Abstract—Blaze et al. in 1998 introduced the concept of proxy re-encryption, which allows a proxy to transform a ciphertext computed under Alice's public key into one that can be opened under Bob's decryption key. In CCS'07, Canetti et al. proposed an IND-PRE-CCA2 bidirectional proxy re-encryption scheme, and later, Libert et al. proposed another IND-PRE-CCA2 secure unidirectional proxy re-encryption. However, both schemes rely on the costly bilinear pairings. In CANS'08, Deng et al. proposed the first IND-PRE-CCA2 secure proxy re-encryption without pairings. Later, in PKC'09, Shao et al. proposed another IND-PRE-CCA2 secure proxy re-encryption without pairings. But both of these schemes prove their security in the standard model. In this paper, based on Hanaoka's efficient chosen ciphertext secure public key encryption under the computational Diffie-Hellman assumption, we present two new interesting bidirectional proxy re-encryption schemes, which are restricted IND-PRE-CCA2 secure without pairing nor random oracle.

I. INTRODUCTION

In 1998, Blaze et al. proposed the rst proxy re-encryption scheme based on ElGamal encryption [3]. But this scheme is bidirectional and colluding unsafe. In 2005, Ateniese et al. proposed the rst unidirectional proxy re-encryption schemes. They proposed three attempts to construct their unidirectional proxy re-encryptions. In CCS'07, Canetti et al. proposed the rst IND-PRE-CCA2 secure proxy re-encryption scheme [6] by applying the CHK transformation [5] to the second scheme in [1], [2]. In PKC'08, Libert et al. further extended their research. They proposed the second IND-PRE-CCA2 secure proxy re-encryption scheme based on the third scheme in [1], [2]. In CANS'08, Deng and Weng et al. proposed the rst bidirectional IND-PRE-CCA2 secure proxy re-encryption scheme without pairings [7], [13]. Recently in PKC'09, Shao et al proposed the st unidirectional collusion-resistance IND-PRE-CCA2 proxy re-encryption without pairing. They constructed their scheme based on the public key encryption with double trapdoors proposed in Asiacrypt'03 [4].

A. Our Contribution

Based on Hanaoka's efficient chosen ciphertext secure public key encryption under the computational Diffie-Hellman assumption [8], We propose two restricted IND-PRE-CCA2 secure proxy re-encryption schemes without pairing in the standard model.

B. Organization

In Section II, we review the concept of IND-PRE-CCA2 secure proxy re-encryption scheme. In the next Sections, we present two proposals toward constructing an IND-PRE-CCA2 secure proxy re-encryption scheme based on Hanaoka and Kurosawa's IND-CCA2 secure PKE. We give our conclusion in the last Section.

II. DEFINITION AND SECURITY MODEL FOR BIDIRECTIONAL PROXY RE-ENCRYPTION

A. De nition

De nition 1: A proxy re-encryption scheme(PRE) is a tuple of algorithms (KeyGen, ReKeyGen, Enc, ReEnc, Dec) :
1) KeyGen (1k) ! (pk; sk). On input the security parameter 1k, the key generation algorithm KeyGen outputs a public key pk and a secret key sk.
2) ReKeyGen (sk1; sk2) ! r k 1$ 2 . On input two secret keys sk1 and sk2, the re-encryption key generation algorithm ReKeyGen outputs a bidirectional re-encryption key r k 1$ 2.
3) Enc (pk; m) ! C. On input a public key pk and a message m 2 f0; 1g, the encryption algorithm Enc outputs a ciphertext C.
4) ReEnc(r k 1$ 2 ; C 1 ) ! C 2 . On input a re-encryption key rk and a ciphertext C 1, the re-encryption algorithm ReEnc outputs a second ciphertext C 2 or the error symbol ?.

B. Security Model

De nition 2: (IND-PRE-CCA game) Let k be the security parameter. Let A be an oracle TM, representing the adversary. The game consists of an execution of A with the following oracles, which can be invoked multiple times in any order, subject to the constraints below:

Uncorrupted key generation oracle: Obtain a new key pair as (pk; sk) KeyGen (1k), A is given pk.
Corrupted key generation oracle: Obtain a new key pair as (pk; sk) KeyGen (1k). A is given both pk and sk.
Re-encryption key generation oracle: On input (pk; pk 0) by the adversary, where pk, pk 0 were generated before by KeyGen, return the re-encryption key r k pk 0 pk 0 = ReKeyGen (sk; sk 0) where sk, sk 0 are the
secret keys that correspond to \( pk, pk' \). We require that either both \( pk \) and \( pk' \) are corrupted, or alternatively both are uncorrupted. We do not allow for re-encryption key generation queries between a corrupted and an un-corrupted key. (This represents the restriction that the identities of parties whose security is compromised should be fixed in advance.)

- **Challenge Oracle**: This oracle can be queried once. On input \((pk^*, m_0, m_1)\), where \( pk^* \) is called the challenge key, the oracle chooses a bit \( b \in \{0, 1\} \) and returns the challenge ciphertext \( C^* = Enc(pk^*, m_b) \). (As we note later, the challenge key must be uncorrupted for \( A \) to win.)

- **Re-encryption oracle**: On input \((pk, pk', C)\), where \((pk, pk')\) were generated before by KeyGen, if \( pk' \) is corrupted and \((pk, C)\) is a derivative of \((pk^*, C^*)\), then return a special symbol \( \bot \) which is not in the domains of messages and ciphertexts. Else, return the re-encrypted ciphertext \( C' = ReEnc(ReKeyGen(sk, sk'), C) \). Derivatives of \((pk^*, sk', C^*)\) are defined inductively, as follows:
  1. \((pk^*, C^*)\) is a derivative of itself.
  2. If \((pk, C)\) is a derivative of \((pk^*, C^*)\) and \((pk', C')\) is a derivative of \((pk, C)\), then \((pk', C')\) is a derivative of \((pk^*, C^*)\).
  3. If \( A \) has queried the re-encryption oracle \( O_{enc} \) on input \((pk, pk', C)\) and obtained response \((pk', C')\), then \((pk', C')\) is a derivative of \((pk, C)\).
  4. If \( A \) has queried the re-encryption key generation oracle \( O_{key} \) on input \((pk, pk') \) or \((pk', pk)\), and \( Dec(pk', C') \in (m_0, m_1)\), then \((pk', C')\) is a derivative of \((pk, C)\).

- **Decryption Oracle**: On input \((pk, C)\), if the pair \((pk, C)\) is a derivative of the challenge ciphertext \( C^* \), or \( pk \) was not generated before by KeyGen, then return a special symbol \( \bot \) which is not in the domain \( D \) of messages. Else, return \( Dec(sk, C) \).

- **Decision Oracle**: This oracle can also be queried only once, on input \( b' \), if \( b' = 0 \) and the challenge key \( pk^* \) is not corrupted, then output 1, else output 0.

We say that \( A \) wins the IND-CPA game with advantage \( \varepsilon \) if the probability, over the random choices of \( A \) and the oracles, that the decision oracle is invoked and outputs 1, is at least \( 1/2 + \varepsilon \). For the details about definitions of derivatives, please refer to [6].

### III. PROPOSAL I

Let \( \mathbb{G} \) be a multiplicative group with prime order \( p \), and \( g \in \mathbb{G} \) be a generator.

1. **Setup(1^k)**: Choose an CCA secure symmetrical encryption scheme \( SE \) and make it to be public. The system generates a prime polynomial \( P(x) \) with degree 2 over \( \mathbb{Z}_p \), which requires satisfying \( g^{P(x)} = 1 \). \( TCR : \mathbb{G} \rightarrow \mathbb{Z}_p^* \) is a target collision resistant hash function and \( h : \mathbb{G} \rightarrow \{0, 1\}^\nu \) is a hash function.

\[
\text{params} = (\mathbb{G}, g, h, TCR, P(x), SE)
\]

Delegator \( A \) generates a random polynomial \( f(x) = a_0 + a_1 x + a_2 x^2 \mod P(x) \) over \( \mathbb{Z}_p \), and compute \( y_i = g^{a_i} \) for \( 0 \leq i \leq 2 \), \( \tilde{y}_i = g^{h(a_i)} \) for \( 0 \leq i \leq 2 \), generate another random polynomial \( g(x) = b_0 + b_1 x + b_2 x^2 \mod P(x) \) over \( \mathbb{Z}_p \), and compute \( t_i = g^{y_i} \) for \( 0 \leq i \leq 2 \).

\[
sk_A = (f(x), g(x), Pk_A = (y_0, y_1, y_2, \tilde{y}_0, \tilde{y}_1, \tilde{y}_2, t_0, t_1, t_2))
\]

Delegator \( B \) generates a random polynomial \( f'(x) = a'_0 + a'_1 x + a'_2 x^2 \mod P(x) \) over \( \mathbb{Z}_p \), and compute \( y'_i = g^{a'_i} \) for \( 0 \leq i \leq 2 \), \( \tilde{y}'_i = g^{h(a'_i)} \) for \( 0 \leq i \leq 2 \), generate another random invertible polynomial \( g'(x) = b'_0 + b'_1 x + b'_2 x^2 \mod P(x) \) over \( \mathbb{Z}_p \), and compute \( t'_i = g'^{y'_i} \) for \( 0 \leq i \leq 2 \).

\[
sk_B = (f'(x), g'(x), Pk_B = (y'_0, y'_1, y'_2, \tilde{y}'_0, \tilde{y}'_1, \tilde{y}'_2, t'_0, t'_1, t'_2))
\]

2. **CheckKeyGen(**\( sk_A, sk_B, \text{params} **)**): On input \( A \) and \( B \)'s decryption key, it outputs

\[
e_k = h'(x) = \frac{g(x)}{f(x)} \mod P(x)
\]

3. **ReKeyGen(**\( sk_A, sk_B, \text{params} **)**): On input \( A \) and \( B \)'s decryption key, it outputs

\[
r_k = h(x) = \frac{f'(x)}{f(x)} \mod P(x)
\]

4. **Encrypt(**\( Pk_A, \text{params}, M **)**): Pick a random \( r \in \mathbb{Z}_p \), and compute

\[
\varphi_A = (C_0, C_1, C_2, C_3) = (g^r, g^{f(r)}g^r g^{g(r)}SE.Enc(m, K))
\]

where \( K = h(y_0, C^0_1) \). \( i = TCR(C_0) \). The final output is \( \varphi \). (Notice that one can easily compute \( g(x)^r, C^0_1 \) as \( g(x)^r = \prod_{i=0}^2 y_i^x \), \( g(x)^r = \prod_{i=0}^2 y_i^x \).

5. **Re-encrypt**\( (\varphi, rk, ck, \text{params} **)**):
   a. On input \( C_A = \varphi_A \) first use check key \( ck \) to check

\[
C_2 = C'^0_1
\]

holds where \( i = TCR(C_0) \). If not, return \( \bot \).
   b. Otherwise it computes

\[
\varphi_B = (C_0', C_1', C_2', C_3') = (C_0, C_1, C_2, C_3, C_0)^r, C_1^r, C_3^r = (g^r, g^{f(r)}g^r g^{g(r)}SE.Enc(m, K))
\]

where \( K = h(y_0, C^0_1) \).

6. **Decrypt**\( (sk, \varphi, \text{params} **)**):
   - For a normal ciphertext

\[
\varphi_A = (C_0, C_1, C_2, C_3) = (g^r, g^{f(r)}g^r g^{g(r)}SE.Enc(m, K))
\]
check whether \((C_0^{f(i)}, C_0^{g(i)}) = (C_1, C_2)\), where \(i = TCR(C_0)\). If not, output \(\perp\). Otherwise, output \(m = SE.Dec(c_3, h(C_0^{c_3}, C_1^{c_3}))\). Or compute \(m = SE.Dec(h(C_0^{c_3}, C_1^{c_3}), C_3)\) and check \(m\)’s validity by using the IND-CCA2 security of \(SE\). If not, output \(\perp\). Otherwise, output \(m\).

For a re-encrypted ciphertext
\[
\varphi_B = (C'_0, C'_1, C'_2, C'_3) = (C_0, C_1, C_0^{x_0}, C_1^{x_0}, C_3)
\]
\[
= (g^{r_i} f(i), g^{r_i g(i)}, g^{r_i f(i)}, \ldots, SE.Enc(m, K))
\]
Compute \(m = SE.Dec(h(C_0^{c_3}, C_1^{c_3}), C_3)\) and check \(m\)’s validity by using the IND-CCA2 security of \(SE\). If not, output \(\perp\). Otherwise, output \(m\).

A. Why It Fails to Achieve CCA2 Security

Given a challenge ciphertext
\[
\varphi_A = (C'_0, C'_1, C'_2, C'_3) = (g^{r_i} f(i), g^{r_i g(i)}, SE.Enc(m, K))
\]
the adversary (the malicious delegatee) can choose any
\[
C_3 \in_R \text{Space}(SE.Enc(M, K))
\]
and query
\[
\varphi_A = (C'_0, C'_1, C'_2, C'_3)
\]
to the proxy, the proxy will let the modified ciphertext pass the “Check” step. Then the adversary (the malicious delegatee) will get
\[
\varphi_B = (C'_0, C'_1, C'_2, C'_3) = (C_0, C_1, C_0^{x_0}, C_1^{x_0}, C_3)
\]
\[
= (g^{r_i} f(i), g^{r_i g(i)}, g^{r_i f(i)}, \ldots, SE.Enc(m, K))
\]
Now the adversary (the malicious delegatee) can replace \(C'_3\) with \(SE.Enc(m_0, K)\) where \(K = h(y_0, C_0^{y_0})\), that is
\[
\varphi_B = (C'_0, C'_1, C'_2, C'_3) = (C_0, C_1, C_0^{x_0}, C_1^{x_0}, C_3)
\]
\[
= (g^{r_i} f(i), g^{r_i g(i)}, g^{r_i f(i)}, \ldots, SE.Enc(m_0, K))
\]
and he can decrypt \(\varphi_B\) to get \(m_0\).

B. Security Result

If we restrict the adversary can not launch the attack as above, then proposal I is restricted IND-PRE-CCA2 secure.

IV. Proposal II

Let \(G\) be a multiplicative group with prime order \(p\), and \(g \in G\) be a generator.

1) Setup(1^k):
Choose an CCA secure symmetrical encryption scheme \(SE\) and make it to be public. The system generates a prime polynomial \(f(x) = a_0 + a_1 x + a_2 x^2\) mod \(P(x)\) over \(\mathbb{Z}_p\), and compute \(y_i = g^{x_i}\) for \(0 \leq i \leq 2\), generate another random polynomial \(g(x) = b_0 + b_1 x + b_2 x^2\) mod \(P(x)\) over \(\mathbb{Z}_p\), and compute \(t_i = g^{y_i}\) for \(0 \leq i \leq 2\).

\[
\text{sk}_A = (f(x), g(x)), PK_A = (y_0, y_1, y_2, t_0, t_1, t_2)
\]

Delegatee \(B\) generates a random polynomial \(f'(x) = a'_0 + a'_1 x + a'_2 x^2\) mod \(P(x)\) over \(\mathbb{Z}_p\), and compute \(y'_i = g^{x'_i}\) for \(0 \leq i \leq 2\), generate another random invertible polynomial \(g'(x) = b'_0 + b'_1 x + b'_2 x^2\) mod \(P(x)\) over \(\mathbb{Z}_p\), and compute \(t'_i = g^{'y_i}\) for \(0 \leq i \leq 2\).

\[
\text{sk}_B = (f'(x), g'(x)), PK_B = (y'_0, y'_1, y'_2, t'_0, t'_1, t'_2)
\]

2) CheckKeyGen(\(sk_A, sk_B, params\)): On input \(A\) and \(B\)’s decryption key, it outputs
\[
ck = h(x) = (g(x) \bmod P(x))
\]

3) ReKeyGen(\(sk_A, sk_B, params\)): On input \(A\) and \(B\)’s decryption key, it outputs
\[
rk = h(x) = \frac{f'(x)}{f(x)} \bmod P(x)
\]

4) Encrypt(\(PK_A, params, M\)): Pick a random \(r \in \mathbb{Z}_p\), and compute
\[
\varphi_A = (C_0, C_1, C_2, C_3) = (g^{r_i} f(i), g^{r_i g(i)}, SE.Enc(m, K))
\]
where \(K = h(y_0, i = TCR(C_0, C_3))\). The final output is \(\varphi\). (Notice that one can easily compute \(g^{'x_i}, g^{'y_i}\) as \(\prod_{0 \leq i \leq 2} g^{'x_i} \cdot g^{'y_i}\).)

5) Re-encrypt(\(\varphi, rk, ck, params\)):

a) On input \(C_A = \varphi_A\) first use check key \(ck\) to check
\[
C_2 = C_0^0
\]
holds where \(j = TCR(C_0, C_3, C_1)\). If not, return \(\perp\).

b) Otherwise it computes
\[
\varphi = (C_0, C_1) = (C_0^{x_0}, C_1^{x_0}) = (g^{r_i \cdot x_0}, SE.Enc(m, K))
\]
\[
\varphi_B = (C_0, C_1) = (C_0, C_1, C_3) = (g^{r_i}, SE.Enc(m, K), SE.Enc(\varphi, K'))
\]
where $K = h(y_0^*)$, $K' = h(g^{r f(i)})$.

6) **Decrypt**($sk, \varphi, \text{params}$):

- For a normal ciphertext
  \[
  \varphi_A = (C_0, C_1, C_2, C_3) = (g^r, g^{r f(i)}, g^{r g(i)}, \text{SE.Enc}(m, K))
  \]
  check whether $(C_0^{(i)}), C_3^{(j)} = (C_1, C_2)$, where \(i = \text{TCR}(C_0, C_3), j = \text{TCR}(C_0, C_3, C_1)\).
  If not, output $\bot$. Otherwise, output $m = \text{SE.Dec}(C_3, h(C_0^{(n)}))$.

- For a re-encrypted ciphertext
  \[
  \varphi_B = (C_0', C_1', C_2') = (g^r, \text{SE.Enc}(m, K), \text{SE.Enc}(\varphi, C_2'))
  \]
  a) Compute \(\varphi' = (C_0', C_1') = \text{SE.Dec}(h(C_0^{(i)}), C_3')\) where \(i = \text{TCR}(C_0', C_3') = \text{TCR}(C_0, C_3)\). and check $\varphi'$'s validity by using the IND-CCA2 security of $SE$. If not, output $\bot$.
  b) Otherwise, compute $m = \text{SE.Dec}(h(C_0^{(n)}), C_1')$ and check $m$'s validity by using the IND-CCA2 security of $SE$. If not, output $\bot$. Otherwise, output $m$.

A. Why It Fails to Achieve CCA2 Security

Suppose there are many proxies, e.g. proxy 1 represents Alice→Bob, proxy 2 represents Alice→Charlie, but proxy 1 can collude with Charlie to get the plaintext corresponding to Alice. Given a challenge ciphertext $\varphi_A = (C_0', C_1', C_2', C_3') = (g^r, g^{r f(i)}, g^{r g(i)}, \text{SE.Enc}(mb, K))$ where $K = h(y_0), i = \text{TCR}(C_0', C_3'), j = \text{TCR}(C_0', C_3', C_1')$

The adversary (proxy 1) chooses randomly $C_1' \in \text{Space}[C_1]$

and modifies $\varphi_A = (C_0', C_1', C_2', C_3') = (g^r, g^{r f(i)}, g^{r g(i)}, \text{SE.Enc}(mb, K))$ to be

\[
\varphi'_A = (C_0', C_1', C_2', C_3') = (g^r, g^{r g(i)}, g^{r f(i)}, C_3')
\]

where $K = h(y_0), i = \text{TCR}(C_0', C_2')$ and $j' = \text{TCR}(C_0', C_3', C_1')$

Note proxy 1 knows $g(x)$. And proxy 1 query $\varphi_A$ to proxy 2's re-encryption oracle, the proxy 2 will return

\[
\varphi'_B = (C_1', C_2') = (C_0^{(m)}, C_3') = (g^{r g(i)}, C_2')
\]

Now Charlie can compute $m_b = \text{SE.Dec}(h(C_1^{(m)}, C_2'))$ to get $mb$.

B. Security Result

If we restrict the adversary as to not employ the attack above, then proposal II is restricted IND-PRE-CCA2 secure.

V. Conclusion

In this paper, we try to solve the open problem of constructing a bidirectional CCA-secure proxy re-encryption without pairing nor random oracle. We remark that solving this open problem is a very difficult task. Toward solving this open problem, we present two new interesting restricted IND-PRE-CCA2 secure bidirectional proxy re-encryption schemes without pairing nor random oracle.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China under contract no. 60842006.

REFERENCES


