Noname manuscript No.
(will be inserted by the editor)

Non-Transferable Proxy Re-Encryption Scheme for Data Dissemination Control

Yi-Jun He · Tat Wing Chim · Lucas Chi Kwong Hui · Siu-Ming Yiu

Received: date / Accepted: date

Abstract A proxy re-encryption (PRE) scheme allows a proxy to re-encrypt a ciphertext for Alice (delegator) to a ciphertext for Bob (delegatee) without seeing the underlying plaintext. With the help of the proxy, Alice can delegate the decryption right to any delegatee. However, existing PRE schemes generally suffer from at least one of the followings. Some schemes fail to provide the *non-transferable property* in which the proxy and the delegatee can collude to further delegate the decryption right to anyone. This is the main open problem left for PRE schemes. Other schemes assume the existence of a fully trusted private key generator (PKG) to generate the re-encryption key to be used by the proxy for re-encrypting a given ciphertext for a target delegatee. But this poses two problems in PRE schemes if the PKG is malicious: the PKG in their schemes may decrypt both original ciphertexts and re-encrypted ciphertexts (referred as the *key escrow* problem); and the PKG can generate re-encryption key for arbitrary delegatees without permission from the delegator (we refer to it as the *PKG despotism* problem).

In this paper, we propose the first *non-transferable* proxy re-encryption scheme which successfully achieves the *non-transferable property*. We also reduce the full trust in PKG, only a limited amount of trust is placed in the proxy and PKG. We show that the new scheme solved the *PKG despotism* problem and *key escrow* problem as well. Further, we find that the new scheme satisfies requirements of data dissemination control which is also a challenging goal for data security. We explore the potential of adopting our new scheme to achieve data dissemination control and implement a non-transferable re-

---

Yi-Jun He
Department of Computer Science, The University of Hong Kong
Tel.: +852-28578440
Fax: +852-25598447
E-mail: yjhe@cs.hku.hk

Tat Wing Chim
Department of Computer Science, The University of Hong Kong

encryption based encrypted PC/USB file system. Performance measurements of our scheme demonstrate that non-transferable re-encryption is practical and efficient.

Keywords proxy re-encryption · certificateless public key encryption · non-transferable property · data dissemination

1 Introduction

1.1 Proxy Re-encryption

In daily life, you may experience one or more of the situations below. When taking holidays overseas, you may still want to read emails regularly for checking if there are urgent matters requiring your attention. You might think that checking emails could easily be done anywhere via a mobile phone or a notebook. But in reality, you could be situated in a place where it is not convenient to access the network, or the network is too slow for checking emails. Then, you may ask a friend to check your emails on your behalf. Have you thought of the risk in telling others your email password or other confidential information? Do you concern about your personal information being leaked to others? Consider another situation. Suppose that you have kept some encrypted photos, videos or sensitive files in the file server to facilitate sharing the data with a group of target users. The distribution of decryption keys to the target users could become a big problem. The file system employed could be similar to Cephesus [6]; it uses a trusted access control server to distribute the keys. So, the group members must contact the access control server to obtain their decryption keys for accessing files. However, the above keys distribution method may not be satisfactory, since the underlying access control server model relies on a complete trust in the server operator. Furthermore, in practice, the server operator could abuse the keys kept by the server to decrypt any data. Even if the access control server operator can be trusted fully, letting all critical key data kept by a single server could make it become an attack target.

The proxy re-encryption, a cryptographic scheme, introduced in [4] can be employed to address the problems mentioned above. It allows a third-party (the proxy) to re-encrypt a ciphertext which has been encrypted for one party without seeing the underlying plaintext so that it can be decrypted by another. This is illustrated in Figure 1, where Alice keeps some photos, videos or sensitive files in encrypted form in the file server; Bob fetches encrypted files from file server, and then transmits the encrypted files to proxy; Alice sends a re-encryption key to the proxy which re-encrypts the encrypted files and sends Bob the re-encrypted ciphertext which can be decrypted by Bob with his own private keys. The above scheme aroused much interest in the encryption community [3][4][9][11][13][14][20][21][22][23][24] since it could be exploited in a number of applications for achieving better information security and privacy, such as:

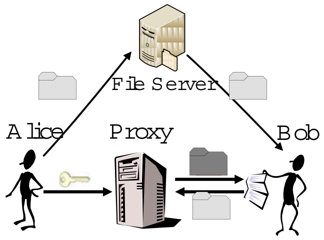


Fig. 1 Proxy Re-Encryption

- Email forwarding: Delegator wishes to delegate his email decryption right to a delegatee. The proxy can "forward" re-encrypted emails to a delegated recipient. The recipient then accesses the emails without needing to know the delegator's decryption key.
- Encrypted files distribution: The encrypted files are stored in a file server. Only the content owners can grant the access right of the files to the target users; even the file server operator has no right to access the files.
- Law-enforcement monitoring: The encrypted communication data is transferred via an Internet service provider (ISP). The ISP can require the content owners to provide the access right to the law enforcement officers to let them monitor the data being transferred to various users; however, the ISP operator cannot access the data.

1.2 Review of the Transferable Problem

However, the main problem of existing PRE schemes (details of existing schemes will be given in the next section) is failing to provide the *non-transferable* property which was first introduced by Ateniese *et al.* in 2005 [3]. A proxy re-encryption scheme is said to be non-transferable if the proxy and a set of colluding delegatees cannot re-delegate decryption rights to other parties. On one hand, this is a very desirable property. For example, user A saves some encrypted private confidential files on the file server. If A delegates B the decryption right for accessing those files, A may need some guarantee that his files "go no further". It requires that the delegatee B plus the proxy cannot re-delegate decryption right to others. On the other hand, researchers [3][11] are even not sure that transferability can be preventable since the delegatee B can always decrypt and forward the plaintext to another party. However, this approach requires that the delegatee remains an active, online participant. What we want to prevent is the delegatee (plus the proxy) providing other parties with a secret value that it can be used offline to decrypt A's ciphertexts. Again, the delegatee can always send its secret key to another party. But in doing so, the delegatee puts itself in a risky situation. Therefore, achieving a non-transferable PRE scheme, in the sense that the only way for delegatee to transfer decryption capabilities to another party is to expose his own secret key, seems to be the main open problem left for PRE.

1.3 Limitations of Existing Solutions

Libert and Vergnaud [11] indicated that it is quite difficult to prevent the proxy and delegatees from colluding to do re-delegation and that discouraging collusion rather than preventing illegitimate re-delegation is an easier approach. Thus, they try to trace the malicious proxy after its collusion with one or more delegatees. No doubt that it works to deter collusion from happening. However, it is more desirable to have a better way to prevent collusion, not just discourage collusion. Some identity-based PRE schemes [20][21][22][23][24][13] assume the existence of a fully trusted private key generator (PKG) which helps to generate the re-encryption key to be used by the proxy for re-encrypting a given ciphertext for a target delegatee. Since the re-encryption key is generated using the master key of the PKG, the proxy and the delegatee(s) cannot further delegate the decryption right to others without the help of the PKG. However, this creates two problems in PRE schemes. First, there is another key escrow problem for which the PKG in their schemes may be able to decrypt both original and re-encrypted ciphertexts; And the PKG despotism problem, in which the PKG itself has the power of generating re-encryption key for transferring decryption right to arbitrary delegatees. Thus those PKG-based PRE schemes just transformed the "delegatee-proxy-collusion transferable problem" to a "PKG alone transferable problem". So it is fair to say that they did not solve the transferable problem.

1.4 Data Dissemination Control

Data Dissemination control [19] seeks to control information and digital objects even after they have been delivered to a legitimate recipient. Control encompasses the usage of the digital object by the recipient (e.g., permission to view a document on a trusted viewer) as well as further dissemination (e.g., permission to distribute a limited number of copies of the document to colleagues but with no further dissemination allowed). Dissemination control is needed in many different domains ranging from the dissemination of digital music and movies, eBooks, business proprietary and sensitive electronic documents, healthcare [10][18].

Non-redissemination is a requirement of dissemination control, i.e. the disseminator, such as A, disseminates an object to a recipient B, but B is not allowed to disseminate the object any further. It would be a nightmare if non-redissemination control does not exist, for example, one day, you suddenly find that your encrypted private information in the file server can be accessed by anyone, but actually you disseminated it only to a recipient B before.

1.5 Our Contributions

To tackle the transferable problem as well as the key escrow problem and PKG despotism problem, a new PRE model based on certificateless public key encryption [2] is built in this paper. We borrow the idea of using PKG to generate a re-encryption key, but our new non-transferable re-encryption scheme successfully solved the problems in previous PKG-based works. The characteristics of our proposed scheme are summarized as follow.

- The proposed scheme has the non-transferable property. The re-encryption key is generated by a key generating centre (PKG); Delegator participants actively to help generating partial decryption key for delegatee using part of his private key. Thus delegatee and proxy cannot collude to re-delegate decryption rights since they do not have knowledge of PKG's master secret and the delegator's private key.
- Without the participation of the delegator, PKG is unable to generate any useful re-encryption key for delegating decryption right, thus completely resolves the PKG despotism problem.
- PKG cannot decrypt the original ciphertext and re-encrypted ciphertexts as well, thus solving the key escrow problem.

Dissemination control literatures [17][19] have been focused on mechanisms and policies. In our research, we find that the non-redissemination requirement is similar to the non-transferable requirement in proxy re-encryption scheme. Thus we propose to use our non-transferable proxy re-encryption scheme to achieve non-redissemination in a cryptographic way. Non-transferable re-encryption may be not the only way to control illegitimate redisseminating digital object, but using PRE scheme brings three main advantages:

- Disseminated digital object is invisible to proxy though it is responsible for doing re-encryption.
- Disseminator does not need to reveal his private key to the recipient for decrypting ciphertext.
- Disseminator and recipient do not need to share the same decryption key. Recipient just needs to use his own private key to decrypt the re-encrypted ciphertext.

2 Related Work

Blaze, Bleumer and Strauss [4] proposed the first proxy re-encryption scheme, which is based on ELGamal encryption. But this scheme is bi-directional, that is, when the proxy is allowed to re-encrypt Alice's messages under Bob's key, it can also re-encrypt Bob's messages under Alice's key. Bob may not like this. Another weakness is that if the proxy colludes with Alice, they can easily learn Bob's secret key SK_B. Likewise, the proxy and Bob may collude to learn Alice's secret key. Furthermore, in order to compute the re-encryption key from A to B, denoted as $rk_{A->B}$, one party must share his or her secret key with the other or they must rely on a trusted third party. The other drawback is that the scheme is transitive in the following sense. Suppose that the proxy is allowed to generate two re-encryption keys $rk_{A->B}$ and $rk_{B->C}$; then the proxy can derive an additional re-encryption key $rk_{A->C}$ for delegation from A to C.

Later, Ivan and Dodis [9] proposed three unidirectional proxy re-encryption schemes based on ElGamal, RSA, and IBE (ID-based encryption) respectively. Their main contribution is that they solved (i) the bi-directional problem and (ii) the transitive problem in [4]. But in their schemes, Alice's private key is split into two parts DK_1 and DK_2, with DK_1 distributed to proxy and DK_2 distributed to Bob. Thus when the proxy colludes with Bob, they can derive Alice's private key.

In 2005, Ateniese *et al.* [3] presented three proxy re-encryption schemes which are considered to be more secure than other approaches. Their major advantages are the following. The schemes are unidirectional and the delegator's private key is protected from being disclosed by the collusion of proxy and a delegatee. They implemented one of their proposed schemes in a secure distributed file system to show that the scheme can work efficiently in practice. They summarized nine important properties of proxy re-encryption schemes, which include the non-transferable property. Lacking the non-transferable property in all existing schemes was considered an open problem of the contemporary PRE schemes.

This open problem was first addressed in 2008 by Libert and Vergnaud [11]. They indicated that it is quite difficult to prevent the proxy and delegatees from colluding to do re-delegation and that discouraging collusion rather than preventing illegitimate re-delegation is an easier approach. Thus, they proposed, instead of preventing the collusion of proxy and delegatee, tracing the malicious proxy after its collusion with one or more delegatees. It is the first attempt to address the open problem. However, it still cannot prevent re-delegation from happening.

Matsuo's PRE schemes [13] use the PKG to help generating re-encryption key for the delegator and the delegatee. Based on this approach, they proposed two PRE schemes: one for the decryption right delegation from a user of PKI-based public key encryption system to IBE system users, and the other for the delegation among IBE system users. This is the first set of schemes that use PKG to generate re-encryption key. However, the PKG in the schemes can decrypt all re-encrypted ciphertexts; so, there is a potential security problem as long as PKG is untrusted or malicious.

In 2008, Wang *et al.*[23] extended the idea of Matsuo's scheme by allowing PKG to generate re-encryption keys based on its master secret key. They proposed several proxy re-encryption schemes:(i) PRE from IBE to Certificate Based Public Key Encryption; (ii) PRE based on a variant of the first system of Selective identity secure IBE [5]; (iii) PRE based on the second system of Selective identity secure IBE [5];and (iv) PRE based on Sakai-Kasahara IBE scheme [15]. Based on this work, Wang *et al.* proposed five other schemes [14][20][21][22][24] to address different problems of proxy re-encryption schemes. However, there are still some issues not yet addressed in each one of them. In [20], the proxy can re-encrypt on its own the ciphertext for the delegator into ciphertext for any delegatee; this is not a desired property of PRE. In [21], it seems that they solved the open problem related to the non-transferable issue, since proxy and delegate cannot collude to re-delegate decryption right; however, in the scheme, the PKG alone can delegate arbitrarily to anyone as it can generate a re-encryption key for any delegatee. In [22][24], the PKG can also delegate arbitrarily as what it could do in [21]. Among the five schemes, [21] seems to be the best in solving the non-transferable issue, we will compare our scheme with [21] in Section 5.

3 Preliminaries

3.1 Bilinear Map

Let G and G_T be multiplicative cyclic groups of prime order p, and g be generator of G. We say that G_T has an admissible bilinear map e: $G \times G \rightarrow G_T$, if the following conditions hold.

- $e(g^a, g^b) = e(g, g)^{ab}$ for all a, b.
- $e(g, g) \neq 1$.
- There is an efficient algorithm to compute $e(g^a, g^b)$ for all a, b and g.

3.2 Assumption

The security of our concrete construction is based on a complexity assumption called "Truncated Decision Augmented Bilinear Diffie-Hellman Exponent Assumption (Truncated *q-ABDHE*)"proposed in [7], which is defined as follows:

Let $e : G \times G \rightarrow G_T$ be a bilinear map, where G and G_T are cyclic groups of large prime order p. Given a vector of q+3 elements:

$$(g', g'^{\left(\alpha^{q+2}\right)}, g, g^{\alpha}, \ldots, g^{(\alpha^q)}) \in G^{q+3}$$

and an element $Z \in G_T$ as input, output 0 if $Z = e(g^{(\alpha^{q+1})}, g')$ and output 1 otherwise.

An algorithm $\mathcal{B}$ has advantage ε in solving the truncated *q-ABDHE* if:

$$| \Pr[\mathcal{B}(g', g'^{\left(\alpha^{q+2}\right)}, g, g^{\alpha}, \ldots, g^{(\alpha^q)}, e(g^{(\alpha^{q+1})}, g')) = 0]$$

$$- \Pr[\mathcal{B}(g', g'^{\left(\alpha^{q+2}\right)}, g, g^{\alpha}, \ldots, g^{(\alpha^q)}, Z) = 0]| \geq \varepsilon$$

where the probability is over the random choice of generators g, g' in G, the random choice of α in Z_p, the random choice of $Z \in G_T$, and the random bits consumed by $\mathcal{B}$.

4 Our Non-Transferable PRE Scheme

4.1 Non-Transferable PRE Model

Our Non-Transferable PRE scheme is based on certificateless public key encryption. It is composed of nine algorithms:

- Setup. On input a security parameter 1^k, the public parameters *mpk* and master secret key *msk* are generated.
- Key Generation.
  - Set-Secret-Value. algorithm generates a secret value which is only known to user himself.
  - Partial-Private-Key-Extract. On input a user's identity *ID*, *msk*, algorithm generates partial private key for user.
  - Set-Private-Key. On input the partial private key and the secret value, algorithm outputs the whole private key for user.
  - Set-Public-Key. On input a user's identity *ID* and secret value, algorithm generates public key.
- Private Key Correctness Check. Algorithm checks the correctness of the private key.
- Encryption. The encryption algorithm takes public key upk_i of delegator i and message m as input, outputs a ciphertext C_i encrypted under upk_i.
- Decryption(delegator). The decryption algorithm takes private key usk_i of delegator i and ciphertext C_i as input, outputs message m. This algorithm actually is not necessary for PRE scheme. We put it here just for indicating that delegator has the ability to decrypt the original ciphertext C_i.
- Re-Encryption Key Generation. Algorithm verifies the delegator i's signature, and extracts delegatee j's *ID* from signature. The re-encryption key generation algorithm outputs a re-encryption key $rk_{i \to j}$ and other relational values.
- Partial-Decryption-Key Generation. Algorithm checks the correctness of the re-encryption key, and generates a partial decryption key.
- Re-Encryption. The re-encryption algorithm takes re-encryption key $rk_{i \to j}$ and ciphertext C_i as input, outputs a re-encrypted ciphertext C_j under upk_j.
- Decryption(delegatee). The decryption algorithm takes private key usk_j of delegatee j, partial decryption key and ciphertext C_j as input, outputs message m.

4.2 Security Model for Identity-Based Encryption

Chosen ciphertext security for proxy re-encryption systems is defined via the following game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$:

Setup. $\mathcal{C}$ runs algorithm Setup, and outputs *params* to $\mathcal{A}$.

Phase 1. $\mathcal{A}$ adaptively issues queries q_1,...,q_m, with query q_i being one of the following:

- $(pkextract, ID_i)$: public key extraction for user ID_i
- $(encrypt, ID_i, m_i)$: encryption of plaintext for user ID_i
- $(pskextract, ID_i)$: partial private key extraction for user ID_i
- $(rkextract, ID_i, ID_{i'})$: re-encryption key extraction for delegator ID_i and delegatee $ID_{i'}$
- $(decrypt, ID_i, c_i)$: decryption of ciphertext for ID_i
- $(reencrypt, ID_i, ID_{i'}, c_i)$: re-encryption of ciphertext for ID_i to $ID_{i'}$

Challenge. The adversary submits two plaintexts M_0, $M_1 \in \mathcal{M}$ and an identity ID_j. ID_j must not have appeared in any key generation query in Phase 1. The challenger selects a random bit $b \in \{0, 1\}$, sets $C = Encrypt(params, ID_j, M_b)$, and sends C to the adversary as its challenge ciphertext.

Phase 2. This phase proceeds as in Phase 1. However $\mathcal{A}$ is restricted from issuing the following queries:

1. $(encrypt, ID_j, M_0)$ and $(encrypt, ID_j, M_1)$
2. $(decrypt, ID_j, c_j)$
3. Any pair of queries $(rkextract, ID_j, ID'_j)$ and $(decrypt, ID'_j, c'_j)$ where c'_j is the re-encrypted ciphertext using $rk_{j \to j'}$.

Guess. Finally, $\mathcal{A}$ submits a guess $b' \in \{0, 1\}$. The adversary wins if $b = b'$. We call an adversary $\mathcal{A}$ in the above game a IND-ID-CCA adversary.

Definition 1. *A proxy re-encryption scheme is said to be (t, q_{ID}, q_c, ϵ)* IND-ID-CCA *secure, if all t-time* IND-ID-CCA *adversaries making at most q_{ID} private key queries and at most q_c chosen ciphertext queries have advantage at most ϵ in winning the above game.*

Recipient-Anonymity. Informally, we say that an system is anonymous if an adversary cannot distinguish the public key *ID* under which a ciphertext was generated. More formally, we can incorporate anonymity into our game above through the following simple modification. In the Challenge phase, the adversary outputs two identities ID_0 and ID_1 not queried in Phase 1 and two messages M_0 and M_1. The challenger picks two random bits $b, c \in \{0, 1\}$, uses ID_b to encrypt M_c, and sends the resulting ciphertext C to the adversary. Phase 2 is like Phase 1, except that the adversary cannot request a private key for ID_0 or ID_1, or the decryption of C under either identity. Finally, in the Guess phase, the adversary guesses two bits b', c' and wins if $b = b'$ and $c = c'$.

Definition 2. *A proxy re-encryption scheme is (t, q_{ID}, q_c, ϵ)* ANON-IND-ID-CCA *secure, if all t-time* ANON-IND-ID-CCA *adversaries making at most q_{ID} private key queries and at most q_c chosen ciphertext queries have advantage at most ϵ in winning the modified game.*

4.3 Non-Transferable PRE Scheme Construction

We construct the Non-Transferable PRE scheme based on the basic IBE system proposed in [8]. However, the IBE system in [8] cannot fully satisfy our security requirement. We transformed this IBE system into a certificateless public key encryption system [2], so that our PRE scheme based on this new certificateless public key encryption system can successfully solve the transferable problem in existing PRE schemes. The main ideas of the scheme are as follow: Before delegation, delegator will send delegatee's identity to PKG. PKG is responsible for generating the re-encryption key, and sending this key and some other information to delegator. Delegator checks the correctness of the re-encryption key, and generates a partial decryption key making use of the information received from PKG. Then, delegator sends the re-encryption key to the proxy, and the partial decryption key to delegatee. The proxy re-encrypts the original ciphertext from delegator, and sends the re-encrypted ciphertext to delegatee. The delegatee can decrypt the ciphertext using his private key and the partial decryption key received from delegator.

In the following sections, we let Alice (A) be the delegator, and Bob (B) be the delegatee.

Setup:

Let G and G_T be groups of order p such that p is a k-bit prime, and let $e : G \times G \to G_T$ be the bilinear map. H_I:$\{0,1\}^* \to \mathrm{Z}_p$, H:$\{0,1\}^* \to \mathrm{Z}_p$ are secure hash functions. The PKG selects four random generators h_1, h_2, h_3, $g \in G$ and randomly chooses $\alpha \in Z_p$. It sets $g_1 = g^{\alpha}$. Define the message space $\mathcal{M} \in G_T$. The public parameters *mpk* and master secret key *msk* are given by

$$mpk = (g, g_1, h_1, h_2, h_3, H_I, H, \mathcal{M}), msk = (\alpha)$$

Key Generation:

This is a protocol through which a user U with an identity ID can securely get his partial private key from PKG.

On input the public key/master secret key pair (*mpk*, *msk*) and an identity $ID_A \in \{0,1\}^k$ of entity A, the PKG computes $id_A = H_I(ID_A)$. If $id_A = \alpha$, it aborts. Otherwise, the protocol proceeds as follow:

- Set-Secret-Value. Entity A selects $r_A \in Z_p$ at random. r_A is A's secret value.
- Partial-Private-Key-Extract.
  1. A sends $R = {h_1}^{r_A}$ to PKG, and gives PKG the following zero-knowledge proof of knowledge:
  $$PK\{r_A : R = {h_1}^{r_A}\}$$
  2. PKG randomly selects r'_A, $r_{\mathrm{A},2}$, $r_{\mathrm{A},3} \in Z_p$ and computes $h'_A = (Rg^{-r'_A})^{1/(\alpha - id_A)}$, $h_{A,2} = (h_2 g^{-r_{A,2}})^{1/(\alpha - id_A)}$, $h_{A,3} = (h_3 g^{-r_{A,3}})^{1/(\alpha - id_A)}$ and sends A's partial private key (r'_A, h'_A, $r_{A,2}$, $h_{A,2}$, $r_{A,3}$, $h_{A,3}$) to A.
- Set-Private-Key. A computes
$$r_{A,1} = r'_A / r_A, h_{A,1} = (h'_A)^{1/r_A} = (h_1 g^{-r_{A,1}})^{1/(\alpha - id_A)}$$

Then, A's private key can be denoted as

$$usk_A = (r_A, r_{A,1}, h_{A,1}, r_{A,2}, h_{A,2}, r_{A,3}, h_{A,3})$$

Similarly, the delegatee B's private key is denoted as

$$usk_B = (r_B, r_{B,1}, h_{B,1}, r_{B,2}, h_{B,2}, r_{B,3}, h_{B,3})$$

- Set-Public-Key. A publishes her public key $upk_A = (p_{A,1}, p_{A,2})$, where $p_{A,1} = {g_1}^{r_A}$, and $p_{A,2} = g^{r_A id_A}$. Anyone can verify the validity of upk_A by checking if the equalities $e(g^{id_A}, p_{A,1}) = e(g_1, p_{A,2})$, and $e(h_1^{r_A}, g_1) = e(h_1, p_{A,1})$ hold (i.e., $h_1^{r_A}$ can be obtained from PKG).

Private Key Correctness Check:

On input (mpk, usk_{ID}) and an identity $ID \in \{0,1\}^k$, A computes $id_A = H_I\left(ID_A\right)$ and checks whether

$$e(h_{A,i}, g_1/g^{id_A}) = e(h_i g^{-r_{A,i}}, g)$$

for i=1,2,3. If correct, output 1. Otherwise, output 0.

Encryption:

To encrypt a message $m \in G_T$ using public key, sender checks that whether the equalities $e(g^{id_A}, p_{A,1}) = e(g_1, p_{A,2})$ and $e(h_1^{r_A}, g_1) = e(h_1, p_{A,1})$ hold. If not, output $\perp$ and abort encryption. Otherwise, sender generates a unique randomly-selected secret parameter $s \in Z_p$, and computes $id_A = H_I\left(ID_A\right)$. Finally, sender outputs the ciphertext C where:
$C = (C_1, C_2, C_3, C_4, C_5) = ({p_{A,1}}^s {p_{A,2}}^{-s}, e(g,g)^s, m \cdot e(g,h_1)^{-s}, e(g,h_2)^s e(g,h_3)^{s\beta}, g^{\beta})$.

We set $\beta = H(C_1, C_2, C_3)$.

Decryption(delegator):

To decrypt a ciphertext $C = (C_1, C_2, C_3, C_4, C_5)$ using secret key usk_A, delegator Alice computes $\beta = H(C_1, C_2, C_3)$ and tests whether

$$e(C_1, C_5) = e(C_1, g)^{\beta}$$

and

$$C_4 = e(C_1, h_{A,2}{h_{A,3}}^{\beta})^{1/r_A} \cdot {C_2}^{r_{A,2}+r_{A,3}\beta}$$

If it is not equal, output $\perp$. Else output

$$m = C_3 \cdot e\left(C_1, h_{A,1}\right)^{1/r_A} \cdot {C_2}^{r_{A,1}}$$

The following Re-Encryption process is done through an interactive protocol among Alice, Bob, PKG and Proxy, which is shown in Figure 2.

Re-Encryption Key Generation:

1. In our PRE scheme, Bob is only allowed to decrypt messages intended for Alice during some specific time period i. To achieve this property, the delegator Alice generates a random value $a_i \in Z_p$ for each time period i, where $i \geq 1$. a_i will be invalid after the period i. Alice signs Bob's identity ID_B, and sends the signature σ, ID_B, a_i to PKG via a secure channel.
   Delegator Sign:
   - Choose $z \in Z_p$, and compute $U = g^z$.
   - Compute $V = H_I\left(ID_B, U\right)$.
   - Compute $W = g^{\alpha r_A + V}$.

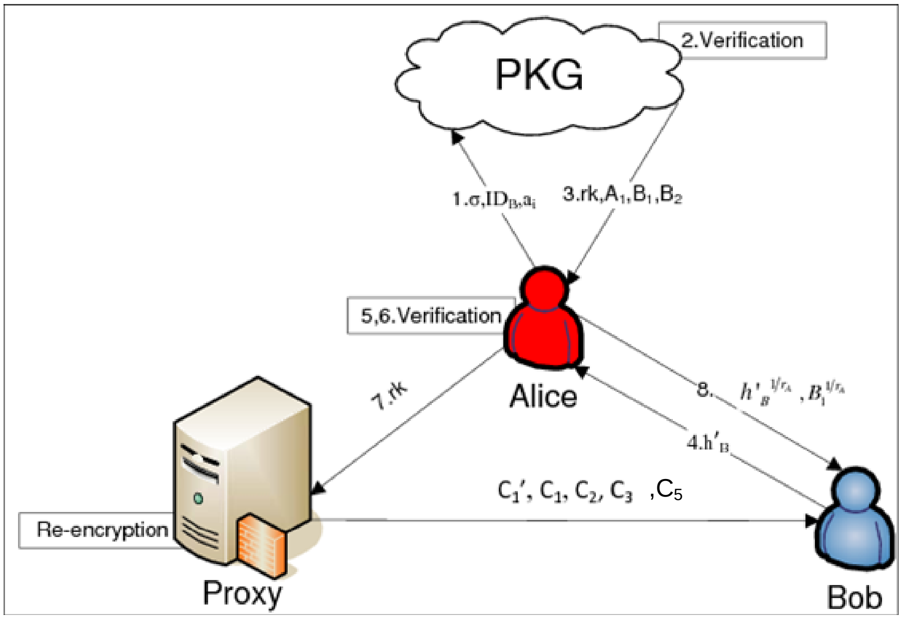


Fig. 2 Proposed Non-Transferable Proxy Re-encryption framework

   - The signature on ID_B is $\sigma = (U, W)$.

2. PKG verifies Alice to identify the identity of the delegator.
   PKG Verify:
   - Compute $V = H_I\,(ID_B, U)$.
   - Accept the signature iff $e(h_1, W) = e({h_1}^{r_A}, g^{\alpha})e(h_1, g)^V$.
3. If verification passes, PKG generates a unique randomly-selected secret parameter $y \in Z_p$, and computes re-encryption key $rk_{A\to B} = (\frac{\alpha - id_B}{\alpha - id_A} + a_i y) \bmod p, A_1 = \left({h_1}^{r_A} g^{-r'_A}\right)^y, B_1 = \left({h_1}^{r_B} g^{-r'_B}\right)^{a_i y/(\alpha - id_B)}, B_2 = {h_1}^{a_i y}$ and sends $rk_{A\to B}$, A_1, B_1, B_2 to Alice.

Partial-Decryption-Key Generation:

4. Delegatee Bob sends h'_B to Alice via a secure and authenticated channel.
5. Alice checks whether

$$e(h_1, B_1) = e(B_2, h'_B)$$

   to ensure B_1 is a valid value which will help delegatee for decryption later. If correct, output 1, otherwise, output 0.

6. Alice checks whether

$$h'_A{}^{(id_A - id_B)} \cdot {A_1}^{a_i} \cdot \left({h_1}^{r_A} g^{-r'_A}\right) = \left({h_1}^{r_A} g^{-r'_A}\right)^{rk_{A\to B}}$$

   to ensure that $rk_{A\to B}$ is a re-encryption key generated properly for delegation from her to Bob.

7. Alice sends the re-encryption key $rk_{A\to B}$ to Proxy via an authenticated channel.
8. Alice computes ${h'_B}^{1/r_A}$ and ${B_1}^{1/r_A}$, and sends them to Bob as partial decryption key.

***Re-Encryption**:*

Proxy computes $\beta = H(C_1, C_2, C_3)$ and tests whether

$$e(C_1, C_5) = e(C_1, g)^{\beta}$$

If it is not equal, output $\perp$. Else computes ${C_1}' = {C_1}^{rk_{A\to B}} = g^{r_A s(\alpha - id_A)(\frac{\alpha - id_B}{\alpha - id_A} + a_i y)}$, and sends $({C_1}', C_1, C_2, C_3, C_5)$ to Bob.

Decryption (delegatee):

Bob computes $\beta = H(C_1, C_2, C_3)$ and tests whether

$$e(C_1, C_5) = e(C_1, g)^{\beta}$$

If it is not equal, output $\perp$. Else Bob computes

$$C_3 \frac{e({C_1}', {h'_B}^{(1/r_A)(1/r_B)}){C_2}^{r_{B,1}}}{e(C_1, {B_1}^{(1/r_A)(1/r_B)})}$$
$$= C_3 \frac{e(g^{r_A s(\alpha - id_A)(\frac{\alpha - id_B}{\alpha - id_A} + a_i y)}, (h_1 g^{-r_{B,1}})^{\frac{1}{(\alpha - id_B) r_A}})(e(g,g)^s)^{r_{B,1}}}{e(g^{r_A s(\alpha - id_A)}, (h_1 g^{-r_{B,1}})^{\frac{a_i y}{(\alpha - id_B) r_A}})}$$
$$= C_3 e(g^{s(\alpha - id_A)(\frac{\alpha - id_B}{\alpha - id_A})}, (h_1 g^{-r_{B,1}})^{\frac{1}{(\alpha - id_B)}}) e(g,g)^{s * r_{B,1}}$$
$$= m \cdot e(g, h_1)^{-s} e(g^{s(\alpha - id_B)}, (h_1 g^{-r_{B,1}})^{\frac{1}{(\alpha - id_B)}}) e(g,g)^{s r_{B,1}}$$
$$= m$$

5 Security Analysis and Proof

5.1 Security Analysis

The main advantage of our scheme is: It achieved Non-transferable property, Non-Key-escrow property and Non-PKG-despotism property, in which Non-Key-escrow property and Non-PKG-despotism property are defined by us especially for estimating security of a PKG involved PRE schemes. To prove that our scheme is able to achieve Non-transferable property, we assume there is a possible attack, and prove how our scheme can resist to this attack.

- Non-transferable: In PRE, the proxy and a set of colluding delegatees cannot re-delegate decryption rights. This is called Non-transferable. For example, from $rk_{A\to B}$, sk_B and pk_C, they cannot produce $rk_{A\to C}$.
  Proof:Now go back to our scheme for a concrete discussion. After one delegation, the proxy holds $rk_{A\to B}$, and the delegatee Bob holds $(r_B, r_{B,1})$ and $(DK_0 = {h'_B}^{1/r_A}$, $DK_1 = B_1^{1/r_A})$. If proxy and Bob want to launch an attack in order to re-delegate the decryption right to others, Bob may compute $DK_2 = DK_0^{1/r_B}$ and $DK_3 = DK_1^{1/r_B}$ by himself, then the proxy

exponentiate the DK_2 by $rk_{A \to B}$. Given (C_1, C_2, C_3) which is the original ciphertext for the delegator, anyone who holds $(DK_2, DK_3, r_{B,1})$ can decrypt by $C_3 \cdot e(C_1, DK_2) \cdot C_2^{r_{B,1}} / e(C_1, DK_3)$. However, notice that whatever method (2-party computation, or oblivious computation) the proxy and delegatee used to compute the DK_2, DK_3, this re-delegation will success only when Bob wishes to send his secret key $r_{B,1}$ explicitly to other parties, because $r_{B,1}$ must be used to exponentiate C_2 for decrypting the ciphertext. Further, C_2 is changeable in each delegation due to the random number s is changing, so $C_2^{r_{B,1}}$ cannot be computed offline, and $r_{B,1}$ must be sent explicitly to other parties. But by doing so, Bob may put himself in danger, because his private key would be known to PKG once other parties report $r_{B,1}$ to PKG. Since the only way for Bob and proxy to transfer decryption capabilities to other parties is to expose Bob's secret key, Bob would not run the risk of launching such attack. Thus we achieved the purpose of preventing delegatees from colluding with proxy to re-delegate the decryption right. Non-transferable property is achieved in our scheme.

- *Non-Key-escrow*: In PRE, PKG should not be allowed to decrypt both original ciphertext and re-encrypted ciphertext for anyone, this is called Non-Key-escrow.

  Most of PKG-based PRE schemes do not achieve this property as discussed in section 2. However, in our proposed scheme, the original ciphertext is decrypted by $m = C_3 \cdot e\left(C_1, h_{A,1}\right)^{1/r_A} \cdot {C_2}^{r_{A,1}}$, in which r_A is needed for decryption. Moreover, the re-encrypted ciphertext is decrypted by $m = C_3 \frac{e({C_1}', {h'_B}^{(1/r_A)(1/r_B)}){C_2}^{r_{B,1}}}{e(C_1, {B_1}^{(1/r_A)(1/r_B)})}$, in which r_B is needed for decryption. Notice that r_A is the secret value of delegator, and r_B is the secret value of delegatee. PKG holds neither r_A nor r_B. Thus only users themselves can decrypt the ciphertext, not the PKG. Non-Key-escrow property is achieved in our scheme.
- *Non-PKG-despotism*: In PRE, PKG is not allowed to generate a proper re-encryption key arbitrarily for delegating decryption right without permission from delegator, this is called Non-PKG-despotism.

  In our proposed scheme, delegator participants actively to help PKG generating re-encryption key by sending the random value a_i, and the validity of the re-encryption key will be verified by delegator by checking if ${h'_A}^{(id_A - id_B)} \cdot {A_1}^{a_i} \cdot \left({h_1}^{r_A} g^{-r'_A}\right) = \left({h_1}^{r_A} g^{-r'_A}\right)^{rk_{A \to B}}$. If PKG misbehaved, delegator can detect it via validity verification. Further, delegator is responsible for generating a partial decryption key $({h'_B}^{1/r_A}, B_1^{1/r_A})$ for delegatee using his own secret value r_A. Without the participation of the delegator, delegatee is unable to decrypt the re-encrypted ciphertext, even if PKG may collude with proxy to do re-encryption illegally (PKG can do this by sending the re-encryption to proxy directly, without passing the delegator). In another word, the re-encryption key generated by PKG alone is useless unless delegator is willing to help to generate the partial decryption key. Thus completely resolves the PKG despotism problem.

To compare some existing proxy re-encryption schemes with our proposed scheme as fully as possible, we also analyze below some important properties defined in [3]. The comparison results are presented in Table 1.

- *Unidirectional*: Delegation from $A \to B$ does not allow re-encryption from $B \to A$.

  In our scheme, $rk_{A \to B}$ is in the form of $(\frac{\alpha - id_B}{\alpha - id_A} + a_i y) \bmod p$. It is impossible to modify it into a valid $rk_{B \to A}$.
- *Non-interactive*: Re-encryption keys can be generated by Alice using Bob's public key; no trusted third party or interaction is required.

  In our proposed scheme, PKG is employed to generate re-encryption keys, so delegator needs to interact with PKG to generate the keys.
- *Proxy invisible to delegator*: In Ateniese's scheme, a property called *proxy invisibility* is achieved. This property in their work means the sender needs to know the existence of proxy, in order to decide whether to generate first-level encryption or second-level encryption, but the delegatee need not to know the proxy existence. However, in our proposed scheme, proxy is invisible to delegator, since there is only one form of encryption, but visible to delegatee because delegatee needs to decide whether to decrypt a normal form of ciphertext or a re-encrypted form of ciphertext. Thus, we differentiate these two situations by using "Proxy invisible to delegator" and "Proxy invisible to delegatee". Our scheme achieved "Proxy invisible to delegator", on the contrary, Ateniese's scheme achieved "Proxy invisible to delegatee".
- *Original-access*: Alice can decrypt re-encrypted ciphertexts that were originally sent to her.

  This has been proved in our scheme construction.
- *Key optimal*: The size of Bob's secret storage remains constant, regardless of how many delegations he accepts.

  Like Ateniese's scheme, delegatee is allowed to decrypt re-encrypted ciphertext during some specific time period i. Thus information received from PKG for decryption only need to exist temporarily in delegatee's side. After a time period i, the information would be invalid. Delegatee can delete the information immediately. Thus in the long run, our scheme is still key optimal.
- *Collusion-"safe"*: Bob and the proxy's collusion cannot recover Alice's secret key.

  In our proposed scheme, secrecy of Alice's secret key depends on a random value r_A. It is chosen by Alice, and is not used in re-encryption key. Although Bob and proxy collude, they cannot recover it.
- *Temporary*: Bob is only able to decrypt messages intended for Alice that were authored during some specific time period i.

  In our scheme, to achieve temporary proxy re-encryption, for each time period $i \geq 1$, Alice generates a random value $a_i \in Z_p$. Because a_i will be invalid after time period i, the re-encryption key's life cycle is also period i. We remark that in most existing schemes we are aware, including

Table 1 Security Comparison of existing PRE schemes and our proposed scheme

Property	BBS [4]	ID [9]	Ateniese [3]	Wang [21]	Our Scheme
Uni-directional	No	Yes	Yes	Yes	Yes
Non-interactive	No	Yes	Yes	No	No
Proxy invisibility	Yes	No	Yes$^{\#}$	Yes	Yes$^{\#}$
Original-access	Yes	Yes	Yes	No	Yes
Key optimal	Yes	No	Yes	Yes	Yes
Collusion-safe	No	No	Yes	Yes	Yes
Temporary	Yes!	Yes!	Yes!	No	Yes!
Non-transitive	No	Yes	Yes	Yes	Yes
Non-transferable	No	No	No	No*	Yes
Non-Key-escrow	−−	No	−−	No	Yes
Non-PKG-despotism	−−	No	−−	No	Yes

($*$) PKG alone can transfer

($^{\#}$) Ateniese [3] can only achieve proxy invisible to delegatee, our scheme can only achieve proxy invisible to delegator.

(!) possible to achieve with additional assumption and overhead.

our scheme, the temporary property is achieved based on the assumption that the proxy will update the re-encryption key after each period expires and re-encrypt the ciphertext using the updated re-encryption key. Otherwise, if proxy always uses the expired re-encryption key to re-encrypt a ciphertext, as long as the re-encryption is correct, the delegatee can always decrypt the re-encrypted ciphertext using expired decryption key. In this case, Temporary property cannot be achieved.

- *Non-transitive*: Based on the re-encryption keys, $rk_{A\to B}$ and $rk_{B\to C}$, the proxy cannot produce $rk_{A\to C}$.

  In our proposed scheme, the re-encryption key is generated using the master secret key of the PKG, proxy cannot generate $rk_{A\to C}$ without knowing the master secret key. And the delegatee's identity is included in the re-encryption key, the proxy is unable to replace the delegatee with another party. So even with the keys $rk_{A\to B}$ and $rk_{B\to C}$, the proxy cannot produce $rk_{A\to C}$.

Our scheme adds more rounds of interaction for the following reasons:

- Private Key Correctness Check is added for checking the partial private key generated by PKG, since PKG is not fully trusted in our assumption.
- In Re-encryption Key Generation, step 1 and 2 are added for PKG to verify delegator and get delegatee's identity, since PKG is responsible for generate re-encryption key. Without verification, attacker may impersonate delegator to trick PKG into generating re-encryption key.
- Partial-Decryption-Key Generation is added to prevent PKG, proxy and delegatee re-delegating decryption right from colluding. With this step, even if PKG, proxy and delegatee's collude, they are unable to generate re-encryption key for re-delegating decryption right without the original delegator's help.

5.2 Security Proof

Theorem 1. Assume the truncated (decision) (t, ε, q)-$ABDHE$ assumption holds for (G, G_T, e). Then, the above non-transferable re-encryption scheme is ANON-IND-ID-CCA secure.

Proof Let $\mathcal{A}$ be an adversary who breaks the ANON-IND-ID-CCA security of our system. We construct an algorithm, $\mathcal{B}$, that solves the truncated decision q-ABDHE problem, as follows. $\mathcal{B}$ takes as input a random truncated decision q-ABDHE challenge $(g', g'_{q+2}, g, g_1, ..., g_q, Z)$ where Z is either $e(g_{q+1}, g')$ or a random element of G_T (recall that $g_i = g^{(\alpha^i)}$). Algorithm $\mathcal{B}$ proceeds as follows.

Setup: $\mathcal{B}$ generates three random polynomials $f_1(x) \in Z_p[x]$, $f_2(x) \in Z_p[x]$ and $f_3(x) \in Z_p[x]$ all of degree q. It sets $h_1 = g^{f_1(\alpha)}$, $h_2 = g^{f_2(\alpha)}$ and $h_3 = g^{f_3(\alpha)}$ by computing them from $(g, g_1, ..., g_q)$. It sends the public parameters (g, g_1, h_1, h_2, h_3) to A. Since g, α, $f_1(x)$, $f_2(x)$ and $f_3(x)$ are chosen uniformly at random, h_1, h_2 and h_3 are uniformly random as well and these public parameters have distribution identical to that in the actual construction.

Phase 1: $\mathcal{A}$ makes key generation queries by giving $\mathcal{B}$ an identity ID_A and $R = h_1^{r_A}$ where $r_A \in Z_p$ is a random number. $\mathcal{A}$ also gives $\mathcal{B}$ the zero-knowledge proof $PK\{r_A : R = h_1^{r_A}\}$.

$\mathcal{B}$ responds to a query on ID_A as follows. If $ID_A = \alpha$, $\mathcal{B}$ uses α to solve the truncated decision q-ABDHE immediately. Otherwise, let $F_{A,2}(x) = (f_2(x) - f_2(ID_A))/(x - ID_A)$ and $F_{A,3}(x) = (f_3(x) - f_3(ID_A))/(x - ID_A)$ be two $(q-1)$-degree polynomials. We use the same technique as in [15] to extract r_A from $\mathcal{A}$, and sets r'_A=$r_A r_{A,1}$ and h'_A=${h_{A,1}}^{r_A}$. $\mathcal{B}$ sets the partial private key $(r'_A, h'_A, r_{A,2}, h_{A,2}, r_{A,3}, h_{A,3})$ to be $(f_1(ID_A)$, $(Rg^{-r'_A})^{1/(\alpha - ID_A)}$, $f_2(ID_A)$, $g^{F_{A,2}(\alpha)}$, $f_3(ID_A)$, $g^{F_{A,3}(\alpha)})$. Note that for $i = 2, 3$, $g^{F_{A,i}(\alpha)} = g^{(f_i(\alpha) - f_i(ID_A))/(\alpha - ID_A)} = (h_i g^{-r_{A,i}})^{1/(\alpha - ID_A)}$.

Next $\mathcal{A}$ computes $r_{A,1} = r'_A / r_A$ and $h_{A,1} = (h'_A)^{1/r_A}$ and sets its private key as $(r_A, r_{A,1}, h_{A,1}, r_{A,2}, h_{A,2}, r_{A,3}, h_{A,3})$. This is a valid private key for ID_A since for $i = 1, 2, 3$, $h_{A,i} = (h_i g^{-r_{A,i}})^{1/(\alpha - ID_A)}$ as required.

$\mathcal{A}$ also makes decryption queries. To respond to a decryption query on (ID_A, C), $\mathcal{B}$ generates a private key for ID_A as before. $\mathcal{B}$ then decrypts C by performing the usual delegator decryption algorithm with this private key.

Challenge: $\mathcal{A}$ outputs identities ID_0, ID_1, random values R_0, R_1 and messages M_0, M_1. Again, if $ID_0 = \alpha$ or $ID_1 = \alpha$, $\mathcal{B}$ uses α to solve the truncated decision q-ABDHE immediately. Otherwise, $\mathcal{B}$ generates bits $b, c \in \{0, 1\}$ and computes a partial private key $(r'_b, h'_b, r_{b,2}, h_{b,2}, r_{b,3}, h_{b,3})$ as in Phase 1.

Let $f_4(x) = x^{q+2}$ and let $F_{4,b}(x) = (f_4(x) - f_4(ID_b))/(x - ID_b)$ be a polynomial of degree $q+1$. $\mathcal{B}$ continues to set $u = g'^{(f_4(\alpha) - f_4(ID_b))r_b}$, $v = Z \times e(g', \prod_{i=0}^{q} g^{F_{4,b,i}\alpha^i})$ and $w = M_c / e(u, h_{b,1})^{1/r_b} v^{r_{b,1}}$ where $F_{4,b,i}$ is the coefficient of x^i in $F_{4,b}(x)$. After setting $\beta = H(u, v, w)$, $\mathcal{B}$ sets $y = e(u, h_{b,2}h_{b,3}{}^{\beta})^{1/r_b} v^{r_{b,2}+r_{b,3}\beta}$ and $z = g^{\beta}$. $\mathcal{B}$ sends (u, v, w, y, z) to $\mathcal{A}$ as the challenge ciphertext.

Let $s = (log_g g')F_{4,b}(\alpha)$. If $Z = e(g_{q+1}, g')$, then $u = g^{sr_b(\alpha - ID_b)}$, $v = e(g, g)^s$, $M_c/w = e(u, h_{b,1})^{1/r_b} v^{r_{b,1}} = e(g, h_1)^s$, and $e(u, z) = e(u, g)^{\beta}$. Since $log_g g'$ and s are uniformly random, (u, v, w, y, z) is a valid, appropriately-distributed challenge to $\mathcal{A}$.

Phase 2: $\mathcal{A}$ makes key generation queries and decryption queries, and $\mathcal{B}$ responds as in Phase 1.

Guess: Finally, the adversary $\mathcal{A}$ outputs guesses $b', c' \in \{0, 1\}$. If $b = b'$ and $c = c'$, $\mathcal{B}$ outputs 0 (indicating that $Z = e(g_{q+1}, g')$). Otherwise, $\mathcal{B}$ outputs 1.

Probability Analysis and Conclusion: If $Z = e(g_{q+1}, g')$, then the simulation is perfect. Assume that $\mathcal{A}$ has made d decryption queries in Phase 1, the average length of a ciphertext be len_c bits and the length of an identity is len_{id} bits. $\mathcal{A}$ will guess the bits (b, c) correctly with probability $1/4 + d/(2^{len_c + len_{id}}) + \epsilon$ where $d/(2^{len_c + len_{id}})$ is the probability that $\mathcal{A}$ has queried the decryption on ID_b and the ciphertext of M_c in Phase 1. If Z is uniformly random, (u, v, w, y) is an invalid ciphertext for (ID_b, M_c) and it carries no information regarding the bits (b, c). In this case, $\mathcal{A}$ will guess the bits (b, c) correctly with probability 1/4. Therefore on overall, $\mathcal{A}$ will guess the bits (b, c) correctly with probability $1/2(1/4 + d/(2^{len_c + len_{id}}) + \epsilon) + 1/2(1/4) = 1/8 + d/(2 \times 2^{len_c + len_{id}}) + 1/2\epsilon + 1/8 = 1/4 + d/(2 \times 2^{len_c + len_{id}}) + 1/2\epsilon$. Thus $\mathcal{A}$'s advantage is non-negligible. $\mathcal{B}$ thus can make use of $\mathcal{A}$ to solve the truncated decision q-ABDHE problem. This leads to a contradiction since the truncated decision q-ABDHE problem is a well-known hard problem. As a result, our scheme is secure under ANON-IND-ID-CCA.

Since no matter before or after re-encryption, the ciphertext resulted from our scheme is of the same format (though involves private keys of different users), the same security proof applies to the cases before and after re-encryption.

Theorem 2*. Assume the truncated decision (t, ε, q)-ABDHE assumption holds for (G, G_T, e). Then, the above non-transferable re-encryption scheme is IND-ID-CCA secure even when the proxy and the delegatees are colluding.*

Proof Let $\mathcal{A}$ be an adversary who breaks the IND-ID-CCA security of our system (in particular, when the proxy and the delegatees are colluding). We construct an algorithm, $\mathcal{B}$, that solves the truncated decision q-ABDHE problem, as follows. $\mathcal{B}$ takes as input a random truncated decision q-ABDHE challenge $(g', g'_{q+2}, g, g_1, ..., g_q, Z)$ where Z is either $e(g_{q+1}, g')$ or a random element of G_T (recall that $g_i = g^{(\alpha^i)}$). Algorithm $\mathcal{B}$ proceeds as follows.

Setup: $\mathcal{B}$ generates three random polynomials $f_1(x) \in Z_p[x]$, $f_2(x) \in Z_p[x]$ and $f_3(x) \in Z_p[x]$ all of degree q. It sets $h_1 = g^{f_1(\alpha)}$, $h_2 = g^{f_2(\alpha)}$ and $h_3 = g^{f_3(\alpha)}$ by computing them from $(g, g_1, ..., g_q)$. It sends the public parameters (g, g_1, h_1, h_2, h_3) to $\mathcal{A}$. Since g, α, $f_1(x)$, $f_2(x)$ and $f_3(x)$ are chosen uniformly at random, h_1, h_2 and h_3 are uniformly random as well and these public parameters have distribution identical to that in the actual construction. $\mathcal{B}$ maintains a list L to store the entry $< ID_Q, pk_Q, psk_Q, sk_Q >$ of every user with identity ID_Q, public key pk_Q, partial private key psk_Q and private key sk_Q it has been queried so far.

Phase 1: During this phase, $\mathcal{A}$ can issue the following queries:

1. $(\mathit{pkextract}, ID_Q)$: public key extraction for user ID_Q
2. $(\mathit{encrypt}, ID_Q, m_Q)$: encryption of plaintext for user ID_Q
3. $(\mathit{pskextract}, ID_Q)$: partial private key extraction for user ID_Q
4. $(\mathit{rkextract}, ID_Q, ID_{Q'})$: reencryption key extraction for delegator ID_Q and delegatee $ID_{Q'}$
5. $(\mathit{decrypt}, ID_Q, c_Q)$: decryption of ciphertext for ID_Q
6. $(\mathit{reencrypt}, ID_Q, ID_{Q'}, c_Q)$: reencryption of ciphertext for ID_Q to $ID_{Q'}$

Note that $\mathcal{A}$ is not allowed to issue private key extraction queries since under our scheme, even PKG only knows the partial private key but not the complete private key of a user.

$\mathcal{B}$ responds to these queries as follows:

On $(\mathit{pkextract}, ID_Q)$, if $ID_Q = \alpha$, $\mathcal{B}$ uses α to solve the truncated decision q-ABDHE immediately. Otherwise, let $F_{Q,2}(x) = (f_2(x) - f_2(ID_Q))/(x - ID_Q)$ and $F_{Q,3}(x) = (f_3(x) - f_3(ID_Q))/(x - ID_Q)$ be two $(q-1)$-degree polynomials. $\mathcal{B}$ sets the partial private key for ID_Q to be $(r'_Q,\ h'_Q,\ r_{Q,2},\ h_{Q,2},\ r_{Q,3},\ h_{Q,3})$ which is $(f_1(ID_Q),\ (Rg^{-r'_Q})^{1/(\alpha - ID_Q)},\ f_2(ID_Q),\ g^{F_{Q,2}(\alpha)},\ f_3(ID_Q),\ g^{F_{Q,3}(\alpha)})$ respectively. For $i = 2, 3$, $g^{F_{Q,i}(\alpha)} = g^{(f_i(\alpha) - f_i(ID_Q))/(\alpha - ID_Q)} = (h_i g^{-r_{Q,i}})^{1/(\alpha - ID_Q)}$. Next, $\mathcal{B}$ computes $r_{Q,1} = r'_Q / r_Q$ and $h_{Q,1} = (h'_Q)^{1/r_Q}$ to complete the private key for ID_Q. The private key for ID_Q thus becomes $(r_Q,\ r_{Q,1},\ h_{Q,1},\ r_{Q,2},\ h_{Q,2},\ r_{Q,3},\ h_{Q,3})$. Note that this is a valid private key for ID_Q since for $i = 1, 2, 3$, $h_{Q,i} = (h_i g^{-r_{Q,i}})^{1/(\alpha - ID_Q)}$ as required. $\mathcal{B}$ then computes the public key for ID_Q as $(g_1^{r_Q},\ (g^{r_Q})^{ID_Q})$, stores all these information into L and returns the public key to $\mathcal{A}$.

On $(\mathit{encrypt}, ID_Q, m_Q)$, if $ID_Q = \alpha$, $\mathcal{B}$ uses α to solve the truncated decision q-ABDHE immediately. If ID_Q is in L, $\mathcal{B}$ simply extracts the public key in the corresponding entry. Otherwise, $\mathcal{B}$ generates the public key, partial private key and private key for ID_Q as in the above, stores them into L, encrypts m_Q by performing the usual encryption algorithm with the public key concerned and returns the ciphertext of m_Q to $\mathcal{A}$.

On $(\mathit{pskextract}, ID_Q)$, if $ID_Q = \alpha$ or $ID_{Q'} = \alpha$, $\mathcal{B}$ uses α to solve the truncated decision q-ABDHE immediately. If ID_Q is in L, $\mathcal{B}$ simply returns the partial private key in the corresponding entry. Otherwise, $\mathcal{B}$ generates the public key, partial private key and private key for ID_Q as in the above, stores them into L and returns the partial private key to $\mathcal{A}$.

On $(rkextract, ID_Q, ID_{Q'})$, if $ID_Q = \alpha$ or $ID_{Q'} = \alpha$, $\mathcal{B}$ uses α to solve the truncated decision q-ABDHE immediately. If ID_Q or $ID_{Q'}$ or both are in L, $\mathcal{B}$ extracts the partial private key(s) in the corresponding entry(entries). Otherwise, $\mathcal{B}$ generates the public key, partial private key and private key for the identity not in L as in the above, stores them into L and uses the partial private keys concerned for further processing as follows. $\mathcal{B}$ computes the re-encryption key $rk_{Q \to Q'}$ using the usual re-encryption key calculation algorithm except that $\mathcal{B}$ generates the random value a_i on behalf of ID_Q and α is replaced by a random value (since $\mathcal{B}$ does not know the value of α). Note that although $rk_{Q \to Q'}$ is an invalid re-encryption key, $\mathcal{A}$ has no way to verify its correctness since it does not possess the private key of ID_Q and it cannot query it from $\mathcal{B}$ either.

On $(decrypt, ID_Q, c_Q)$, if $ID_Q = \alpha$, $\mathcal{B}$ uses α to solve the truncated decision q-ABDHE immediately. If ID_Q is in L, $\mathcal{B}$ simply uses the private key in the corresponding entry to decrypt c_Q by performing the usual delegator decryption algorithm. Otherwise, $\mathcal{B}$ generates the public key, partial private key and private key for ID_Q as in the above, stores them into L and uses the private key concerned decrypt c_Q by performing the usual delegator decryption algorithm.

On $(reencrypt, ID_Q, ID_{Q'}, c_Q)$, if $ID_Q = \alpha$ or $ID_{Q'} = \alpha$, $\mathcal{B}$ uses α to solve the truncated decision q-ABDHE immediately. If ID_Q or ID'_Q or both are in L, $\mathcal{B}$ extracts the public and private keys in the corresponding entry (entries). Otherwise, $\mathcal{B}$ generates the public key, partial private key and private key for the identity not in L as in the above, stores them into L and uses the public and private keys concerned for further processing as follows. $\mathcal{B}$ decrypts c_Q using the private key for ID_Q by performing the usual delegator decryption algorithm and then encrypts the plaintext obtained using the public key for $ID_{Q'}$ by performing the usual encryption algorithm. This ensures that the re-encrypted ciphertext is decryptable by the private key for $ID_{Q'}$.

At the end of Phase 1, $\mathcal{A}$ outputs (ID_A, M_0, M_1) where $\mathcal{A}$ may have queried anything about ID_A but must not have queried $(encrypt, ID_A, M_0)$ and $(encrypt, ID_A, M_1)$ before. If $ID_A = \alpha$, $\mathcal{B}$ uses α to solve the truncated decision q-ABDHE immediately. If ID_A is in L, $\mathcal{B}$ simply extracts the partial private key in the corresponding entry. Otherwise, $\mathcal{B}$ computes a partial private key (r'_A, h'_A, $r_{A,2}$, $h_{A,2}$, $r_{A,3}$, $h_{A,3}$) for ID_A as in the above. Next $\mathcal{B}$ generates bit $c \in \{0,1\}$. Let $f_4(x) = x^{q+2}$ and let $F_{4,A}(x) = (f_4(x) - f_4(ID_A))/(x - ID_A)$ be a polynomial of degree $q+1$. $\mathcal{B}$ continues to set $u = g'^{(f_4(\alpha) - f_4(ID_A)) r_A}$, $v = Z \times e(g', \prod_{i=0}^{q} g^{F_{4,A,i}\alpha^i})$ and $w = M_c / e(u, h_{A,1})^{1/r_A} v^{r_{A,1}}$ where $F_{4,A,i}$ is the coefficient of x^i in $F_{4,A}(x)$. After setting $\beta = H(u, v, w)$, $\mathcal{B}$ sets $y = e(u, h_{A,2} {h_{A,3}}^{\beta})^{1/r_A} v^{r_{A,2} + r_{A,3}\beta}$, $z = g^{\beta}$. $\mathcal{B}$ sends c_A=(u, v, w, y, z) to $\mathcal{A}$ as the challenge ciphertext.

Let $s = (log_g g') F_{4,A}(\alpha)$. If $Z = e(g_{q+1}, g')$, then $u = g^{s r_A(\alpha - ID_A)}$, $v = e(g,g)^s$, $M_c/w = e(u, h_{A,1})^{1/r_A} v^{r_{A,1}} = e(g, h_1)^s$, and $e(u,z) = e(u,g)^{\beta}$. Since $log_g g'$ and s are uniformly random, $c_A = (u, v, w, y, z)$ is a valid, appropriately-distributed challenge to $\mathcal{A}$.

Phase 2: This phase proceeds as in Phase 1. However $\mathcal{A}$ is restricted from issuing the following queries:

1. $(encrypt, ID_A, M_0)$ and $(encrypt, ID_A, M_1)$
2. $(decrypt, ID_A, c_A)$
3. Any pair of queries $(rkextract, ID_A, ID_{A'})$ and (decrypt, $ID_{A'}$, c'_A) where c'_A is the re-encrypted ciphertext using $rk_{A \to A'}$.

At the end of Phase 2, the adversary $\mathcal{A}$ outputs guesses $c' \in \{0, 1\}$. If $c = c'$, $\mathcal{B}$ outputs 0 (indicating that $Z = e(g_{q+1}, g')$). Otherwise, $\mathcal{B}$ outputs 1.

Probability Analysis and Conclusion: If $Z = e(g_{q+1}, g')$, then the simulation is perfect. Assume that A has made d decryption queries in Phase 1, the average length of a ciphertext be len_c bits. A will guess the bit c correctly with probability $1/2 + d/(2^{len_c}) + \epsilon$ where $d/(2^{len_c})$ is the probability that A has queried $(decrypt, ID_A, c_A)$ in Phase 1 where c_A is the ciphertext of M_c. If Z is uniformly random, (u, v, w, y) is an invalid ciphertext for (ID_A, M_c) and it carries no information regarding the bit c. In this case, A will guess the bit c correctly with probability 1/2. Therefore on overall, A will guess the bit c correctly with probability $1/2(1/2 + d/(2^{len_c}) + \epsilon) + 1/2(1/2) = 1/4 + d/(2 \times 2^{len_c}) + 1/2\epsilon + 1/4 = 1/2 + d/(2 \times 2^{len_c}) + 1/2\epsilon$. Thus A's advantage is non-negligible. B thus can make use of A to solve the truncated decision q-ABDHE problem. This leads to a contradiction since the truncated decision q-ABDHE problem is a well-known hard problem. As a result, our scheme is secure under IND-ID-CCA (even when the proxy and the delegatees are colluding).

6 Implementation of Non-transferable Re-encryption based Encrypted USB/PC File Systems

We implemented a non-transferable re-encryption based file system with three goals in mind. First, to show the correctness of our proposed scheme. Second, to prove that our proposed scheme is acceptable and practical to improve real systems (Encrypted USB/PC file systems are used as an example) to achieve non-transferable property. Third, to assess the performance of proposed scheme.

TrueCrypt: We implemented our file system on the basis of Truecrypt [1]. Truecrypt is a software system for establishing and maintaining an encrypted volume (data storage device). No data stored on an encrypted volume can be read (decrypted) without using the correct password/keyfile(s) or correct encryption keys. Entire file system is encrypted (e.g., file names, folder names, contents of every file, free space, meta data, etc).

Files can be copied to and from a mounted TrueCrypt volume just like they are copied to/from any normal disk (for example, by simple drag-and-drop operations). Files are automatically being decrypted on the fly (in memory/RAM) while they are being read or copied from an encrypted TrueCrypt volume. Similarly, files that are being written or copied to the TrueCrypt volume are automatically being encrypted on the fly (right before they are written to the disk) in RAM.

We choose the Truecrypt because it is open source, and allows us to experiment our scheme on top of it without putting much effort on how to establish the file system interface.

Scenario: An encrypted volume (V_1) is for an employee (U_1) of a company, working on a project. When U_1 is on holiday, another user (U_2) takes up the project from U_1. The problem is how to transfer the data in V_1 to U_2? The most direct way is to let U_2 know U_1's key. Certainly, this introduces security problem, for example, U_2 could distribute U_1's key to other parties without getting U_1's permission. Another possible way is to let U_1 decrypt V_1 into plaintext using his key and encrypt again with key of U_2. However, this poses two problems: the existence of plaintext is dangerous; and it is too time consuming to encrypt and decrypt the huge encrypted disk. Thus, we use "Re-Encryption" scheme based encrypted file systems to solve this problem.

6.1 Overview of Non-transferable Re-encryption based Encrypted File Systems

Two kinds of Non-transferable Re-encryption based encrypted file systems are implemented: Non-transferable Re-encryption based Encrypted PC File System and Non-transferable Re-encryption based Encrypted USB File System. We call them *NTR-PC-FS* and *NTR-USB-FS* for short respectively. Each of them has three different encrypted volume creation ways: password only, key files only, and password and keyfiles together.

Overview of NTR-PC-FS:

- **Password only**
  NTR-PC-FS first creates a virtual encrypted volume on PC. The virtual volume is encrypted using the password, encryption algorithm and hash function chosen by U_1. When using the encrypted volume, user need to input the correct password, and mount encrypted volume as a real disk. After that, when user open a file/project stored on a volume (or when user write/copy a file to/from the volume) user will not be asked to enter the password again.
  When U_1 wants U_2 to take up the project from him, the system proceeds as shown in figure 3.
  1. U_1 publishes the encrypted virtual volume on an untrusted content server of the company. The content server makes the encrypted virtual volume available to everyone.
  2. U_2 downloads the encrypted virtual disk from the content server.

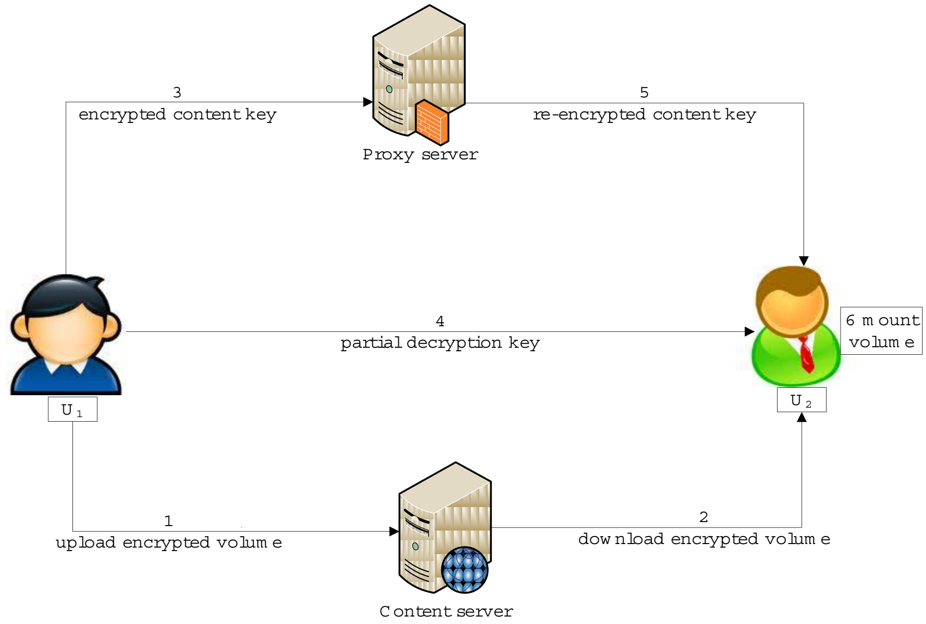


Fig. 3 Proxy Re-Encryption

3. To let U_2 access the encrypted virtual volume, U_1 uses the NTR-PC-FS transforms the password into a symmetric *content key* $\in G_T$ first. Then encrypts the *content key* with U_1's asymmetric public key using the encryption algorithm in section 4.2.
4. U_1 sends U_2 a partial decryption key, and then communicates with an untrusted proxy to re-encrypt the encrypted content key.
5. Proxy uses re-encryption key which is generated by PKG to re-encrypt the content key.
6. After obtaining the re-encrypted content key, NTR-PC-FS on U_2's PC uses U_2's private key, the partial decryption key, and the re-encrypted content key to mount the volume.

Now, U_2 is able to access the project in virtual volume.

- **Keyfile only**

  The difference from the previous volume creation way is that the virtual volume is encrypted using the keyfile randomly generated by NTR-PC-FS or chosen by U_1. The keyfile can be a file on PC or a portable storage device. When using the encrypted volume, user needs to locate the correct keyfile, and mount the encrypted volume as a real disk. For the whole system work flow, please see figure 4.

  To let U_2 access the encrypted virtual volume,

  1. This step is as the same as the password only method.
  2. This step is as the same as the password only method.

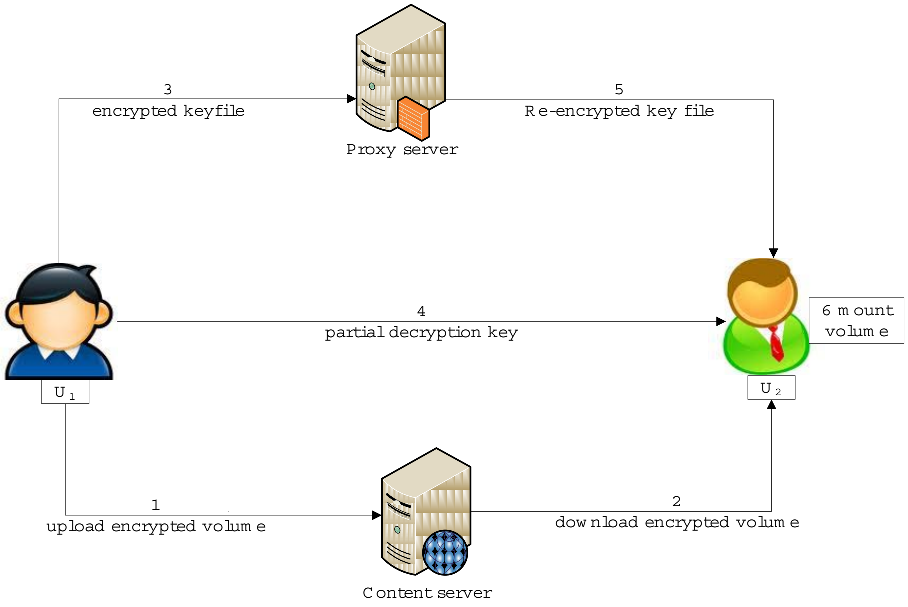


Fig. 4 Proxy Re-Encryption

3. NTR-PC-FS encrypts the keyfile with U_1's asymmetric public key using the encryption algorithm in section 4.2.
4. U_1 sends U_2 a partial decryption key, and then communicates with an untrusted proxy to re-encrypt the encrypted keyfile.
5. Proxy uses re-encryption key which is generated by PKG to re-encrypt the keyfile.
6. After obtaining the re-encrypted keyfile, NTR-PC-FS on U_2's PC uses U_2's private key, the partial decryption key, and the re-encrypted keyfile to mount the volume.

Now, U_2 is able to access the project in virtual volume.

- **Password and Keyfile together**

  The difference from the previous two volume creation ways is that the virtual volume is encrypted using the password chosen by U_1 and keyfile randomly generated by NTR-PC-FS or chosen by U_1. When using the encrypted volume, user need to input the correct password and locate the correct keyfile, and mount the encrypted volume as a real disk.

  To let U_2 access the encrypted virtual volume, NTR-PC-FS encrypts the keyfile and the password respectively with U_1's asymmetric public key using the encryption algorithm in section 4.2. U_1 sends U_2 a partial decryption key, and then communicates with an untrusted proxy to re-encrypt the encrypted keyfile and password. Proxy uses re-encryption key which is generated by PKG to re-encrypt the keyfile and password respectively. After obtaining the re-encrypted keyfile and password, NTR-PC-FS on U_2's PC uses U_2's private key, the partial decryption key, the re-encrypted password, and the re-encrypted keyfile to mount the volume. Now, U_2 is able to access the project in virtual volume.

Overview of NTR-USB-FS:

- **Password only**
  The differences from the previous NTR-USB-FS (Password only) are that the encrypted volume is created on a USB device, not on a PC; and when U_1 wants U_2 to take up the project from him, U_1 does not need to publish the encrypted virtual volume on an untrusted content server of the company. He just passes the USB device to U_2. For the whole work flow, please see figure 5.

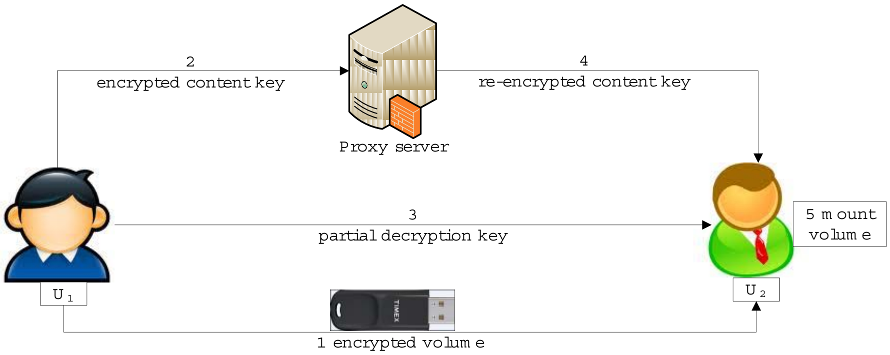


Fig. 5 Proxy Re-Encryption

- **Keyfile only** This volume creation way is similar to the one of NTR-PC-FS. For the differences, please see the above ”Overview of NTR-USB-FS (Password only)”.
- **Password and Keyfile together** This volume creation way is similar to the one of NTR-PC-FS. For the differences, please see the above ”Overview of NTR-USB-FS (Password only)”.

Main Advanced Security Features:

1. We reduces the trust on the proxy server, the content server and PKG. The proxy server re-encrypts the encrypted content key and keyfiles, but it never gets to know the plaintext of the content key and keyfiles; The content server keeps the encrypted volume, but it is unable to access it; The PKG generates the partial private keys for users, but it does not know the whole private keys of users. In case the proxy server, the content server and PKG are compromised, attacker still cannot gain access to the password, keyfiles, and encrypted volume. Thus the proposed file system fundamentally changes the security of the general file systems, because in general file systems, the security relies on the trust of a server operator

or a proxy server; in our proposed file system, the security relies on the strength of the secure cryptosystem.

2. U_2 is not required to input the password or locate the keyfile to mount the encrypted volume got from U_1, since he does not know the password or the keyfile of U_1. The content key decryption, keyfile decryption and volume mounting are proceed by the NTR-PC-FS when provided U_2's private key, the partial decryption key, the re-encrypted content key and the re-encrypted keyfile. The content key and keyfiles are never leaked outside, which is ensured by the security of Truecrypt itself.
3. NTR-PC-FS never saves any decrypted data to a disk - it only stores them temporarily in RAM (memory). Even when the volume is mounted, data stored in the volume is still encrypted. When you restart Windows or turn off your computer, the volume will be dismounted and all files stored on it will be inaccessible (and encrypted). Even when power supply is suddenly interrupted (without proper system shut down), all files stored on the volume will be inaccessible (and encrypted).
4. U_2 is unable to re-delegate the access right to the encrypted volume to others, because he does not know the password or the keyfile of U_1 (discussed in point 2). The only way for him to re-delegate is to expose his own private key, however, U_2 would not be so stupid to run the risk.

6.2 Implementation and Performance Analysis

Our file system consists of five parties: proxy server, content server, PKG, disseminator and recipient. PKG is responsible for key generation. An untrusted proxy is used to manage the dissemination control, and an untrusted content server is used to store encrypted volume for disseminator. Disseminator plays the role of delegator in re-encryption scheme, and recipient plays the role of delegatee in re-encryption scheme. We use our proxy re-encryption scheme to grant encrypted volume access right to legal recipient.

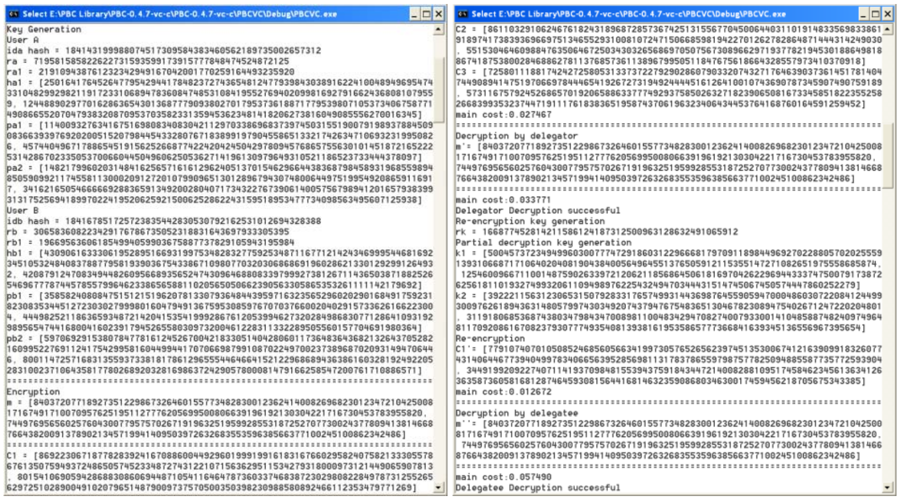
We use two Intel Core 2 Duo CPU E6750 at 2.66GHz with 3GB RAM PCs as the proxy server and the content server. The scheme is implemented in C, with all pairing operations implemented using PBC Library [12]. Note that we do not measure time for setup, private key correctness check, key generation or partial decryption key generation, since these algorithms are performed only once, at initialization time. As in [3], measurements do not take into account the transmission time also, since the encryption, decryption and re-encryption time are the major concerns in re-encryption schemes. We use 512-bit size for order of the base field in proxy re-encryption. Experiments were repeated 10 times using random input points over which timings were averaged.

Our implementation is actually divided into four executable programs. To facilitate reading and checking on the correctness, we combine the four programs into one. A screenshot is shown in figure 6 as below. In the figure, user A represents the delegator, and user B represents the delegatee. We can see the

Table 2 Efficiency Comparison of Ateniese PRE scheme and our proposed scheme

	Ateniese [3]	our scheme
Parameter size	512-bit	512-bit
Encryption	7.7 ms	27.1 ms
Decryption (by delegator)	21.9 ms	33.4 ms
Re-encryption	21.7 ms	12.6 ms
Decryption (by delegatee)	3.4 ms	55.4 ms

message m=m'=m", which means that the message m is correctly decrypted by A and B.



```
Key Generation
User A
ida hash = 184143199988074517309584383460562189735002657312
ra = 719581585822622731593599173915777848474524872125
ra1 = 219109438761232342949167042001770259164493235920
ha1 = [2501641764526477954294417848237274365481247793984303891622410048949695474
33104829929821191723310689478360847485310841955276940209981692791662436808107955
9, 12448890297701628636543013687779093802701795373618871779539807105373406758771
49086655207047938320870953703582331359453623481418206273816049085556270016345]
pa1 = [1140093276341675169808340830421129703386968373974503155190079198937884509
08366393976920200515207984454332807671838991979045586513321742634710693231995082
6, 45744049671788654519156252668774224204245042978094576865755630101451872165222
53142867023350537006604450496062505362714196130979649310521186523733444378097]
pa2 = [1482179960203148416256571616129624051370154629664438368798458931968559894
85059099211745581130002091272010799096513012896794307480064497519954920865911691
7, 34162165054666669288365913492002804071734322767390614005756798941201657938399
31317525694189970224195206259215006252862243159518953477734098563495607125938]
User B
idb hash = 184167851725723835442830530792162531012694328388
rb = 306583608223429176786735052318831643697933305395
rb1 = 196695636061854994059903675887737829105943195984
hb1 = [4309061633306195289516693199753482832775925348711677121424346999544681692
34510532484083788779581939036754338671098077032030686869196028621330129299126493
2, 42087912470834944826095668935652474309646880833979992738126711436503871882526
546967778744578557996462338656588110205650506623905633058653532611111142179692]
pb1 = [3585824080847515121519620781330793648443959716323565296020290168491759231
82308353445127230302799980160479491367595308597670703766002040291573362616622300
4, 44498252118636593487214204153541999286761205399462732028498683077128641093192
98956547441680041602391794526558030973200461228311332289505560157704691980364]
pb2 = [5970692915380784778161245267004218330514042806011736483643682132643705282
16099522769112417542995816044994417070669879910870224970023738968702093149470644
6, 80011472571683135593733818178612965554464664152122968689436386160328192492205
28310023710643581778026892032816986372429057800081479166258547200761710886571]
================================================================================
Encryption
m = [840372077189273512298673264601557734828300123624140082696823012347210425008
171674917100709576251951127776205699500806639196192130304221716730453783955820,
74497695656025760430077957570267191963251959928553187252707730024377809413814668
76643820091378902134571994140950397263268355359638566377100245100862342486]
================================================================================
C1 = [86922306718778283924167088600449296019991991618316766029582407582133305578
67613507594937248650574523348727431221071563629511534279318000973121449065907813
, 801541069059428688308606944871054116464787360337468387230298082284978731255265
6297251028900491020796514879009737570500350398230988580892466112353479771269]
```

```
C2 = [86110329106246761824318968728573674251315567704500644031101914833569833861
91897417383936966975134655293100810724715066859819422701262782864871444314249030
, 551530464609884763506467250343032656869705075673089662971937782194530188649818
8674187538002846886278113768573611389679950511847675618664328557973410370918]
C3 = [72580111881742427258053133737227929028607903320743271764639037361451781404
74490894147519706697844465419267273194924444516126410010743690787345907490759189
, 573116757924526865701920658863377749293758502632718239065081673345851822355258
2668399353237447191117618383651958743706196323406434453764168760164591259452]
main cost:0.027467
================================================================================
Decryption by delegator
m'= [840372077189273512298673264601557734828300123624140082696823012347210425008
171674917100709576251951127776205699500806639196192130304221716730453783955820,
74497695656025760430077957570267191963251959928553187252707730024377809413814668
76643820091378902134571994140950397263268355359638566377100245100862342486]
================================================================================
main cost:0.033771
Delegator Decryption successful
Re-encryption key generation
rk = 166877452814211586124187312500963128632491065912
Partial decryption key generation
k1 = [50045737234949960300777472918603122966681797091189844969270228805702025559
1393106687171064020408190438400564964551376505912115355147271082651975558685874,
 1254600966711001487590263397212062118568645061816970426229694433374750079173872
625618110193274993206110949897622543249470344431514745067450574447860252279]
k2 = [39222115631230653150792833176574993144369876455905947000486030722084124499
30097626189436314805799743034920743794767548365130467823089475402671247220204801
, 311918068536874380347984347008981100483429470827400793300141048588748240974964
8117092086167082379307774935408139381619535865777366841639345136556967395654]
Re-encryption
C1'= [77910740701050852468560566341997305765265623974513530067412163909918326077
43140644677394049978340665639528569811317837865597987577825094885587735772593904
, 344919920922740711419370984815539437591843447214008288109517458462345613634126
3635873605816812874645930815644168146323590868034630017459456218705675343385]
main cost:0.012672
================================================================================
Decryption by delegatee
m''= [84037207718927351229867326460155773482830012362414008269682301234721042500
8171674917100709576251951127776205699500806639196192130304221716730453783955820,
 7449769565602576043007795757026719196325195992855318725270773002437780941381466
876643820091378902134571994140950397263268355359638566377100245100862342486]
================================================================================
main cost:0.057490
Delegatee Decryption successful
```

Fig. 6 Implementation Result

To our knowledge, besides our scheme, the "Third Attempt" of re-encryption schemes in [3] is the only one that has been implemented. However, they implemented their scheme using the MIRACL cryptographic library [16], and we used PBC library. Thus various choices, such as parameter sizes and encryption granularity can greatly affect the efficiency of the scheme. To have a more accurate comparison result of scheme efficiency, we re-implement the "Third Attempt" scheme in [3] using PBC library, and compare our experimental result with [3] in Table 2.

Observation: The experimental results presented in Table 2 show that, when compared with Ateniese's re-encryption scheme, our proposed scheme could cut down the re-encryption time by 9.1ms. This is quite a significant reduction especially when the proxy server has to handle a large number of re-encryption requests. Table 2 shows that our proposed scheme requires more decryption and encryption time; however, these overheads are quite acceptable since (i) decryption and encryption are performed on the client sides(delegators and delegatees), and the time is less than 0.1 second which is acceptable for practical use. (ii) Moreover, a tradeoff between efficiency and security is often unavoidable: in order to achieve the non-transferable property, we have designed a more complicated form of ciphertext which requires additional computation for decryption. (iii) As the clients would perform encryption and decryption for only once, the impact of the extra time is insignificant. Therefore, with the substantial reduction in re-encryption time, the proposed scheme can be considered a promising one.

Limitation: We exclude the discussion on secret issues related to "hack" the program of Truecrypt to steal the password.

7 Conclusions

In this paper, we attempt to solve the open problem pointed out in *NDSS* 2005, in proposing a non-transferable proxy re-encryption scheme, and successfully use this new scheme in data dissemination control. With the proposed PRE scheme, the proxy and a delegatee cannot collude to transfer decryption rights. We also introduced two important properties, namely *Non-Key-escrow* and *Non-PKG-despotism*, into the proposed PRE scheme. The principle behind our solution is that instead of ‘prohibiting’ a party to propagate information, we punish the party who illegitimately propagates information by exposing the important secrets of the party. This method is feasible due to the fact that nobody would run the risk of exposing its own secrets to do illegal decryption right transfer. Thus, our ’punish’ method is more practicable and effective than the ’tracing’ method in [11], because it can strongly prevent illegal decryption right transfer from happening, but not just tracing the malicious proxy after the illegal decryption right transfer.

To the best of our knowledge, our paper is the first paper which practically solves the transferable problem, and the first attempt to use non-transferable re-encryption scheme to achieve data non-redissemination.

8 Acknowledgments

We would like to show our deepest gratitude to Sherman S.M. Chow, for all his kindness and help. Without his valuable comment, we could not have solved the difficult part of this paper.

References

1. Truecrypt. *http://www.truecrypt.org/*.
2. S. S. Al-Riyami and K. G. Paterson. Certificateless public key cryptography. In *ASIACRYPT*, pages 452–473, November 2003.

3. G. Ateniese, K. Fu, M. Green, and S. Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. In *NDSS*, pages 29–43, February 2005.
4. M. Blaze, G. Bleumer, and M. Strauss. Divertible protocols and atomic proxy cryptography. In *EUROCRYPT*, pages 127–144, June 1998.
5. D. Boneh and X. Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *EUROCRYPT*, pages 223–238, May 2004.
6. K. Fu. Group sharing and random access in cryptographic storage file systems. *Master's thesis*, May 1999.
7. C. Gentry. Practical identity-based encryption without random oracles. In *EUROCRYPT*, pages 445–464, May 2006.
8. V. Goyal. Reducing trust in the pkg in identity based cryptosystems. In *CRYPTO*, pages 430–447, August 2007.
9. A. Ivan and Y. Dodis. Proxy cryptography revisited. In *NDSS*, February 2003.
10. J. Jin, G.-J. Ahn, H. Hu, M. J. Covington, and X. W. Zhang. Patient-centric authorization framework for sharing electronic health records. In *SACMAT '09: Proceedings of the 14th ACM symposium on Access control models and technologies*, pages 125–134, June 2009.
11. B. Libert and D. Vergnaud. Tracing malicious proxies in proxy re-encryption. In *Pairing*, pages 332–353, September 2008.
12. B. Lynn. Pbc: The pairing-based cryptography library. *http://crypto.stanford.edu/pbc/*.
13. T. Matsuo. Proxy re-encryption systems for identity-based encryption. In *Pairing*, pages 247–267, July 2007.
14. K. Niu, X. A. Wang, and M. Q. Zhang. How to solve key escrow problem in proxy re-encryption from cbe to ibe. In *DBTA*, pages 95–98, April 2009.
15. R. Sakai and M. Kasahara. Id based cryptosystems with pairing on elliptic curve. *Cryptology ePrint Archive, Report 2003/054*, 2003.
16. M. Scott. Miracl. shamus software. *http://www.shamus.ie/*.
17. N. Shang, M. Nabeel, F. Paci, and E. Bertino. A privacy-preserving approach to policy-based content dissemination. In *ICDE*, pages 944–955, March 2010.
18. J. Singh and J. Bacon. Event-based data dissemination control in healthcare. In *Electronic Healthcare*, pages 167–174, September 2008.
19. R. K. Thomas and R. Sandhu. Towards a multi-dimensional characterization of dissemination control. In *Policies for Distributed Systems and Networks, IEEE International Workshop on*, pages 197–200, June 2004.
20. X. A. Wang and X. Y. Yang. Identity based broadcast encryption based on one to many identity based proxy re-encryption. In *IEEE International Conference on Computer Science and Information Technology*, pages 47–50, August 2009.
21. X. A. Wang and X. Y. Yang. Proxy re-encryption scheme based on bb2 identity based encryption. In *IEEE International Conference on Computer Science and Information Technology*, pages 134–137, August 2009.
22. X. A. Wang and X. Y. Yang. Proxy re-encryption scheme based on sk identity based encryption. In *IAS*, pages 657–660, August 2009.
23. X. A. Wang, X. Y. Yang, and F. G. Li. On the role of pkg for proxy re-encryption in identity based setting. *Cryptology ePrint Archive, Report 2008/410*, 2008.
24. X. A. Wang, X. Y. Yang, and M. Q. Zhang. Proxy re-encryption scheme from ibe to cbe. In *DBTA*, pages 99–102, April 2009.