NTMC REPORT

GROUP NO - 13

Broadcast Proxy Reencryption Based on Certificateless Public Key Cryptography for Secure Data Sharing

BT18CSE015 - SANSKRUTI N

BT18CSE040 - NISARG G

BT18CSE044 - KETAN S

Overview of paper

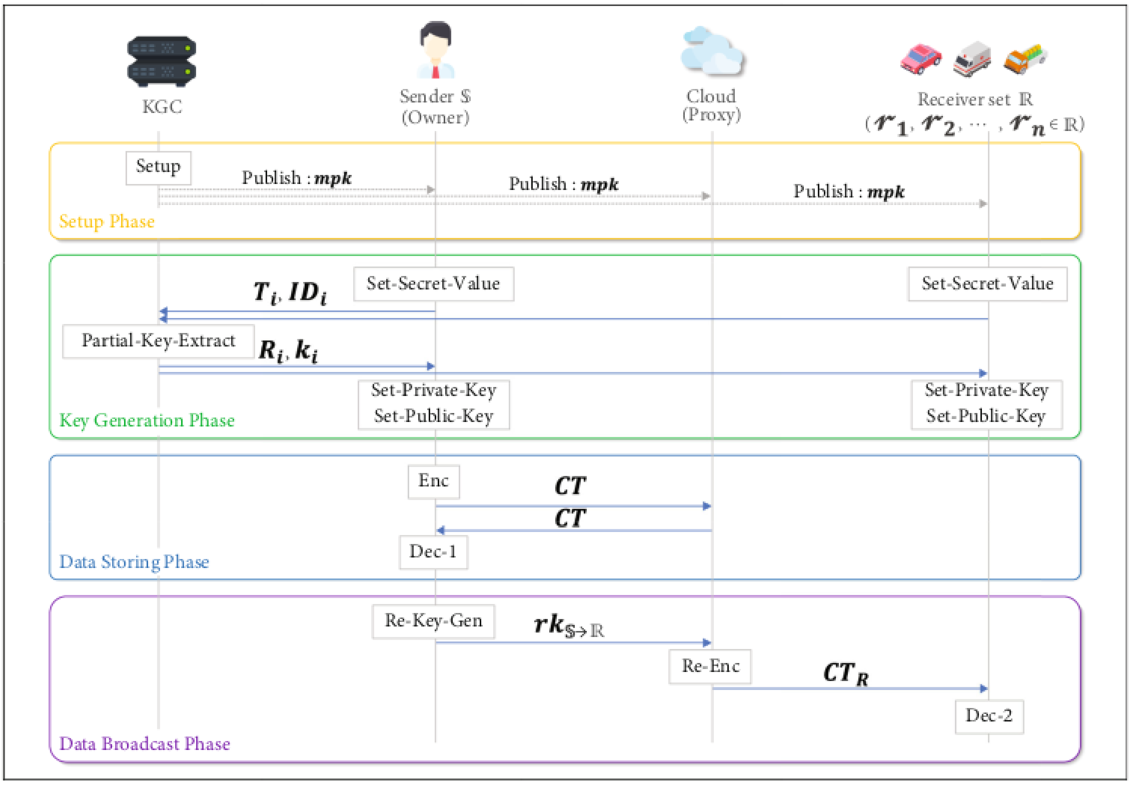

Figure 4: Overview of the proposed scheme.

There are four main phases in this paper:

1. Setup Phase
2. Key Generation Phase
3. Data Storing Phase
4. Data Broadcast Phase

Setup Phase

Setup(λ)⟶(msk, mpk):

This algorithm is an algorithm performed by the KGC.
This phase includes the setup algorithm. This phase is performed by the KGC in advance so that each user can use the cloud. Here, a master public key that can be commonly used by each user and a master secret key known only to the KGC are generated.

```
# 713 is a group there are many such curves defined inside the lib
# NIST/SECG curve over a 224 bit prime field
# 'p/q': 26959946667150639794667015087019630673557916260026308143510066298881,
# 'a': 26959946667150639794667015087019630673557916260026308143510066298878,
# 'b': 18958286285566608000408668544493926415504680968679321075787234672564
def setup(nameId):
    print("\n\n********* KGC ***********")
    # Additive group on E
    G = EcGroup(nameId)
    P, o = G.generator(),G.order()
    # Radom genrator
    d = G.order().random()

    print("d",type(d))

    P_pub = P.pt_mul(d)

    print("d :", d)
    print("P_pub: ", P_pub)
    print("P: ", P)
    set_pub_params(nameId,d)
```

Public parameters are set into the file which can be used by everyone.
Publish the system's master public key mpk = fp, q, l1, l2, E, G, Gq, P, Ppub, H1, H2, H3, H4, H5, H6, H7g and message space M = f0, 1g
Where H1-H7 functions are follows:

H1 : Z∗ q ⟶Z∗ q ,
H2 : f g 0, 1 l1+l2 ⟶Z∗ q ,
H3 : Gq × 0, 1 f g∗⟶Z∗ q ,
H4 : Z∗ q × Z∗ q ⟶f g 0, 1 l1+l2 ,
H5 : Z∗ q × 0, 1 f g∗ × 0, 1 f g∗⟶Z∗ q ,
H6 : Z∗ q ⟶f g 0, 1 l2 ,
H7 : Gq × Gq × 0, 1 f g∗⟶Z∗ q

Key Generation Phase

This phase includes set-secretvalue, partial-key-extract, set-private-key, and set-public-key algorithms. In this phase, each user generates their own private key and public key pair so that they can use the cloud. In this phase, each user communicates with the KGC to receive a partial key and uses the partial key to generate their own public key and private key pair.

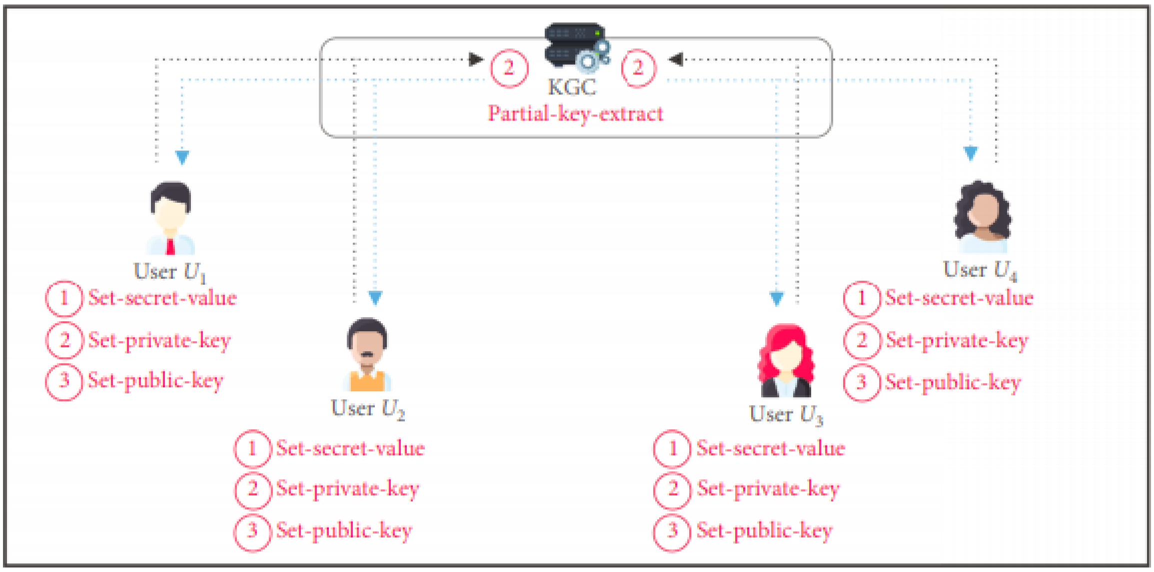

Figure 5: Key generation phase.

Set-secret-value(mpk)⟶(Ti,IDi)

Generation basic user params like IDi. Generates random ti(secret) and uses it to generate Ti.

```python
# User
def setSecretValue():
    print("\n\n********* User ***********")

    nameId, d = get_pub_params()
    IDi = "0001"

    G = EcGroup(nameId)
    P, o = G.generator(),G.order()

    ti = G.order().random()
    Ti = P.pt_mul(ti)

    print("Ti :" , Ti)

    # returning ti to be used by user only
    return Ti,IDi,ti
```

```
********* User ***********
Ti : 02779268f41a82e3347a02ebed9353
```

Partial-key-extract(Ti,IDi, msk, mpk)⟶(Ri,ki)

Using random ri, Ri is generated which is used to generate ki

$$k_i \longleftarrow r_i + dH_3(R_i, T_i, \mathrm{ID}_i) + H_3(dT_i, \mathrm{ID}_i) \pmod{q}$$

ki is the partial secret key generated for the ith user.

Set-private-key:

This algorithm is an algorithm performed by user i. After receiving partial key (Ri , ki) from the KGC, user i verifies these like equations given below. If verification passes, user i compute private key ski = (si , ti)

Verification

$$k_i \bullet P \stackrel{?}{=} R_i + H_7(R_i, T_i, \mathrm{ID}_i)P_{\mathrm{Pub}} + H_3(t_i P_{\mathrm{Pub}}, \mathrm{ID}_i)P$$

Before setting up of user **sk** and **pk** user verifies if everything went correctly.

```
********* KGC ***********
Ri : 0349f99c041baacdc5346431dc0a14
dTi : 03b120d7ab3e53d7438634828e918b
ki: 1352833565931246296220228478494l9938


********* User ***********
Verification
03d4689ba0bf58b6098f0c9838bf0a 03d4689ba0bf58b6098f0c9838bf0a
Verification Successful
```

After that, user i keeps secret ski = (si,ti) as his/her the full private key

Set-public-key:

This algorithm is an algorithm performed by user i. User i keeps pki = (Ri , Ti) as the full public key.

Data Storing Phase

This phase includes the Enc and Dec-1 algorithms. This phase represents the process of the sender encrypting his/her data with his/her public key and storing it in the cloud. In addition, the sender downloads his/her own data stored in the cloud and a decryption process is also included using the private key to obtain the data source again.

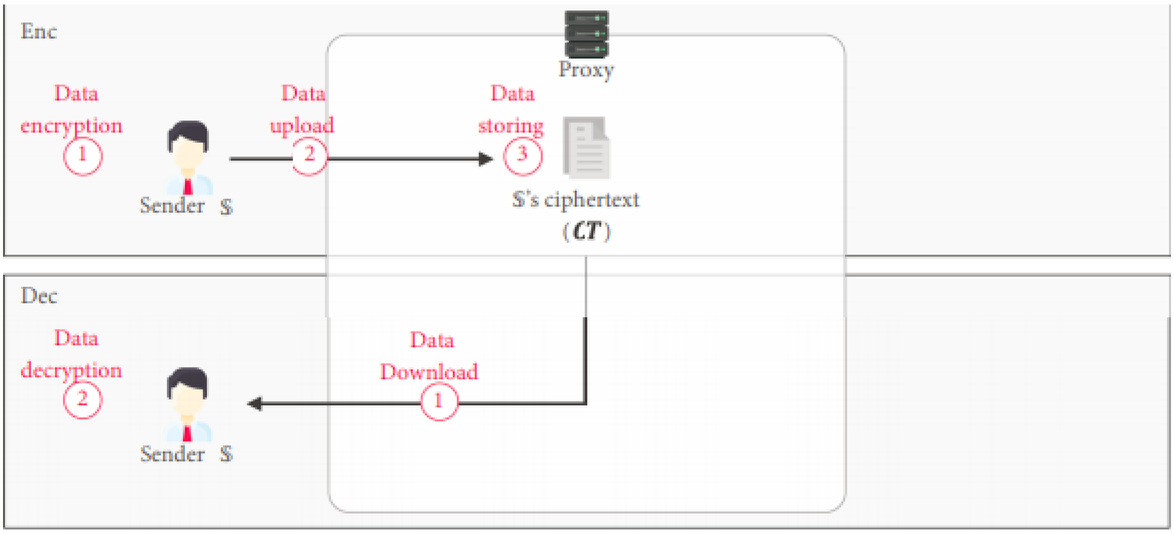

FIGURE 6: Data storing phase.

Encryption:

This algorithm is an algorithm performed by the sender S. Sender S encrypts message m with ciphertext CT by entering his/her public key pkS = (RS,TS) and message m ∈ M. Then, upload the ciphertext CT to the cloud.

Enc (pkS, IDS, m, mpk)⟶CT: This algorithm is performed by the sender, wherein sender S encrypts his/her data m ∈ M using public key pkS to obtain ciphertext CT and uploads it to the cloud.

In this sender S calculates Us using z and pkS = (Rs, Ts)

Us⟵z· (Rs + H7 (Rs,Ts, DS)P_pub + Ts)

Sender S calculates α,θ,C and then generate ciphertext CT⟵(C1, C2)=(Z,C).
Then, the generated CT is uploaded and stored to the cloud.

```python
def encrypt(self,m):
    P, P_pub, Ts, ts, IDs, Rs, ss = self.P, self.P_pub, self.Ti, self.ti, self.IDi, self.Ri, self.si

    w = H7(Rs,Ts, IDs)

    z = H2(bytes(m+str(w), 'utf-8'))
    Z = P.pt_mul(z)

    t21 = P_pub.pt_mul(H7(Rs,Ts,IDs))
    t2 = Rs.pt_add(t21)
    t2 = t2.pt_add(Ts)

    Us =  t2.pt_mul(z)

    alpha = H1(ss+ts)
    theta = H1(alpha) #bug
    C = self.bitwise_xor_bytes(H4(Z,theta) , bytes(m+str(w), 'utf-8'))

    self.w = w
    self.z = z
    self.alpha = alpha

    # tupple of C1,C2
    return (Z, C)
```

Dec-1(CT, skS, IDS, mpk)⟶m:

This algorithm is an algorithm performed by the sender S. The sender S can download the ciphertext CT = (C1, C2)= (Z, C) from cloud. The sender S who has downloaded the ciphertext CT can obtain the plaintext m by decrypting the ciphertext CT with his/her private key skS = (sS, tS)

In this Sender S calculates Us ′ using its private key skS = (sS, tS) and the given ciphertext CT = (C1, C2, C3) , US ′⟵(sS + t S)·C1

Calculate m by inputting C1, C2, θ′

$$(m\|w) \longleftarrow C_2 \oplus H_4\left(C_1, \theta'\right), \tag{11}$$

$$\begin{aligned} \because C_2 \oplus H_4\left(C_1, \theta'\right) &= H_4(Z, \theta) \oplus (m\|w) \oplus H_4\left(C_1, \theta'\right) \\ &= H_4(Z, \theta) \oplus (m\|w) \oplus H_4\left(Z, \theta'\right) = (m\|w), \end{aligned} \tag{12}$$

```python
def decryption1(self, CT):
    C1 = CT[0]
    C2 = CT[1]
    ss, ts, P = self.si, self.ti, self.P

    Us1 = C1.pt_mul(ss+ts)

    alpha = H1(ss+ts)
    theta = H1(alpha)

    # mw = m||w
    mw = self.bitwise_xor_bytes(C2, H4(C1,theta))
    rhs = P.pt_mul(H2(mw))

    print("\n", C1, rhs)

    if(C1 == rhs):
        print("Encrypted message is verified")
    else:
        print("Encrypted message is not verified")
```

Verification II

(7) Verify whether the following equation holds. If not, return $\perp$; otherwise, sender $\mathbb{S}$ keeps plaintext m

$$C_1 \stackrel{?}{=} H_2(m\|H_7(R_{\mathbb{S}}, T_{\mathbb{S}}, ID_{\mathbb{S}}))P, \tag{13}$$

$$\because C_1 = H_2(m\|H_7(R_{\mathbb{S}}, T_{\mathbb{S}}, \mathrm{ID}_{\mathbb{S}}))P = H_2(m\|w)P = zP = Z, \tag{14}$$

Process of Doing verification

```
********* User ***********
03757814e6d3e0c665da1f62e1a315 03757814e6d3e0c665da1f62e1a315
Verification Successful

User Secret Key - si: 158467318898949692210980033317450770, ti: 955228846114802971988698972841719
User Public Key - Ri: 02464d66b05c3984602e26f7630a9d, Ti: 02c776673f89535b8ffc1e5808f743

Cypher Text(CT) : (EcPt(0366ed2c22b6fb269830d502d85041), b'0366ed2c22b6fb269KVS\x16PD_\x01KFQST\x02\x02\x04')

 0366ed2c22b6fb269830d502d85041 0366ed2c22b6fb269830d502d85041
Encrypted message is verified
```

Data Broadcast Phase

This phase includes re-key-gen, re-enc, and dec-2 algorithms. In this phase, the sender generates a reencryption key for a set of recipients and passes it to the proxy. After receiving the reencryption key, the proxy reencrypts the encrypted data and broadcasts it to the recipients. The receiver who has received the broadcast ciphertext can obtain the message by decrypting the ciphertext with their private key as shown

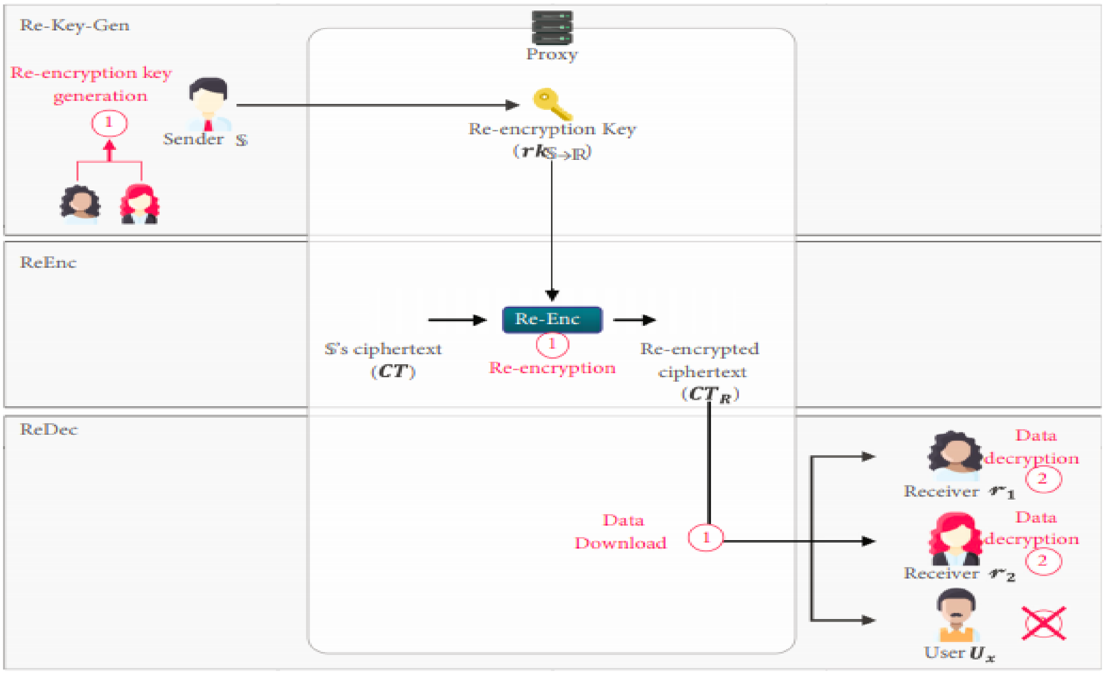

FIGURE 7: Data broadcast phase.

Re-key-gen: This algorithm is executed by sender S to delegate a ciphertext to set of recipients ℝ = (r1, r2, ⋯,rn) of selected receiver rj with identity IDj (1 ≤ j ≤ n). The following steps will be performed in this algorithm

Compute a polynomial f(x) with degree n using β ∈ Z∗ q as follows:
f(x) = Yn(x − Uj)+ β mod q = xn + an−1xn−1 +⋯+a1x + a0
Sender S generates re-encryption key rkS⟶ℝ = (rk1 , rk2 = x, a0, a1,⋯,an−1) and sends rkS⟶ℝ to cloud.

Re-Enc: This algorithm is executed by cloud. This algorithm reencrypts ciphertext CT to ciphertext CTR using reencryption key $rk_{S\longrightarrow\mathbb{R}}$. The following steps will be performed in this algorithm

(1) Compute CTR using ciphertext CT and reencryption key $rk_{S\longrightarrow\mathbb{R}}$

$C_1{}' \longleftarrow C_1$

$C_2{}' \longleftarrow C_2$

$C_3{}' \longleftarrow rk_1 \cdot C_1$,

$C_4{}' \longleftarrow rk_2$

Output CTR = ($C_1{}'$, $C_2{}'$, $C_3{}'$, $C_4{}'$) and send CTR to receivers $\mathbb{R}$

```
def rekeygen(self, Rj, Tj, IDj):
    alpha, z, w, ss, ts, G, P_pub = self.alpha, self.z, self.w, self.si, self.ti,self.G, self.P_pub

    t2 = P_pub.pt_mul(H7(Rj,Tj,IDj))
    t1 = t2.pt_add(Rj)
    t3 = t1.pt_add(Tj)

    Uj = t3.pt_mul(z)
    print("Uj: ",Uj)

    beta = G.order().random()

    uj = H3(Uj,IDj)

    q = 4451685225093714772084598273548427
    # q = EcGroup(nameId).parameters().p

    U = [uj]
    #U is list of Uj
    def polynomial_gen(U,beta,x,q):
        ans = 1
        for ui in U:
            ans *= (x-ui)
        ans += beta % q
        return ans
```

```
def decryption2(self,CT,C31,C41):
    C1,C2,sj,tj,IDj,P = CT[0], CT[1], self.si, self.ti, self.IDi, self.P

    def polynomial(U,x,rk2):
        ans = 1
        for ui in U:
            ans *= (x-ui)
        return rk2

    Uj1 = C1.pt_mul(sj+tj)
    # uj1 = H3(Uji,IDj,w)
    U = [Uj1]
    x = Uj1
    beta1 = polynomial(U,x,C41)
    theta1 = H1(beta1)

    temp = self.bitwise_xor_bytes(C2,H4(C1,theta1)).decode("utf-8")
    m = temp[:len(temp)-3]

    rhs = P.pt_mul(H2(temp.encode("utf-8")))
    print("lhs : ",C1)
    print("rhs : ",rhs)

    if(C1 == rhs):
        print("\nDecrypted Message is verified\n ")
    else:
        print("\nDecrypted Message is not verified\n ")

    return m
```

Dec-2: This algorithm is executed by the selected receiver rj to extract the plaintext from the received ciphertext CTR = (C1 ′, C2 ′, C3 ′, C4). Receiver rj performs following steps:

Generate polynomial fx and compute β′ from
f(x)= xn + an−1xn−1 +⋯+a1x + a0, where β' = f(Uj)
Compute θ′ as input C3 ′ and β′

Calculating m from C1 , C2 and theta

(4) Compute m as input C_1', C_2', θ'

$$m \longleftarrow C_2' \oplus H_4\left(C_1', \theta'\right), \tag{27}$$

$$\begin{aligned} \because C_2' \oplus H_4\left(C_1', \theta'\right) &= H_4(Z, \theta) \oplus m \oplus H_4\left(C_1', \theta'\right) \\ &= H_4(Z, \theta) \oplus m \oplus H_4\left(Z, \theta'\right) = m, \end{aligned} \tag{28}$$

Verification III

(5) Verify message m. If not, return $\perp$; otherwise, receiver i outputs the plaintext m

$$C_1' \stackrel{?}{=} H_2(m)P, \tag{29}$$

$$\because C_1' = H_2(m)P = zP = Z, \tag{30}$$

where $Z = zP$ and $z = H_2(m)$

```
====================== Reciever ==============================

Uj:  03007ca9236f7ee835a141e41fb1f5
Re-encryption key:
 rk1117248203205004952139898555855507021600 rk2 : 183512928163874059528367589992572

Reencryption done sucessfully in Proxy

lhs :  02794b9afe5d44f2b64788bf07e4b5
rhs :  02794b9afe5d44f2b64788bf07e4b5

Decrypted Message is verified

Decrypted message is:  secretmessage
```

Security requirements & Computational Efficiency compared to other schemes

Security requirements

There are a total of 7 security requirements, each of which is confidentiality, integrity, key escrow problem, partial key verifiability, receiver anonymity, and decryption fairness.

Table 1: Comparison of the security requirements.

	Bilinear pairing	Key escrow problem	Receiver anonymity	Re-key-generation
Wang and Yang [41]	Used	Insecure	Offer	KGC/BC
Maiti and Misra [37]	Used	Insecure	Offer	Sender
Sun et al. [38]	Used	Insecure	Offer	Sender
Yin et al. [39]	Used	Insecure	Offer	Sender
Chunpeng et al. [40]	Used	Insecure	Offer	Sender
Proposed scheme	Not used	Secure	Offer	Sender

Requirements: The proposed scheme of this study was designed to satisfy various requirements that the existing schemes do not provide.

Computational Efficiency

Table 2: Comparison of the computation efficiency.

	Enc	Re-key-gen	Re-enc	Dec-2
Wang and Yang [41]	$(2)T_M + (4)T_e + (1)T_P$	$(10+3n)T_M + (1)T_e$	$(6)T_e$	$(7)T_M + (7)T_e + (5)T_P$
Maiti and Misra [37]	$(4)T_M + (3)T_e$	$(3+n^2+n)T_M + (3+n)T_e$	$(1)T_M + (1)T_P$	$(1)T_M + (2)T_P$
Sun et al. [38]	$(2+\mathrm{n})T_M + (5)T_e + (1)T_P$	$(3+n)T_M + (6)T_e + (1)T_p$	$(1+\mathrm{n})T_M + (2)T_P$	$(4+\mathrm{n})T_M + (2)T_e$
Yin et al. [39]	$(4+2n)T_M + (4)T_e$	$(5+n)T_M + (6)T_e$	$(4+3n)T_M + (2)T_e + (2)T_p$	$(7)T_M + (1)T_e + (3)T_P$
Chunpeng et al. [40]	$(2)T_M + (3)T_e$	$(5+n)T_M + (5+\mathrm{n})T_e + (1)T_p$	$(1)T_M + (2)T_p$	$(6)T_M + (2)T_e + (2)T_p$
Proposed scheme	$(2)T_{EM} + (2)T_{EA}$	$(1+n)T_M + (2n)T_{EM} + (2n)T_{EA}$	$(1)T_{EM}$	$(2)T_{EM}$

T_M: computation time of modular multiplication operation; T_{EM}: computation time of ECC multiplication operation; T_{EA}: computation time of ECC point add operation; T_e: computation time of exponent operation; T_p: computation time of bilinear pairing operation.

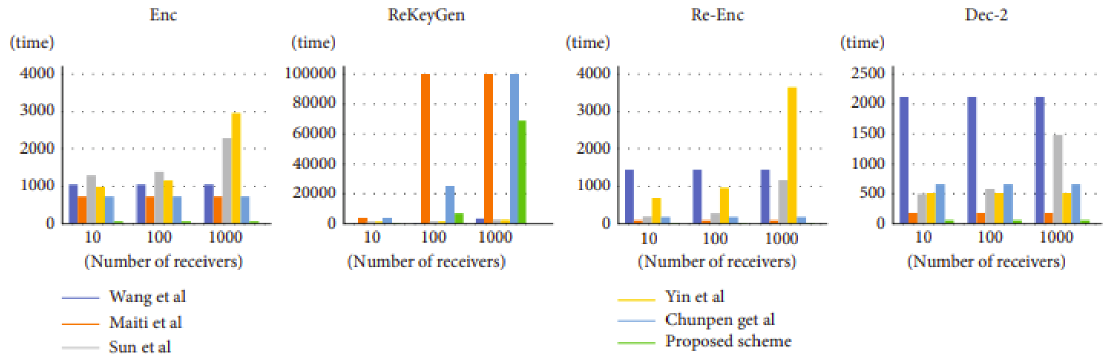

Figure 8: Computation time of schemes.

The proposed scheme uses the CL method and does not use the pairing operation, so that the BPRE can be performed in less time. In addition, it is possible to use BPRE more safely and efficiently by solving the problem of key escrow and recipient anonymity.

Computational efficiency: the proposed scheme of this study was designed with a lower number of calculations compared to the existing schemes . Since the pairing operation is not basically used, BPRE can be performed with less computation time compared to the existing methods.