EAI Endorsed Transactions
on Security and Safety
Research Article EAI.EU

A Rekeying Scheme for Encrypted Deduplication Storage based on NTRU

GuanXiong Ha[1,*], Hang Chen[1], Ruiqi Li[1], Wei Shao[1], Chunfu Jia[1]

[1]Nankai University

Abstract

Rekeying is a common way to protect outsourced data against key compromise and to enable data owners to enforce access control on their data. However, existing rekeying schemes are difficult to apply to the encryption deduplication system which uses message-locked encryption for allowing the server to perform deduplication on users' outsourced data. In this paper, we propose a new rekeying scheme named REEDBN, which leverages a proxy re-encryption based on NTRU to reduce the communicational cost for the system and the computational overheads for clients during rekeying. We implement the prototype ofour scheme and conduct testbed experiments. The results show that our system has much less communicational effort and computational overhead for clients than the previous scheme. Users can even rekey their outsourced data on some mobile terminals which only have limited computation power.

Keywords: Cloud Storage, Deduplication, Rekeying, NTRU.

Received on 01 December 2020, accepted on 05 January 2021, published on 12 January 2021

doi: 10.4108/_____

1. Introduction

In the era of big data, deduplication is an effective way to save storage space. Deduplication improves the utilization of storage space by removing redundant copies in storage. In cloud storage systems, users are usually billed by the amount of data transmitted and stored [1]. By client-side deduplication, it can not only save storage space but even reduce the amount of data being transferred between clients and servers, which greatly decreases the cost for both clients and servers [2]. Recently, the problem of data security has attracted more and more attention. Users incline to upload encrypted data for data confidentiality, so that the server cannot learn anything about the outsourced data. How- ever, traditional encryption algorithms are contradictory with the concept of deduplication since encryption turns the same message into indistinguishable ones and hinders deduplication as the detection of the duplication of encrypted data becomes difficult. Convergent encryption (CE) [3] is the first attempt to achieve encrypted deduplication. In CE, clients use the hash of the original data as the symmetric key to encrypt data. Since the encryption key is the same if users have the same file, the server can detect duplication by the encrypted data. However, CE is inherently vulnerable to brute-force attacks [4], as it cannot provide confidentiality guarantees for predictable messages. DupLESS [4] generates encryption keys through a dedicated key server. It introduces a system-wide secret for the key generation. It is hard for attackers to perform brute- force attacks without knowing the system-wide secret. However, it is inefficient to use DupLESS in chunk-level deduplication because the speed of the key generation is slow [5].

Although current encrypted deduplication systems [6][7][8] can provide data confidentiality to some extent, there are other threats on user data in different scenarios. For example, it is hard for data owners to rekey their outsourced data [9] when the encryption key is compromised. Suppose that a file stored on the server is encrypted by the key $H(F)$, where $H(\cdot)$ is a cryptography hash function. Once $H(F)$ is breached by an adversary, the data owner has to rekey the file by a new convergent key $H'(F)$, where $H'(\cdot)$ is another cryptography hash function. However, to achieve this process, the data owner needs to download the entire file, decrypt it, and upload the new ciphertext encrypted by the new key $hash'(F)$ to the server. Besides, the data owner also needs to broadcast the new hash function $hash'(\cdot)$ to all other owners for

* Corresponding author. Email: 2472727110@qq.com

protect against the compromise of the MLE key. The workflow of it is as follows.

1. Use MLE to encrypt the chunk m_i, get the ciphertext $C_i = Enc_{MLE}(k_i, m_i)$, where

$Enc_{MLE}(\cdot)$ denotes a MLE encryption function.

2. Concatenate the C_i with k_i, obtain $(C_i||k_i)$, compute the hash of the concatenation

$h = H(C_i||k_i)$.

3. Use h and S to compute the pseudo-random mask $G(h) = Enc_h(S)$, where S is a publicly known block of the same size as $C_i||k_i$. Then XOR it with the package $(C_i||k_i)$, compute the $C_i' = (C_i||k_i) \oplus G(h)$.
4. Divide C_i' into several fixed-size pieces whose size is the same as h. XOR all thesepieces and h to compute the tail t.
5. Concatenate C_i' and t, obtain the $(C_i'||t)$. Trim the last few bytes from the package

$(C_i'||t)$ as the stub st_i, and the remaining part as the trimmed package tp_i.

6. Generate a file-level secret key K, encrypt all the stubs of F using AES, get the

$Enc_K(st_1, st_2, \dots, st_n)$.

7. The client uploads the encrypted stubs $Enc_K(st_1, st_2, \dots, st_n)$ and all the trimmed packages $\{tp_1, tp_2, \dots tp_n\}$ to the server.

To reconstruct m_i, the client needs to have both the encrypted stubs and the trimmedpackages. First, the client decrypts the encrypted stub with K, and concatenate C_i' and t to obtain the $(C_i'||t)$. Then we divide C_i' into pieces and XOR them with t to get h. Recover $(C_i||k_i) = C_i' \oplus G(h)$. Finally the client can obtain $m_i = Dec_{MLE}(k_i, C_i)$, where $Dec_{MLE}(\cdot)$ is a MLE decryption function.

3.3 Rekeying

To rekey file F, the client downloads the encrypted stubs from the server, and decrypt them with the file-level secret key K. Then the client encrypts the stubs with a new file- level secret key K' and uploads a new ciphertext $EncK'(stub)$, as is shown in Figure 1. However, if the size of the stubs is relatively large or we need to rekey frequently, rekeying in REED will incur a lot of bandwidth overheads and computational efforts. Besides, when we need to rekey outsourced data on some mobile terminals (e.g., smart phone) which only have limited computational power, frequent encryptions and decryptions make it difficult to rekey on these devices.

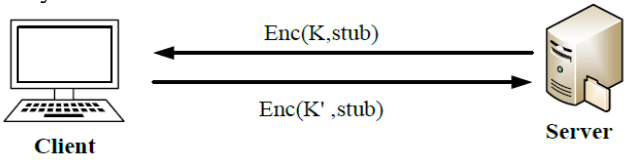


Fig. 1. Rekey in REED

4. Overview of REEDBN

In order to further reduce the cost of rekeying, we propose the REEDBN, which is a cross-user server-side encrypted deduplication storage system. Users can outsource their data to REEDBN for saving local storage space. MLE and NTRU are both used in REEDBN to protect user data. More importantly, REEDBN achieves a new light- weight rekeying scheme where the client just needs to compute a re-encryption key. REEDBN only has negligible bandwidth overhead during rekeying.

4.1 Architecture

REEDBN is composed of clients, a key manager, and a server.

Server: REEDBN deploy a server to provide data storage and data management services for multiple users. It applies the deduplication to user data to save storage space. When receiving data from clients, the server will detect whether there is a duplicate one or not. Only if the data is not duplicated, would the server store it. Furthermore, we assume the server has strong computing power and can perform costly calculations. Clients: Users can outsource their data to the server through clients. Since REEDBN achieves the block-level deduplication, clients need to divide user data into chunks. After chunking, clients generate trimmed packages and stubs based on the chunks. Then clients encrypt the stubs with NTRU. The encrypted stubs and trimmed packages will be uploaded to the server. Besides, clients can send a rekeying request to the server,

achieving the efficient rekeying for outsourced data.

Key Manager: To protect both predictable and unpredictable chunks, REEDBN de- ploys a key manager. It achieves a server-aided MLE by generating a key for a message based on both the message content and a system-wide secret. The key manager provides the system-wide secret to help REEDBN to protect against brute-force attacks.

4.2 Threat Model and Design Goals

In REEDBN, we assume the server and the key manager to be "honest-but-curious". They would follow our proposed protocol, but they are curious about user data and try to learn about them. In this paper, we consider two kinds of adversaries. The first are external adversaries who can compromise the server to access all ciphertexts out- sourced by users. The second are internal adversaries who can collude some valid cli- ents to obtain information about unauthorized data. Since REEDBN is a server-side encrypted deduplication storage system, there is no side channel attack.

REEDBN has two main design goals. The first is data confidentiality. Any kinds of adversaries cannot learn anything about data beyond their access scope, even if they compromise the server or collude other clients. The second

goal is the security and high-efficiency of rekeying. The revoked users cannot access unauthorized data after rekeying and the server also cannot learn anything about the data during rekeying.

Besides, the bandwidth overheads and computational efforts for clients during rekeying is negligible.

5. REEDBN Design

We introduce the NTRUReEncrypt [23] in REED. In REEDBN, a client just needs to upload a re-encryption key without downloading any data from the server during re-keying. REEDBN reduces bandwidth and calculation overheads of clients during re- keying. Hence, users can rekey their data even on their mobile terminals. Even if usersneed to rekey their data frequently, the communication overhead in the system is still little.

5.1 Main Idea

We observe that the only way to avoid the process of data transmission, encryption anddecryption during rekeying is to transform the original encrypted data stored on the server into a new ciphertext. However, if a client directly uploads both the original keyand a new key to the server and allow the server to generate the new ciphertext, the datawill be insecure. The untrusted server can obtain data and learn the keys of the encrypted data. Proxy re-encryption [14] is a good way to achieve the transformation of ciphertext in case that the server does not know the encryption key. It is used commonly in many data sharing schemes for cloud storage [24][25][11]. We apply the proxy re- encryption to the rekeying of encrypted deduplication storage. The purpose of proxy re-encryption in our scheme is to rekey rather than data sharing. Therefore, there is noneed to have a trusted proxy server in REEDBN since it is hard to find it in the real- world scenario. The re-encryption key can be generated directly by data owners them- selves. The proxy re-encryption scheme we choose is NTRUReEncrypt [23] because of its efficiency.

In order to use NTRUReEncrypt [23], the outsourced data must be encrypted by NTRU. If clients encrypt the whole file with NTRU, the performance of encryption willdegrade. This will also incur a lot of storage overheads because of the ciphertext expansion in NTRU. Therefore, we encrypt the stubs in REED with NTRU. Since the sizeof the stubs is small, our scheme will only cause limited overhead during encryption compared with REED.

5.2 Operations

In this subsection, we will give the detailed operations of file uploading, downloadingand rekeying in REEDBN.

File uploading: Suppose that a user wants to upload a file F. A client transforms Finto the stubs and trimmed packages either by the basic encryption or the enhanced encryption just like REED. In contrast to REED, the client encrypts the stubs using NTRU rather than AES in REEDBN. Therefore, the stubs need to be encoded into somepolynomials that NTRU can encrypt. We encode the binary bits in the stubs into the coefficients of polynomials. The client uses the algorithm $KeyGen()$ to output the(pk, sk) as described in section 2.3. Then the client encrypts the encoded stubs using the NTRU encryption algorithm to obtain the ciphertext $C_{stub} = Enc(pk, stub)$, where $Enc(\cdot)$ is the NTRU encryption algorithm. Finally, the client uploads the trimmed packages and C_{stub} to the server (see Figure 2).

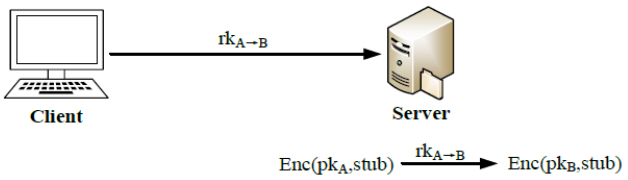


Fig. 2. File uploading in REEDBN

File downloading: The client downloads the trimmed packages and $Cstub$. The client uses the NTRU private key sk to decrypt $Cstub$ to obtain the $stub = Dec(sk, Enc(pk, stub))$, where $Dec(\cdot)$ is the NTRU decryption algorithm. The client can recover the entire file with the stubs and trimmed packages, and the user file can be reconstructed (see Figure 3).

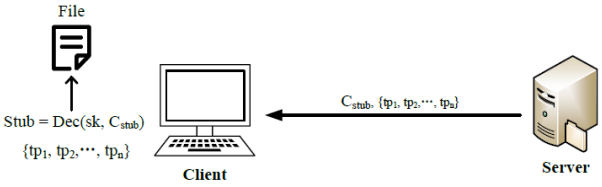


Fig. 3. File downloading in REEDBN

Rekeying: When a user needs to rekey their outsourced data, the client needs to generate a re-encryption key. Suppose that the stubs stored on the server are encrypted by the key pair (pkA, skA) and the user wants to update the key pair to (pkB, skB). The client should input the secret keys skA and skB to the re-encryption key generation algorithm $RekeyGen(\cdot)$ to compute the re-encryption key $rkA\to B = RekeyGen(skA, skB)$. The client uploads the $rkA\to B$ to the server. The server inputs the re-encryption key $rkA\to B$ and the stored ciphertext $Cstub = Enc(pkA, stub)$ to the re-encryption algorithm $ReEnc(\cdot)$. Finally, the server calculates the transformed ciphertext $Cstub' = ReEnc(rkA\to B, Cstub)$ as is shown in Figure 4. At this point, the encrypted stubs stored on the server can only be decrypted with the new key pair (pkB, skB), and the original key pair (pkA, skA) has been invalid. Therefore, the REEDBN can protect against the key compromise.

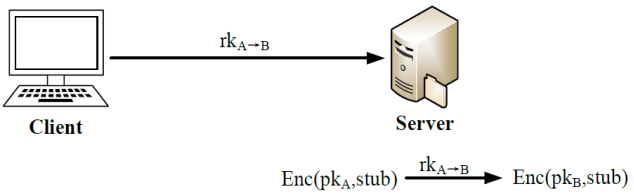


Fig. 4. Rekey in REEDBN

5.3 Dynamic Access Control

Similar to REED, REEDBN can also integrate CP-ABE to control the access privileges to user files. Every file in REEDBN corresponds to a NTRU key pair (pk, sk). The data owner creates an access policy for their outsourced files. The client then encrypts (pk, sk) using CP-ABE based on the access policy and uploads the encrypted key pair to the server. Only the users who satisfy the access policy can decrypt the encrypted key pair and decrypt the entire file.

When the data owner needs to change the access privilege of outsourced files, the client can generate a new NTRU key pair for the file and encrypt the new key pair using CP-ABE based on the new access policy. Then the client uploads the new CP-ABE ciphertext to the server. Only users who satisfy the new access policy can decrypt the new CP-ABE ciphertext at this point.

5.4 Analysis

REEDBN is designed to efficiently rekey for encrypted deduplication storage. In this subsection, we analyze two aspects of REEDBN. The first is data confidentiality and the second is the performance and security of rekeying.

We will use the following four propositions to illustrate the security and performance of REEDBN. In all these propositions, we assume that the cryptographic primitives used in REEDBN, such as MLE, OPRF, NTRU and CP-ABE, are secure.

Proposition 1. Neither the server nor the key manager in REEDBN can obtain any information about user data.

Proof. The clients interact with the key manager to get the server-aided MLE keys using the OPRF protocol. The hash of a file is blinded and the key manager cannot learn anything about the hash. Clients encrypt data before outsourcing them to the server. The server has neither the MLE key nor the NTRU key pair, so it cannot get any information for file plaintext. When users want to rekey their data, clients send $rk_{A\to B}$ to the server. During the process of ciphertext conversion, user data is always encrypted, and the server cannot learn any plaintext. Therefore, the honest but curious server and key manager cannot obtain any information about user data.

Proposition 2. Neither external adversaries nor internal adversaries can access any un-authorized data.

Proof. External adversaries can access all trimmed packages, encrypted stubs and the CP-ABE ciphertext of the NTRU key pair by compromising the server. Since they do not have any key, the encrypted data cannot be decrypted. Therefore, user data is securebecause of the security of MLE, NTRU and CP-ABE. Internal adversaries can colludewith valid clients, but their private keys do not satisfy the access policy define by the CP-ABE ciphertext. Hence, the CP-ABE ciphertext and the encrypted stubs cannot be decrypted. Because of the characteristic of CAONT, internal adversaries also cannot learn anything about the file without the stubs.

Proposition 3. The revoked users cannot access unauthorized data after rekeying.

Proof. Suppose that the stubs of a file are originally encrypted with k_1. The data ownerchanges the encryption key to k_2. The revoked k_1 cannot decrypt the rekeyed stubs be-cause of the security of NTRUReEncrypt. The k_2 is encrypted by CP-ABE and the revoked users' private keys do not satisfy the access policy set in ciphertext. Therefore, revoked k_1 is invalid and the revoked users cannot access k_2. Because of the nature of CAONT, the users will not be able to get data information without decrypting the encrypted stubs.

Proposition 4. REEDBN has negligible bandwidth overhead and computational effortsfor clients during rekeying.

Proof. We show the comparisons of overheads during rekeying in both REED and REEDBN in Table 1. During rekeying, clients do not need to download anything fromthe server in REEDBN and it just needs to upload a re-encryption key $rk_{A\to B}$, the band-width overhead during rekeying is very little. In contrast to REED, the encrypted stubsin REEDBN also do not need to be encrypted and decrypted by clients. The only thingneeds to be done by clients is to generate a re-encryption key. The computational effortfor clients is also negligible. Therefore, users can rekey their data on any device with limited computational power. These characteristics make REEDBN has a good application prospect.

Table 1. The comparisons of overhead during rekeying in REED and REEDBN

Deduplication scheme	Communication efforts	Computational efforts
REED	Twice the size of the stubs	Encryption and decryption of the stubs
REEDBN	The size of a re-encryptedkey (several bytes)	Generation of a re-encrypted key

6. Implementation

6.1 Implementation detail

We extend the open-source system REED to implement the prototype of REEDBN. The cryptographic operations in REEDBN are implemented by OpenSSL [21]. We introduce

NTRU in REED and it is implemented based on NTL [22], which is a portableC++ library. NTL is thread safe and it provides efficient algorithms and data structures for polynomials and big integers. It is worth mentioning that NTL is high-performance.Using it for arithmetical operations can greatly improve the efficiency of the system.
REEDBN consists of three separate C++ programs for clients, a server and a key manager. The setup for REEDBN is the same as REED. A client supports both fixed- size and variable-size chunking scheme and the stub size is 64 bytes for each chunk. We write all the stubs of a file into a stub file, which is encrypted by the NTRU encryption algorithm. REEDBN uses the OPRF protocol to blind the file fingerprint to preventthe key manager from learning data. The server can receive file data from multiple clients and perform cross-user deduplication on the trimmer packages.
In contrast to REED, the client needs to encode the binary bits in stub files into the coefficients of a polynomial since the unit of the NTRU encryption is polynomials. Asfor coding, we choose concatenating. First, the client reads a specific number of char- acters from the stub file and converts them into ASCII characters. As we know, an ASCII character consists of 8 bits. Since the parameter p limits the message space, weshift and concatenate every $p/8$ characters to form a p-bit integer. Finally, we get q p-bits integers and encode them as polynomial coefficients. In this way, we get the encoded messages. After the stub file is encoded into polynomials, the client encrypts them with NTRU.

6.2 Optimization

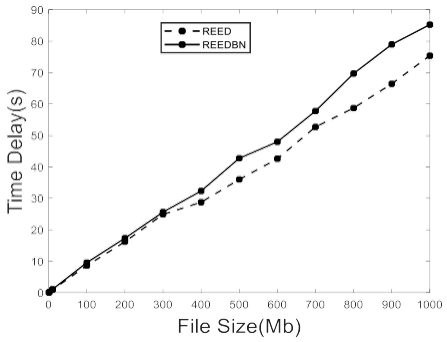
REEDBN leverages some optimization techniques, such as multi-thread and fast flourier transform for better NTRU encryption performance.
Multi-core and Multi-thread: The combination of multi-core and multi-thread al- lows threads to be scheduled to different kernels by the operating system, and complete parallel operations of a program. It achieves the goal of greatly improving the efficiency for program execution. When performing upload and download, we use multi-threading technology to handle multiple chunks in parallel for reducing transmission time. When encrypting and decrypting files, we also use this technique to parallelize the polynomial calculation and increase the computing efficiency.
Fast Fourier Transform (FFT): The basic idea of FFT [19] is to make full use of the symmetric and periodic properties of the exponential factors in the formula. It makes appropriate combinations to achieve the purpose of eliminating duplicate calculations, reducing multiplication operations. Our scheme involves amounts of polynomial multiplication operations in both encryption and decryption. Besides, we need to ensure that the ciphertext C is in the ring Rq and the plaintext M is in R/pR. Hence, modulo operation is required for all results, which also involves polynomial multiplication. In order to shorten the time consumed by above steps, we use FFT algorithm for optimization. It is worth mentioning that FFT can perfectly fit the multi-thread technology. We divide the polynomial coefficients into two parts according to parity and create two threads to perform these two part's DFT transformations [19] in parallel. Butterfly diagram [18] is also used to make sure the output of FFT is in order.

7. Evaluation

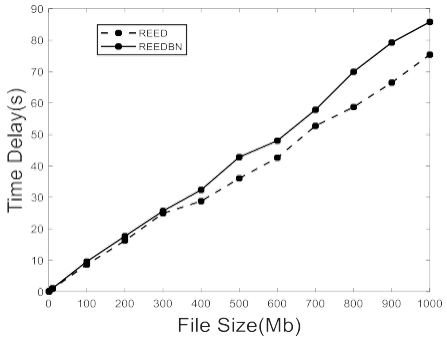
We evaluate REEDBN using machines equipped with a quad-core 2.7GHz Intel Core-i7-7500U, 5400RPM SATA hard disk, and 8GB RAM, and installed with 64-bit Ubuntu 16.04.12. In this section, our simulation results are the average of more than 10 runs. We use synthetic datasets and real-world datasets respectively during the evaluation. The synthetic dataset consists of many artificial files with random binary bits andthe real-world dataset is collected by the File system and Storage Lab (FSL) [20]. To evaluate the performance of REEDBN, especially compared to the enhanced encryption scheme in original REED, we mainly measure the speed of data encryption, upload andrekeying and the storage overhead of REEDBN and REED.

7.1 Evaluation on synthetic datasets

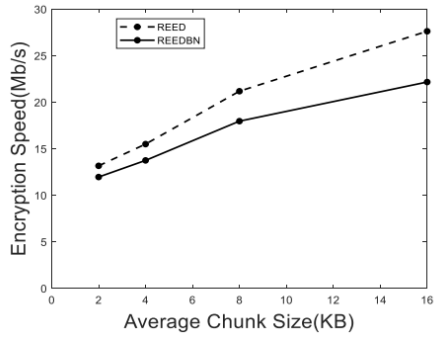
In this subsection, we compare the performance differences between REEDBN and REED in the processes of encrypting, uploading, and rekeying using synthetic dataset.Figure 5(a) shows the time delay in encryption of files with different size in REED andREEDBN. The encryption time consists of the time that transforming file data into theencrypted stubs and trimmed packages. It can be seen that the speed of encryption be- tween these two schemes is close. The speed of encryption in REED is 10.5% higher than REEDBN. This difference mainly comes from the encryption algorithm used in these two schemes. We use NTRU to encrypt the stubs in REEDBN, which is a little bit slower than AES. In Figure 5(b), the upload time of REED and REEDBN is also close. The speed of upload in REED is 10.8% higher than REEDBN. The upload timedenotes the delay of sending all data to the server. Due to the ciphertext expansion problem in NTRU, the amount of data that clients need to upload in REEDBN is morethan that in REED. Since the size of the stub is relatively small, the extra bandwidth overhead in REEDBN is little. Therefore, REEDBN incurs a little bit more upload timeoverhead compared with REED.
Figure 5(c) shows the speeds of encryption under REED and REEDBN versus the average chunk size. As the average block size increased, the encryption performance of both systems improved. This is because the larger the average block size, the fewer blocks need to be processed. For REEDBN, the amount of data encrypted by NTRU isalso reduced, thereby the encryption overhead of the system is further reduced.



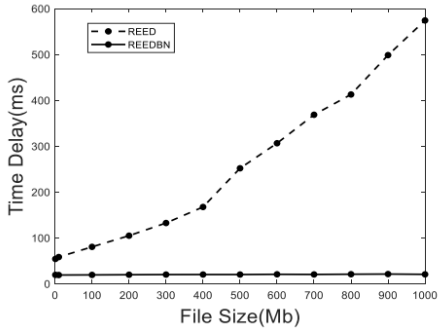
(a) Encryption performance



(b) Upload delay



(c) Varying chunk size



(d) Rekey delay

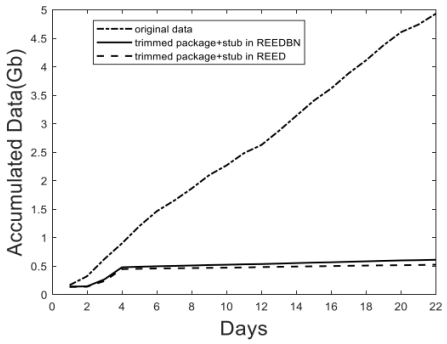
Fig. 5. Performance on synthetic datasets

The most obvious gap between REEDBN and REED is the speed of rekeying, which is also the motivation of this paper. The differences between REEDBN and REED in rekeying focus on the communicational overhead in the system and computational efforts for clients. Figure 5(d) shows the rekeying performance for files with different sizes both in REED and REEDBN. We measure the performance of active revocation scheme in REED. It can be seen that the time delay of rekeying in REED increases with the file size get larger, while the time delay of rekeying in REEDBN is stable and does not change with the increase of file size. This is because the encrypted stubs need to be downloaded, re-encrypted and uploaded to the server in REED. The larger files will be split into larger stubs. Hence, the communicational and computational effortsin rekeying will be increased. As the file size increases, so does the size of the stub, which multiply the data transfer and computational overhead for clients during rekeying in REED. Nevertheless, REEDBN outsources the expensive ciphertext conversion process to the server which has strong computing power. Regardless of the file size, the computational overhead for the clientis only the generation of a re-encrypted key, and the communicational overhead in system is onlythe size (several bytes) of the re-encrypted key.

7.2 Evaluation on real-world datasets

In this subsection, we evaluate the storage cost and encryption performance of REEDBN on a real-world dataset. The real-world dataset we consider is FSL [20].

Figure 6(a) shows the storage cost in REEDBN. Although REEDBN only applies deduplication to trimmed packages and has ciphertext expansion because of NTRU, it still has a good storage performance. We compare the size of the original data before encryption and deduplication with the size of the total of trimmed packages and encrypted stubs stored on the server both in REED and REEDBN. We can see that the data storage overhead decreases significantly after deduplication. After 22 days, the deduplication ratio in REED and REEDBN are respectively 89.3% and 87.6%. The deduplication ratio is defined as the ratio of the duplicate data/original data. The ciphertext expansion in NTRU incurs a little extra storage overhead in REEDBN.



(a) Storage overhead

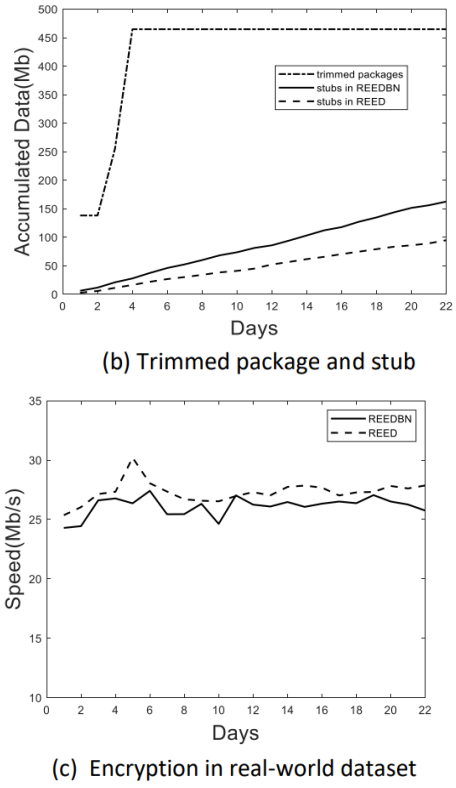


Fig. 6. Performance on real-world datasets

Figure 6(b) compares the size of the trimmed packages with the encrypted stubs both in REED and REEDBN. It can be seen that compared with the trimmed packages, the size of the stub data is relatively low. The storage overhead of encrypted stub in REEDBN is larger than REED because of the ciphertext expansion in NTRU. The ratio of stubs to total storage in REED is 17.5%, while this ratio in REEDBN is 25.9%.

We also measure the encryption performance in REEDBN over days. Figure 6(c) shows that REEDBN also has a good performance on real-world datasets. Its encryption performance is close to REED.

8. Conclusion

In this paper, we present REEDBN, which is a new rekeying scheme based on NTRU in encrypted deduplication storage. It takes advantage of NTRUReEncrypt to significantly mitigate the computational efforts for clients and the bandwidth cost in the system during rekeying. Clients just need to compute a re-encryption key and upload it to the server for rekeying their outsourced data without downloading any data from the server. The complicated process of re-encryption is done on the server side. We implement a REEDBN prototype and evaluate its performance. The results show that REEDBN indeed reduces both the bandwidth overhead in the system and computational efforts for clients.

References

[1] Armknecht, F., Bohli, J.-M., Karame G. O., Youssef, F.: Transparent data deduplication inthe cloud. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, CCS 2015, pp. 886-900. ACM, Denver, USA (2015)

[2] Wallace, G., Douglis, F., Qian, H., Shilane, P., Smaldone S., Chamness M., and Hsu W.: Characteristics of backup workloads in production systems. In: Proceedings of the 10th Conference on File and Storage Technologies, FAST 2012, pp. 1-16. USENIX, San Jose, Cali- fornia, USA. (2012)

[3] Douceur, J. R., Adya, A., Bolosky, W. J., Simon, D., Theimer, M.: Reclaiming space from duplicate files in a serverless distributed file system. 22nd International Conference on Distributed Computing Systems, ICDCS 2002, pp. 617-624. IEEE, Vienna, Austria (2002)

[4] Bellare, M., Keelveedh, S., Ristenpart, T.: DupLESS: Server-aided encryption for deduplicated storage. 22nd USENIX Security Symposium (USENIX Security 2013). USENIX, Washington, USA, 2013: pp. 179-194 (2013)

[5] Zhou, Y., Feng, D., Xia W., Fu M., Huang F., Zhang Y., Li C.: SecDep: A user-aware efficient fine-grained secure deduplication scheme with multi-level key management. In: Proceedings of IEEE MSST, 2015, pp. 1–14. IEEE, Santa Clara, California, USA (2015)

[6] Li J., Li, Y. K., Chen, X., Lee, P. P. C., Lou W.: A hybrid cloud approach for secure author ized deduplication. IEEE Transactions on Parallel and Distributed Systems, 2015, 26(5): pp.1206-1216. (2015)

[7] Li, M, Qin, C, LEE, P.P.C.: CDStore: toward reliable, secure, and cost-efficient cloud storage via convergent dispersal. 2015 USENIX Annual Technical Conference. USENIX, SantaClara, CA, USA, pp. 111-124. (2015)

[8] Liu, J., Asokan, N., Pinkas, B.: Secure deduplication of encrypted data without additional independent servers. The 22nd ACM SIGSAC Conference on Computer and Communications Security, CCS 2015, ACM, Denver, USA, 2015: pp. 874-885. (2015)

[9] Xu, J., Chang, E. C., Zhou, J.: Weak leakage-resilient client-side deduplication of encrypteddata in cloud storage. 8th ACM SIGSAC Symposium on Information, Computer and Communications Security. Hangzhou: ACM, 2013: pp. 195-206. (2013)

[10] Qin, C., Li, J., Lee, P.: The design and implementation of a rekeying-aware encrypted deduplication storage system. ACM Transactions on Storage, 2017, 13(1): 9:1-9:30. (2017)

[11] Lu, Y., Li, J.: A pairing-free certificate-based proxy re-encryption scheme for secure data sharing in public clouds. Future Generation Computer Systems, vol. 62, pp. 140–147. (2016)

[12] Bellare, M., Keelveedhi, S., Ristenpart, T.: Message-locked encryption and secure dedupli cation. In: Proceedings of EUROCRYPT, Springer, Athens, 2013, pp. 296–312 (2013)

[13] Hoffstein, J., Pipher, J., Silverman, J. H.: NTRU: A ring-based public key cryptosystem. InAlgorithmic number theory, pp. 267–288. Springer, 1998. (1998)

[14] Blaze, M., Bleumer, G., Strauss, M.: Divertible protocols and atomic proxy cryptography. In: Proceedings of European Cryptology Conference Springer, EUROCRYPT 1998. Springer, Espoo, Finland. pp. 127–144. (1998)

[15] Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-Policy attribute-based encryption. Proceedings of 2007 IEEE Symposium on Security and Privacy, pp. 321 -334. IEEE, Berkeley,USA, 2007. (2007)
[16] Stehl´e , D., Steinfeld, R.: Making NTRU as secure as worst-case problems over ideal lattices. In: Proceedings of European Cryptology Conference Springer, EUROCRYPT 2011, pp. 27–47. Springer, Tallinn, Estonia. (2011)
[17] Bailey, D. V., Coffin, D., Elbirt, A., Silverman, J. H., Woodbury, A. D.: NTRU in con- strained devices. In Cryptographic Hardware and Embedded Systems, CHES 2001, pp. 262–272. Springer, Paris, France. (2001)
[18] Chide, N., Deshmukh, S., P.B.Borole, P.: Implementation of OFDM System using IFFT andFFT. International Journal of Engineering Research and Applications, IJERA 2013, Vol. 3, pp. 2009-2014, Jan. –Feb. 2013. (2013).
[19] Hermans, J., Vercauteren, F., Preneel, B.: Speed records for NTRU. In Topics in Cryptology-CT-RSA 2010, pp. 73–88. Springer, San Francisco, CA, USA. (2010)
[20] "FSL traces and snapshots public archive," http://tracer.filesystems.org/, 2014
[21] "OpenSSL," https://www.openssl.org.
[22] "NTL," https://www.shoup.net/ntl/.
[23] Nuñez, D., Agudo, I., Lopez, J.: NTRUReEncrypt: An Efficient Proxy Re-Encryption Scheme Based on NTRU. In Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '15, ACM, New York, NY, USA, pp.179–189. (2015)
[24] Xu, P., Xu, J., Wang, Wei, J. Hai, Susilo, W., Zou, D.: Generally hybrid proxy re-encryption: A secure data sharing among cryptographic clouds. In Proceedings of ACM Symposium on Information, Computer and Communications Security, ASIACCS 2016, 2016, pp. 913-918, May 30-June 3. ACM, Xi'an China (2016).
[25] Huang, Q., Yang, Y., Fu, J.: PRECISE: Identity-based private data sharing with conditional proxy re-encryption in online social networks. Future Generation Computer Systems, vol. 86, pp. 1523–153 (2018)