A Multi-hop and Distributed Proxy Re-encryption Scheme with Dynamic Re-sharing

[1]JUNTAO CAO, [1]XIN PEI, [2]XIAOCHUAN WU
[1]The Idol Group, Shanghai, CHINA
[2]School of Computer Science, Fudan University, Shanghai, CHINA

Abstract: In order to achieve delegated data sharing, a reliable proxy is required for both data storage and execution of the delegated authorization. The PRE scheme is a representative technique for delegating data sharing, which involves a single proxy to transform the encryption by reencrypting algorithm with an auth-key, without knowing any knowledge about the plaintext. However, most PRE schemes are performed in a centralized environment, which means the system will crash upon the proxy is off-work. In this paper, we optimize the PRE scheme from two aspects. Firstly, the proxy acting as the key path is decentralized in a thresholdbased network, which will provide continuous PRE service when any t out of N nodes work. Moreover, considering the flexible entry and exit mechanism of the decentralized nodes, this proposal presents a re-share algorithm to ensure N live nodes. Secondly, we adopt the multi-hop re-encryption strategy for transitivity of ciphertext, so that the data owner is released from re-encryption key generation task upon user requests, and the authorized delegatees are able to retransform the encryption to designated users by using its own secret key.

Keywords: data sharing, proxy re-encryption, KMS, reshare, multi-hop, transitivity

Received: April 25, 2021. Revised: July 15, 2022. Accepted: August 12, 2022. Published: September 13, 2022.

1. Introduction

Data sharing is the main way to give full play to the value of data itself. For small amount of data, the generator directly sends the data to the requester peer-to-peer. With the rapid growth of user data scale, the cost of local storage, data security and data services grow exponentially. Cloud storage provides a cost-efficient way for data delegation, and can deploy stable data services. However, data security turns out to be critical since most delegated data are in plaintext. The build-in access control and domain separation methods are major security strategies rather than data encryption. In recent years, several serious data leaking accidents are caused by improper security setting in the cloud, explaining that all sensitive data should be in fine-grained, ciphertext manner, and the data sharing via cloud storage should be authenticated by a safe key, which cannot be used to decrypt the original data. Blaze et al. [1] announced the idea of proxy re-encryption, where a proxy can convert a confidential message encrypted by using one party's public key into another one that can be decrypted by using another party's private key. The Umbral project [2] propose a threshold-based proxy re-encryption scheme following a key encapsulation mechanism (*KEM*) approach. The data owner can delegate decryption rights to any receiver for any ciphertext in intended grained, through a re-encryption process performed by a set of N semi-honest proxies. Under the threshold of (t, N), where at least t out of N of these proxies participate by performing re-encryption algorithm, and then the receiver is able to combine these independent segments and decrypt the original message using its private key.

In *PRE* schemes, the directionality and the transitivity are critical characters for applications. In a bi-directional *PRE* scheme, the re-encryption phase is reversible, which means the proxy can use the same re-encryption key to re-encrypt each ciphertext both from owner to receiver and receiver to owner with zero knowledge to data. In this case, both the owner and receiver must combine their secret keys to produce the re-encryption key. The bi-directionality seems to bridge a peer-to-peer data sharing relationship. On the other hand, a unidirectional *PRE* means the proxy delegation is one-way. The ciphertext can be re-encrypted from owner to receiver, but not the reverse. Thus, the construction of re-encryption key only requires the owner's secret key. As for the transitivity character, it represents the times for a ciphertext to be re-encrypted. Ciphertext that can be re-encrypted only once refers to single-hop *PRE* scheme, while multi-hop *PRE* scheme enables the unlimited-time transitivity of ciphertext re-encryption. The permutation of types for *PRE* characters allows a various of applications such as e-mail forwarding, authorization transfer, and data distribution. However, most *PRE* schemes require a centralized proxy to perform data storage and re-encryption, which means the proxy turns out to be the keypath of the system and has to handle all the delegations.

We improve the system soundness by implementing a decentralized proxy network, where each participant possesses a key-share of re-encryption key, and can execute the re-encrypt task separately. Meanwhile, our proposal also involves multi-hop *PRE* to release data owner from re-encryption key generation upon an authorized request, which frequently calls the owner's secret key and has to be executed locally at owner side. The distributed *PRE* scheme adopts secret sharing and multi-hop re-encryption to provide reliable and full delegation service. Moreover, this proposal achieves a reshare function to process the continuing node variations in decentralized environment.

2. Related Works

The prototype of safe *PRE* scheme is introduced by Mambo and Okamoto [3] by using the partial decryptions, without offering any extra security benefits for delegator's secret key. Their proposal aggregated the decrypt and re-encrypt into an atomic execution by taking the re-encryption key as input, during which the cipher-state original data is never revealed. Later, Blaze, Bleumer and Strauss proposed the ciphertext conversion algorithm [2] on the ElGamal

This work was supported by a grant from the Core Technology Research and Development Program of Lin-Gang.

cryptosystem, which requires taking secret keys of both parties in the construction procedure. In a semi-honest model, the proxy takes a re-encryption key(*RK*) generated by the delegator's private key and delegatee's public key to re-encrypt the ciphertext, which was originally encrypted using the delegator's public key, into a new ciphertext that can be decrypted by the delegatee's private key. The classical *PRE* procedure is shown in Figure 1. Thus, the delegator can share its fine-grained data by simple authentication via generating the re-encryption key, rather than to execute decryption each time or give out its private key.

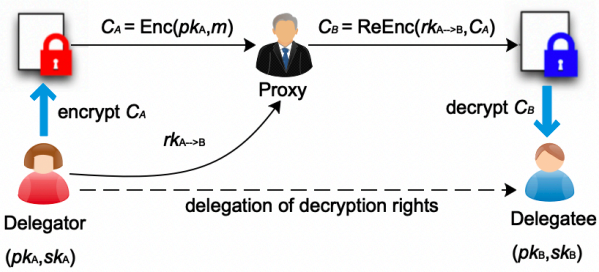

Fig. 1. The classical unidirectional *PRE* scheme

Above re-encryption scheme only supports single-hop type, so that the ciphertext is not transformable after re-encryption. In other words, if the delegatee intends to re-authenticate the ciphertext to another, it has to run decryption with the private key and then encrypt the original data with its public key to a new ciphertext. Ateniese and Fu [4] proposed the first unidirectional proxy re-encryption scheme, where the delegator can assign a proxy to authorize requestors on data access without revealing its private key. In 2009, Wang and Cao[5] proposed the multi-hop re-encryption scheme, enabling the decryption right of cipertext to be transferred multiple times. Their proposal was then enhanced by Cai and Liu [6] to *CCA*-secured. Based on these works, Luo et al. [7] proposed a unidirectional, multi-hop, identity-based proxy re-encryption scheme with *CPA*-security. Sun et al. suggested the identity-based *PRE* encryption schemes with the ability of ciphertext evolution [8,9,10]. However, the length of ciphertext will grow linearly with the times of updating. To solve this, Liang et al. [11] proposed a constant ciphertext size scheme, with the properties of collusion-resistant, bidirectional and multi-hop. To address the directionality problem, Weng et al. proposed a conditional *PRE* approach in which the proxy can re-encrypt ciphertexts for a user only if it satisfies the necessary conditions [12]. The unidirectional, multi-hop *PRE* scheme is shown in Figure 2.

To improve the reliability of secrets, the centralized proxy is distributed to a multi-proxy network, where each sub-proxy possesses a secret share and executes partial computation, so that the aggregation of multi-proxy outputs can meet the demand. Secret sharing is a fundamental primitive in cryptography, and the original secret sharing model was proposed by Shamir. In [13], a (k, n) threshold scheme was proposed. A secret data S can be divided into n share pieces $s_1, \ldots, s_n$ and S can be obtained through any k or more s_i pieces, and S cannot be obtained through any $k-1$or fewer s_i pieces.

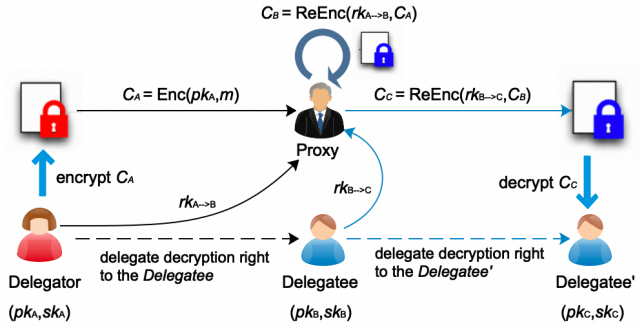

Fig. 2. The unidirectional, multi-hop *PRE* scheme

On the other hand, the nodes in proxy network should periodically prove the possession of secret shares. In the research of remote data checking, Ateniese et al. firstly propose the "challenge-response" mode [14]. For each checking request, the remote server has to calculate a corresponding response with local data as input. Sebe et al. propose a periodically scheme based on *PDP* [15]. The requestor randomly chooses several data blocks as part of the whole data to check. This reduces the overhead of server, but the probability of error recognition decreases from 100% to $1-(1-\rho)^c$ [16] or $1-(1-\rho)^{c \cdot s}$ [17], where c is number of the selected blocks in N rounds, and S is sector number in each block if divided. ρ is the average error probability of each block or sector.

3. The Construction of Our Proposal

Before the construction of the proposed distributed *PRE* scheme with multi-hop delegation, we introduce the secret *Re-share* algorithm.

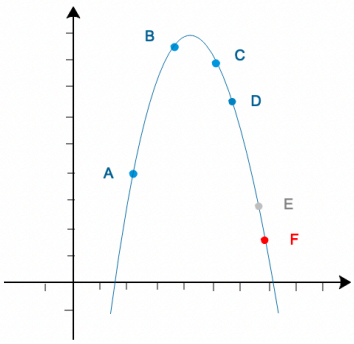

Fig. 3. The *Re-share* algorithm

The secret *Re-share* algorithm provides a mechanism for reliability of the distributed network. The shares are generated by a secret sharing scheme by taking the re-encryption key and the threshold as input. Then, the N shares are distributed to the corresponding nodes, who get the rewards from the contributed storage and computation. However, there are probabilities that several nodes exit or crash. In order to guarantee the retrievability of secret, a heartbeat mechanism of share possession proof is established, so that N live nodes are kept for most of the time and the *Re-share* algorithm will execute when any node fails. Moreover, the *Re-share* algorithm can generate any number of shares unless the amount of live nodes is less then t as defined in the

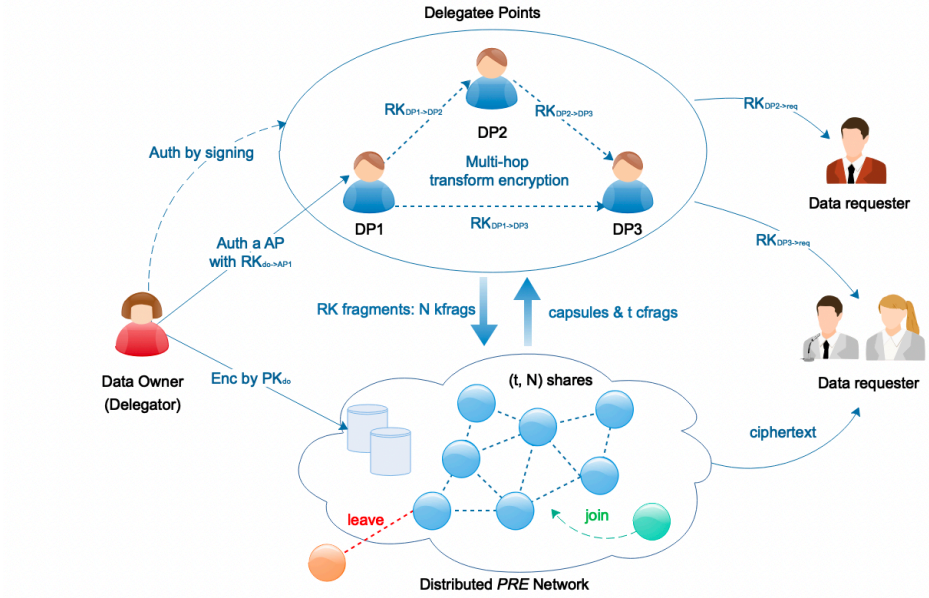

Fig. 4 The System architecture of the multi-hop *PRE* in distributed environment

threshold. Figure 3 illustrates a situation where a node possessing the share E exits from the network under threshold (3, 5). Then, the network executes the *Re-share* algorithm to calculate a new share F on the polynomial via Lagrange interpolation. Let F_x be a random number on $\mathbb{Z}_p$, $F_y = y_A l_A(x) + y_B l_B(x) + y_C l_C(x)$, where $l_A(x) = \frac{F_x - x_B}{x_A - x_B} \cdot \frac{F_x - x_C}{x_A - x_C}$, $l_B(x) = \frac{F_x - x_A}{x_B - x_A} \cdot \frac{F_x - x_C}{x_B - x_C}$, $l_C(x) = \frac{F_x - x_A}{x_C - x_A} \cdot \frac{F_x - x_B}{x_C - x_B}$, and the new share is $F = (F_x, F_y)$.

By adopting the *KEM* methods, the asymmetric encryption only encrypts the data key, which is used to encrypt the original data symmetrically. We use "*capsule*" to term the encryption of the data key to differ from the ciphertext of the original data. The ability to decrypt a capsule means an authentication to access the corresponding original data. Our proposal consists of four pair of functions, the key management pair *Key Generation* and *ReKey-Generation*, the enc/dec pair *Encrypt* and *Decrypt*, the re-enc/retrieve pair *re-encrypt* and *Retrieve*, the updating pair *Resharing* and *Transmit*.

Firstly, we setup the public parameters and define the hash families.

$Setup(\lambda)$: Determine a cyclic group $\mathbb{G}$ of prime order p, according to the security parameter λ. Let $g, U \in \mathbb{G}$ be generators. Let $H: \mathbb{G} \to \mathbb{Z}_p$, be hash functions that behave as random oracles. Let KDF: $\mathbb{G} \to \{0,1\}^l$ be a random oracle key derivation function, where l is the fixed length according to the security parameter λ. The global public parameters are represented by the tuple:

$$params = (\mathbb{G}, g, U, H, \text{KDF}) \qquad (1)$$

$KeyGen()$: Sample $a \in \mathbb{Z}_p$ uniformly at random as private key, compute g^a and output the keypair $(pk, sk) = (g^a, a)$.

In this model, we assume A as the data owner or delegator, B as the requestor or delegatee.

$Encrypt(pk_A)$: On input the public key pk_A, the encrypt algorithm first samples random $r, u \in \mathbb{Z}_p$ and computes $E = g^r$ and $V = g^u$. Computes the value $c = u + r \cdot H_2(E, V)$. The derived data key, used to encrypt the original data symmetrically, is computed as $DK = \text{KDF}({pk_A}^{r+u})$. The tuple (E, V, c) is called *capsule* and allows to derive again the data key *DK*. To examine the validity of the capsule, we can check if the following equation holds:

$$g^c \,?= V \cdot E^{H_2(E,V)} \qquad (2)$$

Then, the $Encrypt$ function runs a *ChaCha20* to encrypt the original data with *DK* into ciphertext *Cipher*. Finally, the encrypt algorithm outputs $(Cipher, capsule)$.

$RekeyGen(sk_A, pk_B, t, N)$: Firstly, pick a random $t-1$ degree polynomial in (3), where a is the secret generated in $keyGen$, and $a_1 \sim a_t$ are randomly chosen.

$$f(x) = a + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1} \qquad (3)$$

We can calculate $O_1 = f(x_1), \ldots, O_n = f(x_n)$, and then we generated n share group data: $(x_1, O_1), \ldots, (x_n, O_n)$, and S can be calculated by any t group data of (x_i, O_i). Note that the picked prime p should be bigger than both S and n, and data range is in $[0, p)$, and the values of $(x_1, O_1), \ldots, (x_n, O_n)$ are computed modulo p. If the secret S is long, it can be break into shorter blocks of m bits. Then, input A' secret key $sk_A = a$ and B' public key $pk_B = g^b$, a threshold (t, N), the $RekeyGen$ algorithm computes N fragments of the re-encryption key between A and B as follows:

(1) Sample a $r_A \in \mathbb{Z}_p$ at random, and compute $R_A = g^{r_A}$

(2) Compute $d = H_2(R_A, pk_B, (pk_B)^{r_A})$, where d is the result of a non-interactive Diffie-Hellman (*DH*) key exchange between B's keypair and the ephemeral key pair (r_A, R_A).

(3) Compute $D = H(pk_A, pk_B, {pk_B}^a)$

(4) Initialize set $kFragSet = \emptyset$ and repeat N times:

a. Sample random y, $id \in \mathbb{Z}_p$

b. Compute $s_x = H(id, D)$ and $Y = g^y$.

c. Compute $rk = f(s_x)$

d. Compute $U_1 = U^{rk}$.

e. Compute $z_1 = H(Y, id, pk_A, pk_B, U_1, R_A)$

f. Compute $z_2 = y - a \cdot z_1$

g. Obtain $kFrag$ as the tuple $(id, rk, R_A, U_1, z_1, z_2)$.

h. Add into $kFragSet$ as $kFragSet \cup \{kFrag\}$.

(5) Finally, output the set $kFragSet$.

This means *A*' grants its own permissions which are in form of encryptions under pk_A to *B*.

$ReEncrypt(kFrag, capsule)$: For each node, on receiving a $kFrag = (id, rk, R_A, U_1, Z_1, Z_2)$ and the $capsule = (E, V, s)$, it first checks the validity of the capsule and outputs $\perp$ if the check fails. Otherwise, it runs the re-encrypt algorithm to compute $E_1 = E^{rk}$ and $V_1 = V^{rk}$, and outputs the capsule fragment $cfrag = (E_1, V_1, id, R_A)$. Due to that the threshold requires *t* shares to retrieve the secret, the $ReEncrypt$ algorithm should appoint at least *t* nodes to calculate *t* pieces of $cfrag$s.

$Retrieve(sk_B, pk_A, \{cfrag_i\}_{i=1}^t)$: Taking *A*'s public key pk_A, *B*'s public key pk_B and corresponding *t* $cfrags$, being each of them $cFrag_i = (E_{1,i}, V_1, id_i, X_A)$ as input, the $Retrieve$ algorithm runs as following to aggregate them into $capsule'$, which can be decrypted by sk_B.

(1) Compute $D = H(pk_A, {pk_A}^b, pk_B)$, where ${pk_A}^b = {pk_B}^a$ by adopting *DH* protocol.

(2) Let $S = \{s_{x,i}\}_{i=1}^t$, where $s_{x,i} = H(id_i, D)$. For all $s_{x,i} \in S$, compute:

$$\lambda_{i,S} = \prod_{j=1, j\neq i}^{t} \frac{s_{x,j}}{s_{x,j} - s_{x,i}}$$

(3) Then, compute the values in $cfrag_i$:

$$E' = \prod_{i=1}^{t} \left(E_{1,i}\right)^{\lambda_{i,S}}, \qquad V' = \prod_{i=1}^{t} (V_{1,i})^{\lambda_{i,S}}$$

(4) Compute $d = H\left(R_A, {R_A}^b, pk_B\right)$, as the result of a non-interactive *DH* key exchange between *B*'s keypair and the ephemeral key pair(r_A, R_A). The value R_A is the same for all the $cfrags$ that are produced by re-encryptions using a $kFrag$ in the set of re-encryption key fragments $kFragSet$.

(5) Finally, output the symmetric key $DK = \text{KDF}((E' \cdot V')^d)$

$Decrypt(Cipher, DK)$: Decrypts the $Cipher$ using the *CHACHA*20 decryption function with key *DK*, which results in message *M* if decryption is correct, and $\perp$ otherwise.

The entire distributed *PRE* procedure is ended with the user decryption with its own secret key. However, this system might confront two reliability issues. One relates to the shares in the distributed *PRE* network, as discussed above, nodes could exit or crash any time, so that we build the *Reshare* algorithm to guarantee enough valid shares live. The other is the delegatee point, acting as the key-path role in this system, transforming the data access right to specified users according to the owner's delegation. Therefore, the chosen *PRE* algorithm must be multi-hop, otherwise, the owner has to generate the re-encryption key upon each user request or reveal its secret key to the delegatee, which is quite unsafe. Besides, the delegatee proxy should have online standbys to ensure continuous service, which is also achieve by the *Key Transform* algorithm.

$Reshare(\{kFrag_i\}_{i=1}^t)$: Under the nodes live checking mechanism, the $Reshare$ algorithm will generate a new $kFrag$ without updating the existing ones. When any node crashes, the system samples an $x \in \mathbb{Z}_q$ at random, and use Lagrange interpolation to calculate the corresponding $y = \sum_{j=0}^{k} y_j l_j(x)$, where $l_j(x) = \prod_{i=0, i\neq j}^{k} \frac{x - x_i}{x_j - x_i}$ on the given polynomial. Thus, the calculated point (x, y) turns out to be a new $kFrag$ assigned to a live node.

$Transmi\,t(\{kFrag_{A\to B,i}\}_{i=1}^t, sk_B, pk_C)$: To transmit an authorization from $A \to B$ to $B \to C$ requires $\{kFrag_{A\to B,i}\}_{i=1}^t$, sk_B , pk_C as input. Compared with the $RekeyGen$ algorithm, the difference is that the $Transmit$ algorithm achieves transitivity, which enables the delegator to authorize any delegatee without using his own secret key sk_A.

The proposal *PRE* scheme has the following properties:

Unidirectionality. The delegation from the data owner to the delegatee is unidirectional, while the reversal requires another setup.

Non-interactivity. The adopted Diffie-Hellman protocol between the delegator and delegatee's keypair makes a shared secret, enabling the re-encryption key generation of the scheme non-interactive.

Transitivity. The encryption can be re-delegated more than once by a series of transform keys.

Multi-hop. The designated delegatees are allowed to be the delegators that can re-delegate the encryption under the owner's authorizations, chaining the transformations.

Collusion safety. During the re-encryption, none of the delegatees is able to collude with the distributed proxy nodes that hold the transform key to recover the private key of the delegator.

4. Experiments of Our Proposal

In this section discuss, we evaluate the performances of re-encryption key generation algorithm, capsule re-encryption algorithm, data key retrieval algorithm and $kFrag$ reshare algorithm. The results of all tests are the average of 20 times. Figure 5 illustrates a complete text process on above algorithms by a given threshold (3, 5).

Figure 6 illustrates the time cost of $kFragset$ generation. There are 8 tuples of threshold included, where any *t* out of *N* pieces can recover the re-encryption key. The growing rate is decided by both *t* and *N*, due to that the *t* factor decides the order of the polynomial, and *N* determines the total shares.

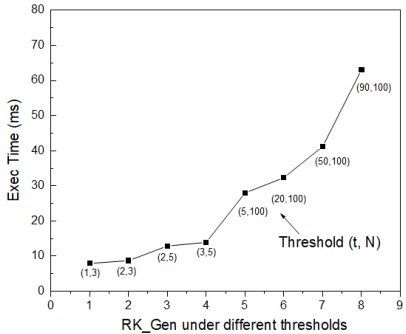

Fig. 6. Re-encryption key shares generation time

Figure 7 shows the computation cost in re-encryption. Since out proposal adopts *KEM* strategy, the complexity only relates to *t* in the threshold, with no regards to *N* or the size of original file. However, this proposal ignores the node crash probability during the re-encryption procedure, and several

backup nodes possessing their $kFrag$ can be arranged to run re-encryption algorithm to enhance the correctness.

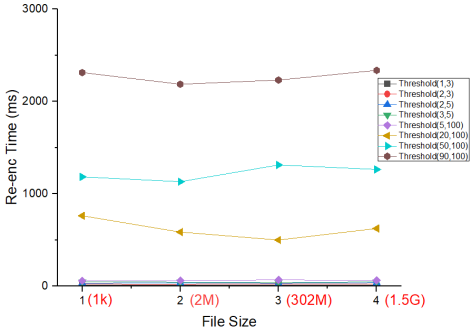

Fig. 7. Re-encryption time of an increasing size of files

Figure 8 shows the cost of retrieving a data key with different thresholds. This procedure can be regarded as the reverse execution of data key encryption. We compare 8 tuples of thresholds, similarly, the cost only relates to t numbers of $cfrag$, corresponding to the re-encrypted $kFrags$.

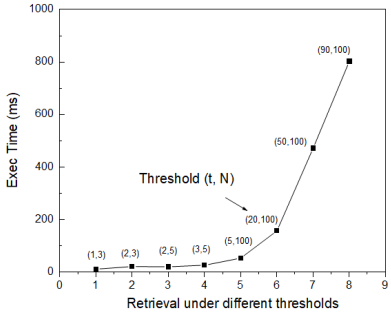

Fig. 8. Data key retrieval time

We also test the cost of *Reshare* algorithm. Given a fixed polynomial, to sample an x at random and calculate corresponding y is quite easy. But taking any t points to recover the polynomial by using Lagrange interpolation is costly. For a threshold with t = 50, the calculation time

1. Key generation phase:

INPUT
{ "type": "ECC", "account": "xinpei", "role": "sender"/"receiver"
}
OUTPUT
{
"owner_enc_pk": "03aea98c72d3d0c2fca73ffa9e4c08c89931f36c9c8a7d23ad9761a1fe4d925fe5",
"owner_enc_sk": "ccdcfbfb7fe36467b31678c8c9a7b659d7194784817cd7b856eb4652f5ea528a",
"owner_signing_key": "8a5c5df44c9b1c2f0385783d658a9f84ced7bd5f041a5c027728611292dc7b96",
"owner_verifying_key": "0306114c67af0ee020f39e09f9839983331df6d096effaed8ba5691d1d3d9310b1"
}

{
"rcv_enc_pk": "039bd8e354b7c6e4d5c38ea6bca263a337c3b659243c5a7f597bb30a726dd8ff58",
" rcv_enc_sk": "70688498bb9b608288c606b69c31b1c52ae060c2bf1db05e209145df5f039fda"
}

2. Encrypt phase:

INPUT
{ "plaintext": "aGFoYWhh",
"public_key": "03aea98c72d3d0c2fca73ffa9e4c08c89931f36c9c8a7d23ad9761a1fe4d925fe5"
}
OUTPUT
{
"aescipherhtext": "QmXPpuq5yyWDdgQpeWzDzVmA9dfbGDEQjTQJfDzaWq18j4",
"capsule": "QmdT3WVsrP4MvB48N7KmcseiEdcZaqYiqf9piTTE3gkuJL",
"ciphertext": "QmZPE6MahiDxUkbkYLWtEFATLMwrdokxFqy89vug7nvzi9"
}

3. Generate the distributed re-encryption key shares:

INPUT
{
"delegatekey": "ccdcfbfb7fe36467b31678c8c9a7b659d7194784817cd7b856eb4652f5ea528a",
"signersk":"8a5c5df44c9b1c2f0385783d658a9f84ced7bd5f041a5c027728611292dc7b96",
"publickey": "039bd8e354b7c6e4d5c38ea6bca263a337c3b659243c5a7f597bb30a726dd8ff58",
"threshold": (3, 5)
}
OUTPUT
[
"QmNZ2RYqkPNu6bGiJUdFzu6CvzoQWH4YJ7ABFZeYcAbbtu",
"QmUEWPFUZZCZCnV1616asBXrJQRDAsnCkrYgpdFq4ZuJQ8",
"QmQFHCVfGKGTBpFneDxXE12nTMMNqRTupoeBdtFNy3178N",
"QmP3VpRvwy4W7jYxXy8f5m3FS4KzVMDVnVfDktiG4fZu3N",
"Qma6hemUex7Wd6kooUPtogyQMYk3B7Z4hfaRdkG8kjywmC"
]

4. Distributed re-encrypt for each node:

INPUT
{
"threshold":3,
"capsule": "QmdT3WVsrP4MvB48N7KmcseiEdcZaqYiqf9piTTE3gkuJL",
"addresses":[
"QmNZ2RYqkPNu6bGiJUdFzu6CvzoQWH4YJ7ABFZeYcAbbtu",
"QmUEWPFUZZCZCnV1616asBXrJQRDAsnCkrYgpdFq4ZuJQ8",
"QmQFHCVfGKGTBpFneDxXE12nTMMNqRTupoeBdtFNy3178N"],
"delegating": "03aea98c72d3d0c2fca73ffa9e4c08c89931f36c9c8a7d23ad9761a1fe4d925fe5",
"receiving":"039bd8e354b7c6e4d5c38ea6bca263a337c3b659243c5a7f597bb30a726dd8ff58",
"verifying":"0306114c67af0ee020f39e09f9839983331df6d096effaed8ba5691d1d3d9310b1"
}

OUTPUT
{
"caddrs": [
"QmPjo4AqwxNJ51RAKWNqEbWT4UrgF1nW3GNVUmqcomjAqb",
"QmSzfiiMgqCYvaKfQ39Wbmi444TJmyE7sfEmVb9q6pTVsn",
"QmRHD7Uq2Kytf6sf8Pcsdc1xotAM5bLb3coejHjdqatJJD"
],
"capsule": "QmV6CxPQ6sGY8ZALQ4Bhvi749GbW81R1GBj5yphPmuZVxH"
}

5. Fetch the re-encrypted cipher shares:

INPUT
{
"capsule": "QmV6CxPQ6sGY8ZALQ4Bhvi749GbW81R1GBj5yphPmuZVxH",
"caddrs": [
"QmPjo4AqwxNJ51RAKWNqEbWT4UrgF1nW3GNVUmqcomjAqb",
"QmSzfiiMgqCYvaKfQ39Wbmi444TJmyE7sfEmVb9q6pTVsn",
"QmRHD7Uq2Kytf6sf8Pcsdc1xotAM5bLb3coejHjdqatJJD"
]
}
OUTPUT
{
"capsule": "QmbQb4HC3KNkvD7GQqxN8ee2FUjPP6q4HUWcUFBuG1LPEG"
}

6. Decrypt phase:

INPUT
{
"ciphertext": "QmZPE6MahiDxUkbkYLWtEFATLMwrdokxFqy89vug7nvzi9",
"aescipherhtext":"QmXPpuq5yyWDdgQpeWzDzVmA9dfbGDEQjTQJfDzaWq18j4",
"decryptkey":"70688498bb9b608288c606b69c31b1c52ae060c2bf1db05e209145df5f039fda",
"capsule":"QmbQb4HC3KNkvD7GQqxN8ee2FUjPP6q4HUWcUFBuG1LPEG"
}

OUTPUT
{"plaintext": "aGFoYWhh"}

7. Reshare the secret in PGF256Interpolator:

INPUT:
{
"msg": b'TndT3W';
"threshold": (3, 5)
}
OUTPUT
[
{"coeffs": [84, 119, 30],
"points": [(235, 102), (92, 70), (244, 233), (36, 174), (34, 4)]},
{"coeffs": [110, 119, 249],
"points": [(79, 191), (66, 78), (23, 168), (21, 143), (11, 118)]},
{"coeffs": [100, 5, 130],
"points": [(138, 25), (204, 17), (179, 246), (240, 149), (176, 69)]},
{"coeffs": [84, 13, 146];
"points": [(89, 216), (204, 81), (180, 7), (142, 141), (59, 65)]},
{"coeffs": [51, 223, 71];
"points": [(210, 223), (64, 75), (19, 141), (195, 195), (225, 158)]},
{"coeffs": [87, 171, 227];
"points: [(145, 131), (229, 121), (136, 145), (154, 246), (111, 83)]}
]

8. Multi-hop delegation and verification:

INPUT
{
"plaintext": { bytes: [113, 119, 101, 114, 49, 50, 51, 52, 53, 54, 55, 56, 57, 48, 49, 50, 51,"…]};
}
OUTPUT
{
"Original Encryption": {EncryptedOnceValue { x: [78, 222, 104, 185, 213, 158, 204, 209, 51, 191, 2…],
Internal_key { x: [1133076089, 477841103, 151329648, 12592386…], y:[…]…}
"First Re-encrypt": {EncryptedOnceValue { x: [78, 222, 104, 185, 213, 158, 204, 209, 51, 191, 2…],
Internal_key { x: [1133076089, 477841103, 151329648, 12592386…], y:[…]…};
"Second Re-encrypt": {EncryptedOnceValue { x: [78, 222, 104, 185, 213, 158, 204, 209, 51, 191, 2…]
Internal_key { x: [1133076089, 477841103, 151329648, 12592386…], y:[…]…};
"Decrypted": { bytes: [113, 119, 101, 114, 49, 50, 51, 52, 53, 54, 55, 56, 57, 48, 49, 50, 51,"…]}
}

Fig. 5. A complete verification on all algorithms of this proposal by a threshold (3, 5)

requires 4872 milli seconds. In a decentralized network environment, the participants are required to pledge their assets for providing stable service, so that the *Reshare* algorithm also means the crash times of participated nodes.

5. Conclutions

This paper proposes a distributed proxy re-encryption network with multi-hop delegatees for reliable encryption transformation. The proposed scheme is unidirectional, non-interactive, transitive, multi-hop and collusion-safe. Also, we have implemented this scheme in a test environment, which only involves one local node that performed multiple roles in the scheme. Experiments show that most of the overhead of algorithm computation is in millisecond level except for resharing in huge threshold, while the generations of re-encryption key and multi-hop re-encrypting as well as the reshare algorithms are regardless of the raw data size. Next, we will combine this proposal with a *DAG* structured blockchain to provide decentralized *KMS* service.

Acknowledgement

This work was supported by a grant from the Core Technology Research and Development Program of Lin-Gang.

References

[1] Matt Blaze, G. Bleumer, and M. Strauss. 1998. Divertible protocols and atomic proxy cryptography. In Proceedings of Eurocrypt '98, volume 1403, 127–144.

[2] Egorov M, Wilkison M L . NuCypher KMS: Decentralized key management system[J]. 2017.

[3] M. Mambo and E. Okamoto. Proxy cryptosystems: Delegation of the power to decrypt ciphertexts[J]. IEICE Trans Fundament. Electron. Commun. Comput. Sci. 1997, 54–63.

[4] Ateniese G, Fu K, Green M, et al. Improved proxy re-encryption schemes with applications to secure distributed storage[J]. Acm Transactions on Information & System Security. 2006, 9(1):1-30.

[5] Wang H , Cao Z. A Fully Secure Unidirectional and Multi-use Proxy Re-encryption Scheme[J]. ACM CCS poster session, 2009.

[6] Cai Y , Liu X . A Multi-use CCA-Secure Proxy Re-encryption Scheme[J]. IEEE, 2014:39-44.

[7] Song L, Shen Q, Chen Z. Fullysecureunidirectionalidentity- based proxy re-encryption. In Proceedings of the International Conference on Infor- mation Security and Cryptology (ICISC 2011). Springer, Berlin, Heidelberg, 109-126, 2011.

[8] Sun Y, Willy S, Zhang F, Anmin Fu. CCA-Secure Revocable Identity-Based Encryption With Ciphertext Evolution in the Cloud. IEEE Access 6 (Oct, 2018), 56977-56983, 2018.

[9] Sun Y, Mu Y, Willy S, Zhang F,Anmin F. Revocableidentity- based encryption with server-aided ciphertext evolution. Theoretical Computer Science 815 (May, 2020), 11-24, 2020.

[10] Benhamouda F, Krawczyk H, Rabin T. "Robust Non-interactive Multiparty Computation Against Constant-Size Collusion", 2017.

[11] Liang K, Joseph Liu, Wong D,Willy S. An Efficient Cloud-Based Revocable Identity-based Proxy Re-encryption Scheme for Public Clouds Data Sharing. In Proceedings of the European Symposium on Research in Computer Security (ESORICS 2014). Springer, Cham, 257-272, 2014.

[12] W. Chang and Y. F. Wang. 2015. Seeing through the Appearance: Body Shape Estimation using Multi-view Clothing Images. In Proc. IEEE Int. Conf. on Multimedia and Expo. 1–6.

[13] Shamir. 1979. How to Share a Secret. Commun. ACM 22, 11 (Nov. 1979), 612–613.

[14] Ateniese G, Burns R, Curtmola R, et al. "Provable data possession at untrusted stores". ACM Conference on Computer & Communications Security. ACM, 2007.

[15] Sebé F, Domingo-Ferrer J, Martinez-Balleste A, et. al. "Efficient remote data possession checking in critical information infrastructures". IEEE Transactions on Knowledge and Data Engineering, 2008, 20(8): 1034-1038.

[16] Wang Q, Wang C, Ren K, et. al. "Enabling public auditability and data dynamics for storage security in cloud computing". IEEE Transactions on Parallel and Distributed Systems, 2011, 22(5): 847-859.

[17] Yang K, Jia X, Ren K. "DAC-MACS: Effective data access control for multiauthority cloud storage systems". IEEE Transactions on Information Forensics and Security, 2013, 8(11): 1790-1801.