Efficient Key-Aggregate Proxy Re-Encryption for Secure Data Sharing in Clouds

1st Wei-Hao Chen
Computer Science and Engineering
National Sun Yat-sen University
Kaohsiung, Taiwan, ROC
ferrans.chen@gmail.com

2nd Chun-I Fan*
Information Security Research Center
National Sun Yat-sen University
Kaohsiung, Taiwan, ROC
cifan@mail.cse.nsysu.edu.tw
(*The corresponding author)

3rd Yi-Fan Tseng
Computer Science and Engineering
National Sun Yat-sen University
Kaohsiung, Taiwan, ROC
yftseng1989@gmail.com

***Abstract*—Cloud computing undoubtedly is the most unparalleled technique in rapidly developing industries. Protecting sensitive files stored in the clouds from being accessed by malicious attackers is essential to the success of the clouds. In proxy re-encryption schemes, users delegate their encrypted files to other users by using re-encryption keys, which elegantly transfers the users' burden to the cloud servers. Moreover, one can adopt conditional proxy re-encryption schemes to employ their access control policy on the files to be shared. However, we recognize that the size of re-encryption keys will grow linearly with the number of the condition values, which may be impractical in low computational devices. In this paper, we combine a key-aggregate approach and a proxy re-encryption scheme into a key-aggregate proxy re-encryption scheme. It is worth mentioning that the proposed scheme is the first key-aggregate proxy re-encryption scheme. As a side note, the size of re-encryption keys is constant.**

***Index Terms*—Cloud Computing, Proxy Re-Encryption, Key-Aggregate Cryptosystem, Access Control**

I. Introduction

In recent years, cloud computing has grown from a small concept to a rapidly growing part of IT industries. It enables numerous people to share information without geographical restrictions. Therefore, protecting sensitive files stored in the clouds from being tampered by malicious attackers is essential to the success of the clouds. Nowadays, data security has become a critical issue in various kinds of applications. Users may prefer storing their files in an encrypted manner and delegating decryption rights efficiently. In order to protect the files stored in the clouds, the owners can encrypt the files by using their keys before uploading the files to the clouds. Still, a user needs to be online to share her encrypted files because she needs to send her keys to her friends. It is extremely inefficient because of the heavy overhead on the user. Fortunately, proxy re-encryption(PRE) schemes [1], [2], [3], [4] enable users to share their encrypted files with other users by using re-encryption keys. For instance, Alice encrypts her files and uploads these files to the clouds. Then Alice generates a re-encryption key and sends it to the cloud. The cloud operator then re-encrypts Alice's encrypted files into Bob's encrypted files whenever Bob downloads Alice's files. The scenario of PRE in clouds is depicted in FIGURE 1. PRE schemes elegantly transfer the users' burden to the clouds. Nonetheless, the existing PRE schemes should face the following problems. First, the re-keys may be abused by the delegatees and the proxy. For instance, Bob can conspire with cloud administrators to decrypt all Alice's ciphertexts while Bob is not authorized to access these encrypted files. Alice may only allow the proxy to process her partial ciphertexts. Second, most existing PRE algorithms do not efficiently support flexible delegation. Consider that Alice only allows Bob to access the files containing the keyword "Customer". Obviously, the existing PREs are difficult to have this flexibility. The cloud server simply follows the access control policies indicated by the delegator, which is impractical in diverse applications. To deal with this issue, Weng et al. [5] introduced conditional proxy re-encryption (C-PRE), where ciphertexts containing the certain condition value can be re-encrypted. In C-PREs [5], [6], [7], [8], a data owner generates both encrypted files and re-encryption keys, or condition keys, with certain condition values. For instance, Alice selects a value w. Meanwhile, she creates a re-encryption key, or condition key, by using w and Bob's public key. Thus the decryption right is limited to the certain condition value w. However, we recognize that the number of re-encryption keys, or condition keys, will grow linearly with the number of condition values, which may be impractical in resource-constrained devices. In our proposed scheme, we integrate a PRE scheme [9] and a key-aggregate approach [10] into a key-aggregate proxy re-encryption scheme. The proposed algorithm handles performance issues in reducing the number of re-keys while achieving fine-grained access control on the encrypted files, simultaneously.

978-1-5386-5790-4/18/$31.00 ©2018 IEEE

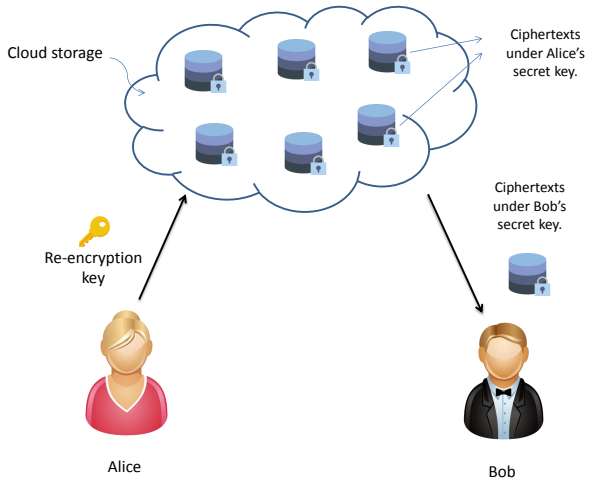

Fig. 1. Alice generates the re-key. The cloud will transform Alice's ciphertexts into Bob's ciphertexts when Bob downloads the files.

II. Preliminaries

A. Bilinear Mapping

Definition 1: Groups $(\mathbb{G}, \mathbb{G}_T)$ are two multiplicative cyclic groups of prime order p.
A bilinear mapping $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ has three properties.

- **Bilinearity:** $e(g^a, h^b) = e(g, h)^{ab}$ for any $a, b \in \mathbb{Z}_p$ and $g, h \in \mathbb{G}$.
- **Non-Degeneracy:** Whenever element $g, h \neq 1_{\mathbb{G}}$, $e(g, h) \neq 1$.
- **Computability:** One can use efficient algorithms to compute $e(g, h)$ for any element $g, h \in \mathbb{G}$.

III. Our Construction

A. Notations

The following table shows the meanings of the notations in our scheme.

TABLE I
The Notations

Notation	Meaning
$\mathbb{G}$	a cyclic multiplicative group
$\mathbb{G}_T$	a cyclic multiplicative group
e	a bilinear mapping
par	the public parameters
n	the number of total file types

B. The Proposed Scheme

Our scheme is composed of eight algorithms, i.e. Setup, KeyGen, ReKeyGen, Enc_2, Enc_1, ReEnc, Dec_2, and Dec_1, which will be defined in the following subsections.

1) Setup (λ): This algorithm outputs the parameters $par = \langle p, g, d, u, v, w, e, \mathbb{G}, \mathbb{G}_T, Z, H, F, l_1, l \rangle$ upon an input security parameter λ. Each component of par is set as follows. $\mathbb{G}, \mathbb{G}_T$ are two cyclic multiplicative groups of prime order p where $2^\lambda \leq p \leq 2^{\lambda+1}$ and g, d, u, v, w be generators of $\mathbb{G}$. Let $H : \mathbb{G} \times \mathbb{G} \to \mathbb{Z}_p^*, H_1 : \mathbb{G}_T \times \mathbb{G} \to \mathbb{G}^*$ be two hash functions and $Z = e(g, g)$.

2) KeyGen (i): The user i's keys are $pk_i = (g^{a_{i,1}}, g^{a_{i,2}}, \Delta_i)$ and $sk_i = (a_{i,1}, a_{i,2}, a_{i,3})$, where $a_{i,1}, a_{i,2}, a_{i,3} \in_R \mathbb{Z}_p$, $\Delta_i = \langle g_1, g_2, \ldots, g_n, g_{n+2}, \ldots, g_{2n} \rangle$, where $g_\rho = g^{a_{i,3}^\rho}$ for each $\rho \in \{1, ..., 2n\}/\{n+1\}$, and n represents the number of the file types specified by user i.

3) ReKeyGen (S, sk_i, pk_j): For the set S of user i's file types that are able to be re-encrypted, user i (a delegator) with $sk_i = (a_{i,1}, a_{i,2}, a_{i,3})$ can delegate decryption rights to a delegatee, say user j, with $pk_j = (g^{a_{j,1}}, g^{a_{j,2}}, \Delta_j)$ by computing $rk_{i \underset{S}{\to} j}$ as:

$$rk_{i \underset{S}{\to} j} = ((g^{a_{j,1}})^{1/a_{i,1}}), (\textstyle\prod_{\nu \in S} g_{n+1-\nu})^{a_{i,2}}) = (g^{a_{j,1}/a_{i,1}}, \textstyle\prod_{\nu \in S} g_{n+1-\nu}^{a_{i,2}}).$$

The example of ReKeyGen is shown below:

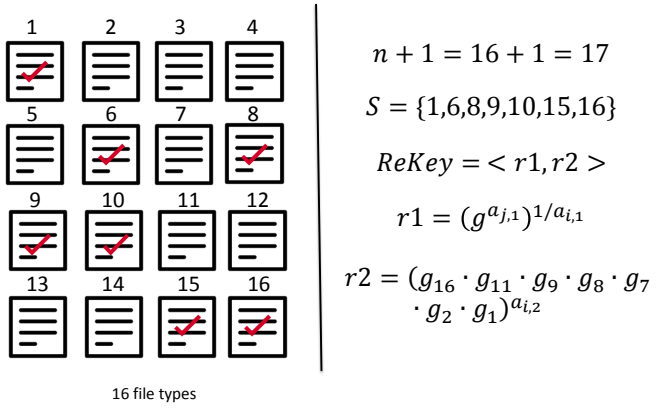

Fig. 2. In this figure, there are totally 16 different file types. In fact, each index of file type represents an individual condition value. Assuming that user i wants to delegate decryption rights of files types {1, 6, 8, 9, 10, 15, 16} to user j, she computes a constant-size re-encryption key with user j's public key.

4) Enc_2 (pk_i, m): Given $m \in \mathbb{G}$ and i's public key $pk_i = (g^{a_{i,1}}, g^{a_{i,2}}, \Delta_i)$, where m's type is $\rho \in \{1, ..., n\}$, select $t, k, r, \eta \in_R \mathbb{Z}_p$ and compute:

$$C = (k, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9)$$
$$= (k, d^r, g^{a_{i,1} r}, g^t, (g^{a_{i,2}} \cdot g_\rho)^t, H_1(K, d^r) \oplus m \oplus H_1$$
$$(e(g_1, g_n)^t, d^r), (u^h \cdot v^k \cdot w)^r, (u^{h'} \cdot v^k \cdot w)^r, g^\eta, H_1(K, g^\eta))$$

where $h = H(c_1, c_5), h' = H(c_1, H_1(K, c_1) \oplus m), K = Z^r$. Finally, this algorithm outputs the ciphertext:

$$C = (k, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9)$$

We obtain the validity of the ciphertext by checking the following formulas:

$$e(c_1, u^h \cdot v^k \cdot w) = e(c_6, d) \quad \text{(Eq. 1)}$$

$$e(c_1, g^{a_1}) = e(c_2, d) \quad \text{(Eq. 2)}$$

$$e(c_3, g^{a_{i,2}} \cdot g_\rho) = e(c_4, g) \quad \text{(Eq. 3)}$$

5) Enc$_1$ (pk_j, m): Given $m \in \mathbb{G}$ and j's public key $pk_j = (g^{a_{j,1}}, g^{a_{j,2}}, \Delta_j)$, select $k, r, \eta \in_R \mathbb{Z}_p$ and compute:

$$C' = (k, c'_1, c'_2, c'_3, c'_4, c'_5, c'_6) = (k, d^r, e(g^{a_{j,1}}, g)^r, H_1(K, d^r) \oplus m, (u^h \cdot v^k \cdot w)^r, g^\eta, H_1(K, g^\eta))$$

where $h = H(c'_1, c'_3)$, and $K = Z^r$. Finally, output the ciphertext $C' = (k, c'_1, c'_2, c'_3, c'_4, c'_5, c'_6)$.
Note that the validity of k, c'_1, c'_3, c'_4 can be checked by:

$$e(c'_1, u^h \cdot v^k \cdot w) = e(c'_4, d) \quad \text{(Eq. 4)}$$

We can verify c'_2 by Eq. 6. If true, it is accepted as a valid ciphertext.

6) ReEnc ($S, rk_{i \underset{S}{\rightarrow} j}, C$): In order to generate a first level ciphertext $C' = (k, c'_1, c'_2, c'_3, c'_4, c'_5, c'_6)$ of user j with $rk_{i \underset{S}{\rightarrow} j} = (r_1, r_2)$ and $C = (k, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9)$ of user i, the proxy first checks if C's type $\rho \in S$. Then it checks Eq. 1, Eq. 2, and Eq. 3. If true, it performs the following steps; otherwise, outputs $\perp$.

1) Set $c'_1 = c_1$, $c'_4 = c_7$, $c'_5 = c_8$, $c'_6 = c_9$.
2) Compute $c'_2 = e(c_2, r_1)$.
3) Compute $c'_3 = c_5 \oplus H_1(\frac{e(\prod_{\nu \in S} g_{n+1-\nu}, c_4)}{e(r_2 \cdot \prod_{\nu \in S, \nu \neq \rho} g_{n+1-\nu+\rho}, c_3)}, c_1)$.
4) Send $C' = (k, c'_1, c'_2, c'_3, c'_4, c'_5, c'_6)$ to user j.

Note that, for ciphertext C, the verification of Eq. 1 and Eq. 2 can be alternately done by picking $d_1, d_2 \in_R \mathbb{Z}_p^*$ and testing if

$$e(c_1, (g^{a_1})^{d_1} \cdot (u^h \cdot v^k \cdot w)^{d_2}) = e(c_2^{d_1} \cdot c_6^{d_2}, d) \quad \text{(Eq. 5)}$$

7) Dec$_2$ (sk_i, C): In order to decrypt $C = (k, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9)$ of user i, one performs the following steps:

1) Check Eq. 3 and Eq. 5. If FALSE, output $\perp$.
2) With $sk_i = (a_{i,1}, a_{i,2}, a_{i,3})$, we compute $K = e(c_2, g)^{1/a_{i,1}}$. If $H_1(K, c_8) = c_9$ holds, then generate $m = H_1(K, c_1) \oplus c_5 \oplus H_1(e(c_3, g_{n+1}), c_1)$; otherwise, output $\perp$.

8) Dec$_1$ (sk_j, C'): In order to decrypt $C' = (k, c'_1, c'_2, c'_3, c'_4, c'_5, c'_6)$ of user j, one performs the following steps:

1) Check Eq. 4. If FALSE, output $\perp$.
2) With $sk_j = (a_{j,1}, a_{j,2}, a_{j,3})$, we can compute $K = (c'_2)^{1/a_{j,1}}$ then generate $m = H_1(K, c'_1) \oplus c'_3$ if the following formula holds:

$$H_1(K, c'_5) = c'_6 \quad \text{(Eq. 6)}$$

Otherwise, output $\perp$.

C. Correctness

1) We demonstrate the correctness of the re-encryption as below:

$$\begin{aligned}
c'_1 &= c_1 = d^r. \\
c'_2 &= e(c_2, r_1) \\
&= e(g^{a_{i,1} r}, g^{a_{j,1}/a_{i,1}}) \\
&= e(g, g)^{a_{j,1} r}. \\
c'_3 &= c_5 \oplus H_1(\tfrac{e(\prod_{\nu \in S} g_{n+1-\nu}, c_4)}{e(r_2 \cdot \prod_{\nu \in S, \nu \neq \rho} g_{n+1-\nu+\rho}, c_3)}, c_1) \\
&= c_5 \oplus H_1(\tfrac{e(\prod_{\nu \in S} g_{n+1-\nu}, (g^{a_{i,2}} \cdot g_\rho)^t)}{e(\prod_{\nu \in S} g_{n+1-\nu}^{a_{i,2}} \cdot \prod_{\nu \in S, \nu \neq \rho} g_{n+1-\nu+\rho}, g^t)}, c_1) \\
&= c_5 \oplus H_1(\tfrac{e(\prod_{\nu \in S} g_{n+1-\nu}, g_\rho^t)}{e(\prod_{\nu \in S, \nu \neq \rho} g_{n+1-\nu+\rho}, g^t)}, c_1) \\
&= c_5 \oplus H_1(\tfrac{e(\prod_{\nu \in S} g_{n+1-\nu+\rho}, g^t)}{e(\prod_{\nu \in S} g_{n+1-\nu+\rho}, g^t)/e(g_{n+1}, g^t)}, c_1) \\
&= c_5 \oplus H_1(e(g_{n+1}, g^t), c_1)) \\
&= H_1(K, c_1) \oplus m. \\
c'_4 &= c_7 \\
&= (u^{h'} \cdot v^k \cdot w)^r \\
&= (u^{H(c_1, H_1(K, c_1) \oplus m)} \cdot v^k \cdot w)^r. \\
c'_5 &= c_8 = g^\eta. \\
c'_6 &= c_9 = H_1(K, g^\eta).
\end{aligned}$$

2) The correctness of the decryption on a first level ciphertext is demonstrated below:

$$\begin{aligned}
&K = (c'_2)^{1/a_{j,1}} = e(g^{a_{j,1}}, g)^{r/a_{j,1}} = e(g, g)^r, \\
&H_1(K, c'_1) \oplus c'_3 = H_1(K, c'_1) \oplus H_1(K, c'_1) \oplus m \\
&= m.
\end{aligned}$$

where $sk_i = (a_{j,1}, a_{j,2}, a_{j,3})$.

3) The correctness of the decryption on a second level ciphertext is as follows:

$$\begin{aligned}
&K = e(c_2, g)^{1/a_{i,1}} = e(g^{a_{i,1} r}, g)^{1/a_{i,1}} = e(g, g)^r, \\
&H_1(K, c_1) \oplus c_5 \oplus H_1(e(c_3, g_{n+1}), c_1) = \\
&H_1(K, c_1) \oplus H_1(e(g_1, g_n)^t, c_1) \oplus c_5 = \\
&H_1(K, c_1) \oplus H_1(K, c_1) \oplus H_1(e(g_1, g_n)^t, c_1) \oplus \\
&H_1(e(g_1, g_n)^t, c_1) \oplus m = m.
\end{aligned}$$

where $sk_i = (a_{i,1}, a_{i,2}, a_{i,3})$.

IV. Comparison

In this chapter, we examine our scheme with the correlated C-PRE schemes [5], [6], [7], [8]. We focus on the comparison of the storage cost of the re-encryption keys and the computational cost of $ReKeyGen$. As stated in [11], [12], [13], [14], [15] we learn that $T_s \approx 29T_m, T_a \approx 0.12T_m$ and $T_h \approx 7.75T_m$ and that a modular multiplication uses 66 clock cycles in a 256-bit finite field [16]. TABLE II indicates that our scheme can efficiently support fine-grained access control on the files with constant-size re-encryption keys.

V. Conclusion

In this paper, we deal with the performance issues in reducing the number of re-encryption keys while achieving fine-grained access control on the encrypted files, simultaneously. Noteworthily, the proposed scheme is the first key-aggregate proxy re-encryption scheme. However, the standard proof of our scheme is still lacking. In order to achieve a more

TABLE II
THE COMPARISONS BETWEEN C-PREs AND OUR SCHEME.

	Weng [5]	Chu [6]	Fang [7]	Liang [8]	Ours
Fine-grained access control	Yes	Yes	Yes	Yes	Yes
ReKeyGen computational cost	$2nT_s + nT_h$ $\approx 4340nCs$	$2nT_s + 2nT_a$ $\approx 3844nCs$	$7nT_s + 3nT_a$ $\approx 13422nCs$	$\triangle$	$2T_s + (n-1)T_a$ $\approx 8n + 3820Cs$
Re-encryption keys Length	$n(\lvert\mathbb{G}\rvert + \lvert\mathbb{G}\rvert) = 512nbits$	$n(\lvert\mathbb{G}\rvert) = 256nbits$	$n(\lvert\mathbb{G}\rvert + 3\lvert\mathbb{G}\rvert) = 1024nbits$	$\triangle$	$\lvert\mathbb{G}\rvert + \lvert\mathbb{G}\rvert = 512bits$

- $|\mathbb{G}|$: the length of $\mathbb{G}$ element
- $|\mathbb{G}_T|$: the length of $\mathbb{G}_T$ element
- n : the number of total file types
- T_s : the cost of a scalar multiplication
- T_a : the cost of an addition
- T_h : the cost of a hash operation
- Cs : clock cycles
- $\triangle$: the cost is dependent on the cryptographic primitives

secure and practical level of our algorithm, we will prove and implement our scheme in the future work.

VI. ACKNOWLEDGEMENT

This work was under the support of the Information Security Research Center at National Sun Yat-sen University.

REFERENCES

[1] G. Ateniese, K. Benson, and S. Hohenberger, "Key-private proxy re-encryption," CT-RSA, Springer, 2009, pp. 279–294.
[2] R. Canetti and S. Hohenberger, "Chosen-ciphertext secure proxy re-encryption," Proceedings of the 14th ACM conference on Computer and communications security, ACM, 2007, pp. 185–194.
[3] R. H. Deng, J. Weng, S. Liu, and K. Chen, "Chosen-ciphertext secure proxy re-encryption without pairings," International Conference on Cryptology and Network Security, Springer, 2008, pp. 1–17.
[4] M. Green and G. Ateniese, "Identity-based proxy re-encryption," Applied Cryptography and Network Security, Springer, 2007, pp. 288–306.
[5] J. Weng, R. H. Deng, X. Ding, C. K. Chu, and J. Lai, "Conditional proxy re-encryption secure against chosen-ciphertext attack," Proceedings of the 4th International Symposium on Information, Computer, and Communications Security, ACM, 2009, pp. 322–332.
[6] C. K. Chu, J. Weng, S. S. Chow, J. Zhou, and R. H. Deng, "Conditional proxy broadcast re-encryption," ACISP, Springer, 2009, pp. 327–342.
[7] L. Fang, W. Susilo, and J. Wang, "Anonymous conditional proxy re-encryption without random oracle," International Conference on Provable Security, Springer, 2009, pp. 47–60.
[8] K. Liang, Z. Liu, X. Tan, D. S. Wong, and C. Tang, "A cca-secure identity-based conditional proxy re-encryption without random oracles," International Conference on Information Security and Cryptology, Springer, 2012, pp. 231–246.
[9] J. Weng, M. Chen, Y. Yang, R. H. Deng, K. Chen, and F. Bao, "Cca-secure unidirectional proxy re-encryption in the adaptive corruption model without random oracles," Science China Information Sciences, vol. 53, 2010, pp. 593–606.
[10] C. K. Chu, S. S. Chow, W. G. Tzeng, J. Zhou, and R. H. Deng, "Key-aggregate cryptosystem for scalable data sharing in cloud storage," IEEE transactions on parallel and distributed systems, vol. 25, 2014, pp. 468–477.
[11] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, "Handbook of applied cryptography," CRC Press, 2001.
[12] M. Scott, "Implementing cryptographic pairings," Proceedings of the First International Conference on Pairing-Based Cryptography, Springer-Verlag, 2007, pp. 177–196.
[13] N. Koblitz, A. Menezes, and S. Vanstone, "The state of elliptic curve cryptography," Des. Codes Cryptography, vol. 19, 2000, pp. 173–193.
[14] Y. C. Zhang, W. Liu, W. Lou, and Y. Fang, "Securing mobile ad hoc networks with certificateless public keys," IEEE transactions on dependable and secure computing, vol. 3, 2006, pp. 386–399.
[15] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson, "Performance comparison of the aes submissions," In Proceedings of the Second AES Candidate Conference, 1999, pp. 15–34.
[16] A. Mrabet, N. El Mrabet, R. Lashermes, J. B. Rigaud, B. Bouallegue, S. Mesnager, and M. Machhout, "A systolic hardware architectures of montgomery modular multiplication for public key cryptosystems.," IACR Cryptology ePrint Archive, 2016, pp. 487.