# A Generic Construction of Predicate Proxy Key Re-encapsulation Mechanism[1]

Yi-Fan Tseng, Zi-Yuan Liu, Raylin Tso

Department of Computer Science, National Chengchi University, Taiwan
{yftseng, zyliu, raylin}@cs.nccu.edu.tw

## Abstract

Proxy re-encryption (PRE), first formalized by Blaze *et al*. in 1998, allows a proxy entity to delegate the decryption right of a ciphertext from one party to another without obtaining the information of the plaintext. In order to achieve more flexible access control, the predicate proxy re-encryption (PPRE) is studied further. However, existing PPRE is restricted with the inner product predicate function. The problem of how to realize the PPRE of arbitrary predicate functions is still to be solved. In this paper, we propose two secure generic predicate proxy key re-encapsulation mechanisms (PPKREM). By applying the key encapsulation mechanism/ data encapsulation mechanism paradigm, our PPKREM can be converted to a PPRE. Consequently, the results open new avenues for building more flexible and fine-grained PPRE.

**Keywords:** Predicate encryption, Predicate proxy re-encryption, Generic construction, Single-hop, Unidirectional

## 1 Introduction

Proxy re-encryption (PRE), first formalized by Blaze *et al.* in 1998 [2], allows a proxy entity to re-encrypt a ciphertext that has been encrypted for Alice and to generate a new ciphertext that can be decrypted using Bob's private key. The proxy entity only needs a re-key provided by Alice, without obtaining any other information of the plaintext or needing to access Alice's and Bob's private keys. In other words, the proxy entity can delegate the decryption right from one party to another. With this flexible property, PRE yields numerous real-world applications [3] such as outsourcing cryptography, distributed file storage systems, and law enforcement, etc. To support more flexibility on access control, some studies focus on supporting more complex access control mechanisms, such as identity-based PRE [4-6] and attribute-based PRE [7-9].

On the other hand, predicate encryption (PE), formalized by Katz *et al.* in 2008 [10], is a paradigm for public-key encryption that conceptually generalizes public-key encryption, supporting fine-grained and role-based access to encrypted data. More precisely, in a PE for a predicate function $R_k$, a private key is associated with a key attribute $y$, while the ciphertext is associated with a ciphertext attribute $x$, where $k$ is the description of a predicate. A ciphertext with ciphertext attribute $x$ can be decrypted by a private key with key attribute $y$ if, and only if, $R_k(x, y) = 1$. Thus, PE captures wide classes of encryption in cryptography. For example, identity-based encryption can be viewed as PE supporting "equality" predicate functions, and both ciphertext attributes and key attributes are strings.

Although many identity-based PRE and attribute-based PRE have been studied, only a few of the studies have researched how to construct proxy re-encryption (PPRE) [11-13]. Unfortunately, these schemes consider only the case where the predicate function is an inner product predicate. At present, many flexible and fine-grained PPRE schemes have not been implemented and discussed. Hence, how to realize a PPRE of an arbitrary predicate function remains an open problem.

### 1.1 Contributions

In this paper, we affirmatively solve this by proposing two generic constructions that can transform any linear predicate key encapsulation mechanism (PKEM) or any linear PE scheme to a predicate proxy key re-encapsulation mechanism (PPKREM).

We prove that our construction is payload hiding of second/first-level ciphertext (i.e., original/re-encapsulation ciphertext) secure in the standard model, if the underlying PKEM satisfies indistinguishability under chosen-ciphertext attacks (IND-CCA).

Then, since secure key encapsulation mechanism (KEM) can be used as a building block to construct public key encryption, i.e., combining with a secure symmetric encryption scheme, we can use our construction to obtain a secure PPRE.

---

[1] The extended abstract version of this work is appeared in the Proceedings of The 15th Asia Joint Conference on Information Security (AsiaJCIS 2020) [1].

*Corresponding Author: Raylin Tso; E-mail: raylin@cs.nccu.edu.tw

DOI: 10.53106/160792642021092205020

Besides, we adopt our proposed generic construction for Water's identity-based encryption [14]. More precisely, we first obtain an identity-based KEM from Water's work and then obtain an identity-based proxy key re-encapsulation mechanism using our proposed construction. Furthermore, by applying the KEM/DEM paradigm, anyone can easily obtain an identity-based PRE.

Our result, compared with the previous identity-based [4-6] and attribute-based [7-9] constructions, is more flexible to use in various scenarios. However, our generic construction limits the underlying building block requirements to meet the linear property, which does not exist in all PKEM schemes.

### 1.2 Comparison with the Previous Version [1]

In this paper, we formally prove that the proposed generic construction is payload hiding of second-/first-level ciphertext secure. In addition, the previous version considers only how to obtain a PPKREM scheme from linear PKEM. Here, we further propose another generic construction that can obtain a PPKREM scheme from any linear PE scheme.

### 1.3 Organization

The remainder of the work is organized as follows. In Sections 2 and 3, we introduce the definition and the security requirement of PE, PKEM, and PPKREM, respectively. In Sections 4 and 5, we propose our generic construction and provide the security proofs, respectively. In Section 6, we give an instantiates of identity-based proxy key re-encapsulation mechanism from Water's identity-based encryption. Finally, we conclude the work in Section 7.

## 2 Preliminary

### 2.1 Notations and Abbreviations

For simplicity and convenience, we use the following notations and abbreviations detailed in Table 1 throughout the paper.

**Table 1.** Notations and abbreviations

| Symbols | Description |
|---|---|
| $\lambda$ | Security parameter |
| $\mathbb{N}$ | The set of positive integers |
| $\mathbb{Z}$ | The set of integers |
| $\mathbb{Z}_p$ | The set of integers module $p$ |
| PE | Predicate encryption |
| PRE | Proxy re-encryption |
| PPRE | Predicate proxy re-encryption |
| KEM | Key encapsulation mechanism |
| PKEM | Predicate key encapsulation mechanism |
| PPKREM | Predicate proxy key re-encapsulation mechanism |
| PPT | Probabilistic polynomial-time |

### 2.2 Predicate Key Encapsulation Mechanism (PKEM)

In this Subsection, we first recall the definition of the predicate family in [15-16], and then recall the definition of PKEM in [17] described by a binary relation.

**Definition 1 (Predicate Family).** We consider a predicate family $R=\{R_k \in \mathbb{N}^c\}$ for some constant $c \in \mathbb{N}$, where a relation $R_k : \mathbb{X}_k \times \mathbb{Y}_k \to \{0,1\}$ is a predicate function that maps a pair of ciphertext attributes in a ciphertext attribute space $\mathbb{X}_k$ and key attributes in a key attribute space $\mathbb{Y}_k$ to $\{0,1\}$. The family index $k=(n_1, n_2, \ldots)$ specifies the description of a predicate from the family.

**Definition 2 (Predicate Key Encapsulation Mechanism).** Let $\psi$ be the encapsulation ciphertext space and $K$ be the encapsulation key space, a PKEM scheme PKEM for predicate family $R$ consists of the following four algorithms:

- $\text{Setup}(1^\lambda, k) \to (params, msk)$: Taking as input the security parameter $\lambda \in \mathbb{N}$ and a description $k \in \mathbb{N}$, the algorithm outputs the system parameter params, where the description of k is implicitly included, and the master secret key msk. Note that params will be an implicit input for the following algorithms.
- $\text{Encaps}(x) \to (CT_x, k)$: Taking as inputs a ciphertext attribute $x \in \mathbb{N}_k$, the algorithm outputs a ciphertext $CT_x \in \psi$ and an encapsulation key $k \in K$.
- $\text{KeyGen}(msk, y) \to SK_y$: Taking as inputs the master secret key msk and a key attribute $y \in \mathbb{N}_k$, the algorithm outputs a private key $SK_y$ associated with y.
- $\text{Decaps}(CT_x, SK_y) \to k / \perp$: Taking as inputs a ciphertext $CT_x \in \psi$ for some ciphertext attribute $x \in \mathbb{X}_k$ and a private key $SK_y$ for some key attribute $y \in \mathbb{Y}_k$, the algorithm outputs an encapsulation key $k \in K$ if $R_k(x, y)=1$. Otherwise, it outputs $\perp$.

**Correctness.** A PKEM scheme PKEM is correct if, for all $\lambda, k \in \mathbb{N}$, we have $k \leftarrow \text{Decaps}(CT_x, SK_y)$, if $R_k(x, y)=1$; $\perp \leftarrow \text{Decaps}(CT_x, SK_y)$, otherwise, where $(CT_x, k) \leftarrow \text{Encaps}(x)$, $SK_y \leftarrow \text{KeyGen}(msk, y)$, and $(params, msk) \leftarrow \text{Setup}(1^\lambda, k)$.

**Security.** In order to describe the security of the PKEM, we define the following IND-CCA game between a challenger *C* and an adversary *A*.

**Game IND-CCA:**

- **Setup.** The challenger *C* runs the algorithm $\text{Setup}(1^\lambda, k)$ to generate system parameter params

and the master secret key msk. It then sends params to the adversary $A$.

- **Phase 1.** The adversary $A$ makes polynomial times of queries to the following oracles.
  - **Key generation oracle $O_{ke}$ :** On input of a key attribute $y \in \mathbb{Y}_k$, the oracle returns the corresponding private key $SK_y$.
  - **Decapsulation oracle $O_{de}$ :** On input of a ciphertext $CT_x \in \psi$ and a key attribute $y \in \mathbb{Y}_k$, the oracle returns an encapsulation key $k$ or $\perp$.
- **Challenge.** The adversary submits a target ciphertext attribute $x^* \in \mathbb{X}_k$, where $R_k(x^*, y) = 0$ for all $y \in \mathbb{Y}_k$ queried in **Phase 1.** Then the challenger $C$ randomly chooses a bit $b \leftarrow \{0,1\}$, runs $(CT^*_{x^*}, k^*_0) \leftarrow \text{Encaps}(x^*)$, and chooses $k^*_1 \leftarrow K$. Finally, $C$ returns $(CT^*_{x^*}, k^*_b)$ to $A$.
- **Phase 2.** It is the same as **Phase 1** except that $\text{Decaps}(CT^*_{x^*}, y)$ and $\text{KeyGen}(y)$ are not allowed if $R_k(x^*, y) = 1$.
- **Guess.** The adversary $A$ outputs a bit $b'$, and wins the game if $b' = b$.

The advantage of the adversary $A$ in winning the game is defined as:

$$Adv^{IND-CCA}_{PKEM,A}(\lambda) = |\Pr[b' = b] - \frac{1}{2}|.$$

## 2.3 Predicate Encryption (PE)

**Definition 3 (Predicate Encryption).** A predicate encryption scheme PE for predicate family R consists of the following four algorithms:

- $\text{Setup}(1^\lambda, k) \rightarrow (params, msk)$ : Taking as input the security parameter $\lambda$ and a description k, the algorithm outputs the system parameter params, where the description of k is implicitly included, and the master secret key msk. Note that params will be an implicit input for the following algorithms.
- $\text{Encaps}(x, M) \rightarrow CT_x$ : Taking as inputs a ciphertext attribute $x \in \mathbb{X}_k$ and a message $M \in M$, the algorithm outputs a ciphertext $CT_x$.
- $\text{KeyGen}(msk, y) \rightarrow SK_y$ : Taking as inputs the master secret key msk and a key attribute $y \in \mathbb{Y}_k$, the algorithm outputs a private key $SK_y$.
- $\text{Decaps}(CT_x, SK_y) \rightarrow M$ : Taking as inputs a ciphertext $CT_x$ for ciphertext attribute $x \in \mathbb{X}_k$ and a private key $SK_y$ for key attribute $y \in \mathbb{Y}_k$, the algorithm outputs a message $M \in M$.

**Correctness.** A predicate encryption scheme PE is correct if, for all $\lambda, k \in \mathbb{N}$, we have $M \leftarrow \text{Decrypt}(CT_x, SK_y)$ if $R_k(x, y) = 1$, where $CT_x \leftarrow \text{Encrypt}(x, M)$, $SK_y \leftarrow \text{KeyGen}(msk, y)$, and $(params, msk) \leftarrow \text{Setup}(1^\lambda, k)$.

**Security.** In order to describe the security of the predicate encryption scheme, we define the following IND-CCA game between a challenger $C$ and an adversary $A$.

**Game IND-CCA:**

- **Setup.** The challenger $C$ runs the algorithm $\text{Setup}(1^\lambda, k)$ and sends params to the adversary $A$.
- **Phase 1.** The adversary $A$ makes polynomial times of queries to the following oracles.
  - $\text{KeyGen}(y_i)$ : Upon inputting a key attribute $y_i \in \mathbb{Y}$, the oracle returns the corresponding private key $SK_{y_i}$.
  - $\text{Decrypt}(CT, y_i)$ : Upon inputting a ciphertext $CT$ and a key attribute $y_i \in \mathbb{Y}$, the oracle returns the output of $\text{Decrypt}(CT, SK_{y_i})$.
- **Challenge.** The adversary submits two distinct messages $M_0, M_1 \in M$ of the same length, and a target ciphertext attribute $x^* \in \mathbb{N}_K$, where $R_k(x^*, y_i) = 0$ for all $y_i$ queried in **Phase 1.** Then the challenger $C$ randomly chooses a bit $b \leftarrow \{0,1\}$, and returns $CT^* \leftarrow \text{Encrypt}(x^*, M_b)$.
- **Phase 2.** It is the same as **Phase 1** except $\text{Decrypt}(CT^*, y_i)$ and $\text{KeyGen}(y_i)$ such that $R_k(x^*, y_i) = 1$ are not allowed.
- **Guess.** The adversary $A$ outputs a bit $b'$, and wins the game if $b' = b$.

The advantage of the adversary $A$ in winning the game is defined as:

$$Adv^{IND-CCA}_{PE,A}(\lambda) = |\Pr[b' = b] - \frac{1}{2}|.$$

**Definition 4 (IND-CCA Security).** We say that a PE scheme PE for predicate family R is IND-CCA secure if, for all PPT adversary A, $Adv^{IND-CCA}_{PE,A}(\lambda)$ is negligible.

The model can be easily changed for CPA security and selective security by removing the Decrypt oracle and forcing the adversary to submit its target first, respectively.

**Linearity.** We say that a correct predicate encryption scheme $PE = (\text{Setup, Encrypt, KeyGen, Decrypt})$ is linear if for all $\gamma \in \mathbb{Z}$, $CT_x \leftarrow \text{Encrypt}(x, M)$ and $SK_y \leftarrow \text{KeyGen}(msk, y)$, where $(params, msk) \leftarrow \text{Setup}(1^\lambda, k)$, the following equation is satisfied: $\text{Decrypt}(CT_x, (SK_y)^\gamma) = \text{Decrypt}(CT_x, SK_y)^\gamma$.

**Definition 5 (IND-CCA Security of PKEM).** We say that a PEKM schemePKEM for predicate family R is IND-CCA secure if, for all PPT adversary A, $Adv^{IND-CCA}_{PKEM,A}(\lambda)$ is negligible.

The model can be easily changed for CPA security and selective security by removing the **Decapsulation oracle** and forcing the adversary to submit its target first, respectively.

**Linearity.** In this work, the whole correctness of the proposed construction is based on the linearity of the PKEM, defined as follows.

**Definition 6 (Linearity of PKEM).** We say that a correct PKEM scheme $PKEM = (\text{Setup}, \text{Encrypt}, \text{KeyGen}, \text{Decrypt})$ for predicate family $R$ is linear if for all $\gamma \in \mathbb{Z}, \lambda, k \in \mathbb{N} (CT_x, k) \leftarrow \text{Encaps}(x)$, and $SK_y \leftarrow \text{KeyGen}(msk, y)$, where $(params, msk) \leftarrow \text{Setup}(1^\lambda, k)$ and $R_k(x, y) = 1$, the following equation is satisfied: $\text{Decaps}(CT_x, (SK_y)^\gamma) = k^\gamma$, where $(SK_y)^\gamma$ and $k^\gamma$ denote the component-wise exponentiation to $SK_y$ and k, respectively.

## 3 Predicate Proxy Key Re-encapsulation Mechanism (PPKREM)

In this Section, we introduce the definition and security models of a single-hop unidirectional PPKREM. More precisely, we adopt the security game in [18]. However, the game in [18] is defined for an identity-based cryptography scheme, thus we revise it and provide new security games for our scheme. Additionally, for consistency and ease of interpretation, we use the terminologies defined in [19-20], that is, an original ciphertext is called the second-level ciphertext and a re-encapsulation ciphertext is called the first-level ciphertext.

**Definition 7 (Single-hop Unidirectional Predicate Proxy Key Re-encapsulation Mechanism).** Let $\psi$ be the encapsulation ciphertext space and $K$ be the encapsulation key space, a PPKREM scheme PPKREM for predicate family R consists of seven PPT algorithms (Setup, KeyGen, Encaps, ReKey, ReEncaps, $\text{Decaps}_{\text{oct}}$, $\text{Decaps}_{\text{rct}}$):

- $\text{Setup}(1^\lambda, k) \rightarrow (params, msk)$: Taking as input the security parameter $\lambda \in \mathbb{N}$, and a description $k \in \mathbb{N}$, the algorithm outputs the system parameter params, where the description of k is implicitly included, and the master secret key msk. Note that params will be an implicit input for the following algorithms.
- $\text{KeyGen}(msk, y) \rightarrow SK_y$: Taking as input the master secret key msk and a key attribute $y \in \mathbb{Y}_k$, the algorithm outputs a private key $SK_y$.
- $\text{Encaps}(x) \rightarrow (oct_x, k_x)$: Taking as input a ciphertext attribute $x \in \mathbb{X}_k$, the algorithm outputs a second-level ciphertext $oct_x \in \psi$ and an encapsulation key $k_x \in K$.
- $\text{ReKey}(SK_y, x') \rightarrow rk_{y,x'}$: Taking as input a private key $SK_y$ for some key attribute $y \in \mathbb{Y}_k$ and a ciphertext attribute $x' \in \mathbb{X}_k$, the algorithm outputs a re-key $rk_{y,x'}$.
- $\text{ReEncaps}(oct_x, rk_{y,x'}) \rightarrow rct_{x'}$: Taking as input a ciphertext $oct_x \in \psi$ for some ciphertext attribute $x \in \mathbb{X}_k$ and a re-key $rk_{y,x'}$, the algorithm outputs a first-level ciphertext $rct_x \in \psi$ which can be decaps by the private key $SK_{y'}$ for some key attribute $y' \in \mathbb{Y}_k$ where $R_k(x', y') = 1$.
- $\text{Decaps}(oct_x, SK_y) \rightarrow k$: Taking as input a second-level ciphertext $oct_x \in \psi$ for some ciphertext attribute $\mathbb{X}_k$ and a private key $SK_y$ for key attribute $y \in \mathbb{Y}_k$, the algorithm outputs a key $k \in K$ if $R_k(x, y) = 1$. Otherwise, it outputs $\perp$.
- $\text{Decaps}_{rct}(rct_{x'}, SK_{y'}) \rightarrow k$: Takeing as input a first-level ciphertext $oct_{x'} \in \psi$ for some ciphertext attribute $x' \in \mathbb{X}_k$ and a private key $SK_y$ for some key attribute $y' \in \mathbb{Y}_k$, the algorithm outputs an encapsulation key $k \in K$ if $R_k(x', y') = 1$. Otherwise, it outputs $\perp$.

**Correctness.** A single-hop unidirectional PPKREM scheme PPKREM is correct if, for all $\lambda, k \in \mathbb{N}$, $x, x' \in \mathbb{X}_k$, and $y, y' \in \mathbb{Y}_k$, we have:

- $k = \text{Decaps}(oct_x, SK_y)$ if $R_k(x, y) = 1$;
- $\perp = \text{Decaps}(oct_x, SK_y)$ if $R_k(x, y) = 0$;
- $k = \text{Decaps}_{rct}(\text{ReEncaps}(oct_x, \text{ReKey}(SK_y, x')), SK_{y'})$ if $R_k(x, y) = 1 \wedge R_k(x', y') = 1$;
- $\perp = \text{Decaps}_{rct}(\text{ReEncaps}(oct_x, \text{ReKey}(SK_y, x')), SK_{y'})$ if $R_k(x, y) = 0 \vee R_k(x', y') = 0$, where $SK_y \leftarrow \text{KeyGen}(msk, y)$, $SK_{y'} \leftarrow \text{KeyGen}(msk, y')$, and $(params, msk) \leftarrow \text{Setup}(1^\lambda, k)$.

**Security.** Before introducing the security models, we follow [18] to define the derivatives for single-hop unidirectional PPKREM.

**Definition 8 (Derivatives).** Let $x, x', x'' \in \mathbb{X}_k$ be the ciphertext attributes, let $y \in \mathbb{Y}_k$ be the key attribute, and let $ct, ct', ct'' \in \psi$ be the ciphertexts. The derivatives of $(x, ct)$ are defined as follows:

- $(x, ct)$ is a derivative of itself;
- If $(x', ct')$ is a derivative of $(x, ct)$ and $(x'', ct'')$ is also a derivative of $(x', ct')$, then $(x'', ct'')$ is a

derivative of $(x, ct)$;

- If an adversary A has issued a query $(y, x', ct)$ on re-encapsulation oracle and obtained $ct'$, where $R_k(x, y)=1$, then $(x', ct')$ is a derivative of $(x, ct)$;
- If an adversary A has issued a query $(y, x')$ on re-encapsulation key generation oracle, obtained $rk_{y,x'}$, then for a $ct' = \mathrm{ReEncaps}(ct, rk_{y,x'})$, where $R_k(x, y)=1$, $(x', ct')$ is a derivative of $(x, ct)$.

In the following, we introduce two security games to describe the security of the PPKREM between a challenger *C* and an adversary *A*.

**Game - Payload-hiding for Second-level Ciphertext:**

- **Setup.** The challenger *C* runs the algorithm $\mathrm{Setup}(1^\lambda, k)$ to generate parameter params and the master secret key msk. It then sends params to the adversary *A*.
- **Phase 1.** The *A* may adaptively make polynomial times of queries to the following oracles.
  - **Key generation oracle $O_{ke}$:** On input of $y \in \mathbb{Y}_k$ by *A*, the challenger *C* computes $SK_y \leftarrow \mathrm{KeyGen}(msk, y)$. It then gives $SK_y$ to *A*.
  - **Re-encapsulation key generation oracle $O_{rk}$:** On input $(y \in \mathbb{Y}_k, x' \in \mathbb{X}_k)$ by *A*, the challenger *C* computes $rk_{y,x'} \leftarrow \mathrm{ReKey}(SK_y, x')$, where $SK_y \leftarrow \mathrm{KeyGen}(msk, y)$. It then gives $rk_{y,x'}$ to *A*.
  - **Re-encapsulation oracle $O_{re}$:** On input of $(y \in \mathbb{Y}_k, x' \in \mathbb{X}_k \in \psi)$ by *A*, the challenger *C* first computes $rk_{y,x'} \leftarrow \mathrm{ReKey}(SK_y, x')$ where $SK_y \leftarrow \mathrm{KeyGen}(msk, y)$. It then computes $rct_{x'} \leftarrow \mathrm{ReEncaps}(oct_x, rk_{y,x'})$. Finally, it gives $rct_{x'}$ to *A*.
  - **Second-level ciphertext decapsulation oracle $O_{sde}$:** On input of $(x \in \mathbb{X}_k, oct_x \in \psi)$ by *A*, the challenger *C* computes $k \leftarrow \mathrm{Decaps}_{oct}(oct_x, SK_y)$ where $SK_y \leftarrow \mathrm{KeyGen}(msk, y)$ and $R_k(x, y)=1$. It then returns *k* to *A*.
  - **First-level ciphertext decapsulation oracle $O_{fde}$:** On input of $(x' \in \mathbb{X}_k, rct_{x'} \in \psi)$ by *A*, the challenger *C* computes $k \leftarrow \mathrm{Decaps}_{rct}(rct_x, SK_y)$ where $SK_{y'} \leftarrow \mathrm{KeyGen}(msk, y')$ and $R_k(x', y')=1$. It then returns *k* to *A*.
- **Challenge.** *A* outputs a ciphertext attribute $x^* \in \mathbb{X}_k$ with restriction that:
  - $R_k(x^*, y)=0$ for all $y \in \mathbb{Y}_k$ submitted to $O_{ke}$;
  - for all $(y \in \mathbb{Y}_k, x' \in \mathbb{X}_k)$, $(x' \in \mathbb{X}_k)$ submitted to $O_{rk}$, $R_k(x^*, y)=0$.

  If $x^*$ satisfies the above requirements, the challenger *C* then randomly chooses a bit $b \in \{0,1\}$, and responds with $(oct^*_{x^*}, k^*_b)$, where $(oct^*_{x^*}, k^*_0) \leftarrow \mathrm{Encaps}(x^*)$ and $k^*_1$ is randomly chosen from *K*.
- **Phase 2.** *A* can continue to issue more queries to the oracles as follows:
  - **Key generation oracle $O_{ke}$:** The oracle is the same as **Phase 1** with three additional restrictions:
- $R_k(x^*, y)=0$;
- for all $y' \in \mathbb{Y}_k$ such that $R_k(x^*, y')=1 \wedge R_k(x, y)=1$, the tuple $(y', x)$ must not have been queried to $O_{rk}$ before:
- for all $y' \in \mathbb{Y}_k$, $x, x' \in \mathbb{X}_k$, and $oct' \in \psi$ such that $R_k(x, y)=1 \wedge R_k(x', y')=1$, and $(x', oct')$ is a derivative of $(x^*, oct^*_{x^*})$, the tuple $(y, x, oct')$ has not been queried to $O_{re}$ before.
  - **Re-encapsulation key generation oracle $O_{rk}$:** The oracle is the same as **Phase 1** with a restriction: if $x = x^*$, then for all $y' \in \mathbb{Y}_k$ such that $R_k(x, y)=1 \wedge R_k(x', y')=1$, $y'$ must not have been queried to $O_{ke}$ before.
  - **Re-encapsulation oracle $O_{re}$:** The oracle is the same as **Phase 1** with a restriction: if $(x, oct_x)$ is a derivative of $(x^*, oct^*_{x^*})$, then for all $y' \in \mathbb{Y}_k$ such that $R_k(x, y)=1 \wedge R_k(x', y')=1$, $y'$ must not have been queried to $O_{ke}$ before.
  - **Second-level ciphertext decapsulation oracle $O_{sde}$:** The oracle is the same as **Phase 1** with a restriction: $(x, oct_x)$ is not a derivative of $(x^*, oct^*_{x^*})$.
  - **First-level ciphertext decapsulation oracle $O_{fde}$:** The oracle is the same as **Phase 1** with a restriction: $(x', rct_{x'})$ is not a derivative of $(x^*, oct^*_{x^*})$.
- **Guess.** In the end, *A* outputs a guess $b \in \{0,1\}$ and wins the game if $b = b'$.

The advantage of the adversary *A* in winning the above game is defined as:

$$Adv^{PH-SC}_{PPKREM,A}(\lambda) = |\Pr[b = b'] - \frac{1}{2}|.$$

**Definition 9 (Payload-hiding Security for Second-level Ciphertext).** We say that a single-hop unidirectional PPKREM scheme PPKREM for predicate family R is payload-hiding secure for second-level ciphertext if, for any polynomial time adversary A, the function $Adv^{PH-SC}_{PPKREM,A}(\lambda)$ is negligible.

**Game Payload-hiding for First-level Ciphertext:**

- **Setup.** The challenger $C$ runs the algorithm $\text{Setup}(1^{\lambda}, k)$ to generate parameter params and the master secret key msk. It then sends params to the adversary $A$.
- **Phase 1.** The $A$ may adaptively make a polynomial time of queries to the following oracles.
  - **Key generation oracle $O_{ke}$:** On input of $y \in \mathbb{Y}_k$ by $A$, the challenger $C$ computes $SK_y \leftarrow \text{KeyGen}(msk, y)$. It then gives $SK_y$ to $A$.
  - **Re-encapsulation key generation oracle $O_{rk}$:** On input of $(y \in \mathbb{Y}_k, x' \in \mathbb{X}_k)$ by $A$, the challenger $C$ computes $rk_{y,x'} \leftarrow \text{ReKey}(SK_y, x')$, where $SK_y \leftarrow \text{KeyGen}(msk, y)$. It then gives $rk_{y,x'}$ to $A$.
  - **Re-encapsulation oracle $O_{re}$:** On input of $(y \in \mathbb{Y}_k, x' \in \mathbb{X}_k, oct_x \in \psi)$ by $A$, the challenger $C$ first computes $rk_{y,x'} \leftarrow \text{ReKey}(SK_y, x')$ where $SK_y \leftarrow \text{KeyGen}(msk, y)$. It then computes $rct_x \leftarrow \text{ReEncaps}(oct_x, rk_{y,x'})$. Finally, it gives $rct_{x'}$ to $A$.
  - **Second-level ciphertext decapsulation oracle $O_{sde}$:** On input of $(x \in \mathbb{X}_k, oct_x \in \psi)$ by $A$, the challenger $C$ computes $k \leftarrow \text{Decaps}_{oct}(oct_x, SK_y)$ where $SK_y \leftarrow \text{KeyGen}(msk, y)$ and $R_k(x, y) = 1$. It then returns $k$ to $A$.
  - **First-level ciphertext decapsulation oracle $O_{fde}$:** On input of $(x' \in \mathbb{X}_k, rct_{x'} \in \psi)$ by $A$, the challenger $C$ computes $k \leftarrow \text{Decaps}_{rct}(rct_{x'}, SK_{y'})$ where $SK_{y'} \leftarrow \text{KeyGen}(msk, y')$ and $R_k(x', y') = 1$. It then returns $k$ to $A$.
- **Challenge.** $A$ outputs a ciphertext attribute $x^* \in \mathbb{X}_k$ with restriction: for all $y \in \mathbb{Y}_k$ submitted to $O_{ke}$, $R_k(x^*, y) = 0$. If $x^*$ satisfies the above requirements, the challenger $C$ first computes $SK_{y^*} \leftarrow \text{KeyGen}(msk, y^*)$ where $R_k(x^*, y^*) = 1$. Then, it chooses a ciphertext attribute $\hat{x} \in \mathbb{X}_k$, and randomly chooses a bit $b \in \{0,1\}$. Next, it computes:

  a. $rk_{y^*,\hat{x}} \leftarrow \text{ReKey}(SK_{y^*}, \hat{x})$;

  b. $rct^*_{\hat{x}} \leftarrow \text{ReEncaps}(oct^*_{x^*}, rk_{y^*,\hat{x}})$,

  where $(oct^*_x, k^*_0) \leftarrow \text{Encaps}(x^*)$ and $k^*_1$ is randomly chosen from K. Finally, it responds $(rct^*_{\hat{x}}, k^*_b)$ to A.
- **Phase 2.** $A$ can continue to issue more queries to the oracles as in **Phase 1** with two additional restrictions:
  - **Key generation oracle $O_{ke}$:** for all $y \in \mathbb{Y}_k$, $R_k(\hat{x}, y) = 0$.
  - **First-level ciphertext decapsulation oracle $O_{fde}$:** it cannot be queried with the challenge ciphertext $rct^*_{\hat{x}}$ as input.
- **Guess.** In the end, $A$ outputs a guess $b \in \{0,1\}$ and wins the game if $c$.

The advantage of the adversary $A$ in winning the above game is defined as:

$$Adv^{PH-FC}_{PPKREM,A}(\lambda) = |\Pr[b = b'] - \frac{1}{2}|.$$

**Definition 10 (Payload-hiding Security for First-level Ciphertext).** We say that a single-hop unidirectional PPKREM scheme PPKREM for predicate family R is payload-hiding secure for first-level ciphertext if for PPT adversary A the function $Adv^{PH-FC}_{PPKREM,A}(\lambda)$ is negligible.

## 4 Generic Construction of Predicate Proxy Key Re-encapsulation Mechanism (PPKREM)

In this Section, we first give a generic construction that can obtain a PPKREM scheme from a secure linear PKEM scheme, then we give a generic PPKREM construction by using a secure linear PE scheme. At a high level, to generate a re-encapsulation key $rk_{y,x'}$, we first encaps the ciphertext attribute $x'$ to obtain a pair $(CT_x, k')$, then compute $h = H(k')$, where $H(\cdot)$ is a cryptographic hash function. Next, we let the re-encapsulation key be $rk_{y,x'} = \{(SK_y)^h, CT_{x'}\}$, where $(SK_y)^h$ denotes the $h$ component-wise exponentiation to $SK_y$. Note that, due to the complexity of the discrete-log problem, the proxy entity is impossible to obtain $h$ from $(SK_y)^h$. In other words, the proxy entity cannot recover $SK_y$ from $rk_{y,x'}$. In order to generate a first-level ciphertext $rct_{x'}$ from the second-level ciphertext $oct_x$ using the re-encapsulation key $rk_{y,x'}$, we directly run:

$$\delta \leftarrow \text{PKEM.Decaps}(oct_x, (SK_y)^n)..$$

With the linear property of PKEM (Definition 6), if $R_k(x, y) = 1$, $\delta$ actually equals to $(k)^h$, where $(oct_x, k) \leftarrow \text{PKEM.Encaps}(x)$. Then, the first-level ciphertext $rk_{y,x'}$ is set as $\{\delta, CT_{x'}\}$. Only the proxy receiver can decaps $CT_{x'}$ using her/his private key to obtain $k'$, and recover the value hidden in the encapsulation key, *i.e.,* $h = H(k')$. Finally, the proxy receiver can obtain:

$$(\delta)^{h^{-1}} = (k)^{h \cdot h^{-1}} = k.$$

Let PKEM = (Steup, KeyGen, Encaps, Decaps) be an IND-CCA secure PKEM with linear property for predicates family $R = \{R_k\}$ and let $H : K \to \mathbb{Z}$ be a cryptographic hash function, we define the construction of PPRKEM as follows:

- Setup$(1^\lambda, k)$: On input of a security parameter $\lambda \in \mathbb{N}$ and a description $k \in \mathbb{N}$, this algorithm runs $(params, msk) \leftarrow PKEM.\text{Setup}(1^\lambda, k)$. It then outputs the parameter *params* and the master secret key *msk*.
- KeyGen$(msk, y)$: On input of a master secret key msk and a key attribute $y \in \mathbb{Y}_k$, this algorithm runs $PKEM.\text{KeyGen}(msk, y)$ to output a private key $SK_y$ for key attribute y and outputs it.
- Encaps$(x)$ : On input of a ciphertext attribute $x \in \mathbb{X}_k$, this algorithm runs $(oct_x, k) \leftarrow PKEM.\text{Encaps}(x)$. It then outputs a second-level ciphertext $oct_x$ and an encapsulation key *k*.
- ReKey$(SK_y, x')$: On input of a private key $SK_y$ for some key attribute $y \in \mathbb{Y}_k$ and a ciphertext attribute $x' \in \mathbb{X}_k$, this algorithm runs the following steps to generate a re-encapsulation key:
  - Computes $(CT_{x'}, k') \leftarrow PKEM.\text{Encaps}(x')$;
  - Computes $h = H(k')$;
  - Outputs $rk_{y,x'} = \{(SK_y)^h, CT_{x'}\}$.
- ReEncaps$(oct_x, k_{y,x'})$ : On input of a second-level ciphertext $oct_x$ encapsed by ciphertext attribute $x \in \mathbb{X}_k$ and a re-encapsulation key $rk_{y,x'} = \{(SK_y)^h, CT_{x'}\}$, to generate a first-level ciphertext $rct_{x'}$ which can be decapsed by the private key $SK_y$ for some key attribute $y' \in \mathbb{Y}_k$ where $R(x', y') = 1$, this algorithm runs $\delta \leftarrow PKEM.\text{Decaps}(oct_x, (SK_y)^h)$, and outputs $rct_{x'} = \{\delta, CT_{x'}\}$.
- $\text{Decaps}_{\text{oct}}(oct_x, SK_y)$: On input of a second-level ciphertext $oct_x$ and a private key $SK_y$ for some key attribute $y \in \mathbb{Y}_k$, this algorithm runs $PKEM.\text{Decaps}(oct_x, SK_y)$ to obtains an encapsulation key *k* or $\perp$, and outputs it.
- $\text{Decaps}_{\text{rct}}(rct_{x'}, SK_{y'})$: On input of a first-level ciphertext $rct_{x'} = \{\delta, CT_{x'}\}$ and a private key $SK_y$ for some key attribute $y' \in \mathbb{Y}_k$, this algorithm runs the following steps:
  - Runs $PKEM.\text{Decaps}(oct_{x'}, SK_{y'})$ to obtain $k'$ if $R_k(x', y') = 1$. Otherwise, outputs $\perp$;
  - Computes $h = H(k')$;
  - Computes $k = (\delta)^{h^{-1}}$.

**Lemma 1.** *The proposed PPKREM scheme PPKREM described above is correct if the underlying PKEM scheme PKEM is correct and linear.*

*Proof.* We separate this proof into two parts: one for the second-level ciphertext and the other for the first-level ciphertext. For all security parameter $\lambda \in \mathbb{N}$ and description $k \in \mathbb{N}$, WLOG., we assume that the second-level ciphertext $oct_x$ and the key $k$ are generated from $PPKREM.\text{Encaps}(x)$ for some $x \in \mathbb{X}_k$ and the first-level ciphertext $rct_{x'} = \{\delta, CT_{x'}\}$ is generated from $PPKREM.\text{ReEncaps}(oct_x, rd_{y,x'})$ where $rk_{y,x'} \leftarrow \text{ReKey}(SK_y, x')$. Besides, $SK_y \leftarrow \text{KeyGen}(msk, y \in \mathbb{Y}_k)$, $SK_y \leftarrow \text{KeyGen}(msk, y' \in \mathbb{Y}_k)$, and $(params, msk) \leftarrow \text{Setup}(1^\lambda, k)$.

- Second-level ciphertext: Since the pair of second-level ciphertext and encapsulation key $(oct_x, k)$ is actually generated from $PKEM.\text{Encaps}(x \in \mathbb{X}_k)$, with the correctness of the underlying PKEM, it is trivial that the same encapsulation key $k$ can be obtained by running $PKEM.\text{Decaps}(oct_x, SK_y)$ if $R_k(x, y) = 1$. Thus, the encapsulation key $k$ can be correctly obtained.
- First-level ciphertext: Since the pair of $(CT_{x'}, k')$ is generated from $PKEM.\text{Encaps}(x')$, with the correctness of the underlying PKEM, $k'$ can be obtain using private key $SK_y$ where $R_k(x', y') = 1$ is satisfied. On the other hand, since $\delta \leftarrow PKEM.\text{Decrypt}(oct_y, (SK_y)^h)$ and the underlying PKEM is linear, $\delta$ actually equals to $PKEM.\text{Decaps}((oct_x, SK_y))^h$, that is $\delta = k^h$ if $R_k(x, y) = 1$. Therefore, we can compute:

$$(\delta)^{h^{-1}} = k^{h \cdot h^{-1}} = k.$$

In the following, we give a generic construction of a predicate proxy key re-encapsulation mechanism scheme from a secure linear predicate encryption scheme. Let $PE = (\text{Setup, KeyGen, Enc, Dec})$ be an IND-CCA secure predicate encryption scheme with linear property for predicates family $R = \{R_k\}$ and let $H_1 : M \to K, H_2 : M \to \mathbb{Z}$ be two cryptographic hash functions. We define the construction of predicate proxy key re-encapsulation mechanism as follows:

- Setup$(1^\lambda, k)$: On input of a security parameter $\lambda$ and a description *k*, this algorithm runs $(params, msk) \leftarrow PE.\text{Setup}(1^\lambda, k)$. It then outputs the parameter

*params* and the master secret key *msk*.

- Setup($1^\lambda, k$): On input of a security parameter $\lambda$ and a description *k*, this algorithm runs ($params, msk$) $\leftarrow PE.\text{Setup}(1^\lambda, k)$. It then outputs the parameter *params* and the master secret key *msk*.
- KeyGen($msk, y$): On input of a master secret key msk and a key attribute $y \in \mathbb{Y}_k$, this algorithm runs $PE.\text{KeyGen}(msk, y)$ to output a private key $SK_y$ for key attribute *y* and outputs it.
- Encaps($x$): On input of a ciphertext attribute $x \in \mathbb{X}_k$, this algorithm first randomly chooses $m_1 \in M$, and then runs $oct_x \leftarrow PE.\text{Encrypt}(x, m_1)$. Finally, it outputs a second-level ciphertext $oct_x$ and an encapsulation key $k = H_1(m_1)$.
- ReKey($SK_y, x'$): On input of a private key $SK_y$ for key attribute $y \in \mathbb{Y}_k$ and a ciphertext attribute $x' \in \mathbb{X}_k$, to generate a re-encapsulation key, this algorithm runs the following steps:
  - Randomly chooses a message $m_2 \in M$, and computes $CT_{x'} \leftarrow PE.\text{Encrypt}(x', m_2)$;
  - Computes $h = H_2(m_2)$;
  - Outputs $rk_{y,x'} = \{(SK_y)^h, CT_{x'}\}$.
- ReEncaps($oct_x, rk_{y,x'}$): On input of a second-level ciphertext $oct_x$ encapsed by ciphertext attribute $x \in \mathbb{X}_k$ and a re-encapsulation key $rk_{y,x'} = \{(SK_y)^h, CT_{x'}\}$, to generate a first-level ciphertext $rct_x$ which can be decaps by the private key $SK_y$ for key attribute $y' \in \mathbb{Y}_k$ where $R_k(x', y') = 1$, this algorithm runs $\delta \leftarrow PE.\text{Decrypt}(oct_x, (SK_y)^h)$, and outputs $rct_x = \{\delta, CT_{x'}\}$.
- $\text{Decaps}_{oct}(oct_x, SK_y)$: On input of a second-level ciphertext $oct_x$ and a private key $SK_y$ for key attribute $y \in \mathbb{Y}_k$, this algorithm runs $m_1 \leftarrow PE.\text{Decrypt}(oct_x, SK_y)$, and outputs $k = H_1(m_1)$ if $R_k(x, y) = 1$, outputs $\perp$, otherwise.
- Decrypt($rct_x, SK_{y'}$): On input of a first-level ciphertext $rct_x = \{\delta, CT_{x'}\}$ and a private key $SK_y$ for key attribute $y' \in \mathbb{Y}_k$, this algorithm runs the following steps:
  - Runs $m_2 \leftarrow PE.\text{Decrypt}(oct_{x'}, SK_{y'})$ to obtain $m_2$ if $R_k(x', y')$. Otherwise, it terminals and outputs $\perp$;
  - Computes $h = H_2(m_2)$;
  - Computes $m' = (\delta)^{h^{-1}}$
  - Outputs $k_{x'} = H_1(m')$.

**Lemma 2.** *The generic construction of PPKREM described above is correct if the underlying PE scheme PE is correct and linear.*

*Proof.* We separate this proof into two parts: one for the second-level ciphertext and the other for the first-level ciphertext. For all security parameters $\lambda$ and descriptions *k*, WLOG., we assume that the second-level ciphertext $oct_x$ and the key $k_x$ are generated from $\text{Encaps}(x \in \mathbb{X}_k)$ and the first-level ciphertext $rct_{x'} = \{\delta, CT_{x'}\}$ is generated from $\text{ReEncaps}(oct_x, rk_{y,x'})$ where $rk_{y,x'} \leftarrow \text{ReKey}(SK_y, x')$. Besides, $SK_y \leftarrow \text{KeyGen}(msk, y \in \mathbb{Y}_k)$, $SK_{y'} \leftarrow \text{KeyGen}(msk, y' \in \mathbb{Y}_k)$, and $(params, msk) \leftarrow \text{Setup}(1^\lambda, k)$.

- Second-level ciphertext: Since the second-level ciphertext $oct_x$ is actually generated from $PE.\text{Encrypt}(x \in \mathbb{X}_k, m_1 \in M)$, with the correctness of the underlying predicate encryption scheme, it is obvious that $m_1 \leftarrow PE.\text{Decrypt}(oct_x, SK_y)$ if $R_k(x, y) = 1$. Thus, the encapsulation key $k_{x'} = H_1(m_1)$ can be correctly obtained.
- First-level ciphertext: Since $CT_{x'}$ is generated from $PE.\text{Encrypt}(x', m_2)$, with the correctness of the underlying predicate encryption scheme, $m_2$ can be obtain using private key $SK_y$ where $R_k(x', y') = 1$ is satisfied. On the other hand, since $\delta \leftarrow PE.\text{Decrypt}(oct_x, (SK_y)^h)$, with the linear property of the underlying predicate encryption scheme, $\delta$ actually equals to $(m_1)^h$ if $R_k(x, y) = 1$. Therefore, we can compute:

$$(\delta)^{h^{-1}} = (m_1)^{h \cdot h^{-1}} = m,$$

and obtain the encapsulation key:

$$k_{x'} = H_1(m_1) = k_x.$$

## 5 Security Proofs

In this Section, we provide the security proofs for the payload-hiding security of the proposed constructions.

**Theorem 1.** *The first proposed construction is payload-hiding secure for second-level ciphertext under predicate family R if the underlying PKEM scheme PKEM is IND-CCA secure under the same predicate family, and the underlying hash function H is collision-resistant.*

*Proof.* Suppose there exists an adversary *A* against the payload-hiding security for second-level ciphertext of the proposed construction that has non-negligible

advantage. Then, there exists another adversary $B$ that can use $A$ to break the IND-CCA game of the underlying PKEM scheme *PKEM* with non-negligible advantage. $B$ constructs a hybrid game interacting with $A$ as follows.

- **Setup.** $B$ first invokes the IND-CCA game of *PKEM* to obtain the system parameters params. $B$ then passes params to $A$.
- **Phase 1.** In this phase, $A$ can adaptively make polynomial times of queries to the following oracles.
  - **Key generation oracle $O_{ke}$ :** When $A$ queries this oracle for a key attribute $y \in \mathbb{Y}_k$, $B$ invokes the key generation oracle of *PKEM* on the same $y$, and is given a private key $SK_y$. $B$ then passes $SK_y$ to $A$.
  - **Re-encapsulation key generation oracle $O_{rk}$ :** When $A$ queries to this oracle for a key attribute $y \in \mathbb{Y}_k$, and a ciphertext attribute $x' \in \mathbb{X}_k$, $B$ first invokes the key generation oracle of *PKEM* on the same $y$, and is given a private key $SK_y$. Then, $B$ runs $(CT_{x'}, k') \leftarrow PKEM.\text{Encaps}(x')$ and $h = H(k')$. Finally, $B$ returns $rk_{y,x'} = \{(SK_y)^h, CT_{x'}\}$ to $A$.
  - **Re-encapsulation oracle $O_{re}$ :** When $A$ queries this oracle for a key attribute $y \in \mathbb{Y}_k$, a ciphertext attribute $x' \in \mathbb{X}_k$, and a second-level ciphertext $oct_x \in \psi$, $B$ first invokes the key generation oracle of *PKEM* on the same $y$, and is given a private key $SK_y$. Then, $B$ runs $(CT_{x'}, k') \leftarrow PKEM.\text{Encaps}(x')$, computes $h = H(k')$, and sets $rk_{y,x'} = \{(SK_y)^h, CT_{x'}\}$. Finally, $B$ runs ReEncaps $(oct_x, rk_{y,x'})$ as the proposed construction to obtain a first-level ciphertext $rct_{x'} = \{\delta, CT_{x'}\}$, and returns $rct_x$ to $A$.
  - **Second-level ciphertext decapsulation oracle $O_{sde}$ :** When $A$ queries to this oracle for a ciphertext attribute $x \in \mathbb{X}_k$, and a second-level ciphertext $oct_x \in \psi$, $B$ first randomly chooses a key attribute $y \in \mathbb{Y}_k$ such that $R_k(x, y) = 1$. $B$ then invokes the decapsulation oracle of *PKEM* on $(oct_x, y)$, and is given an encapsulation key $k \in K$. In the end, $B$ returns $k$ to $A$.
  - **First-level ciphertext decapsulation oracle $O_{fde}$ :** When $A$ queries this oracle for a ciphertext attribute $x' \in \mathbb{X}_k$ and a first-level ciphertext $rct_{x'} = \{\delta, CT_{x'}\}$, $B$ first randomly chooses a key attribute $y' \in \mathbb{Y}_k$ such that $R_k(x', y') = 1$. $B$ then invokes the decapsulation oracle of *PKEM* on $(CT_{x'}, y')$, and is given a key $k' \in M$. $B$ computes $h = H(k')$ and computes $k = (\delta)^{h^{-1}}$. Finally, $B$ returns $k$ to $A$.
- **Challenge.** In this phase, $A$ submits a target ciphertext attribute $x^* \in \mathbb{X}_k$ to $B$ with following restrictions:
  - $R_k(x^*, y) = 0$ for all $y \in \mathbb{Y}_k$ submitted to $O_{ke}$;
  - for all $(y \in \mathbb{Y}_k, x' \in \mathbb{X}_k)$ submitted to $O_{rk}$, $R_k(x^*, y) = 0$

After receiving $x^*$ from A, B invokes the challenge phase of PKEM on $x^*$, and is given $(CT^*, k^*)$. B then returns $(CT^*, k^*)$ to A.

- **Phase 2.** This phase is the same as **Phase 1** with the additionally restrictions described in the payload-hiding security for second-level ciphertext game in Section 3.
- **Guess.** Finally, After $A$ outputs a guess $b'$, $B$ takes $b'$ as its own guess.

If $k^*$ is indeed an encapsulation key of $CT^*$, then $(CT^*, k^*)$ is a valid second-level ciphertext. On the other hand, if $k^*$ is sampled from the key space $K$, to the view of $A$, $(CT^*, k^*)$ is still a valid second-level ciphertext. Therefore, if $A$ can distinguish whether $k^*$ is an encapsulation key of the ciphertext $CT^*$ or not and wins the payload-hiding game for second-level ciphertext with non-negligible advantage, then $B$ can follow $A$'s answer to win the IND-CCA security game of the underlying PKEM scheme with the non-negligible advantage. Thus, the proof is completed.

**Theorem 2.** *The first proposed construction is payload-hiding secure for the first-level ciphertext under predicate family R if the underlying PKEM scheme PKEM is IND-CCA secure under the same predicate family, and the underlying hash function H is collision-resistant.*

*Proof.* Suppose there exists an adversary $A$ against the payload-hiding security for the first-level ciphertext of the proposed construction that has non-negligible advantage. Then, there exists another adversary $B$ that can use $A$ to break the IND-CCA game of the underlying PKEM scheme *PKEM* with non-negligible advantage. $B$ constructs a hybrid game interacting with $A$ as follows.

- **Setup.** $B$ first invokes the IND-CCA game of *PKEM* to obtain the system parameter params. $B$ then passes params to $A$.
- **Phase 1.** In this phase, $A$ can adaptively make polynomial times of queries to the following oracles.
  - **Key generation oracle $O_{ke}$ :** When $A$ queries this oracle for a key attribute $y \in \mathbb{Y}_k$, $B$ invokes the key generation oracle of *PKEM* on the same $y$, and is given a private key $SK_y$. $B$ then passes

$SK_y$ to $A$.

— **Re-encapsulation key generation oracle $O_{rk}$ :** When A queries this oracle for a key attribute $y \in \mathbb{Y}_k$, and a ciphertext attribute $x' \in \mathbb{X}_k$, $B$ first invokes the key generation oracle of PKEM on the same y, and is given a private key $SK_y$. Then, $B$ runs $(CT_{x'}, k') \leftarrow PKEM.\text{Encaps}(x')$ and computes $h = H(k')$. Finally, B returns $rk_{y,x'} = \{(SK_y)^h, CT_{x'}\}$ to A.

— **Re-encapsulation oracle $O_{re}$ :** When $A$ queries this oracle for a key attribute $y \in \mathbb{Y}_k$, a ciphertext attribute $x' \in \mathbb{X}_k$, and a second-level ciphertext $oct_x \in \psi$, $B$ invokes the key generation oracle of $PKEM$ on the same y, and is given a private key $SK_y$. Then, $B$ runs $(CT_{x'}, k') \leftarrow PKEM.\text{Encaps}(x')$, computes $h = H(k')$, and sets $rk_{y,x'} = \{(SK_y)^h, CT_{x'}\}$. Finally, $B$ runs $\text{ReEncaps}(oct_x, rk_{y,x'})$ as the proposed construction to obtain a first-level ciphertext $rct_{x'} = \{\delta, CT_{x'}\}$, and returns $rct_{x'}$ to $A$.

— **Second-level ciphertext decapsulation oracle $O_{sde}$**: When $A$ queries this oracle for a ciphertext attribute $x \in \mathbb{X}_k$, and a second-level ciphertext $oct_x \in \psi$, $B$ first randomly chooses a key attribute $y \in \mathbb{Y}_k$ such that $R_k(x, y) = 1$. $B$ then invokes the decapsulation oracle of $PKEM$ on $(oct_x, y)$, and is given an encapsulation key $k \in K$. In the end, $B$ returns $k$ to $A$.

— **First-level ciphertext decapsulation oracle $O_{fde}$**: When $A$ queries this oracle for a ciphertext attribute $x'$ and a first-level ciphertext $rct_{x'} = \{\delta, CT_{x'}\}$, $B$ first randomly chooses a key attribute $y' \in \mathbb{Y}_k$ such that $R_k(x', y') = 1$. $B$ then invokes the decapsulation oracle of $PKEM$ on $(CT_{x'}, y')$, and is given a key $k' \in M$. $B$ computes $h = H(k')$ and computes $k = (\delta)^{h^{-1}}$. Finally, $B$ returns $k$ to $A$.

- **Challenge.** In this phase, $A$ submits a target ciphertext attribute $x^* \in \mathbb{X}_k$ to $B$ with the restriction: $R_k(x^*, y) = 0$ for all $y \in \mathbb{Y}_k$ submitted to $O_{ke}$. After receiving $x^*$ from $A$, $B$ invokes the challenge phase of $PKEM$ on $x^*$, and is given $(CT^*, x^*)$. $B$ then randomly chooses $\tilde{x} \leftarrow \mathbb{X}_k$ and computes $(CT_{\tilde{x}}, \tilde{k}) \leftarrow PKEM.\text{Encaps}(\tilde{x})$. Next, $B$ returns $rct_x = \{(k^*)^{H(\tilde{k})}, CT_{\tilde{x}}\}$ to $A$.
- **Phase 2.** This phase is the same as **Phase 1,** with the additional restrictions described in the payload-hiding security for the first-level ciphertext game in Section 3.
- **Guess.** Finally, After $A$ outputs a guess $b'$, $B$ takes $b'$ as its own guess.

We first analyze the distribution of the first-level ciphertext $rct_{x^*} = \{(k^*)^{H(\tilde{k})}, CT_{\tilde{x}}\}$. First, the distribution of $CT_{\tilde{x}}$ is trivially the same as $CT_{x'}$ returned from $O_{re}$. Second, $\delta$ in $O_{re}$ is equal to $k^h$ if the linear property is hold, where $k$ is the encapsulation key of the second-level ciphertext and $h \in \mathbb{Z}$. That is, the distribution of $(k^*)^{H(\tilde{k})}$ is the same as $\delta$ returned from $O_{re}$. Therefore, the distribution of $rct_x$ and the first-level ciphertext queried from $O_{re}$ are the same to $A$.

In the following, we discuss the advantage of $B$ that wins the game. If $A$ wins the payload-hiding security game for first-level ciphertext of PPKREM scheme with non-negligible advantage, it implies that $A$ has the ability to distinguish whether $k^*$ is an encapsulation key of the $CT^*$. $B$ can follow $A$'s answer to win the IND-CCA security game of the underlying PKEM scheme with non-negligible advantage. Therefore, the proof is completed.

**Theorem 3.** *The second proposed construction is payload-hiding secure for the second-level ciphertext and first-level ciphertext under predicate family R, if the underlying PE scheme PE is IND-CCA secure under the same predicate family, and the underlying hash function H is collision-resistant.*

*Proof.* The proof is intuitive and similar to Theorem 1 and Theorem 2, except that the oracles invoked by $B$. More concretely, as the underlying building block is PE, $B$ can only invoke the key generation oracle and decrypte oracle. In addition, in the **Challenge** phase, $B$ is given a challenge ciphertext attribute $x^* \in \mathbb{X}_k$. Then, it randomly chooses two messages $m_1, m_2$ with the same length and sends $(x^*, m_1, m_2)$ as the challenge for IND-CCA game of the underlying PE scheme. After receiving the challege ciphertext $CT^*$, $B$ randomly chooses a message $m_b, b \in \{0,1\}$, and submits $((CT^*, k^*) = H(m_b))$ to $A$. Then, if $A$ can distinguish whether $k^*$ is an encapsulation key of the $CT^*$, $B$ can follow $A$'s answer to win the IND-CCA security game of the underlying PE scheme with non-negligible advantage. Therefore, the proof is completed.

## 6 Concrete Instantiation

In this Section, we propose a (single-hop, unidirectional) identity-based proxy key re-encapsulation scheme from Water's identity-based encryption [14]. More precisely, we first obtain an identity-based KEM from [14]. Then, since the scheme satisfies the linear property, we can adopt our proposed

generic construction to obtain an identity-based PKREM. Here, we note that identity-based KEM is actually a kind of PKEM over the predicate function $R_k$ such that $R_k(x, y)=1$ if $x=y$; $R_k(x, y)=0$, otherwise.

### 6.1 Identity-based Key Encapsulation Mechanism

Let $\mathbb{G}, \mathbb{G}_1$ be two groups with the same order p. Let $g \in \mathbb{G}$ be the generator of $\mathbb{G}$ and $e: \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$ be a bilinear mapping that maps two elements of $\mathbb{G}$ to group $\mathbb{G}_1$. The identity-based key encapsulation mechanism is presented as follows:

- Setup$(1^\lambda, k)$: On input of the security parameter $\lambda \in \mathbb{N}$, this algorithm runs the following steps to generate system parameter *params* and master secret key *msk*:
  - Randomly chooses $a \in \mathbb{Z}_p$;
  - Randomly chooses a generator $g \in \mathbb{G}$;
  - Sets $g_1 = g^a$, and randomly chooses $g_2 \in \mathbb{G}$;
  - Chooses an encode function $F: \{0,1\}^* \to \mathbb{G}$ that maps an arbitrary length string to a group element of $\mathbb{G}$;
  - Finally outputs system parameter $params = \{g, g_1, g_2, F\}$ and master secret key $msk = g_2^a$.

  Here, we note that the system parameter params will be implicit input for the following algorithms:
- Encaps$(id)$: On input of an identity $id \in \{0,1\}^*$, this algorithm first randomly selects $t \in \mathbb{Z}_p$ and then computes $c_1 = g^t$ and $c_2 = eid^t$. Finally, it outputs a ciphertext $CT = \{c_1, c_2\}$ and an encapsulation key $k = e(g_1, g_2)^t$.
- KeyGen$(msk, id)$: On input of a master secret key $msk = g_2^a$ and an identity $id \in \{0,1\}^*$, this algorithm first encodes user's identity to a group element, that is $eid = F(id)$. Then, it randomly selects $r \in \mathbb{Z}_p$ and computes $d_1 = g_2^a \cdot eid^r$ and $d_2 = g^r$. Finally, it sets the private key $SK_{id} = \{d_1, d_2\}$ for the identity *id*, and output $SK_{id}$.
- Decaps$(CT, SK_{id})$: On input of a ciphertext $CT = \{c_1, c_2\}$ and a private key $SK_{id} = \{d_1, d_2\}$, this algorithm decrypts the ciphertext by computing:

$$k = \frac{e(d_1, c_1)}{e(d_2, c_2)} = e(g_1, g_2)^t.$$

Finally, it outputs an encapsulation key $k \in \mathbb{G}_1$.

### 6.2 Identity-based Proxy Key Re-encapsulation Mechanism

In the following, we obtain an identity-based proxy key re-encapsulation mechanism scheme from the above scheme. Here we use the same notation setting as Section 6.1.

- Setup$(1^\lambda, k)$: On input of the security parameter $\lambda \in \mathbb{N}$, this algorithm runs the following steps to generate system parameter params and master secret key msk:
  - Randomly chooses $a \in \mathbb{Z}_p$;
  - Randomly chooses a generator $g \in \mathbb{G}$;
  - Sets $g_1 = g^a$, and randomly chooses $g_2 \in \mathbb{G}$;.
  - Randomly chooses $u \in \mathbb{G}$;
  - Chooses an encode function $F: \{0,1\}^* \to \mathbb{G}$ that maps an arbitrary length string to a group element of $\mathbb{G}$;
  - Chooses a cryptographic hash function $H: \mathbb{G}_1 \to \mathbb{Z}_p$;
  - Finally, outputs system parameter $params = \{g, g_1, g_2, F, H\}$ and master secret key $msk = g_2^a$. Here, we note that the system parameter *params* will be implicit input for the following algorithms.
- Encaps$(id)$: On input of an identity $id \in \{0,1\}^*$, this algorithm first randomly selects $t \in \mathbb{Z}_p$ and then computes $c_1 = g^t$, $c_2 = edi^t$. Finally, it outputs a second-level ciphertext $CT = \{c_1, c_2\}$ and an encapsulation key $k = e(g_1, g_2)^t$.
- KeyGen$(msk, id)$: On input of a master secret key $msk = g_2^a$ and an identity $id \in \{0,1\}^*$, this algorithm first encodes user's identity to a group element, that is $eid = F(id)$. Then, it randomly selects $r \in \mathbb{Z}_p$ and computes $d_1 = g_2^a \cdot (eid)^r$, $d_2 = g^r$. Finally, it sets the private key $SK_{id} = \{d_1, d_2\}$ for the identity *id*, and output $SK_{id}$.
- ReKey$(SK_{id}, id')$: On input of an identity's private key $SK_{id} = \{d_1, d_2\}$ and a target identity $id'$, this algorithm first randomly chooses $t' \in \mathbb{Z}_p$. Then, it encodes the identity, that is $eid' = F(id')$. Next, it computes $r_1 = g^{t'}$, $r_2 = edi'^{t'}$, and sets $CT_{id'} = \{r_1, r_2\}$. It also computes $h = H(e(g_1, g_2)^{t'})$. Finally, it outputs a re-encryption key:

$$rk_{id,id'} = \{SK_{id}^h = \{d_1^h, d_2^h\}, CT_{id'} = \{r_1, r_2\}\}.$$

- ReEncaps$(oct_{id}, RK_{id,id'})$: On input of a first-level ciphertext $oct_{id} = \{c_1, c_2\}$ and a re-encryption key $rk_{id,id'} = \{SK_{id}^h = \{d_1^h, d_2^h\}, CT_{id'} = \{r_1, r_2\}\}$, this algorithm computes:

$$\delta = \frac{e(d_1^h, c_1)}{e(d_2^h, c_2)} = (e(g_1, g_2)^t)^h.$$

- $\text{Decaps}_{oct}(oct_{id}, SK_{id})$: On input of a second-level ciphertext $oct_{id} = \{c_1, c_2\}$ and a private key $SK_{id} = \{d_1, d_2\}$, this algorithm decrypts the ciphertext by computing $k = \frac{e(d_1, c_1)}{e(d_2, c_2)} = e(g_1, g_2)^t$.
- $\text{Decaps}_{rct}(rct_{id}, SK_{id'})$: On input of a first-level ciphertext $rct_{id'} = \{\delta = (e(g_1, g_2)^t)^h, CT_{id'} = \{r_1, r_2\}\}$ and a private key $SK_{id} = \{d_1', d_2'\}$, this algorithm first computes:

$$H\left(\frac{e(d_1', r_1)}{e(d_2', r_2)}\right)$$
$$= H\left(\frac{e(g_2^a \cdot eid'^{r'}, g^{t'})}{e(g^{r'}, eid'^{t'})}\right)$$
$$= H\left(\frac{e(g_1, g_2)^{t'}, e(eid'^{t'}, g^{r'})}{e(g^{r'}, eid'^{t'})}\right)$$
$$= H(e(g_1, g_2)^{t'})$$
$$= h.$$

Finally, it outputs an encapsulation key:

$$k = (\delta)^{h^{-1}} = (e(g_1, g_2)^{th})^{h^{-1}} = e(g_1, g_2)^t.$$

Note that we use $r' \in \mathbb{Z}_p$ to represent the random number used in the key generation algorithm for identity $id'$.

## 7 Conclusions and Future Work

In this paper, we present two novel generic constructions that can obtain a (single-hop, unidirectional) PPKREM from a linear PKEM or from a linear PE. By combining with a secure symmetric encryption, a (single-hop, unidirectional) PPRE is also obtained. Hence, the result provides a new solution for constructing a PPRE that supports any predicate function and solves the problem that the current PPRE only supports the inner product predicate function. In further work, we will expand the single-hop setting to multi-hop setting to support more complex scenarios, while considering a bidirectional setting.

## Acknowledgments

This research was supported by the Ministry of Science and Technology, Taiwan (ROC), under Project Numbers MOST 108-2218-E-004-002-MY2, MOST 109-2221-E-004-011-MY3, MOST 109-3111-8-004-001-, MOST 110-2218-E-004-001-MBK, MOST 110-2221-E-004-003-.

## References

[1] Y.-F. Tseng, Z.-Y. Liu, R. Tso, A Generic Construction of Predicate Proxy Key Re-encapsulation Mechanism, *Asia Joint Conference on Information Security*, Taipei, Taiwan, 2020, pp. 1-8.

[2] M. Blaze, G. Bleumer, M. Strauss, Divertible Protocols and Atomic Proxy Cryptography, *International Conference on the Theory and Application of Cryptographic Techniques*, Espoo, Finland, 1998, pp. 127-144.

[3] S. S. Chow, J. Weng, Y. Yang, R. H. Deng, Efficient Unidirectional Proxy Re-encryption, *International Conference on Cryptology in Africa*, Stellenbosch, South Africa, 2010, pp. 316-332.

[4] C.-K. Chu, W.-G. Tzeng, Identity-based Proxy Re-encryption without Random Oracles, *Information Security Conference*, Valparaíso, Chile, 2007, pp. 189-202.

[5] M. Green, G. Ateniese, Identity-based Proxy Re-encryption, *International Conference on Applied Cryptography and Network Security*, Zhuhai, China, 2007, pp. 288-306.

[6] L. Wang, L. Wang, M. Mambo, E. Okamoto, New Identity-based Proxy Re-encryption Schemes to Prevent Collusion Attacks, *Pairing-Based Cryptography*, Yamanaka Hot Spring, Japan, 2010, pp. 327-346.

[7] K. Liang, L. Fang, W. Susilo, D. S. Wong, A Ciphertext-policy Attribute-based Proxy Re-encryption with Chosen-ciphertext Security, *International Conference on Intelligent Networking and Collaborative Systems*, Shaanxi Province, China, 2013, pp. 552-559.

[8] X. Liang, Z. Cao, H. Lin, J. Shao, Attribute based Proxy Re-encryption with Delegating Capabilities, *ACM Symposium on Information, Computer and Communications Security*, Sydney, Australia, 2009, pp. 276-286.

[9] S. Luo, J. Hu, Z. Chen, Ciphertext Policy Attribute-based Proxy Re-encryption, *International Conference on Information and Communications Security*, Barcelona, Spain, 2010, pp. 401-415.

[10] J. Katz, A. Sahai, B. Waters, Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products, *International Conference on the Theory and Applications of Cryptographic Techniques*, Istanbul, Turkey, 2008, pp. 146-162.

[11] M. Backes, M. Gagné, S. A. K. Thyagarajan, Fully Secure Inner-product Proxy Re-encryption with Constant Size Ciphertext, *International Workshop on Security in Cloud Computing*, Singapore, Singapore, 2015, pp. 31-40.

[12] M. Sepehri, A. Trombetta, M. Sepehri, Secure Data Sharing in Cloud using an Efficient Inner-Product Proxy Re-Encryption Scheme, *Journal of Cyber Security and Mobility*, Vol. 6, No. 3, pp. 339-378, July, 2017.

[13] M. Sepehri, A. Trombetta, M. Sepehri, E. Damiani, An Efficient Cryptography-Based Access Control using Inner-Product Proxy Re-Encryption Scheme, *International Conference on Availability, Reliability and Security*, Hamburg, Germany, 2018, pp. 1-10.

[14] B. Waters, Efficient Identity-based Encryption without

Random Oracles, *International Conference on the Theory and Applications of Cryptographic Techniques*, Aarhus, Denmark, 2005, pp. 114-127.

[15] S. Agrawal, M. Chase, A Study of Pair Encodings: Predicate Encryption in Prime Order Groups, *Theory of Cryptography*, Tel Aviv, Israel, 2016, pp. 259-288.

[16] N. Attrapadung, Dual System Encryption via Doubly Selective Security: Framework, Fully Secure Functional Encryption for Regular Languages, and More, *International Conference on the Theory and Applications of Cryptographic Techniques*, Copenhagen, Denmark, 2014, pp. 557-577.

[17] H. Feng, J. Liu, Q. Wu, W. Liu, Predicate Fully Homomorphic Encryption: Achieving Fine-Grained Access Control over Manipulable Ciphertext, *Conference on Information Security and Cryptology*, Xi'an, China, 2017, pp. 278-298.

[18] R. Canetti, S. Hohenberger, Chosen-ciphertext Secure Proxy Re-encryption, *ACM Conference on Computer and Communications Security*, Alexandria, Virginia, USA, 2007, pp. 185-194.

[19] R. H. Deng, J. Weng, S. Liu, K. Chen, Chosen-ciphertext Secure Proxy Re-encryption without Pairings, *Cryptology and Network Security*, Hong-Kong, China, 2008, pp. 1-17.

[20] B. Libert, D. Vergnaud, Unidirectional Chosen-ciphertext Secure Proxy Re-encryption, *International Conference on Practice and Theory in Public Key Cryptography*, Barcelona, Spain, 2008, pp. 360-379.

## Biographies

**Yi-Fan Tseng** was born in Kaohsiung, Taiwan. He received the Ph.D. degree and MS degree in computer science and engineering from National Sun Yat-sen University, Taiwan, in 2014 and 2018, respectively. In 2019, he has joined the faculty of the Department of Computer Science, National Chengchi University, Taipei, Taiwan.

**Zi-Yuan Liu** received the B.E. degree in computer science from National Tseng Hua University, Taiwan in 2016 and the M.E. degree in computer science from National Chengchi University, Taiwan in 2018. He is currently pursuing the Ph.D. degree in computer science at National Chengchi University, Taiwan.

**Raylin Tso** is currently a professor in the Department of Computer Science, National Chengchi University, Taiwan. He has authored or coauthored over 60 papers in referred journals and conferences in the area of information security. His research interests are mainly in the areas of cryptography, IoT security, and blockchain technology.