applied sciences

Article

Proxy Re-Encryption Scheme for Decentralized Storage Networks

Jia Kan [1,2,†], Jie Zhang [1,†], Dawei Liu [3] and Xin Huang [2,*]

[1] Department of Communications and Networking, Xi'an Jiaotong-Liverpool University, Suzhou 215123, China; jia.kan19@student.xjtlu.edu.cn (J.K.); jie.zhang01@xjtlu.edu.cn (J.Z.)
[2] College of Data Science, Taiyuan University of Technology, Taiyuan 030024, China
[3] Cyber Technology Institute, De Montfort University, Leicester LE1 9BH, UK; dawei.liu@dmu.ac.uk
* Correspondence: huangxin@tyut.edu.cn
† These authors contributed equally to this work.

Abstract: Storage is a promising application for permission-less blockchains. Before blockchain, cloud storage was hosted by a trusted service provider. The centralized system controls the permission of the data access. In web3, users own their data. Data must be encrypted in a permission-less decentralized storage network, and the permission control should be pure cryptographic. Proxy re-encryption (PRE) is ideal for cryptographic access control, which allows a proxy to transfer Alice's ciphertext to Bob with Alice's authorization. The encrypted data are stored in several copies for redundancy in a permission-less decentralized storage network. The redundancy suffers from the outsourcing attack. The malicious resource provider may fetch the content from others and respond to the verifiers. This harms data integrity security. Thus, proof-of-replication (PoRep) must be applied to convince the user that the storage provider is using dedicated storage. PoRep is an expensive operation that encodes the original content into a replication. Existing PRE schemes cannot satisfy PoRep, as the cryptographic permission granting generates an extra ciphertext. A new ciphertext would result in several expensive replication operations. We searched most of the PRE schemes for the combination of the cryptographic methods to avoid transforming the ciphertext. Therefore, we propose a new PRE scheme. The proposed scheme does not require the proxy to transfer the ciphertext into a new one. It reduces the computation and operation time when allowing a new user to access a file. Furthermore, the PRE scheme is CCA (chosen-ciphertext attack) security and only needs one key pair.

Keywords: proxy re-encryption; blockchain; storage; proof-of-replication

check for updates

Citation: Kan, J.; Zhang, J.; Liu, D. ; Huang, X. Proxy Re-Encryption Scheme for Decentralized Storage Networks. *Appl. Sci.* **2022**, *12*, 4260. https://doi.org/10.3390/app12094260

Academic Editor: Arcangelo Castiglione

Received: 14 March 2022
Accepted: 9 April 2022
Published: 22 April 2022

1. Introduction

Blockchain technology has been actively developing in recent years. A decentralized storage network [1] based on the blockchain is a promising application direction. The decentralized storage network would redefine data ownership, privacy, and accessibility. Taking the example of the traffic surveillance cameras, the data may be stored on a decentralized storage network. Therefore, the public can verify that the data exist, but only authorized parties can access it. Multiple institutions (such as insurance companies) to access data require an encryption scheme with access control. Traditional symmetric or asymmetric cryptography cannot meet this requirement, as these schemes require specifying who can decrypt before encrypting. The proxy re-encryption (PRE) is a suitable scheme for data sharing.

PRE allows a user to grant access permission in a cryptographic method. Alice would allow Bob to visit her data under Alice's authorization. However, the ciphertext must be transferred to the new one (Figure 1). In a decentralized storage network, data integration suffers from the challenge of the outsourcing attack. Blockchain consists of many semi-trusted resource providers. When asked for proof, the malicious provider would download the data content from other honest providers on the fly. Proof-of-replication (PoRep) brings the concept against the outsourcing attack. The idea is to encode the user data with a unique key, e.g., the provider's public key. Meanwhile, the encoding algorithm should be expensive, and decoding is cheap, so the resource provider would not drop the replicated data, as regenerating the replication would cost more. Since everyone tends to reduce the cost, the data would lose redundancy without PoRep. PoRep is the mandatory algorithm to convenience the verifiers that the dedicating storage resource is spending.

Existing PRE is not ideal for a decentralized storage network because the extra ciphertext would trigger an expensive replicating operation (Figure 2). Combined with PoRep, the cost of PRE sharing is too high.

We propose a CCA (chosen ciphertext attack)-secure and collusion-resilience (collusion safe) proxy re-encryption scheme for the decentralized storage network (Figure 3).

1. No new ciphertext is generated for the permission grant in a decentralized storage network. It brings down the cost for proof-of-replication in a permission-less decentralized storage network.
2. The collusion-resilience scheme in group algebra requires only one key pair.

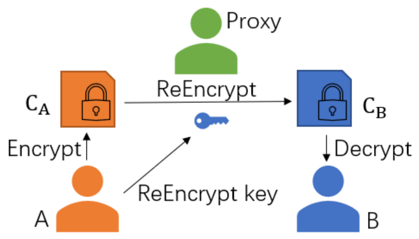

Figure 1. Traditional PRE requires proxy computation to re-encrypt.

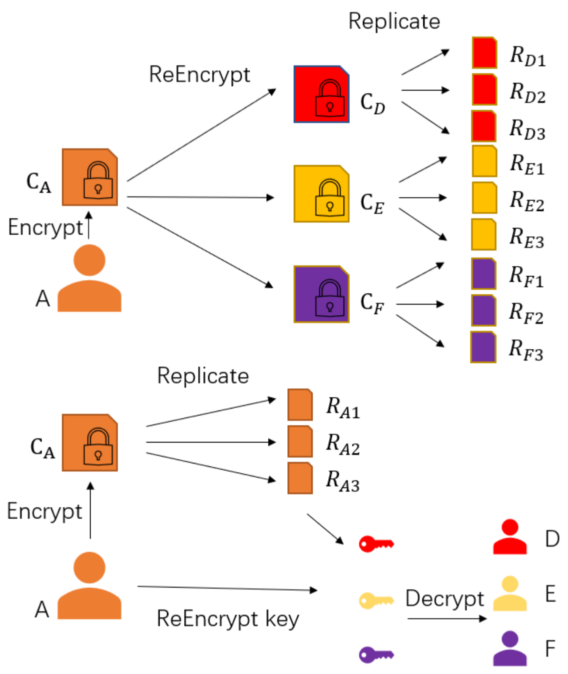

Figure 2. Comparison of the replication in decentralized storage network with or without transferred ciphertext.

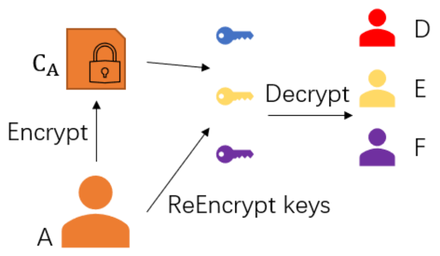

Figure 3. Our PRE scheme use C_A and ReEncrypt key to decrypt.

The rest of the paper is organized as follows. In Section 2, we talk about the detail of decentralized storage networks and proxy re-encryption. In Section 3, we dive into the knowledge, terms, and formula used in this work. The proposed scheme and security analysis are presented in Section 4. In Section 5 we give a practical implementation and show the experiment result. The various extended questions are talked about in the evaluation, Section 6. Finally, Sections 7 and 8 provide the conclusion and future work.

2. Related Work

This section will dive deep into the background of the decentralized storage network, blockchain, and its relationship with proxy re-encryption.

2.1. Decentralized Storage Network

The cloud service is provided by a trusted third party. The data permission is controlled with a centralized mechanism. With the blockchain innovation, the recent study shows that the decentralized storage network is viable [1–4]. In the decentralized storage network, the user is not required to trust any providers, just the cryptography. It brings enormous confidence to the data owner.

The decentralized storage network is ideally built upon a permission-less blockchain. Blockchain miners provide the storage resource. The content that a user uploads to the blockchain is kept by miners. The blockchain makes the storage network permission-less. The miners can freely join or quit.

In the decentralized storage network, permission control must be cryptographic since the storage providers are semi-trusted. The existing public-key crypto is required to specify the target user to decrypt before encryption. Beyond this, PRE allows users to add the target users by re-encrypt the existing ciphertext into a new one. It is identical to permission granting at the application level. Thus, PRE schemes are helpful for blockchain storage.

However, a decentralized storage network must periodically check content integrity. The malicious miner may cheat by fetching content from other honest miners and responding to the checking. The cheater is committing to keep the content but never spending the storage resource. A decentralized storage network must be resilient to the outsourcing attack. Proof-of-replication requires the miners to encode the user content into the replication with each miner's unique identity. It is intended to make the encoding more expensive. The miner is willing to save the replication on disk, as it is impossible to fetch the content from other miners and finish the encoding in the limited time.

2.2. Proxy Re-Encryption and Proof-of-Replication

In 1998, Blaze, Bleumer, and Strauss [5] proposed the first proxy re-encryption scheme based on a cyclic group [6]. In 2010, Weng et al. [7] proposed a CCA and collusion-resilience PRE scheme. After 2010 [8,9], most PRE schemes were based on bilinear pairing [10–12] or lattice algebra structure [13]. The data uploaded must be encrypted in the decentralized storage network. However, most the PRE schemes are generating the new ciphertext during the re-encryption. A ciphertext would be replicated with each miners' unique identity. Any modification of the ciphertext would lead to more expensive PoRep operations. A PRE scheme is ideal not to generate new ciphertext during the frequent permission sharing actions under the decentralize storage network scenario. Thus, we propose the new PRE scheme.

While the new PRE schemes are diving into more complex algebra structure, the use scenarios of PRE are still limited. As business companies back the cloud service, the cloud and mobile do not fully utilize PRE schemes. The encryption will prevent data analysis and take extra cost of storage and computation. Owing to the blockchain, we foresee that blockchain-based decentralized applications will heavily rely on cryptographic schemes. Web3 allows the user to own their data. The decentralized storage network requires a pure cryptographic access control feature. PRE is ideal, but PoRep is mandatory.

The first blockchain, Bitcoin, proposed the proof-of-work (PoW) as the consensus algorithm [14] after PoW was used earlier for anti-spam purpose [15]. Computation was used as the resource for consensus, such as voting. Later, the storage space as a resource was studied, which can be classified into two categories. The proof-of-space intends to replace proof-of-work as the consensus algorithm. This replacement can bring down the cost of electricity by PoW, but junk data needs to be filled in the hard disk so far. Conversely, the proof-of-storage algorithm focuses on storing useful data. However, this algorithm cannot agree on a consensus. It only shows the proof of the data stored. Proof-of-replication is an extension of proof-of-storage, which convinces the owner that the unique storage resource keeps the data. In the permission-less blockchain, the PoRep is the key algorithm. It is nice to design schemes working with PoRep for the storage features. Filecoin uses SDR as the PoRep encoding and proves with the zero-knowledge-based algorithm [2,3]. Filecoin lets users decide how to encrypt their data. Therefore, there is no cryptographic access control for decentralized storage networks yet. The improved PRE scheme is worth studying.

In 1998, Blaze, Bleumer, and Strauss [5] proposed the first proxy re-encryption scheme. The ciphertext can be re-encrypted into another by the proxy authorized by the owner. Although the first PRE scheme is not collusion-resilient, it shows the possibility to change the key or password of the ciphertext without decryption. In 2003, Ivan and Dodis [16] proposed the group-based proxy cryptography scheme. In their unidirectional scheme, the secret key is first divided into two parts. This is the main technique that is used for collusion safety by many schemes. This illuminates our idea. In 2009, Shao et al. [17] proposed a CCA-secure scheme without pairing. Their scheme uses double trapdoors with the big prime multiplication as the secret key. One year later, Weng et al. [7] proposed a CCA-secure scheme WDLC10 without pairing. Two key pairs are used to avoid collusion for the secret key in their scheme, which is similar to Ivan and Dodis [16]. In the following years, most CCA-secure schemes were based on bilinear pairing or lattice. The PRE schemes such as AFGH06 [18] and GA07 [19] are based on bilinear pairing. XT10 [20] and ABPW13 [21] are based on lattice (LWE). NAL15a [22] is based on lattice (NTRU).

Let us take a look at how WDLC10 [7] works. To achieve collusion resilience, WDLC10 uses two key pairs. The core idea of preventing collusion is to use the sum of two secret keys instead of one. This results from the fact that the delegatee and proxy can only work together to obtain the sum of the keys, but cannot learn the value of each secret. The two keys are used for two different layers of ciphertext.

Our scheme achieved collusion resilience with one regular public/secret key pair. Under the general concept of asymmetry encryption, the cleartext can be encrypted with a public key, and the ciphertext is decrypted with the corresponding secret key. However, in the scenario of PRE, we slightly changed the definition. Assuming that anyone can create PRE ciphertext with the given public key, the malicious user could keep the crucial internal value which should be discarded during the encryption. The internal value can be used to generate new re-keys, where the access permission to the ciphertext should be controlled by the owner. In this case, the ciphertext generated with Alice's public key may not actually be controlled by Alice. This may lead to security issues. The proposed scheme uses the secret key for encryption and decryption to ensure that only the owner can create the ciphertext. Alice can generate a re-key with her secret key and Bob's public key.

To summarize, BBS98 [5], Dodis and Ivan [16], and WDLC10 [7] use groups. WDLC10 [7] improved many features compared to BBS98 (including the most important feature, collusion resilience). To archive collusion resilience in "hashed" ElGamal, two key pairs are required for WDLC10 [7]. Shao et al. [17] used double trapdoors. For the other PRE schemes, most of them are based on bilinear pairing or lattice.

3. Preliminaries

We briefly talk about preliminary knowledge for the decentralized storage network and proxy re-encryption.

3.1. Outsourcing Attack and Proof-of-Replication

In a permission-less decentralized storage network, whoever joins the network can provide resources and make incoming attacks. The resource providers may make arbitrary attacks to reduce their costs and increase the margin. One of the most critical is the outsourcing attack. The solution to prevent outsourcing attacks is proof-of-replication [1].

The data must have redundancy stored in a permission-less decentralized storage network. An individual resource provider who deleted the local copy of data to save the storage cost makes an outsourcing attack. When the request to retrieve data comes, the resource provider can fetch the content from another provider and send it back to the requester. The data must be preprocessed into the replication format with the unique key to prevent the outsourcing attack. The cost of replication should be more expensive and time-costly than honestly storing the data. The replicated data are hard for another to utilize, as the replication key is unique. Proof-of-replication ensures the dedicating unique physical store for the data.

3.2. Public Key Encryption

Public key encryption has advantages in key management compared with symmetry key encryption. Users only need to keep their secret keys safe instead of memorizing many passwords. RSA and ElGamal (including ECC) are the most commonly used public key encryption schemes. In RSA, we can either encrypt with a public key or secret key and decrypt with the other, respectively. In ElGamal, the public key is for encryption via Equation (2), and the secret key is for decryption via Equation (3). Public key encryption allows anyone to create an encrypted message and send it to the secret key owner to establish secure communication. We mainly focus on ElGamal here:

$$a, r \in \mathbb{Z}_p, \tag{1}$$

$$pk_A = g^a, \tag{2}$$

$$sk_A = a. \tag{3}$$

where pk_A is the public key, and sk_A is the secret key. Ciphertext c is encrypted with the public key pk_A and the clear message m via Equation (4):

$$c = \langle c_0, c_1 \rangle = \langle g^r, m \cdot pk_A^r \rangle = \langle g^r, m \cdot g^{ar} \rangle. \tag{4}$$

The secret key is required for decryption via Equation (5):

$$m = \frac{c_1}{(c_0)^{sk_A}} = \frac{m \cdot g^{ar}}{(g^r)^a}. \tag{5}$$

The ElGamal scheme satisfies the CPA security. Given the same input m, the output c is different each time according to the random value r. To achieve CCA security, validation is required before the decryption. It detects if the adversary had modified the ciphertext.

3.3. Proxy Re-Encryption

We review the model of collusion-resilience PRE. A CCA collusion-resilience proxy re-encryption scheme is an algorithm:

$$(KeyGen, RenKeyGen, Enc, ReEnc, Dec). \tag{6}$$

$KeyGen()$: The algorithm generates the public/secret key pair (pk, sk).

$ReKeyGen(sk_A, pk_B)$: The re-encryption key generation algorithm accepts the secret key sk_A of Alice and the public key pk_B of Bob. It outputs a re-encryption key $rk_{A\to B}$.

$Enc(sk, m)$: The encryption algorithm takes the public key sk and the clear message returns the encrypted message c.

$ReEnc(rk_{A\to B}, c_A)$: The re-encryption algorithm transfers the encrypted message c_A into the ciphertext c_B using the re-encryption key $rk_{A\to B}$. Bob's secret key can decrypt the transformed ciphertext c_B.

$Dec(sk, c)$: The user decrypts the ciphertext with his secret key and encrypted c, e.g., c_A or c_B. It outputs the cleartext m.

Correctness. Correctness is ensured for any $m \in \mathcal{M}$ and any key pair of (pk_A, sk_A), (pk_B, sk_B), following the conditions Equations (7) and (8):

$$Dec(sk_A, Enc(sk_A, m)) = m, \tag{7}$$

$$Dec(sk_B, ReEnc(ReKeyGen(sk_A, pk_B), Enc(sk_A, m))) = m. \tag{8}$$

Security definition. Security for a CCA collusion-resilience PRE scheme is defined in the game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. There are two ciphertexts from the cleartext message for the PRE scheme: encrypted cipher $m \to c_A$ and re-encrypted $m \to c_B$ are required for chosen-ciphertext security.

Phase 1. The adversary $\mathcal{A}$ issues queries $q_1, \ldots, q_m$, of which q_i is one of the following:

- Uncorrupted key generation query: The challenger $\mathcal{C}$ computes $(pk_i, sk_i) \leftarrow KeyGen()$, and sends the pk_i to the adversary $\mathcal{A}$.
- Corrupted key generation query: The challenger $\mathcal{C}$ computes $(pk_j, sk_j) \leftarrow KeyGen()$, and sends the (pk_j, sk_j) to the adversary $\mathcal{A}$.
- Re-encryption key generation query: The challenger $\mathcal{C}$ computes $rk_{1\to 2} \leftarrow ReKeyGen(sk_1, pk_2)$, and sends the $rk_{1\to 2}$ to the adversary $\mathcal{A}$. Here, sk_1 and pk_2 must be from different key pairs. This query allows any key pair except that $\mathcal{A}$ cannot know the value of sk_1.
- Re-encryption query: The challenger $\mathcal{C}$ computes $c_2 \leftarrow ReEnc(rk_{1\to 2}, c_1)$, and sends the ciphertext c_2 to the adversary $\mathcal{A}$.
- Decryption query: The challenger $\mathcal{C}$ computes $m \leftarrow Dec(sk, c)$, and sends the cleartext m to the adversary $\mathcal{A}$. Here, sk cannot be sk_1 or sk_2.

Challenge. After the adversary $\mathcal{A}$ ends up Phase 1 , $\mathcal{A}$ chooses from two equal-length messages $m_0, m_1 \in \mathcal{M}$, and sends to the challenger $\mathcal{C}$.

The challenger $\mathcal{C}$ receives m_0, m_1. $\mathcal{C}$ flips a random coin δ Equation (9), and computes c,

$$\delta \leftarrow \{0, 1\}, \tag{9}$$

$$c \leftarrow Enc(sk_A, m_\delta), \tag{10}$$

then sends the c Equation (10) back to the adversary $\mathcal{A}$.

Phase 2. The adversary $\mathcal{A}$ continues to issue queries $q_{m+1}, \ldots, q_{max}$, which q_i can be one of the queries:

- Uncorrupted key generation query: The challenger $\mathcal{C}$ responses are the same as in Phase 1.

- Corrupted key generation query: The challenger $\mathcal{C}$ responses are the same as in Phase 1.
- Re-encryption key generation query: The challenger $\mathcal{C}$ responses are the same as in Phase 1.
- Re-encryption query: The challenger $\mathcal{C}$ responses are the same as in Phase 1.
- Decryption query: The challenger $\mathcal{C}$ responses are the same as in Phase 1, except that $c \neq c_A$ and $sk \neq sk_A$, or $c \neq c_B$ and $sk \neq sk_B$.

  Guess. The adversary $\mathcal{A}$ outputs $\hat{\delta} \in \{0,1\}$.

Referring to adversary $\mathcal{A}$ as an IND-PRE-CCA adversary, we define the advantage of the adversary $\mathcal{A}$ in attacking scheme Π as

$$\mathbf{Adv}_{\Pi,\mathcal{A}}^{IND-PRE-CCA} = |\Pr[\delta = \hat{\delta}] - \frac{1}{2}|. \tag{11}$$

Definition 1. *A PRE scheme Π is said to be $(t, q_u, q_c, q_{rk}, q_{re}, q_d, \epsilon)$-IND-PRE-CCA secure, if for any t-time, IND-PRE-CCA adversary $\mathcal{A}$ makes at most q_u uncorrupted key generation queries, at most q_c corrupted key generation queries, at most q_{rk} re-encryption key generation queries, at most q_{re} re-encryption queries, and at most q_d decryption queries; thus we have*

$$\mathbf{Adv}_{\Pi,\mathcal{A}}^{IND-PRE-CCA} \leq \epsilon. \tag{12}$$

3.4. Complexity Assumptions

The computational assumption of Diffie–Hellman (CDH) is defined as

Definition 2. *$\mathbb{G}$ is a cyclic multiplicative group with prime order p. The CDH problem is said in group $\mathbb{G}$, given a tuple $(g, g^x, g^y) \in \mathbb{G}^3$ with unknown $x, y \leftarrow \mathbb{Z}_p$, to compute g^{xy}.*

A variant of the CDH problem named divisible computation Diffie–Hellman (DCDH) [23] problem is defined as follows.

Definition 3. *Let $\mathbb{G}$ be a cyclic multiplicative group with prime order p. The DCDH problem in group $\mathbb{G}$ is, given $(g, g^{\frac{1}{x}}, g^y) \in \mathbb{G}^3$ with unknown $x, y \leftarrow \mathbb{Z}_p$, to compute g^{xy}.*

The construction of our chosen ciphertext-secure PRE scheme is based on the assumption of modified computational Diffie–Hellman (mCDH). The mCDH problem is a combination of the CDH problem and the DCDH problem.

Definition 4. *Let $\mathbb{G}$ be a cyclic multiplicative group with prime order p. The mCDH problem in group $\mathbb{G}$ is, given a tuple $(g, g^{\frac{1}{x}}, g^x, g^y) \in \mathbb{G}^4$ with unknown $x, y \leftarrow \mathbb{Z}_p$, to compute g^{xy}.*

Definition 5. *For a polynomial time adversary $\mathcal{B}$, the advantage is defined as solving the mCDH problem in group $\mathbb{G}$:*

$$\mathbf{Adv}_{\mathcal{B}}^{mCDH} = \Pr[\mathcal{B}(g, g^{\frac{1}{x}}, g^x, g^y) = g^{xy}]. \tag{13}$$

4. Proxy Re-Encryption Scheme

Our PRE scheme is adopted with the PoRep in a permission-less decentralized storage network. The ciphertext *ReEnc* is an optional operation in the definition. This CCA and collusion-resilience PRE scheme is based on “hashed" ElGamal. ElGamal is one of the most important asymmetry cryptographic schemes based on CDH assumption. Both discrete logarithm and ECC can be used for the ElGamal implementation.

4.1. Features

Collusion resilience. Collusion resilience (collusion safe) states that the proxy and the delegate (Bob) can collude to obtain the delegator’s (Alice) secret key. In BBS98 [5],

$rk_{A\to B} = \frac{sk_B}{sk_A}$, proxy and delegate (Bob) can calculate $sk_A = rk_{A\to B} \cdot sk_B$. Collusion resilience (collusion safe) is an important feature. In any case, sk_A should be safe because it is related to more than the current ciphertext. All the ciphertexts generated by Alice are bound with the security of sk_A. Our scheme is collusion resilience due to the novel method of re-key generation, inspired by the bidirectional scheme of Ivan and Dodis [16].

Bidirectional. Delegation from $A \to B$ allows re-encryption from $B \to A$. It is observed that unidirectional and bidirectional delegation can be applied in different use cases. It is nice to distinguish between unidirectional and bidirectional proxy re-encryption. The bidirectional PRE refers to the fact that it can generate $rk_{B\to A}$ from $rk_{A\to B}$. WDLC10 [7] used the two layers for unidirectional encryption, where layer 2 cipher can be converted into layer 1 cipher by $rk_{A\to B} = \frac{\Delta}{sk_{A1}+sk_{A2}}$.

The bidirectional scheme means the re-encrypted ciphertext can transfer back to the original cipher. It depends on how the re-encryption key is designed. In BBS98, $rk_{A\to B} = \frac{sk_B}{sk_A}$ and the reversed key $rk_{B\to A} = \frac{sk_A}{sk_B}$ can be easily calculated. Obviously, this reversed encryption key can be applied to all the ciphertext generated by Bob. In the Ivan and Dodis 2003 bidirectional ElGamal scheme, $rk = g^{r(x_2-x_1)}$ also can be reversed, but due to the random r, the reversed key can be only applied to Bob's current ciphertext. Comparing the two scenarios, BBS98's bidirectional feature leads to more privacy issues than Ivan and Dodis [16].

Noninteractive. The generation of the re-encryption key requires Alice to use Bob's public key. Bob is not involved in the interaction of re-key generation.

Proxy invisibility. The user sending messages to Alice does not need to be aware of the existence of the proxy. The same applies to Bob, the delegate.

Key optimality. Bob should keep a constant number of secrets, regardless of how many delegations he accepts.

Nontransitive. A proxy re-encryption scheme is transitive if the proxy has right to re-delegate decryption permission. Moreover, it combines several re-encryption keys to produce a new re-key (e.g., from $rk_{A\to B}$ and $rk_{B\to C}$ one can obtain $rk_{A\to C}$). Our scheme is nontransitive, as generating a re-key requires Alice's authorization to prevent transitive action on a proxy.

Transferability. This property, first considered by Ateniese et al. in [18], catches the inability of collusion of the proxy and the delegates to re-delegate decryption rights (i.e., producing new re-encryption keys). The proxy has rk and Bob knows g^r and sk_B, which can generate a new re-key for another user.

4.2. Proposed Scheme

Setup

In the CCA-secure and collusion-resilience PRE scheme, g is the generator of a cyclic multiplicative group $\mathbb{G}$ of prime order p. sk_A Equation (16) is the secret key and pk_A Equation (15) is the public key of Alice. sk_B Equation (18) is the secret key and pk_B Equation (17) is the public key of Bob.

m is the clear message of l_0 bits length in the binary message space denoted by $\mathcal{M}$. w is the random bits of l_1 length. H is the hash function, where $H_1 : \mathbb{Z}_p \cdot \mathbb{Z}_p \to \mathbb{Z}_p$, $H_2 : \{0,1\}^{l_0} \cdot \{0,1\}^{l_1} \to \mathbb{Z}_p$, $H_3 : \mathbb{G}^2 \to \{0,1\}^{l_0+l_1}$, $H_4 : \mathbb{G} \cdot \{0,1\}^{l_0+l_1} \to \mathbb{Z}_p$.

$KeyGen()$: The key generation algorithm generates the public/secret key pair (pk, sk) for the user:

$$a, b \in \mathbb{Z}_p, \tag{14}$$

$$pk_A = g^a, \tag{15}$$

$$sk_A = a, \tag{16}$$

$$pk_B = g^b, \tag{17}$$

$$sk_B = b. \tag{18}$$

$ReKeyGen(sk_A, pk_B)$: The re-encryption-key-generating algorithm accepts the secret key sk_A from Alice and the public key pk_B from Bob. The algorithm returns the re-encryption key $rk_{A\to B}$.

Re-key $rk_{A\to B} = (\frac{pk_B}{pk_A})^d = (\frac{g^b}{g^a})^d = g^{bd-ad}$, where the pk_A can be derived from sk_A. When re-encrypting $D_B = D_A \cdot rk_{A\to B} = (g^a)^d \cdot g^{bd-ad} = g^{bd}$, the re-key can only be issued by Alice, who knows $d = H_1(sk_A, r)$.

$Enc(sk_A, m)$: The encryption algorithm takes the secret key sk_A, and the clear message m returns the encrypted message c_A via Equation (19):

$$(m||w) \xrightarrow{Enc} c_A = \langle D_A, r, E, F, V, S\rangle. \tag{19}$$

where $D, r, E, F, V, andS$ are defined as Equations (20)–(24):

$$D_A = (pk_A)^d, d = H_1(sk_A, r), r \leftarrow \mathbb{Z}_p, \tag{20}$$

$$E = g^e, e = H_2(m, w), w \leftarrow \{0,1\}^{l_1}, \tag{21}$$

$$F = H_3(g^d, E) \oplus (m||w), \tag{22}$$

$$V = g^v, v \leftarrow \mathbb{Z}_p, \tag{23}$$

$$S = g^s, s = v + sk_A \cdot r. \tag{24}$$

$ReEnc(rk_{A\to B}, c_A)$: The re-encryption algorithm transfers the encrypted message c_A into the ciphertext c_B using the generated re-encryption key $rk_{A\to B}$ via Equation (25). Bob's secret key can decrypt the transformed ciphertext c_B. Here, d is used to generate the permission of delegation. Only the content owner can create new D or rk with sk_A and r.

Before the transferring, the validation $S \overset{?}{=} V \cdot pk_A^r$ of ciphertext should be checked to ensure Alice generates the ciphertext. Otherwise, the algorithm outputs $\perp$:

$$\begin{aligned} c_A \xrightarrow{ReEnc} c_B &= \langle D_B, r, E, F, V, S\rangle \\ &= \langle D_A \cdot rk_{A\to B}, r, E, F, V, S\rangle. \end{aligned} \tag{25}$$

$Dec(sk, c)$: The user decrypts the ciphertext with his secret key and encrypted c, e.g., c_A or c_B. It outputs the cleartext m and random bits w via Equation (26). After decryption, the validation of ciphertext should be checked $E \overset{?}{=} g^{H_2(m,w)}$. If not, the algorithm outputs $\perp$:

$$\begin{aligned} c_B \xrightarrow{Dec} (m||w) &= F \oplus H_3(g^d, E), g^d = D_B^{\frac{1}{sk_B}} \\ &= F \oplus H_3(D_B^{\frac{1}{sk_B}}, E). \end{aligned} \tag{26}$$

In WDLC10 [7], CCA-secure "hashed" ElGamal and modified version is used. The textbook ElGamal is CPA-secure and risky in the rounded attack. To enhance the security, "hashed" ElGamal is applied with a message authenticated mechanism.

4.3. Security Analysis

Our collusion-resilience PRE scheme is CCA-secure in a random oracle model, under the modified-computation Diffie–Hellman (mCDH) assumption [24].

In this section, we prove the scheme under mCDH assumption [24] that any efficient algorithm's mCDH advantage is negligible.

Theorem 1. *Our PRE scheme $\prod$ is IND-PRE-CCA-secure under the assumption of the mCDH [24] in group $\mathbb{G}$, and the Schnorr signature [25] is EUF-CMA-secure in the random oracle model. An* *adversary $\mathcal{A}$, who asks at most q_{H_i} random oracle queries to H_i with $i \in \{1, \ldots, 4\}$, can effectively break the $(t, q_u, q_c, q_{re}, q_d, \epsilon)$-IND-PRE-CCA of our scheme $\prod$, for any $0 < \nu < \epsilon$. Thus we have:*

- *The (t', ϵ')-mCDH problem [24] in group $\mathbb{G}$ can be solved by an algorithm $\mathcal{B}$ with Equations (27) and (28):*

$$\begin{aligned} t' \leq t &+ (q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} \\ &+ q_u + q_c + q_{rk} + q_{re} + q_d)\mathcal{O}(1) \\ &+ (q_u + q_c + 4q_{re} + 3q_d + (2q_d + q_{re})q_{H_2})t_e, \end{aligned} \tag{27}$$

$$\epsilon' \geq \frac{1}{q_{H_3}}(2(\epsilon - \nu) - \frac{q_{H_2} + (q_{H_2} + q_{H_3})q_d}{2^{l_0 + l_1}} - \frac{2q_d + q_{re}}{q}). \tag{28}$$

  where t_e is the exponential running time in the group $\mathbb{G}$.
- *The EUF-CMA security of the Schnorr signature [25] can be broken by an attacker with advantage ν within time t'.*

Proof. It is assumed that the Schnorr signature [25] is (t', ϵ')-EUF-CMA-secure for the probability $0 < \nu < \epsilon$. While the CDH problem (given g, g^x, g^y output g^{xy}) is as hard as the mCDH problem [24] (given $g, g^{\frac{1}{x}}, g^x, g^y$ outputs g^{xy}), this theorem is proved under the mCDH problem [24]. A t-time adversary $\mathcal{A}$ can break the IND-PRE-CCA security of scheme $\prod$ with advantage $\epsilon - \nu$. We show how an algorithm $\mathcal{B}$ solves the (t', ϵ')-mCDH problem [24] in group $\mathbb{G}$. □

Suppose algorithm $\mathcal{B}$ accepts the input of mCDH challenge tuple $(g, g^x, g^{\frac{1}{x}}, g^y) \in \mathbb{G}^4$, and $x, y \leftarrow \mathbb{Z}_p$ is unknown. Algorithm $\mathcal{B}$ plays the role of challenger playing the IND-PRE-CCA game with adversary $\mathcal{A}$. Algorithm $\mathcal{B}$'s goal is to output g^{xy}.

Setup. Algorithm $\mathcal{B}$ passes parameters $(p, \mathbb{G}, g, H_1, H_2, H_3, H_4, l_0, l_1)$ to adversary $\mathcal{A}$. H_1, H_2, H_3, H_4 are random hash oracles controlled by the algorithm $\mathcal{B}$.

Hash Oracle Queries. Adversary $\mathcal{A}$ may send the queries to random oracle H_1, H_2, H_3, and H_4 at any time. Algorithm $\mathcal{B}$ has four empty lists H_1^{list}, H_2^{list}, H_3^{list}, and H_4^{list} initially, used for storing the query parameters and result value tuples.

- H_1 queries. With the parameters (a, r), if this query exists in the H_1^{list} as a tuple (a, r, d), output the value d as the result to adversary $\mathcal{A}$. Otherwise, choose $d \leftarrow \mathbb{Z}_p$ and add the tuple (a, r, d) to the hash list H_1^{list}, and respond with $H_1(a, r) = d$ to adversary $\mathcal{A}$.
- H_2 queries. With the parameters (m, w), if this query exists in the H_2^{list} as a tuple (m, w, v), output the value v as the result to adversary $\mathcal{A}$. Otherwise, choose $v \leftarrow \mathbb{Z}_p$ and add the tuple (m, w, v) to the hash list H_2^{list}, and respond with $H_2(m, w) = v$ to adversary $\mathcal{A}$.
- H_3 queries. With the parameters (g^d, E), if this query exists in the H_3^{list} as a tuple (g^d, E, α), output the value α as the result to adversary $\mathcal{A}$. Otherwise, choose $\alpha \leftarrow \{0,1\}^l$ and add the tuple (g^d, E, α) to the hash list H_3^{list}, and respond with $H_3(g^d, E) = \alpha$ to adversary $\mathcal{A}$.
- H_4 queries. With the parameters (E, F), if this query exists in the H_4^{list} as a tuple (E, F, β), output the value β as the result to adversary $\mathcal{A}$. Otherwise, choose $\beta \leftarrow \mathbb{Z}_p$ and add the tuple (E, F, β) to the hash list H_4^{list}, and respond with $H_4(E, F) = \beta$ to adversary $\mathcal{A}$.

Phase 1. The adversary $\mathcal{A}$ sends a series of queries as in the definition of the IND-PRE-CCA game. The algorithm $\mathcal{B}$ holds three hash lists $K_{Uncorrupted}^{list}$, $K_{Corrupted}^{list}$, and R^{list}, answering the adversary $\mathcal{A}$ as follows:

- Uncorrupted key generation query q_u. If the tuple (a, g^a) is not in the hash list $K_{Uncorrupted}^{list}$, the algorithm $\mathcal{B}$ chooses $a \leftarrow \mathbb{Z}_p$; add the tuple (a, g^a) to the hash list $K_{Uncorrupted}^{list}$. Respond with $pk = g^a$ to adversary $\mathcal{A}$.

- Corrupted key generation query q_c. If the tuple (a, g^a) is not in the hash list $K^{list}_{Corrupted}$, the algorithm $\mathcal{B}$ chooses $a \leftarrow \mathbb{Z}_p$; add the tuple (a, g^a) to the hash list $K^{list}_{Corrupted}$. Respond with $(sk, pk) = (a, g^a)$ to adversary $\mathcal{A}$.
- Re-encryption key generation query q_{rk}. The re-key generation is from Alice's secret key and Bob's public key; both key pairs can be uncorrupted or corrupted. It is because in the re-encryption from c_A to c_B, ciphertexts can be decrypted by either sk_A or sk_B. In the case algorithm, $\mathcal{B}$ recovers $(sk_A, pk_A), (sk_B, pk_B)$ from $K^{list}_{Uncorrupted}$ or $K^{list}_{Corrupted}$. Then, algorithm $\mathcal{B}$ generates re-key $rk_{A\to B} = (\frac{pk_B}{pk_A})^{H_1(sk_A, r)} = (\frac{g^b}{g^a})^{H_1(a,r)}$. The tuple $(sk_A, pk_A, sk_B, pk_B, rk_{A\to B})$ is added to the R^{list}. Then, the $rk_{A\to B}$ is returned to adversary $\mathcal{A}$.
  For the challenge purpose, both sk_A and sk_B should be uncorrupted.
- Re-Encryption query q_{re}. Given $rk_{A\to B}$ and $c_A = \langle D_A, r, E, F, V, S\rangle$: If $S \neq V \cdot pk^r_A$, it outputs $\perp$. Otherwise, the algorithm returns the re-encrypted ciphertext $c_B = \langle D_A \cdot rk_{A\to B}, r, E, F, V, S\rangle$ to adversary $\mathcal{A}$.
- Decryption query q_d. The algorithm recovers sk from $K^{list}_{Uncorrupted}$ or $K^{list}_{Corrupted}$. Run $(m, w) = Dec(sk, c)$. If $E = g^{H_2(m,w)}$, give m back to the adversary, otherwise it outputs $\perp$.

Challenge. When adversary $\mathcal{A}$ ends Phase 1, the adversary outputs a target public key pk^* and two equal-length messages $m_0, m_1 \in \{0,1\}^{l_0}$, queries to algorithm $\mathcal{B}$. Algorithm $\mathcal{B}$ responds as follows:

1. Recovers (sk^*, pk^*) from $K^{list}_{Uncorrupted}$ and let $pk^* = g^a := g^{\frac{1}{x}}$.
2. Let $D^* = (pk^*)^d = (g^a)^d := g^y$, so that $(g^a)^d = (g^{\frac{1}{x}})^{xy}$. We can obtain $d = xy$ as $g^a = g^{\frac{1}{x}}$. Then, $g^d = g^{xy}$.
3. As $F = H_3(g^d, E) \oplus (m||w)$ defined, choose $\delta \leftarrow \{0,1\}, w^* \leftarrow \{0,1\}^{l_1}$ and $F^* = H_3(g^d, E^*) \oplus (m_\delta||w^*)$.
4. Return $c^* = \langle D^*, r^*, E^*, F^*, V^*, S^*\rangle$ as the challenged ciphertext to adversary $\mathcal{A}$.

Phase 2. The adversary $\mathcal{A}$ issues the queries as in Phase 1. Algorithm $\mathcal{B}$ responds to those queries to $\mathcal{A}$ as in Phase 1.

Guess. The adversary $\mathcal{A}$ responds a guess $\hat{\delta} \in \{0,1\}$ to $\mathcal{B}$. Algorithm $\mathcal{B}$ calculates $H_3(g^d, E^*) = H_3(g^{xy}, E^*) = F^* \oplus m_{\hat{\delta}} = \hat{\alpha}$. $\mathcal{B}$ looks up the hash list H^{list}_3 for the tuple (g^d, E^*, α) where $\alpha = \hat{\alpha}$, then returns the g^d as the solution g^{xy} to the given mCDH instance.

Analysis. First, let us evaluate the simulation of random oracles. H_1, H_4 are perfect As long as $\mathcal{A}$ does not query (m_δ, w) to H_2 or (g^{xy}, E) to H_3, so H_2 and H_3 are perfect . We denote $AskH^*_2$ the event (m_δ, w) has been queried to H_2, and $AskH^*_3$ the event that (g^{xy}, E) has been queried to H_3.

The challenged ciphertext is identically distributed.

Second, the simulation for the re-encryption oracle. The re-encryption query is perfect unless the adversary $\mathcal{A}$ can transfer the ciphertext into the new one without querying hash H_1 to obtain the rk. We denote this event as $ReEncErr$. Since H_1 plays the role of the random oracle, which is queried by adversary $\mathcal{A}$ at most q_{re} times, we have

$$\Pr[ReEncErr] \leq \frac{q_{re}}{q}. \tag{29}$$

Third, the simulation for the decryption oracle. Suppose that $(pk, c), c = (D, r, E, F, V, S)$ is a valid ciphertext, as the validation $S \overset{?}{=} V \cdot pk^r_A$ of ciphertext can be checked. There is still a chance that c can be generated by querying other random values to H_3 instead of g^d, where $d = H_1(sk_A, r)$. Denote $Valid$ to be an event that c is valid. Let $AskH_3$ be the event that (g^d, E) has been queried to H_3 and $AskH_2$ be the event that (m, w) has been queried to H_2. Then, we have

$$\begin{aligned}
&\Pr[\mathit{Valid}|\neg \mathit{AskH}_2] \\
&= \Pr[\mathit{Valid} \wedge \mathit{AskH}_3|\neg \mathit{AskH}_2] \\
&\quad + \Pr[\mathit{Valid} \wedge \neg \mathit{AskH}_3|\neg \mathit{AskH}_2] \\
&\leq \Pr[\mathit{AskH}_3|\neg \mathit{AskH}_2] + \Pr[\mathit{Valid}|\neg \mathit{AskH}_3 \wedge \neg \mathit{AskH}_2] \\
&\leq \frac{q_{H_3}}{2^{l_0+l_1}} + \frac{1}{q},
\end{aligned} \tag{30}$$

similarly,

$$\begin{aligned}
&\Pr[\mathit{Valid}|\neg \mathit{AskH}_3] \\
&= \Pr[\mathit{Valid} \wedge \mathit{AskH}_2|\neg \mathit{AskH}_3] \\
&\quad + \Pr[\mathit{Valid} \wedge \neg \mathit{AskH}_2|\neg \mathit{AskH}_3] \\
&\leq \Pr[\mathit{AskH}_2|\neg \mathit{AskH}_3] + \Pr[\mathit{Valid}|\neg \mathit{AskH}_2 \wedge \neg \mathit{AskH}_3] \\
&\leq \frac{q_{H_2}}{2^{l_0+l_1}} + \frac{1}{q}.
\end{aligned} \tag{31}$$

Thus, we have

$$\begin{aligned}
&\Pr[\mathit{Valid}|\neg \mathit{AskH}_2 \vee \neg \mathit{AskH}_3] \\
&\qquad \leq \Pr[\mathit{Valid}|\neg \mathit{AskH}_2] + \Pr[\mathit{Valid}|\neg \mathit{AskH}_3] \\
&\qquad \leq \frac{q_{H_2} + q_{H_3}}{2^{l_0+l_1}} + \frac{2}{q}.
\end{aligned} \tag{32}$$

Denote DecErr as the event that $\mathit{Valid}|(\neg \mathit{AskH}_2 \vee \neg \mathit{AskH}_3)$ happens during the entire simulation. The q_d times of decryption queries have been issued to a decryption oracle, and we have

$$\Pr[\mathit{DecErr}] \leq \frac{(q_{H_2} + q_{H_3})q_d}{2^{l_0+l_1}} + \frac{2q_d}{q}. \tag{33}$$

Denote Good as the event $\mathit{AskH}_3^* \vee (\mathit{AskH}_2^*|\neg \mathit{AskH}_3^*) \vee \mathit{ReEncErr} \vee \mathit{DecErr}$. If Good has not happened, the adversary $\mathcal{A}$ cannot gain any advantage in guessing δ from m_0, m_1, due to the random E as one of the input of $H_3(g^d, E)$ and $E = g^e = g^{H_2(m,w)}$ is generated with the random bits $w \leftarrow \{0,1\}^{l_1}$. We have $\Pr[\delta = \delta'|\neg \mathit{Good}] = \frac{1}{2}$

$$\begin{aligned}
&\Pr[\delta = \delta'] \\
&= \Pr[\delta = \delta'|\neg \mathit{Good}] \Pr[\neg \mathit{Good}] + \Pr[\delta = \delta'|\mathit{Good}] \Pr[\mathit{Good}] \\
&\leq \frac{1}{2} \Pr[\neg \mathit{Good}] + \Pr[\mathit{Good}] \\
&= \frac{1}{2}(1 - \Pr[\mathit{Good}]) + \Pr[\mathit{Good}] \\
&= \frac{1}{2} + \frac{1}{2} \Pr[\mathit{Good}],
\end{aligned} \tag{34}$$

and

$$\begin{aligned}
&\Pr[\delta = \delta'] \\
&\qquad \geq \Pr[\delta = \delta'|\neg \mathit{Good}] \Pr[\neg \mathit{Good}] \\
&\qquad = \frac{1}{2}(1 - \Pr[\mathit{Good}]) \\
&\qquad = \frac{1}{2} - \frac{1}{2} \Pr[\mathit{Good}],
\end{aligned} \tag{35}$$

we have

$$|\Pr[\delta = \delta'] - \frac{1}{2}| \leq \frac{1}{2}\Pr[Good]. \tag{36}$$

By the definition, the advantage $(\epsilon - \nu)$ for IND-PRE-CCA adversary:

$$\begin{aligned}
&\epsilon - \nu \\
=&|\Pr[\delta = \delta'] - \frac{1}{2}| \\
\leq&\frac{1}{2}\Pr[Good] \\
=&\frac{1}{2}(\Pr[AskH_3^* \vee (AskH_2^*|\neg AskH_3^*) \vee ReEncErr \vee DecErr]) \\
=&\frac{1}{2}(\Pr[AskH_3^*] + \Pr[AskH_2^*|\neg AskH_3^*] \\
&+ \Pr[ReEncErr] + \Pr[DecErr]).
\end{aligned} \tag{37}$$

Since $\Pr[ReEncErr] \leq \frac{q_{re}}{q}$, $\Pr[DecErr] \leq \frac{(q_{H_2}+q_{H_3})q_d}{2^{l_0+l_1}} + \frac{2q_d}{q}$ and $\Pr[AskH_2^*|\neg AskH_3^*] \leq \frac{q_{H_2}}{2^{l_0+l_1}}$, we obtain

$$\begin{aligned}
\Pr[AskH_3^*]& \\
\geq&2(\epsilon - \nu) - [AskH_2^*|\neg AskH_3^*] \\
&- \Pr[DecErr] - \Pr[ReEncErr] \\
\geq&2(\epsilon - \nu) - \frac{q_{H_2}}{2^{l_0+l_1}} \\
&- \frac{(q_{H_2}+q_{H_3})q_d}{2^{l_0+l_1}} - \frac{2q_d}{q} - \frac{q_{re}}{q} \\
=&2(\epsilon - \nu) - \frac{q_{H_2} + (q_{H_2}+q_{H_3})q_d}{2^{l_0+l_1}} - \frac{2q_d + q_{re}}{q}.
\end{aligned} \tag{38}$$

In event $AskH_3^*$, algorithm $\mathcal{B}$ will be able to solve the mCDH instance, and, consequentially, the following is obtained:

$$\epsilon' \geq \frac{1}{q_{H_3}}(2(\epsilon - \nu) - \frac{q_{H_2} + (q_{H_2}+q_{H_3})q_d}{2^{l_0+l_1}} - \frac{2q_d + q_{re}}{q}). \tag{39}$$

From the description of the simulation, the running time of algorithm $\mathcal{B}$ can be bounded by

$$\begin{aligned}
t' \leq t &+ (q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} \\
&+ q_u + q_c + q_{rk} + q_{re} + q_d)\mathcal{O}(1) \\
&+ (q_u + q_c + 4q_{re} + 3q_d + (2q_d + q_{re})q_{H_2})t_e.
\end{aligned} \tag{40}$$

This completes the proof of Theorem 1.

5. Experiment

In this section, we analyze the computation cost. With the development of the blockchain in the past decade, elliptic curves cryptography (ECC), including ECDSA and 25519, has become the standard user access credential. The group with big integers is less used nowadays. As ECC computation is still heavy even for a modern CPU, we propose the practical implementation which cached the ECC operation to speed up for practice. Otherwise, the encryption and decryption over ECC would take too long a time and become meaningless.

5.1. Schemes Comparison

The two group-based schemes without pairings are using double trapdoors [26] and "hashed: ElGamal [7]. In Table 1, the comparison results indicate that the proposed scheme is slower than WDLC10 for encryption, since "hashed" ElGamal is used. Our scheme does not differentiate the first and the second level of ciphertext. t_{eN} is the time in exponential operation over the N^2 group, where N is the safe prime. Let $N = pq$ be a safe prime modulus, such that $p = 2p' + 1$, $q = 2q' + 1$, and p, p', q, q' are primes. t_e is the time in exponential operation over the group. k is the length of generated key in $KeyGen(1^k)$. k_1 is the hash algorithm $H : \{0,1\}^* \to \{0,1\}^{k_1}$. N_x and N_y are the safe-prime modulus corresponding to the delegator and the delegatee, respectively. The ReEnc is not available in our scheme as we do not transfer the ciphertext, but it only generates a re-key.

Table 1. Comparison with Shao09 and WDLC10.

	Schemes	**Shao09**	**WDLC10**	**Ours**
Compute Cost	ReKenGen	$2t_{eN}$	t_e	t_e
	Enc	$5t_{eN}$	$3t_e$	$5t_e$
	Dec	$4t_{eN}$	$3t_e$	$2t_e$
	ReEnc	$5t_{eN}$	$3t_e$	N/A
Ciphertext Size	1st level	$2k + 3\|N_x^2\| + \|m\|$	$3\|\mathbb{G}\|$	$4\|\mathbb{G}\| + \|m\| + \|w\|$
	2nd level	$k_1 + 3\|N_x^2\| + 2\|N_y^2\| + \|m\|$	$3\|\mathbb{G}\| + \|\mathbb{Z}_q\|$	
Security	Security Level	collusion resistant, CCA	collusion resistant, CCA	collusion resistant, CCA
	Standard model	Yes	Yes	Yes
	Underlying Assumptions	DDH	CDH	CDH

5.2. Practical Implementation

Due to the computation inefficiency of ECC, in the practical implementation, we can use elliptic curves and cyclic multiplicative group together to boost the encryption.

For the practical encryption and decryption, the message is divided into small chunks m, as each time that group operation is involved in the computation, using ECC will cost much longer time. For the calculation $F = H_3(g^d, E) \oplus (m||w)$, where $E = g^e, e = H_2(m, w), w \leftarrow \{0,1\}^{l_1}$, the operation $E = g^e$ will be too expensive if using ECC. On the other hand, ECC provides better security with less secret key length in bits for the user's public key and secret key. Roughly speaking, 160 bits of the ECC secret key are as strong as 1024 bits of the secret key required in RSA or ElGamal over the multiplicative integer group.

It is better to divide and conquer the problem by using both the elliptic curves group and the cyclic multiplicative integer group $g \in \mathbb{G}_p$. In the calculation $D_A = (pk_A)^d, d = H_1(sk_A, r), r \leftarrow \mathbb{Z}_p$, for the user key pair (pk, sk), we use generator $g_{ECC} \in \mathbb{G}_{ECC}$, then $pk = g_{ECC}^{sk}$. Meanwhile $F = H_3(g^d, E) \oplus (m||w)$ the exponential operation g^d in H_3 can be calculated only once in ECC and cached. Another exponential calculation $E = g^e$ must be evaluated for every m and w. Thus, for g^e, the ECC operation will be too heavy. Here, we have the modification below:

$$(m||w) \xrightarrow{Enc} c_A = \langle D_A, r, E, F, V, S \rangle. \tag{41}$$

where D, r, E, F, V, S is defined as:

$$D_A = (pk_A)^d, d = H_1(sk_A, r), r \leftarrow \mathbb{Z}_p, \tag{42}$$

$$E = g^e, e = H_2(m, w), w \leftarrow \{0,1\}^{l_1}, \tag{43}$$

$$F = H_3(g_{ECC}^d, E) \oplus (m||w), \tag{44}$$

$$V = g_{ECC}^{v}, v \leftarrow \mathbb{Z}_p, \tag{45}$$

$$S = g_{ECC}^{s}, s = v + sk_A \cdot r. \tag{46}$$

For the re-encryption $c_A \xrightarrow{ReEnc} c_B$, re-encrypting $D_B = D_A \cdot rk_{A\to B} = (g_{ECC}^{a})^d \cdot g_{ECC}^{bd-ad} = g_{ECC}^{bd}$, where re-key $rk_{A\to B} = (\frac{pk_B}{pk_A})^d = (\frac{g_{ECC}^{b}}{g_{ECC}^{a}})^d = g_{ECC}^{bd-ad}$. For the validation $E \overset{?}{=} g^{H_2(m,w)}$ after decryption, as pk is not involved, no ECC operation needs to be performed.

5.3. Performance Comparison

Python 3.8 is used to implement our PRE scheme and WDLC10. Since Shao09 uses a different theory over the large prime numbers, it was hard to make a fair comparison by choosing the parameters. The code is modified from an open-source pure Python library named python-ecdsa, which is licensed in the public domain.

We implemented the ECC version of our PRE scheme and the practical modification in Python 3.8 and tested it on a MacBook Air of Intel Core i5 at a processor speed of 1.3 GHz. For every 64 bytes of data (including 48 bytes clear message m and 16 bytes of random initialization vector w), we repeat encryption 100 thousand times and note the time cost.

Our scheme is slightly slower than WDLC10 for encrypting in theory, as we described in Table 1. However, for the modification for practical cached exponential operation for g_{ECC}, from Figure 4, we can observe that practical modification can speed up the encryption by avoiding ECC computation. In most cases, elliptic curve-based asymmetry cryptography is commonly used in the signature or key exchange due to its slowness. However, the decentralized storage network shows the scenarios where an asymmetry encryption is required. Meanwhile, it is nice to have the key strength and reasonable encryption/decryption speed. The result of Figure 4 shows that even for a modern CPU, the computation is insufficient for fast encryption. The practical implementation is helpful to speed up the operation while maintaining the security from ECC with less key length.

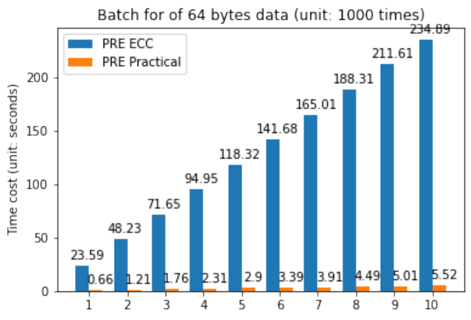

Figure 4. Time cost for PRE ECC and practical encryption for every 1000 times of encryption of 64 bytes data on MacBook 1.3 GHz Intel Core i5.

In the experiment, we use the cyclic multiplicative group over an integer of 196 bits for both ours and WDLC10. The ECC curve is NIST192p. The message m size is 48 bytes and w is 16 bytes since the blake2b outputs up to 64 bytes (512 bits) each time.

In Figure 5, we compare our practical modification PRE scheme with pure-Python-implemented WDLC10. PRE brings useful access control, privacy features, and better key management than symmetry encryption. Unlike WDLC10, there is no requirement for layer 1 and layer 2 ciphertext. Our scheme needs only one public/secret key pair. For the hash function, we use blake2b, which can output a flexible length of the hash digest. For the curve, we use NIST192p. Our proxy re-encryption is ready for practice. Figure 5 shows that with the modification on our scheme, the ECC computing is cached, so our scheme can be slightly faster than WDLC10 even if the theory shows our scheme was slower, shown in Table 1.

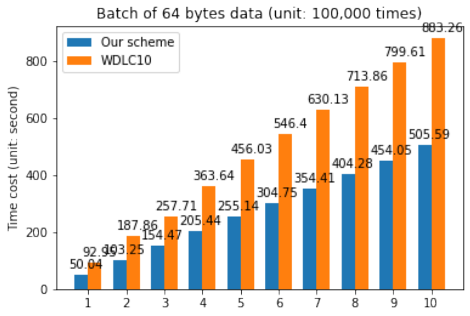

Figure 5. Time cost for our PRE scheme and WDLC10 encryption for every 100 k times of encryption of 64 bytes data on MacBook 1.3 GHz Intel Core i5.

5.4. Performance on the Embedded Device

The PRE encryption and decryption are highly likely to be performed on an embedded device, such as an IP camera or mobile phone. Figure 6 shows the performance of our PRE scheme and WDLC10 on an early model of Raspberry Pi microcomputer. The early version of Raspberry Pi has quite low computation capacity; however, it achieves a reasonable performance. In addition, the permission grant in our scheme does not require transfer of the ciphertext, which would be friendly to the embedded devices. Recently, the embed device has attained a faster CPU with multi-cores. The feature of skipping the re-encryption makes it fit better for the embedded device.

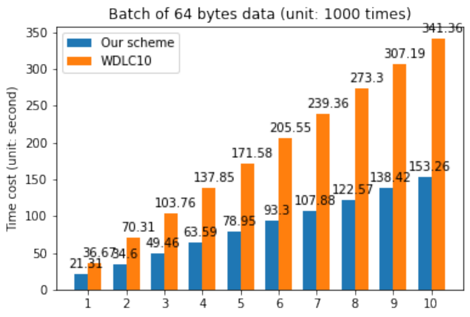

Figure 6. Time cost for our PRE scheme and WDLC10 encryption for every 1000 times of encryption of 64 bytes data on a 1st generation of Raspberry Pi model B 700 Hz.

6. Evaluation

6.1. Remove Access or Corrupted Ciphertext

Thus far, our scheme covers the encryption and decryption of data and shares the data with the public key of another user. We can either choose to generate a $rk_{A\to B}$ or D_B, and it is possible to generate a new ciphertext by replacing D_B with D_A or reuse c_A by placing D_B in a separate file.

$$c_A \xrightarrow{ReEnc} c_B$$

How about removing the access permission of a user? The concept of forwarding secrecy [27] was introduced in cryptography. Strictly, since a message is sent to another one, it is not a secret anymore, as the content could be copied and shared again. It is impossible to revoke a message or erase information with cryptography. However, in practice, people come and leave the organization, and granting or revoking data access permission is the daily operation. If a user intends to expose the critical information g^d_{ECC} to the public, the cipher c is no longer a secret. In those cases, the cryptography method can not ensure security in practice. The only choice is to remove the existing ciphertext to prevent further information leaking.

We provide another ciphertext transfer operation. It is not a part of our scheme but it is useful when transferring the ciphertext into a new one when the message is leaking, and this operation could be applied after any data access permission is revoked.

Transfer operation $c_A \xrightarrow{Transfer} c'_A$, where $c'_A = \langle D'_A, r', E, F', V', S' \rangle$:

$$D'_A = (pk_A)^{d'}, d' = H_1(sk_A, r'), r' \leftarrow \mathbb{Z}_p, \tag{47}$$

$$\begin{aligned} F' &= F \oplus re \\ &= F \oplus H_3(g^d, E) \oplus H_3(g^{d'}, E) \\ &= H_3(g^d, E) \oplus (m||w) \oplus H_3(g^d, E) \oplus H_3(g^{d'}, E) \\ &= H_3(g^{d'}, E) \oplus (m||w), \end{aligned} \tag{48}$$

$$V' = g^{v'}, v' \leftarrow \mathbb{Z}_p, \tag{49}$$

$$S' = g^{s'}, s' = v' + sk_A \cdot r'. \tag{50}$$

The ciphertext transfering key *re* is defined:

$$re = H_3(g^d, E) \oplus H_3(g^{d'}, E). \tag{51}$$

It is safe to send the transfer key to the proxy and transfer the existing ciphertext into a new one. By this operation, the previous ciphertext is discarded, and the new permission of access should be regenerated.

The concept above is also helpful in blockchain storage content for key renewal. Periodically changing the secret key is recommended to avoid potential confidential key leakage. A finance blockchain such as Bitcoin can create another secret/public key pair and transfer existing coin assets to the new wallet address. The signature is the evidence of a coin transaction. As opposed to this, the storage blockchain uses a secret key to decrypt. Losing a key is losing the data unless an algorithm can transfer the old ciphertext to the new one under the new key. Our PRE scheme is suitable for this scenario.

6.2. Search with PRE

The commercial applications are interested in searching [28,29]. Nowadays, searching in data is more than just full-text matching. Complexity algorithms are applied to texts, images, videos, and even speech. A CPA-secure encryption works against searching over the ciphertext in concept. Even if, in the future, the full homomorphic encryption [30] is ready, it might be hard to perform a search over CPA ciphertexts.

With proxy re-encryption, it is possible to design the application that outsources the information processing to the trusted party. Data could be stored safely on the cloud with versions, and the index for the recent version will be processed in-house.

6.3. Applications with PRE

We introduced PRE for decentralized storage network scenarios, which are the fundamental components for lots of blockchain applications. It is possible to build a media store for movies and music based on blockchain. Once a user purchases the movie, he has the right to download the movie file content freely. A purchase record is marked on the blockchain, as evidence of the right to use from the intellectual property owner. It is publicly verifiable. The PRE re-key can be used as evidence. The evidence can be listed publicly and it is meaningful only to the purchaser who owns the secret key.

7. Conclusions

In this paper, we proposed a PRE scheme satisfying the PoRep scenario. Since the PoRep is the key algorithm for a decentralized storage network, the proposed PRE would be an important candidate for a future blockchain storage network. In a decentralized storage network, access control must be cryptography-based. Meanwhile, the decentralized storage network suffers from outsourcing attacks. Proof-of-replication helps to convince users that their content is kept by the dedicated storage resource. The proposed PRE scheme is suited for proof-of-replication, which does not generate the extra ciphertext. The scheme reduces the cost of cryptographic access control. Moreover, our PRE scheme is CCA-secure and only requires one key pair. With the practical implementation, it is reasonably fast to use in applications.

Nowadays, users become used to placing their data on the public cloud. Although the data access is permission-controlled, it might be transparent to cloud storage providers. Employing the PRE scheme will bring true privacy to user data, even if the service provider is semi-trusted.

Due to the computation efficiency, symmetry encryption is widely used for encryption. However, the key management for symmetry encryption is complex, leading to more privacy issues. With the maturity of the PRE scheme, it can bring more flexibility to data storage and access control. Asymmetry encryption will play a more important role in blockchain-based systems.

8. Future Work

In this paper, we proposed the proxy re-encryption for the decentralized storage networks. It is a CCA-secure, collusion-resilience PRE scheme that requires only one key pair. The PRE scheme works under the concept of proof-of-replication, which is the core algorithm of the decentralized storage network, and the proposed scheme is reasonable, fast, and practical. It can be used for mobile and IoT devices. In the future, we will keep working on speeding up the scheme. Furthermore, it is possible to add more features based on the current scheme, e.g., the multiply public keys re-encryption, or the time-limited re-key issuing. To enable searching within PRE is also an interesting topic.

Author Contributions: Funding acquisition, D.L.; Supervision, X.H.; Writing—original draft, J.K.; Writing—review & editing, J.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded in part by the XJTLU Key Program Special Fund under grant number KSF-E-05. This work is supported by Research Project Supported by Shanxi Scholarship Council of China 2021-038, and Applied Basic Research Project of Shanxi Province No. 20210302123130, No. 20210302124273. This work was partially supported by the National Natural Science Foundation of China under Grant No. 62002296; the Natural Science Foundation of Jiangsu Province under Grant No. BK20200250; and the XJTLU Key Programme Special Fund under Grant No. KSF-E-54.

Conflicts of Interest: The authors declare no conflict of interest.

Glossaries

pk	The public key is a part of the key pair. The pk is a big integer internally.
sk	The secret key is a part of the key pair. The sk is a generator of a multiplicative group or the generator point in ECC.
rk	The re-key rk is from the concept of proxy re-encryption, which converts the ciphertext c_A encrypted by Alice's pk_A to a new ciphertext c_B which can be decrypted with Bob's sk_B. The $rk_{A\to B}$ is generated under Alice's permission. It requires Alice's sk_A and Bob's pk_B to generate $rk_{A\to B}$.
M	is the message space that contains all the combinations of message m.
m	is the clear message which will be encrypted. It is represented in binary in length l. In our scheme, we have $m \in \{0,1\}^{l_0}$.
w	is the random bits of length l_1 generated when encrypting message m. This provides CPA-level security that the same message will output different ciphertext under multiple times of encryption.

l	is the message length in bits. We have $l = l_0 + l_1$, where l_0 is the length of m and l_1 is the length of w.
c	is the ciphertext. In proxy re-encryption, beside encryption and decryption, the ciphertext c_A can be transfered to another ciphertext c_B with the $rk_{A \to B}$ under Alice(A)'s permission grant.
d	is the hash value generated by user Alice's secret key sk_A and the random integer r. d is an important value during the calculation which needs to be discarded to keep the ciphertext safe. Otherwise, g^d can be used for decryption without a secret key.
D	is the first element of the ciphertext. When decrypting, g^d will be calculated from D with sk, and when re-encrypting, $rk_{A \to B}$ is applied to D_A to obtain D_B as $D_B = D_A \cdot rk_{A \to B}$.
r	is the second element of the ciphertext. The random r makes sure the value d is random.
E	is a part of the ciphertext. E is calculated from m and w. It is a signature used for satisfying the CCA security. This signature $E \overset{?}{=} g^{H_2(m,w)}$ will be checked after decryption by the user to verify m.
F	is a part of the ciphertext. The concatenation of message m and w is hidden in F. Value g^d is required for encryption or decryption.
v	is a random value used in the Schnorr signature. It is an internal value, which needs to be discarded to keep the secret key safe in s.
V	is a part of the ciphertext. Together with s and pk, it performs the Schnorr signature check before re-encryption.
s	is used for Schnorr's signature to verify the identity of who encrypts the ciphertext. s is hiding in $S = g^s$ to stay safe. Even if m is too long and split into $m = m_1 \vert\vert m_2 \vert\vert ... \vert\vert m_i$, the v and s only need to generate once.
S	is the last element of the ciphertext. It is a value for Schnorr's signature. This signature $S \overset{?}{=} V \cdot pk_A^r$ will be checked before re-encrypting c by the proxy.
H	is the hash function. The hash function is a one-way function, which is easy to calculate the function's output from the input value, but it is hard to obtain the input from the given output. In our scheme, we have H_1, H_2, H_3, and H_4, which take different types of input and return different outputs. $H_1 : \mathbb{G} \cdot \mathbb{Z}_p \to \mathbb{Z}_p$, $H_2 : \{0,1\}^{l_0} \cdot \{0,1\}^{l_1} \to \mathbb{Z}_p$, $H_3 : \mathbb{G}^2 \to \{0,1\}^{l_0+l_1}$, $H_4 : \mathbb{G} \cdot \{0,1\}^{l_0+l_1} \to \mathbb{Z}_p$. In practice, those H functions can use any standard hash function such as sha2 or blake2b with data type converting between bytes and integer.
CDH	is computational Diffie–Hellman (CDH) assumption. The CDH problem in group $\mathbb{G}$ is, given a tuple $(g, g^x, g^y) \in \mathbb{G}^3$ with unknown $x, y \leftarrow \mathbb{Z}_p$, to compute g^{xy}.
mCDH	is the modified computational Diffie–Hellman (mCDH) assumption. Given a tuple $(g, g^{\frac{1}{x}}, g^x, g^y) \in \mathbb{G}^4$ with unknown $x, y \leftarrow \mathbb{Z}_p$, it is hard to compute g^{xy}.
g	In our PRE scheme, g stands for the generator of the multiplicative group.
g_{ECC}	In our PRE scheme, g_{ECC} stands for the generator point of the group of the elliptic curve (ECC).
$\mathbb{Z}_p$	is the non-negative integer set less than a prime integer p, and p is the prime order of a cyclic multiplicative group.
Adversary $\mathcal{A}$	is an efficient adversary who attempts to solve the problem in the security game. Adversary $\mathcal{A}$ issues the queries to the challenger $\mathcal{C}$, and the challenger responds.
Algorithm $\mathcal{B}$	is an algorithm which can break the mCDH problem. In a security reduction, adversary $\mathcal{A}$ transforms the existing problem to the mCDH problem, which algorithm $\mathcal{B}$ can solve, to show the hardness of security.
Challenger $\mathcal{C}$	is the role in the security game that responds to adversary $\mathcal{A}$'s queries following CCA-secure rules.
K^{list}	is a hash list used to simulate the random oracle behavior. The algorithm $\mathcal{B}$ maintains two hash list $K^{list}_{Uncorrupted}$, $K^{list}_{Corrupted}$ and R^{list}, answering the adversary $\mathcal{A}$'s queries.
H^{list}	is a hash list used to simulate the random oracle behavior. Algorithm $\mathcal{B}$ has four lists H_1^{list}, H_2^{list}, H_3^{list}, and H_4^{list}, answering the adversary $\mathcal{A}$'s queries.

References

1. Benet, J.; Dalrymple, D.; Greco, N. *Proof of Replication*; Protocol Labs: San Francisco, CA, USA, 2017.
2. Fisch, B. *PoReps: Proofs of Space on Useful Data*; Protocol Labs: San Francisco, CA, USA, 2018.
3. Fisch, B.; Bonneau, J.; Greco, N.; Benet, J. *Scaling proof-of-replication for Filecoin Mining*; Protocol Labs: San Francisco, CA, USA, 2018.
4. Kan, J. Economic Proof of Work. In *Cryptology ePrint Archive*; Report 2020/1117; Springer: Berlin/Heidelberg, Germany, 2020.
5. Blaze, M.; Bleumer, G.; Strauss, M.J. Divertible protocols and atomic proxy cryptography. In *Theory and Application of Cryptographic Techniques*; Springer: Berlin/Heidelberg, Germany, 1998; pp. 127–144.
6. Elgamal, T. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory* **1985**, *31*, 469–472. [CrossRef]
7. Weng, J.; Deng, R.H.; Liu, S.; Chen, K. Chosen-ciphertext secure bidirectional proxy re-encryption schemes without pairings. *Inf. Sci.* **2010**, *180*, 5077–5089. [CrossRef]
8. Ran, C.; Susan, H. Chosen-ciphertext secure proxy reencryption. In Proceedings of the 14th ACM Conference on Computer and Communications Security, Alexandria, VA, USA, 2 November–31 October 2007; pp. 185–194.
9. Benoît, L.; Damien, V. Unidirectional chosen-ciphertext secure proxy re-encryption. Information Theory. *IEEE Trans.* **2011**, *57*, 1786–1802.
10. Chu, C.-K.; Tzeng, W.-G. *Identity-Based proxy re-encryption without Random Oracles*; Springer: Berlin/Heidelberg, Germany, 2007.
11. Toshihiko, M. Proxy re-encryption systems for identity-based encryption. In *Pairing-Based Cryptography–Pairing 2007*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 247–267.
12. Giuseppe, A.; Karyn, B.; Susan, H. *Key-Private proxy re-encryption*; Springer: Berlin/Heidelberg, Germany, 2009.
13. Elena Kirshanova. Proxy re-encryption from lattices. In P*ublic-Key Cryptography–PKC 2014*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 77–94.
14. Satoshi, N. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Bus. Rev.* **2008**, 21260.
15. Dwork, C.; Naor, M. Pricing via Processing or Combatting Junk Mail. In *International Cryptology Conference*; Springer: Berlin/Heidelberg, Germany, 1992.
16. Ivan, A.; Dodis, Y. Proxy Cryptography Revisited. In *Network and Distributed System Security Symposium*; Springer: Berlin/Heidelberg, Germany, 2003.
17. Nuez, D.; Agudo, I.; Lopez, J. proxy re-encryption. *J. Netw. Comput. Appl.* **2017**, *87*, 193–209. [CrossRef]
18. Ateniese, G.; Fu, K.; Green, M.; Hohenberger, S. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Trans. Inf. Syst. Secur.* **2006**, *9*, 1–30. [CrossRef]
19. Green, M.; Ateniese, G. Identity-Based Proxy Re-encryption. In *Applied Cryptography and Network Security*; Springer: Berlin/Heidelberg, Germany, 2007; 288–306.
20. Keita, X.; Keisuke, T. Proxy re-encryption based on learning with errors. In Proceedings of the 2010 Symposium on Cryptography and Information Security, Amalfi, Italy, 13–15 September 201 0.
21. Yoshinori, A.; Xavier, B.; Le Trieu, P.; Lihua, W. Keyprivate proxy re-encryption under LWE. In *Progress in Cryptology–INDOCRYPT 2013*; Springer: Cham, Switzerland, 2013; pp. 1–18.
22. David, N.; Isaac, A.; Javier, L. NTRUReEncrypt: An efficient proxy re-encryption scheme based on NTRU. In Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '15, New York, NY, USA, 14–17 April 2015; pp. 179–189.
23. Bao, F.; Deng, R.H.; Zhu, H. Variations of Diffie-Hellman Problem. In *International Conference on Information and Communication Security*; Springer: Berlin/Heidelberg, Germany, 2003.
24. Libert, B.; Vergnaud, D. Multi-use unidirectional proxy re-signatures. In Proceedings of the 15th ACM Conference on Computer and Communications Security, Alexandria, VA, USA, 27–31 October 2008.
25. Schnorr, C. Efficient Identification and Signatures for Smart Cards. In *International Cryptology Conference*; Springer: New York, NY, USA, 1989.
26. Shao, J.; Cao, Z. CCA-Secure Proxy Re-encryption without Pairings. In *Public Key Cryptography*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 357–376.
27. Derler, D.; Krenn, S.; Lorünser, T.; Ramacher, S.; Slamanig, D.; Striecks, C. Revisiting Proxy Re-encryption: Forward Secrecy, Improved Security, and Applications. In Proceedings of the Public-Key Cryptography, Rio de Janeiro, Brazil, 25–29 March 2018; pp. 219–250.
28. Boneh, D. *Public Key Encryption with Keyword Search*; Springer: Berlin/Heidelberg, Germany, 2004.
29. Kamara, S.; Lauter, K.E. *Cryptographic Cloud Storage*; Financial Cryptography; Springer: Berlin/Heidelberg, Germany, 2010.
30. Van Dijk, M.; Gentry, C.; Halevi, S.; Vaikuntanathan, V. Fully homomorphic encryption over the integers. In *Theory and Application of Cryptographic Techniques*; Springer: Berlin/Heidelberg, Germany, 2010.