol. 26, no. 5, 2015, pp. 1206–1216.
6. P. Meye et al., "A Secure Two-Phase Data Deduplication Scheme," *Proc. IEEE 6th Int'l Symp. Cyberspace Safety and Security, IEEE 11th Int'l Conf. Embedded Software and Systems* (HPCC, CSS, ICESS), 2014, pp. 802–809.
7. P. Puzio et al., "ClouDedup: Secure Deduplication with Encrypted Data for Cloud Storage," *Proc. IEEE 5th Int'l Conf. Cloud Computing Technology and Science*, 2013, pp. 363–370.
8. Z. Yan, W. Ding, and H. Zhu, "Manage Encrypted Data Storage with Deduplication in Cloud," *Proc. Int'l Conf. Algorithms and Architectures for Parallel Processing* (ICA3PP), 2015, pp. 547–561.

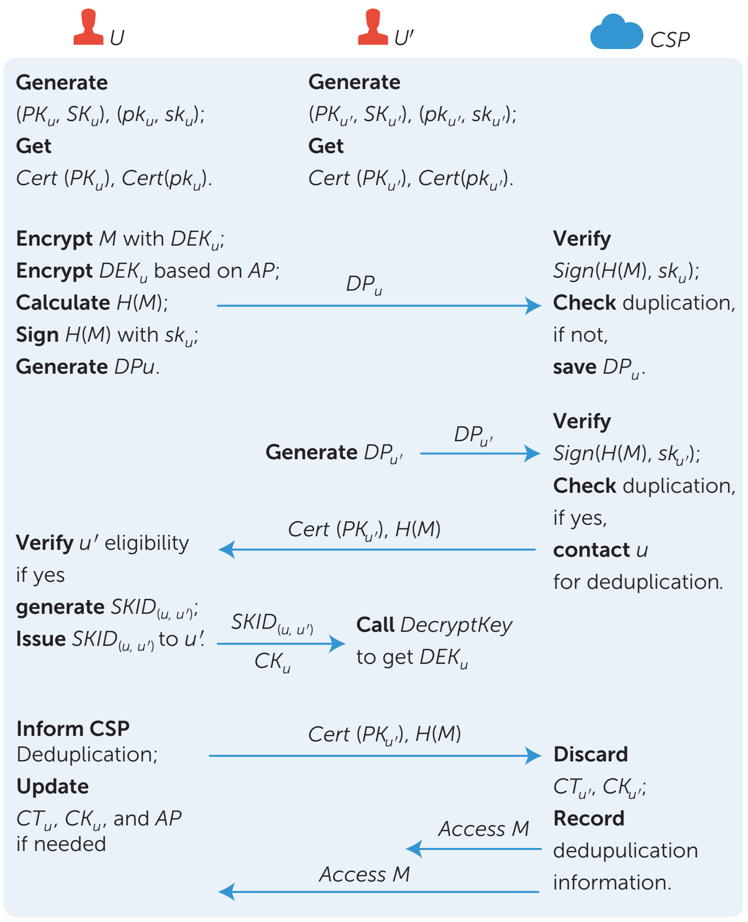
ID-related policies. If the result is positive, the algorithm outputs a public attribute key about the ID for user u, denoted $PKID_u$; otherwise, it outputs NULL.

IssueIDSK. The IssueIDSK algorithm checks whether u' with public key $PK_{u'}$ is eligible to hold the data. If it is, IssueIDSK outputs a secret attribute key $SKID_{(u,u')}$ for user u'. Otherwise, it outputs NULL. This process is executed by u based on identity verification by checking that $Cert(PK_{u'})$ is a valid certificate and the owner of $PK_{u'}$ is an eligible data holder. Data owner u sends $SKID_{(u,u')}$ to u' through a secure channel or using the PKC.

For example, data owner u would like a data holder with $ID = PK_{u'}$ to share its data storage. It encrypts its data encryption key DEK_u with an access policy AP: $ID = PK_{u'}$. The access policy can also contain multiple IDs or other control attributes when the data owner wants fine-grained control over the stored data in the CSP, for example, $AP : V_{i=1}^{n} ID_i$.

EncryptKey. The EncryptKey algorithm takes as input symmetric key DEK_u, access policy AP, and public key $PKID_u$ corresponding to the identity attribute occurring in AP. The algorithm encrypts DEK_u with the policy AP and outputs the cipherkey CK_u. This process is conducted at u to support deduplication of data storage at the CSP.

DecryptKey. The DecryptKey algorithm takes as input cipherkey CK_u produced by the EncryptKey algorithm; the access policy, AP, under which cipher key CK_u was encrypted; $SK_{u'}$; and $SKID_{(u,u')}$. It decrypts CK_u and outputs the corresponding plain key DEK_u if the attributes are sufficient to satisfy AP; otherwise, it outputs NULL. This process is executed at u' if duplicated storage occur. It first checks



U
U′
CSP
Generate (PK_u, SK_u), (pk_u, sk_u); Get Cert (PK_u), Cert(pk_u).
Generate $(PK_{u'}, SK_{u'})$, $(pk_{u'}, sk_{u'})$; Get Cert $(PK_{u'})$, Cert$(pk_{u'})$.
Encrypt M with DEK_u; Encrypt DEK_u based on AP; Calculate $H(M)$; Sign $H(M)$ with sk_u; Generate DPu.
DP_u
Verify Sign$(H(M), sk_u)$; Check duplication, if not, save DP_u.
Generate $DP_{u'}$
$DP_{u'}$
Verify Sign$(H(M), sk_{u'})$; Check duplication, if yes, contact u for deduplication.
Cert $(PK_{u'})$, $H(M)$
Verify u' eligibility if yes generate $SKID_{(u, u')}$; Issue $SKID_{(u, u')}$ to u'.
$SKID_{(u, u')}$
CK_u
Call DecryptKey to get DEK_u
Inform CSP Deduplication; Update CT_u, CK_u, and AP if needed
Cert $(PK_{u'})$, $H(M)$
Discard $CT_{u'}$, $CK_{u'}$; Record deduplication information.
Access M
Access M

FIGURE 1. Data deduplication process.

the access policy AP, then conducts decryption to get DEK_u.

Encrypt. The algorithm encrypts data M with DEK_u and outputs ciphertext CT_u. This process is conducted at user u to protect its data with DEK_u.

Decrypt. This algorithm is conducted at user u' or u to decrypt CT_u with DEK_u and output plain data M.

Scheme

Designing a deduplication scheme with ABE has several advantages. First, the scheme can easily realize data access control by introducing control policies into AP when calling $EncryptKey(DEK_u, AP, PKID_u)$ by updating AP to support both deduplication and access control based on practical demands. Our scheme can also support digital rights management based on the data owner's expectations. Second, the scheme saves CSP storage space since it only stores one copy of the same data. Storing deduplication functional records could occupy some storage memory, but this cost is minimal compared to the cost of storing a large volume of duplicated data. Third, the proposed scheme can support many duplication instances and a huge volume of duplicated data. In this case, data holder u' only sends data package $\{H(M), Sign(H(M), sk_{u'}), Cert(PK_{u'}), Cert(pk_{u'})\}$ without $CT_{u'}$ and $CK_{u'}$ for a duplication check before actually uploading the data. If duplication occurs, data holder u' can get $SKID_{(u,u')}$ from the data owner if it's eligible.

Data Deduplication

Figure 1 illustrates the data deduplication process at a CSP with the data owner's instruction and control. We assume that user u is a data owner who saves data M at the CSP, using DEK_u to protect the data, while user u' is a data holder who tries to save the same data at that CSP.

First, each CSP user generates personal credentials and two key pairs (PK_u, SK_u) and (pk_u, sk_u) and gets the certificates of its public keys $Cert(PK_u)$ and $Cert(pk_u)$.

User u saves data M at the CSP. The user generates $CT_u = Encrypt(DEK_u, M)$ and encrypts DEK_u with $PKID_u$ by calling $EncryptKey(DEK_u, AP, PKID_u)$ to get CK_u. It generates $PKID_u$ according to u's data storage and access policy. User u calculates $H(M)$ and signs it with sk_u as $Sign(H(M), sk_u)$. Next, u sends the data package, $DP_u = \{CT_u, CK_u, H(M), Sign(H(M), sk_u), Cert(PK_u), Cert(pk_u)\}$, to the CSP.

After receiving DP_u, the CSP verifies $Cert(PK_u)$ and $Cert(pk_u)$. If the verification is positive, the CSP uses verify $Sign(H(M), sk_u)$ to check if duplicate data is saved by finding whether the same $H(M)$ is in its storage. If the check is negative, the CSP saves DP_u. If the check is positive and the prestored data is from the same user, the CSP notifies that user. If a different user is storing the same data, the CSP performs deduplication.

If data holder u' uploads the same data to the CSP, the CSP contacts data owner u by sending $H(M)$ and $Cert(PK_{u'})$ for deduplication. User u verifies the eligibility of u'. If verification is positive, user u calls $IssueIDSK(ID, SK_u, PK_{u'})$ to generate $SKID_{(u,u')}$ and issues $SKID_{(u,u')}$ to u' to allow it to access M. Next, user u reports the successful data deduplication to the CSP. After receiving this notification, the CSP discards $CT_{u'}$ and $CK_{u'}$ and recodes the corresponding deduplication information in the system. In this step, the data owner can also update DEK_u, upload new CT_u and CK_u to the CSP, and send the newly encrypted DEK_u (with ABE) to eligible data holders through the CSP or directly.

At this moment, both u and u' can access data

IEEE Cloud Computing,Volume:3,Issue:2,Mar.-Apr.2016.

Table 1. Computation complexity comparison.*

Entity	Algorithm		Computations		Complexity	
	This article	Yan and colleagues[9]	This article	Yan and colleagues[9]	This article	Yan and colleagues[9]
Data owner	InitiateNode	InitiateNode	1*ModInv+2*Exp	1*ModInv+1*Exp	$O(1)$	$O(1)$
	CreateIDPK	–	2*Exp	–		
	IssueIDSK	–	1*Exp	–		
	EncryptKey	Data upload	3*Exp	3*Exp		
CSP	Verify signature	Reencryption	1*Exp	1*BiPair	$O(n)$	$O(n)$
Data holder	DecryptKey	DecryptKey	2*BiPair	1*Exp	$O(1)$	$O(1)$
Access policy	System setup	System setup	–	1*Exp	–	$O(n)$
	Reencryption key generation	Reencryption key generation	–	1*Exp		

*BiPair: bilinear pairing; Exp: exponentiation; *n*: number of data holders who share the same data; ModInv: modular inversion.

M stored at the CSP. User u uses DEK_u directly, whereas u' gets DEK_u by calling $DecryptKey(CK_u, AP, SK_{u'}, SKID_{(u,u')})$.

Data Deletion at the CSP

Data holder u' sends a deletion request to the CSP by providing $Cert(pk_{u'})$ and $H(M)$. The CSP checks the request's validity, then removes the deduplication record of u', and blocks its later access to M. The CSP further checks if the deduplication record is empty. If it is, the CSP deletes encrypted data CT_u and all related records.

When data owner u sends a deletion request to the CSP with $Cert(pk_u)$ and $H(M)$, the CSP checks the request's validity, then removes the deduplication record of u and blocks its later access to data M. The CSP further checks whether the deduplication record is empty. If it is, it deletes encrypted data CT_u and all related records. Otherwise, the CSP asks u to continuously manage deduplication. If u agrees, no action is performed at the CSP. If u doesn't agree, the CSP contacts data holder u' with the earliest data storage or data holder u' designated by u for later deduplication support. In this case, u' encrypts DEK_u by calling $CK_{u'} = EncryptKey(DEK_u, AP, PKID_{u'})$ and then performs as a delegate of the data owner to support deduplication by sending $CK_{u'}$ to CSP.

Data Owner Management

During data deletion, the CSP also asks the data owner to allow it to decide if the raw data needs to be reencrypted. For security purposes, the data owner could select a new DEK_u, reencrypt data M, update CT_u and CK_u at the CSP, and issue a new CK_u to existing eligible users for managing data deletion. If the real data owner uploads the data after the data holder, the CSP can save the data encrypted by the real data owner at the cloud and allow the data owner to manage and issue corresponding access keys to other data holders.

Performance Analysis and Evaluation

Our scheme ensures data confidentiality through ABE, symmetric key encryption, and PKC. The original user data is encrypted using symmetric encryption with DEK, which is then encrypted using the EncryptKey algorithm under access policy AP. Assuming that the symmetric key algorithm is secure (for example, using a standard algorithm such as AES), the scheme's data confidentiality merely relies on the security of the EncryptKey algorithm. The CSP and other parties have no way to know the original data without DEK. We use ABE to preserve DEK. A security proof has been given under an attribute-based selective-set module.[8] The CSP and other users with unmatched identities can't get the secret key $SKID_{(u,u')}$ from the data owner, so they can get nothing about DEK and the original data.

The proposed scheme involves three system entities: data owner, data holder, and CSP. We adopt AES for symmetric encryption, RSA for PKC. and CP-ABE for data deduplication. Table 1 compares the computation complexity of the current scheme with that of work presented elsewhere.[9]

The InitiateNode algorithm includes the key generation operation of two key pairs, which are PKC (RSA) key pair generation and ABE key pair generation. The RSA key generation needs one modular inversion. The key generation of ABE includes an exponentiation for PK_u and an exponentiation for SK_u on a group. The CreateIDPK algorithm contains two exponentiations on a group and the IssueIDSK

IEEE Cloud Computing,Volume:3,Issue:2,Mar.-Apr.2016.

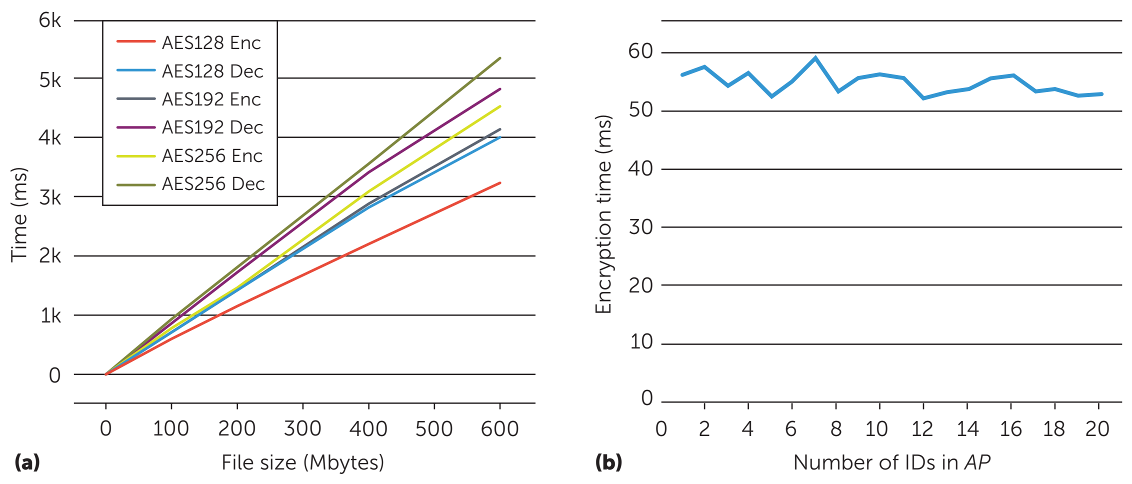


FIGURE 2. Operation time of (a) file encryption and decryption with AES and (b) the EncryptKey algorithm.

algorithm contains only one exponentiation. The computation complexity of encrypting data using AES depends on the data size, which is inevitable in any cryptographic method for protecting data. However, for the EncryptKey algorithm, the computation contains three exponentiations on a group no matter how many IDs are in access policy *AP*. The signing operation requires one exponentiation.

Each user uploads its data to the CSP. The CSP should first check the signature and decide if saving the appropriate copy of the data. It needs only one exponentiation for signature verification. Its computation complexity depends on the number of data holders.

When a data holder wants to store data to a CSP, it operates the same way as a data owner. But if it wants to access the data, it needs to conduct a decryption operation. Here, we only consider the DecryptKey algorithm since the computation complexity of the Decrypt algorithm depends on the data size. The DecryptKey algorithm contains two bilinear pairings, which are cost constant.

We can see that the computation complexity of the data owner and the data holder are both $O(1)$. The CSP's computation complexity is $O(n)$, where n is the number of data holders, but generally a CSP has sufficient computation capability. This fact suggests that our scheme is computationally efficient. As Table 1 shows, the proposed scheme has similar computation complexity to the scheme presented elsewhere.[9] Although key generation at the data owner's end takes more computation time, it doesn't depend on a trusted third party. The system deployment and operation costs should be much lower than in the other work.[9]

When user u saves data M at the CSP, the communication cost is 1,979 bytes (excluding CT_u) if the same data isn't prestored. If there's duplication, user u uploads data package DP_u to the CSP. Next, $H(M)$, $Cert(PK_u)$, and $SKID_{(u,u')}$ are transferred among CSPs, u, and u'. In this case, the total communication cost is 6,151 bytes (excluding CT_u). Thus, the extra communication cost introduced by our scheme is trivial compared to the large volume of saved data storage.

Implementation and Evaluation

We implemented the proposed scheme using CP-ABE with the pairing-based cryptography (PBC) library (http://crypto.stanford.edu/pbc) and the OpenSSL library. We tested our scheme's performance in a Linux operating system (Linux 2.6.32-71.el6.i686 CentOS 6.0) running an Intel CPU (Intel Core i5-2450 Quad CPU 2.5 GHz, 2 Gbytes SDRAM). We used SHA-1 as the hash function in implementation.

We tested the operation time of data encryption and decryption with AES by applying different AES key sizes (128, 196, and 256 bits) and different data size (from 10 to 600 Mbytes). As Figure 2a shows, even when the data is as large as 600 Mbytes, the encryption/decryption time is less than 6,000 milliseconds (ms) using a 256-bit AES key. Therefore, applying symmetric encryption for data protection is a reasonable and practical choice.

Based on our implementation, we found the operation time of InitiateNode, CreateIDPK, IssueIDSK, and DecryptKey to be 69.2, 4.3, 3.9, and 12.7 ms. All of them are constant. For EncryptKey, the operation time is about 54 ms, as Figure 2b shows, approximately constant, which doesn't vary much with the number of IDs in *AP*. This is because *AP* only contains one type of attribute ID. This result proves that our scheme is efficient, with sound scalability and extensibility. The operation time of fundamental algorithms is stable and doesn't change with the number of data holders.

Managing encrypted data with deduplication is significant in practice for running a secure, dependable, and green cloud storage service, especially for big data processes. Future work includes efficient data ownership verification, scheme optimization with hardware acceleration at IoT devices for practical deployment, and development of a flexible solution to support deduplication and data access controlled by either the data owner or its representative agent.

Acknowledgments

This work is sponsored by the NSFC (grant U1536202), the 111 project (grant B08038), the PhD grant of the Chinese Educational Ministry (grant JY0300130104), the initial grant of the Chinese Educational Ministry for researchers from abroad (grant JY0600132901), and the grant of the Shaanxi Province for excellent researchers from abroad (grant 680F1303), as well as Aalto University. The corresponding author is Zheng Yan.

References

1. D.T. Meyer and W.J. Bolosky, "A Study of Practical Deduplication," *ACM Trans. Storage*, vol. 7, no. 4, 2012, pp. 1–20.
2. M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-Locked Encryption and Secure Deduplication," *Advances in Cryptology* (EUROCRYPT 13), LNCS 7881, 2013, pp. 296–312.
3. J. Li et al., "A Hybrid Cloud Approach for Secure Authorized Deduplication," *IEEE Trans. Parallel Distributed Systems*, vol. 26, no. 5, 2015, pp. 1206–1216.
4. Z. Wan, J. Liu, and R.H. Deng, "HASBE: A Hierarchical Attribute-Based Solution for Flexible and Scalable Access Control in Cloud Computing," *IEEE Trans. Information Forensics and Security*, vol. 7, no. 2, 2012, pp. 743–754.
5. M. Fu et al., "Accelerating Restore and Garbage Collection in Deduplication-Based Backup Systems via Exploiting Historical Information," *Proc. Usenix Ann. Technical Conf.*, 2014, pp. 181–192.
6. M. Kaczmarczyk et al., "Reducing Impact of Data Fragmentation Caused by In-Line Deduplication," *Proc. 5th Ann. Int'l Systems and Storage Conf.*, 2012, pp. 1–12.
7. M. Lillibridge, K. Eshghi, and D. Bhagwat, "Improving Restore Speed for Backup Systems That Use Inline Chunk-Based Deduplication," *Proc. 11th Usenix Conf. File and Storage Technologies*, 2013, pp. 183–198.
8. Z. Yan and M.J. Wang, "Protect Pervasive Social Networking Based on Two Dimensional Trust Levels," *IEEE Systems J.*, Sept. 2014, pp. 1–12; doi: 10.1109/JSYST.2014.2347259.
9. Z. Yan, W. Ding, and H. Zhu, "Manage Encrypted Data Storage with Deduplication in Cloud," *Proc. Int'l Conf. Algorithms and Architectures for Parallel Processing* (ICA3PP), 2015, pp. 547–561.

ZHENG YAN *is a professor in the School of Cyber Engineering at Xidian University, China, and a visiting professor in the Department of Communications and Networking at Aalto University, Finland. Her research interests include trust, security and privacy, social networking, cloud computing, networking systems, and data mining. Yan has a doctor of science in technology degree in electrical engineering from Helsinki University of Technology, Finland. She's a senior member of IEEE. Contact her at zyan@xidian.edu.cn.*

MINGJUN WANG *is a PhD student in information security at Xidian University, Xi'an, China. His research interests include security, privacy, and trust management in social networking, 5G, and cloud computing. Wang has a BSc in communication and information systems from Henan Normal University, Xinxiang, China. Contact him at xdmjwang@hotmail.com.*

YUXIANG LI *is a student at the State Key Lab on Integrated Services Networks at Xidian University, China. His research interests include information security and cloud computing. Li has a BSc in telecommunications engineering from Xidian University. Contact him at 18717319433@163.com.*

ATHANASIOS V. VASILAKOS *is a professor in the Department of Computer Science at the Luleå University of Technology, Sweden. His research interests include networks, cloud computing, security, medical informatics. Vasilakos has a PhD in networks from University of Patras. He's a senior member of IEEE. Contact him at th.vasilakos@gmail.com.*