A Generic Construction of Predicate Proxy Key Re-encapsulation Mechanism

Yi-Fan Tseng, Zi-Yuan Liu, Raylin Tso
Department of Computer Science, National Chengchi University, Taipei, Taiwan
{yftesng, zyliu, raylin}@cs.nccu.edu.tw

***Abstract*—Proxy re-encryption (PRE), 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 recent years, many studies have explored how to construct PRE schemes that support fine-grained access control for complex application scenarios, such as identity-based PRE and attribute-based PRE. Besides, in order to achieve more flexible access control, the predicate proxy re-encryption (PPRE) is further studied. However, existing PPRE is restricted with the inner product predicate function. Therefore, how to realize the PPRE of arbitrary predicate function is still a problem to be solved. In this manuscript, we propose a secure generic construction of predicate proxy key re-encapsulation mechanism built from a "linear" predicate key encapsulation mechanism. Since the secure key encapsulation mechanism can be used as a building block to construct public key encryption, we can obtain a PPRE from our construction. As a result, the results open up new avenues for building more flexible and fine-grained PPRE.**

***Index Terms*—Predicate encryption, predicate proxy re-encryption, generic construction, single-hop, unidirectional**

I. Introduction

Proxy re-encryption (PRE), first formalized by Blaze *et al.* in 1998 [1], 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 a word, the proxy entity can delegate the decryption right from one party to another. With this flexible property, PRE yields numerous real-world applications [2], 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 mechanism, such as identity-based PRE [3], [4], [5] and attribute-based PRE [6], [7], [8].

On the other hand, predicate encryption (PE), formalized by Katz *et al.* in 2008 [9], is a paradigm for public-key encryption that conceptually generalizes the public-key encryption supporting fine-grained and role-based access to an encrypted data. More preciously, in a PE for a predicate function R_κ, a private key is associated with a key attribute y, while the ciphertext is associated with a ciphertext attribute x, where κ 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 $R_\kappa(\mathsf{x}, \mathsf{y}) = 1$. Thus, PE captures wide classes of encryption in cryptography. For example, identity-based encryption can be viewed as PE supporting "equality" predicate function, and both ciphertext attribute and key attribute are strings.

Although many identity-based PRE and attribute-based PRE have been studied, only a few researches on how to construct predicate proxy re-encryption (PPRE) [10], [11], [12]. Unfortunately, these schemes consider only the case where the predicate function is an inner product predicate. Therefore, at present, many more flexible and fine-grained predicate proxy re-encryption schemes have not been implemented and discussed. Hence, how to realize a PPRE of arbitrary predicate function remains an open problem.

A. Contributions

In this manuscript, we affirmatively solve this by proposing a generic construction that can transform any *linear* predicate key encapsulation mechanism (PKEM) to a predicate proxy key re-encapsulation mechanism (PPKREM). 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.

We also 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). Besides, we adopt our proposed generic construction for Water's identity-based encryption [13]. More preciously, 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.

B. Organization

The rest of the work is organized as follows. In Section II and III, we introduce the definition and the security requirement of PKEM and PPKREM, respectively. In Section IV and V, we propose our generic construction and provide the security proofs, respectively. In Section VI, we give an instantiate of identity-based proxy key re-encapsulation mechanism from Water's identity-based encryption. Finally, we conclude the work in Section VII.

II. Preliminary

A. Notations

For simplicity and convenience, we use the following notations and abbreviations throughout the manuscript. We use λ

978-1-7281-9922-1/20/$31.00 ©2020 IEEE
DOI 10.1109/AsiaJCIS50894.2020.00013

Authorized licensed use limited to: National Cheng Chi University. Downloaded on December 09,2021 at 03:16:03 UTC from IEEE Xplore. Restrictions apply.

to denote the security parameter. We let $\mathbb{N}$ and $\mathbb{Z}$ denote the set of positive integer and the set of integer, respectively. Besides, for a prime p, $\mathbb{Z}_p$ denotes the set of integers module p. PRE, PPRE, KEM, PKEM, PPKREM are the abbreviations of proxy re-encryption, predicate proxy re-encryption, key encapsulation mechanism, predicate key encapsulation mechanism, and predicate proxy key re-encapsulation mechanism, respectively. We also use PPT as the abbreviation of the probabilistic polynomial-time.

B. Predicate Key Encapsulation Mechanism

In this section, we recall the definition of the predicate family in [14], [15], and the definition of PKEM in [16] described by a binary relation.

Definition 1 (Predicate Family [15]). *We consider a predicate family $R = \{R_\kappa \in \mathbb{N}^c\}$ for some constant $c \in \mathbb{N}$, where a relation $R_\kappa : \mathbb{X}_\kappa \times \mathbb{Y}_\kappa \rightarrow \{0,1\}$ is a predicate function that maps a pair of ciphertext attribute in a ciphertext attribute space $\mathbb{X}_\kappa$ and key attribute in a key attribute space $\mathbb{Y}_\kappa$ to $\{0,1\}$. The family index $\kappa = (n_1, n_2, \dots)$ specifies the description of a predicate from the family.*

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

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

Correctness. A PKEM scheme $\mathcal{PKEM}$ is correct if for all $\lambda, \kappa \in \mathbb{N}$, we have

$$\mathsf{k} \leftarrow \mathsf{Decaps}(\mathsf{CT_x}, \mathsf{SK_y}), \text{ if } R_\kappa(\mathsf{x}, \mathsf{y}) = 1; \\ \perp \leftarrow \mathsf{Decaps}(\mathsf{CT_x}, \mathsf{SK_y}), \text{ otherwise,}$$

where $(\mathsf{CT_x}, \mathsf{k}) \leftarrow \mathsf{Encaps}(\mathsf{x})$, $\mathsf{SK_y} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{y})$, and $(\mathsf{params}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, \kappa)$.

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

Game - IND-CCA:

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

The advantage of the adversary $\mathcal{A}$ in winning the above game is defined as

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{PKEM},\mathcal{A}}(\lambda) = \left|\Pr[b' = b] - \tfrac{1}{2}\right|.$$

Definition 3 (IND-CCA security of PKEM). *We say that a PEKM scheme $\mathcal{PKEM}$ for predicate family R is IND-CCA secure if, for all PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{PKEM},\mathcal{A}}(\lambda)$ 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 4 (Linearity of PKEM). *We say that a correct PKEM scheme $\mathcal{PKEM} = (\mathsf{Setup}, \mathsf{Encaps}, \mathsf{KeyGen}, \mathsf{Decaps})$ for predicate family R is linear if for all $\gamma \in \mathbb{Z}, \lambda, \kappa \in \mathbb{N}, (\mathsf{CT_x}, \mathsf{k}) \leftarrow \mathsf{Encaps}(\mathsf{x})$, and $\mathsf{SK_y} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{y})$, where $(\mathsf{params}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, \kappa)$ and $R_\kappa(\mathsf{x}, \mathsf{y}) = 1$, the following equation is satisfied.*

$$\mathsf{Decaps}(\mathsf{CT_x}, (\mathsf{SK_y})^\gamma) = \mathsf{k}^\gamma,$$

where $(\mathsf{SK_y})^\gamma$ and k^γ denote the component-wise exponentiation to $\mathsf{SK_y}$ and k*, respectively.*

Authorized licensed use limited to: National Cheng Chi University. Downloaded on December 09,2021 at 03:16:03 UTC from IEEE Xplore. Restrictions apply.

III. Predicate Proxy Key Re-encapsulation Mechanism

In this section, we introduce the definition and security models of a single-hop unidirectional PPKREM. More preciously, we adopt the security game in [17], however, the game in [17] is defined for 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 [18], [19], that is, an original ciphertext is called the second-level ciphertext and a re-encapsulation ciphertext is called the first-level ciphertext.

Definition 5 (Single-hop Unidirectional Predicate Proxy Key Re-encapusulation Mechanism)**.** *Let Ψ be the encapsulation ciphertext space and $\mathcal{K}$ be the encapsulation key space, a PPKREM scheme $\mathcal{PPKREM}$ for predicate family R consists of seven PPT algorithms (*Setup, KeyGen, Encaps, ReKey, $\mathsf{ReEncaps}$, $\mathsf{Decaps_{oct}}$, $\mathsf{Decaps_{rct}}$*):*

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

Correctness. A single-hop unidirectional PPKREM scheme $\mathcal{PPKREM}$ is correct if for all $\lambda, \kappa \in \mathbb{N}$, $\mathsf{x}, \mathsf{x'} \in \mathbb{X}_\kappa$, and $\mathsf{y}, \mathsf{y'} \in \mathbb{Y}_\kappa$, we have

- $\mathsf{k} = \mathsf{Decaps_{oct}}(\mathsf{oct_x}, \mathsf{SK_y})$ if $R_\kappa(\mathsf{x}, \mathsf{y}) = 1$;
- $\perp = \mathsf{Decaps_{oct}}(\mathsf{oct_x}, \mathsf{SK_y})$ if $R_\kappa(\mathsf{x}, \mathsf{y}) = 0$;
- $\mathsf{k} = \mathsf{Decaps_{rct}}(\mathsf{ReEncaps}(\mathsf{oct_x}, \mathsf{ReKey}(\mathsf{SK_y}, \mathsf{x'})), \mathsf{SK_{y'}})$ if $R_\kappa(\mathsf{x}, \mathsf{y}) = 1 \wedge R_\kappa(\mathsf{x'}, \mathsf{y'}) = 1$;
- $\perp = \mathsf{Decaps_{rct}}(\mathsf{ReEncaps}(\mathsf{oct_x}, \mathsf{ReKey}(\mathsf{SK_y}, \mathsf{x'})), \mathsf{SK_{y'}})$ if $R_\kappa(\mathsf{x}, \mathsf{y}) = 0 \vee R_\kappa(\mathsf{x'}, \mathsf{y'}) = 0$,

where $(\mathsf{oct_x}, \mathsf{k}) \leftarrow \mathsf{Encaps}(\mathsf{x})$, $\mathsf{SK_y} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{y})$, $\mathsf{SK_{y'}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{y'})$, and $(\mathsf{params}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, \kappa)$.

Security. Before introducing the security models, we follow [17] to define the derivatives for single-hop unidirectional PPKREM.

Definition 6 (Derivatives)**.** *Let $\mathsf{x}, \mathsf{x'}, \mathsf{x''} \in \mathbb{X}_\kappa$ be the ciphertext attributes, let $\mathsf{y} \in \mathbb{Y}_\kappa$ be the key attribute, and let $\mathsf{ct}, \mathsf{ct'}, \mathsf{ct''} \in \Psi$ be the ciphertexts. The derivatives of $(\mathsf{x}, \mathsf{ct})$ is defined as follows:*

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

The following we introduce two security games to describe the security of the PPKREM between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

Game - Payload-hiding for Second-level Ciphertext:

- **Setup.** The challenger $\mathcal{C}$ runs the algorithm $\mathsf{Setup}(1^\lambda, \kappa)$ to generate parameter params and the master secret key msk. It then sends params to the adversary $\mathcal{A}$.
- **Phase 1.** The $\mathcal{A}$ may adaptively make polynomial times of queries to the following oracles.
  - **Key generation oracle** $\mathcal{O}_{\mathsf{ke}}$: On input $\mathsf{y} \in \mathbb{Y}_\kappa$ by $\mathcal{A}$, the challenger $\mathcal{C}$ computes $\mathsf{SK_y} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{y})$. It then gives $\mathsf{SK_y}$ to $\mathcal{A}$.
  - **Re-encapsulation key generation oracle** $\mathcal{O}_{\mathsf{rk}}$: On input $(\mathsf{y} \in \mathbb{Y}_\kappa, \mathsf{x'} \in \mathbb{X}_\kappa)$ by $\mathcal{A}$, the challenger $\mathcal{C}$ computes $\mathsf{rk_{y,x'}} \leftarrow \mathsf{ReKey}(\mathsf{SK_y}, \mathsf{x'})$, where $\mathsf{SK_y} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{y})$. It then gives $\mathsf{rk_{y,x'}}$ to $\mathcal{A}$.

Authorized licensed use limited to: National Cheng Chi University. Downloaded on December 09,2021 at 03:16:03 UTC from IEEE Xplore. Restrictions apply.

- **Re-encapsulation oracle** $\mathcal{O}_{\mathsf{re}}$: On input $(\mathsf{y} \in \mathbb{Y}_\kappa, \mathsf{x}' \in \mathbb{X}_\kappa, \mathsf{oct}_\mathsf{x} \in \Psi)$ by $\mathcal{A}$, the challenger $\mathcal{C}$ first computes $\mathsf{rk}_{\mathsf{y},\mathsf{x}'} \leftarrow \mathsf{ReKey}(\mathsf{SK}_\mathsf{y}, \mathsf{x}')$ where $\mathsf{SK}_\mathsf{y} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{y})$. It then computes $\mathsf{rct}_{\mathsf{x}'} \leftarrow \mathsf{ReEncaps}(\mathsf{oct}_\mathsf{x}, \mathsf{rk}_{\mathsf{y},\mathsf{x}'})$. Finally, it gives $\mathsf{rct}_{\mathsf{x}'}$ to $\mathcal{A}$.

- **Second-level ciphertext decapsulation oracle** $\mathcal{O}_{\mathsf{sde}}$: On input $(\mathsf{x} \in \mathbb{X}_\kappa, \mathsf{oct}_\mathsf{x} \in \Psi)$ by $\mathcal{A}$, the challenger $\mathcal{C}$ computes $\mathsf{k} \leftarrow \mathsf{Decaps}_{\mathsf{oct}}(\mathsf{oct}_\mathsf{x}, \mathsf{SK}_\mathsf{y})$ where $\mathsf{SK}_\mathsf{y} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{y})$ and $R_\kappa(\mathsf{x}, \mathsf{y}) = 1$. It then returns k to $\mathcal{A}$.

- **First-level ciphertext decapsulation oracle** $\mathcal{O}_{\mathsf{fde}}$: On input $(\mathsf{x}' \in \mathbb{X}_\kappa, \mathsf{rct}_{\mathsf{x}'} \in \Psi)$ by $\mathcal{A}$, the challenger $\mathcal{C}$ computes $\mathsf{k} \leftarrow \mathsf{Decaps}_{\mathsf{rct}}(\mathsf{rct}_{\mathsf{x}'}, \mathsf{SK}_{\mathsf{y}'})$ where $\mathsf{SK}_{\mathsf{y}'} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{y}')$ and $R_\kappa(\mathsf{x}', \mathsf{y}') = 1$. It then returns k to $\mathcal{A}$.

- **Challenge.** $\mathcal{A}$ outputs a ciphertext attribute $\mathsf{x}^* \in \mathbb{X}_\kappa$ with restriction that
  1) $R_\kappa(\mathsf{x}^*, \mathsf{y}) = 0$ for all $\mathsf{y} \in \mathbb{Y}_\kappa$ submitted to $\mathcal{O}_{\mathsf{ke}}$;
  2) for all $(\mathsf{y} \in \mathbb{Y}_\kappa, \mathsf{x}' \in \mathbb{X}_\kappa)$ submitted to $\mathcal{O}_{\mathsf{rk}}$, $R_\kappa(\mathsf{x}^*, \mathsf{y}) = 0$.

  If x^* satisfies the above requirements, the challenger $\mathcal{C}$ then randomly chooses a bit $b \in \{0, 1\}$, and responds with $(\mathsf{oct}^*_{\mathsf{x}^*}, \mathsf{k}^*_b)$, where $(\mathsf{oct}^*_{\mathsf{x}^*}, \mathsf{k}^*_0) \leftarrow \mathsf{Encaps}(\mathsf{x}^*)$ and k^*_1 is randomly chosen from $\mathcal{K}$.

- **Phase 2.** $\mathcal{A}$ can continue to issue more queries to the oracles as follows:
  - **Key generation oracle** $\mathcal{O}_{\mathsf{ke}}$: The oracle is the same as **Phase 1** with three additional restrictions:
    * $R_\kappa(\mathsf{x}^*, \mathsf{y}) = 0$;
    * for all $\mathsf{y}' \in \mathbb{Y}_\kappa$ such that $R_\kappa(\mathsf{x}^*, \mathsf{y}') = 1 \wedge R_\kappa(\mathsf{x}, \mathsf{y}) = 1$, the tuple $(\mathsf{y}', \mathsf{x})$ must not have been queried to $\mathcal{O}_{\mathsf{rk}}$ before;
    * for all $\mathsf{y}' \in \mathbb{Y}_\kappa$, $\mathsf{x}, \mathsf{x}' \in \mathbb{X}_\kappa$, and $\mathsf{oct}' \in \Psi$ such that $R_\kappa(\mathsf{x}, \mathsf{y}) = 1 \wedge R_\kappa(\mathsf{x}', \mathsf{y}') = 1$, and $(\mathsf{x}', \mathsf{oct}')$ is a derivative of $(\mathsf{x}^*, \mathsf{oct}^*_{\mathsf{x}^*})$, the tuple $(\mathsf{y}', \mathsf{x}, \mathsf{oct}')$ has not been queried to $\mathcal{O}_{\mathsf{re}}$ before.
  - **Re-encapsulation key generation oracle** $\mathcal{O}_{\mathsf{rk}}$: The oracle is the same as **Phase 1** with a restriction: if $\mathsf{x} = \mathsf{x}^*$, then for all $\mathsf{y}' \in \mathbb{Y}_\kappa$ such that $R_\kappa(\mathsf{x}, \mathsf{y}) = 1 \wedge R_\kappa(\mathsf{x}', \mathsf{y}') = 1$, y' must not have been queried to $\mathcal{O}_{\mathsf{ke}}$ before.
  - **Re-encapsulation oracle** $\mathcal{O}_{\mathsf{re}}$: The oracle is the same as **Phase 1** with a restriction: if $(\mathsf{x}, \mathsf{oct}_\mathsf{x})$ is a derivative of $(\mathsf{x}^*, \mathsf{oct}^*_{\mathsf{x}^*})$, then for all $\mathsf{y}' \in \mathbb{Y}_\kappa$ such that $R_\kappa(\mathsf{x}, \mathsf{y}) = 1 \wedge R_\kappa(\mathsf{x}', \mathsf{y}') = 1$, y' must not have been queried to $\mathcal{O}_{\mathsf{ke}}$ before.
  - **Second-level ciphertext decapsulation oracle** $\mathcal{O}_{\mathsf{sde}}$: The oracle is the same as **Phase 1** with a restriction: $(\mathsf{x}, \mathsf{oct}_\mathsf{x})$ is not a derivative of $(\mathsf{x}^*, \mathsf{oct}^*_{\mathsf{x}^*})$.
  - **First-level ciphertext decapsulation oracle** $\mathcal{O}_{\mathsf{fde}}$: The oracle is the same as **Phase 1** with a restriction: $(\mathsf{x}', \mathsf{rct}_{\mathsf{x}'})$ is not a derivative of $(\mathsf{x}^*, \mathsf{oct}^*_{\mathsf{x}^*})$.

- **Guess.** In the end, $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$ and wins the game if $b = b'$.

The advantage of the adversary $\mathcal{A}$ in winning the above game is defined as

$$\mathsf{Adv}^{\mathsf{PH\text{-}SC}}_{\mathcal{PPKREM},\mathcal{A}}(\lambda) = \left|\Pr[b = b'] - \tfrac{1}{2}\right|.$$

Definition 7 (Payload-hiding Security for Second-level Ciphertext)**.** *We say that a single-hop unidirectional PPKREM scheme $\mathcal{PPKREM}$ for predicate family R is payload-hiding secure for second-level ciphertext if for any polynomial time adversary $\mathcal{A}$ the function* $\mathsf{Adv}^{\mathsf{PH\text{-}SC}}_{\mathcal{PPKREM},\mathcal{A}}(\lambda)$ *is negligible.*

Game - Payload-hiding for First-level Ciphertext:

- **Setup.** The challenger $\mathcal{C}$ runs the algorithm $\mathsf{Setup}(1^\lambda, \kappa)$ to generate parameter params and the master secret key msk. It then sends params to the adversary $\mathcal{A}$.

- **Phase 1.** The $\mathcal{A}$ may adaptively make a polynomial times of queries to the following oracles.
  - **Key generation oracle** $\mathcal{O}_{\mathsf{ke}}$: On input $\mathsf{y} \in \mathbb{Y}_\kappa$ by $\mathcal{A}$, the challenger $\mathcal{C}$ computes $\mathsf{SK}_\mathsf{y} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{y})$. It then gives SK_y to $\mathcal{A}$.

  - **Re-encapsulation key generation oracle** $\mathcal{O}_{\mathsf{rk}}$: On input $(\mathsf{y} \in \mathbb{Y}_\kappa, \mathsf{x}' \in \mathbb{X}_\kappa)$ by $\mathcal{A}$, the challenger $\mathcal{C}$ computes $\mathsf{rk}_{\mathsf{y},\mathsf{x}'} \leftarrow \mathsf{ReKey}(\mathsf{SK}_\mathsf{y}, \mathsf{x}')$, where $\mathsf{SK}_\mathsf{y} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{y})$. It then gives $\mathsf{rk}_{\mathsf{y},\mathsf{x}'}$ to $\mathcal{A}$.

  - **Re-encapsulation oracle** $\mathcal{O}_{\mathsf{re}}$: On input $(\mathsf{y} \in \mathbb{Y}_\kappa, \mathsf{x}' \in \mathbb{X}_\kappa, \mathsf{oct}_\mathsf{x} \in \Psi)$ by $\mathcal{A}$, the challenger $\mathcal{C}$ first computes $\mathsf{rk}_{\mathsf{y},\mathsf{x}'} \leftarrow \mathsf{ReKey}(\mathsf{SK}_\mathsf{y}, \mathsf{x}')$ where $\mathsf{SK}_\mathsf{y} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{y})$. It then computes $\mathsf{rct}_{\mathsf{x}'} \leftarrow \mathsf{ReEncaps}(\mathsf{oct}_\mathsf{x}, \mathsf{rk}_{\mathsf{y},\mathsf{x}'})$. Finally, it gives $\mathsf{rct}_{\mathsf{x}'}$ to $\mathcal{A}$.

  - **Second-level ciphertext decapsulation oracle** $\mathcal{O}_{\mathsf{sde}}$: On input $(\mathsf{x} \in \mathbb{X}_\kappa, \mathsf{oct}_\mathsf{x} \in \Psi)$ by $\mathcal{A}$, the challenger $\mathcal{C}$ computes $\mathsf{k} \leftarrow \mathsf{Decaps}_{\mathsf{oct}}(\mathsf{oct}_\mathsf{x}, \mathsf{SK}_\mathsf{y})$ where $\mathsf{SK}_\mathsf{y} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{y})$ and $R_\kappa(\mathsf{x}, \mathsf{y}) = 1$. It then returns k to $\mathcal{A}$.

  - **First-level ciphertext decapsulation oracle** $\mathcal{O}_{\mathsf{fde}}$: On input $(\mathsf{x}' \in \mathbb{X}_\kappa, \mathsf{rct}_{\mathsf{x}'} \in \Psi)$ by $\mathcal{A}$, the challenger $\mathcal{C}$ computes $\mathsf{k} \leftarrow \mathsf{Decaps}_{\mathsf{rct}}(\mathsf{rct}_{\mathsf{x}'}, \mathsf{SK}_{\mathsf{y}'})$ where $\mathsf{SK}_{\mathsf{y}'} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{y}')$ and $R_\kappa(\mathsf{x}', \mathsf{y}') = 1$. It then returns k to $\mathcal{A}$.

- **Challenge.** $\mathcal{A}$ outputs a ciphertext attribute $\mathsf{x}^* \in \mathbb{X}_\kappa$ with restriction: for all $\mathsf{y} \in \mathbb{Y}_\kappa$ submitted to $\mathcal{O}_{\mathsf{ke}}$, $R_\kappa(\mathsf{x}^*, \mathsf{y}) = 0$. If x^* satisfies the above requirements, the challenger $\mathcal{C}$ first computes $\mathsf{SK}_{\mathsf{y}^*} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{y}^*)$ where $R_\kappa(\mathsf{x}^*, \mathsf{y}^*) = 1$. Then, it chooses a ciphertext

Authorized licensed use limited to: National Cheng Chi University. Downloaded on December 09,2021 at 03:16:03 UTC from IEEE Xplore. Restrictions apply.

attribute $\hat{\mathsf{x}} \in \mathbb{X}_\kappa$, and randomly chooses a bit $b \in \{0,1\}$. Next, it computes

1) $\mathsf{rk}_{\mathsf{y}^*,\hat{\mathsf{x}}} \leftarrow \mathsf{ReKey}(\mathsf{SK}_{\mathsf{y}^*}, \hat{\mathsf{x}})$;
2) $\mathsf{rct}^*_{\hat{\mathsf{x}}} \leftarrow \mathsf{ReEncaps}(\mathsf{oct}^*_{\mathsf{x}^*}, \mathsf{rk}_{\mathsf{y}^*,\hat{\mathsf{x}}})$,

where $(\mathsf{oct}^*_{\mathsf{x}^*}, \mathsf{k}^*_0) \leftarrow \mathsf{Encaps}(\mathsf{x}^*)$ and k^*_1 is randomly chosen from $\mathcal{K}$. Finally, it responds $(\mathsf{rct}^*_{\hat{\mathsf{x}}}, \mathsf{k}^*_b)$ to $\mathcal{A}$.

- **Phase 2.** $\mathcal{A}$ can continue to issue more queries to the oracles as in **Phase 1** with two additional restrictions:
  - **Key generation oracle** $\mathcal{O}_{\mathsf{ke}}$: for all $\mathsf{y} \in \mathbb{Y}_\kappa$, $R_\kappa(\hat{\mathsf{x}}, \mathsf{y}) = 0$.
  - **First-level ciphertext decapsulation oracle** $\mathcal{O}_{\mathsf{fde}}$: it cannot be queried with the challenge ciphertext $\mathsf{rct}^*_{\hat{\mathsf{x}}}$ as input.
- **Guess.** In the end, $\mathcal{A}$ outputs a guess $b' \in \{0,1\}$ and wins the game if $b = b'$.

The advantage of the adversary $\mathcal{A}$ in winning the above game if defined as

$$\mathsf{Adv}^{\mathsf{PH\text{-}FC}}_{\mathcal{PPKREM},\mathcal{A}}(\lambda) = \left|\Pr[b = b'] - \tfrac{1}{2}\right|.$$

Definition 8 (Payload-hiding Security for First-level Ciphertext)**.** *We say that a single-hop unidirectional PPKREM scheme $\mathcal{PPKREM}$ for predicate family R is payload-hiding secure for first-level ciphertext if for PPT adversary $\mathcal{A}$ the function $\mathsf{Adv}^{\mathsf{PH\text{-}FC}}_{\mathcal{PPKREM},\mathcal{A}}(\lambda)$ is negligible.*

IV. Generic Construction of Predicate Proxy Key Re-encapsulation Mechanism

In this section, we give a generic construction that can obtain a PPKREM scheme from a secure linear PKEM scheme. At a high level, to generate a re-encapsulation key $\mathsf{rk}_{\mathsf{y},\mathsf{x}'}$, we first encaps the ciphertext attribute x' to obtain a pair $(\mathsf{CT}_{\mathsf{x}'}, \mathsf{k}')$, then compute $h = \mathcal{H}(\mathsf{k}')$, where $\mathcal{H}(\cdot)$ is a cryptographic hash function. Next we let the re-encapsulation key be

$$\mathsf{rk}_{\mathsf{y},\mathsf{x}'} = \{(\mathsf{SK}_\mathsf{y})^h, \mathsf{CT}_{\mathsf{x}'}\},$$

where $(\mathsf{SK}_\mathsf{y})^h$ denotes the h component-wise exponentiation to SK_y. Note that due to the hardness of the discrete-log problem, the proxy entity is impossible to obtain h from $(\mathsf{SK}_\mathsf{y})^h$. In other word, the proxy entity is also impossible to recover SK_y from $\mathsf{rk}_{\mathsf{y},\mathsf{x}'}$. In order to generate a first-level ciphertext $\mathsf{rct}_{\mathsf{x}'}$ from the second-level ciphertext oct_x using the re-encapsulation key $\mathsf{rk}_{\mathsf{y},\mathsf{x}'}$, we directly runs

$$\delta \leftarrow \mathsf{PKEM.Decaps}(\mathsf{oct}_\mathsf{x}, (\mathsf{SK}_\mathsf{y})^h).$$

With the linear property of PKEM (Definition 4), if $R_\kappa(\mathsf{x}, \mathsf{y}) = 1$, δ actually equals to $(\mathsf{k})^h$, where $(\mathsf{oct}_\mathsf{x}, \mathsf{k}) \leftarrow \mathsf{PKEM.Encaps}(\mathsf{x})$. Then, the first-level ciphertext $\mathsf{rk}_{\mathsf{y},\mathsf{x}'}$ is set as $\{\delta, \mathsf{CT}_{\mathsf{x}'}\}$. Besides, only the proxy receiver can decaps $\mathsf{CT}_{\mathsf{x}'}$ using her/his private key to obtain k', and recovery the value hided in the encapsulation key, i.e., $h = \mathcal{H}(\mathsf{k}')$. Finally, the proxy receiver can obtain

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

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

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

V. Security Proofs

In this section, we provide the security proofs for the payload-hiding security of the proposed construction. Intu-

Authorized licensed use limited to: National Cheng Chi University. Downloaded on December 09,2021 at 03:16:03 UTC from IEEE Xplore. Restrictions apply.

itively, to show that our proposed scheme is secure, we define two hybrid games in which the challenger $\mathcal{B}$ and the adversary $\mathcal{A}$ interact in the payload-hiding game for second-level ciphertext and the first-level ciphertext, respectively. In this two hybrid games, we show that if exists an adversary $\mathcal{A}$ against the payload-hiding security of the proposed construction that has non-negligible advantage, then $\mathcal{B}$ can use $\mathcal{A}$ to break the IND-CCA game of the underlying PKEM scheme $\mathcal{PKEM}$ with non-negligible advantage.

- Payload-hiding security for second-level ciphertext: In challenge phase, $\mathcal{B}$ takes the response k^* from $\mathcal{PKEM}$ as the challenge response for $\mathcal{A}$. If k^* is indeed an encapsulation key of CT^*, then $(\mathsf{CT}^*, \mathsf{k}^*)$ is a valid second-level ciphertext. On the other hand, if k^* is sampled from the key space $\mathcal{K}$, to the view of $\mathcal{A}$, $(\mathsf{CT}^*, \mathsf{k}^*)$ is still a valid second-level ciphertext. Therefore, if $\mathcal{A}$ 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 $\mathcal{B}$ can follow $\mathcal{A}$'s answer to win the IND-CCA security game of the underlying PKEM scheme with the non-negligible advantage.
- Payload-hiding security for first-level ciphertext: In challenge phase, $\mathcal{B}$ takes the response k^* from $\mathcal{PKEM}$ as the challenge response for $\mathcal{A}$. If $\mathcal{A}$ wins the payload-hiding security game for first-level ciphertext of PP-KREM scheme with non-negligible advantage it implies that $\mathcal{A}$ has the ability to distinguish whether k^* is an encapsulation key of the CT^*. $\mathcal{B}$ can follow $\mathcal{A}$'s answer to win the IND-CCA security game of the underlying PKEM scheme with non-negligible advantage.

Due to the space limit, we include the security proofs in full version [20].

Remark 1. *In order to provide a more general construction, we start our proposed scheme from a PKEM rather than a PE. Since a secure KEM combines a secure symmetric encryption implies a secure public-key encryption, we can combine our PPRKEM with a secure symmetric encryption to obtain a PPRE. On the other hand, a secure PE implies a secure PKEM, our construction can also be obtained from a secure PE. Due to the page limit, we leave these parts in the full version work.*

VI. Instantiate

In this section, we propose a (single-hop unidirectional) identity-based proxy key re-encapsulation scheme from Water's identity-based encryption [13]. More preciously, we first obtain an identity-based KEM from [13]. Then, since the scheme satisfies the linear property, we can adopt our proposed generic construction to obtain an identity-based proxy key re-encapsulation mechanism scheme. Here, we note that identity-based KEM actually is a kind of PKEM over the predicate function R_κ such that $R_\kappa(\mathsf{x},\mathsf{y}) = 1$ if $\mathsf{x} = \mathsf{y}$; $R_\kappa(\mathsf{x},\mathsf{y}) = 0$, otherwise.

A. Identity-based Key Encapsulation Mechanism

Let $\mathbb{G}, \mathbb{G}_1$ be two groups with the same order p. Besides, 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 from [13] is presented as follows.

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

  Here, we note that the system parameter params will be an implicitly input for the following algorithms.
- $\mathsf{Encaps}(\mathsf{id})$: On input an identity $\mathsf{id} \in \{0,1\}^*$, this algorithm first randomly selects $t \in \mathbb{Z}_p$ and then computes
  - $\mathsf{c}_1 = g^t$;
  - $\mathsf{c}_2 = \mathsf{eid}^t$.

  Finally, it outputs a ciphertext $\mathsf{CT} = \{\mathsf{c}_1, \mathsf{c}_2\}$ and an encapsulation key $\mathsf{k} = e(g_1, g_2)^t$.
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$: On input a master secret key $\mathsf{msk} = g_2^\alpha$ and an identity $\mathsf{id} \in \{0,1\}^*$, this algorithm first encodes user's identity to a group element, that is $\mathsf{eid} = \mathcal{F}(\mathsf{id})$. Then, it randomly selects $r \in \mathbb{Z}_p$ and computes
  - $\mathsf{d}_1 = g_2^\alpha \cdot \mathsf{eid}^r$;
  - $\mathsf{d}_2 = g^r$.

  Finally, it sets the private key $\mathsf{SK}_{\mathsf{id}} = \{\mathsf{d}_1, \mathsf{d}_2\}$ for the identity id, and output $\mathsf{SK}_{\mathsf{id}}$.
- $\mathsf{Decaps}(\mathsf{CT}, \mathsf{SK}_{\mathsf{id}})$: On input a ciphertext $\mathsf{CT} = \{\mathsf{c}_1, \mathsf{c}_2\}$ and a private key $\mathsf{SK}_{\mathsf{id}} = \{\mathsf{d}_1, \mathsf{d}_2\}$, this algorithm decrypts the ciphertext by computing: $\mathsf{k} = \frac{e(\mathsf{d}_1, \mathsf{c}_1)}{e(\mathsf{d}_2, \mathsf{c}_2)} = e(g_1, g_2)^t$. Finally, it outputs an encapsulation key $\mathsf{k} \in \mathbb{G}_1$.

B. Identity-based Proxy Key Re-encapsulation Mechanism

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 VI-A.

- $\mathsf{Setup}(1^\lambda)$: On input 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 $\alpha \in \mathbb{Z}_p$;

Authorized licensed use limited to: National Cheng Chi University. Downloaded on December 09,2021 at 03:16:03 UTC from IEEE Xplore. Restrictions apply.

- Randomly chooses a generator $g \in \mathbb{G}$;
- Sets $g_1 = g^{\alpha}$, and randomly chooses $g_2 \in \mathbb{G}$;
- Randomly chooses $u \in \mathbb{G}$;
- Chooses an encode function $\mathcal{F} : \{0,1\}^* \to \mathbb{G}$ that maps an arbitrary length string to a group element of $\mathbb{G}$;
- Chooses a cryptographic hash function $\mathcal{H} : \mathbb{G}_1 \to \mathbb{Z}_p$;
- Finally outputs system parameter $\mathsf{params} = \{g, g_1, g_2, \mathcal{F}, \mathcal{H}\}$ and master secret key $\mathsf{msk} = g_2^{\alpha}$.

Here, we note that the system parameter params will be an implicitly input for the following algorithms.

- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$: On input a master secret key $\mathsf{msk} = g_2^{\alpha}$ and an identity $\mathsf{id} \in \{0,1\}^*$, this algorithm first encodes user's identity to a group element, that is $\mathsf{eid} = \mathcal{F}(\mathsf{id})$. Then, it randomly selects $r \in \mathbb{Z}_p$ and computes $\mathsf{d}_1 = g_2^{\alpha} \cdot (u \cdot \mathsf{eid})^r$, $\mathsf{d}_2 = g^r$. Finally, it sets the private key $\mathsf{SK}_{\mathsf{id}} = \{\mathsf{d}_1, \mathsf{d}_2\}$ for the identity id, and output $\mathsf{SK}_{\mathsf{id}}$.
- $\mathsf{Encaps}(\mathsf{id})$: On input an identity $\mathsf{id} \in \{0,1\}^*$, this algorithm first randomly selects $t \in \mathbb{Z}_p$ and then computes $\mathsf{c}_1 = g^t$, $\mathsf{c}_2 = \mathsf{eid}^t$. Finally, it outputs a second-level ciphertext $\mathsf{oct} = \{\mathsf{c}_1, \mathsf{c}_2\}$ and an encapsulation key $\mathsf{k} = e(g_1, g_2)^t$.
- $\mathsf{ReKey}(\mathsf{SK}_{\mathsf{id}}, \mathsf{id}')$: On input an identity's private key $\mathsf{SK}_{\mathsf{id}} = \{\mathsf{d}_1, \mathsf{d}_2\}$ and a target identity id', this algorithm first randomly chooses $t' \in \mathbb{Z}_p$. Then, it encodes the identity, that is $\mathsf{eid}' = \mathcal{F}(\mathsf{id}')$. Next, it computes $\mathsf{r}_1 = g^{t'}$, $\mathsf{r}_2 = \mathsf{eid}'^{t'}$, and sets $\mathsf{CT}_{\mathsf{id}'} = \{\mathsf{r}_1, \mathsf{r}_2\}$. It also computes $h = \mathcal{H}(e(g_1, g_2)^{t'})$. Finally, it outputs a re-encryption key $\mathsf{rk}_{\mathsf{id},\mathsf{id}'} = \{\mathsf{SK}_{\mathsf{id}}^h = \{\mathsf{d}_1^h, \mathsf{d}_2^h\}, \mathsf{CT}_{\mathsf{id}'} = \{\mathsf{r}_1, \mathsf{r}_2\}\}$.
- $\mathsf{ReEncaps}(\mathsf{oct}_{\mathsf{id}}, \mathsf{rk}_{\mathsf{id},\mathsf{id}'})$: On input a first-level ciphertext $\mathsf{oct}_{\mathsf{id}} = \{\mathsf{c}_1, \mathsf{c}_2\}$ and a re-encryption key $\mathsf{rk}_{\mathsf{id},\mathsf{id}'} = \{\mathsf{SK}_{\mathsf{id}}^h = \{\mathsf{d}_1^h, \mathsf{d}_2^h\}, \mathsf{CT}_{\mathsf{id}'} = \{\mathsf{r}_1, \mathsf{r}_2\}\}$, this algorithm computes $\delta = \frac{e(\mathsf{d}_1^h, \mathsf{c}_1)}{e(\mathsf{d}_2^h, \mathsf{c}_2)} = (e(g_1, g_2)^t)^h$. Finally, it outputs a first-level ciphertext $\mathsf{rct}_{\mathsf{id}'} = \{\delta, \mathsf{CT}_{\mathsf{id}'}\}$.
- $\mathsf{Decaps}_{\mathsf{oct}}(\mathsf{oct}_{\mathsf{id}}, \mathsf{SK}_{\mathsf{id}})$: On input a second-level ciphertext $\mathsf{oct}_{\mathsf{id}} = \{\mathsf{c}_1, \mathsf{c}_2\}$ and a private key $\mathsf{SK}_{\mathsf{id}} = \{\mathsf{d}_1, \mathsf{d}_2\}$, this algorithm decrypts the ciphertex by computing: $\mathsf{k} = \frac{e(\mathsf{d}_1, \mathsf{c}_1)}{e(\mathsf{d}_2, \mathsf{c}_2)} = e(g_1, g_2)^t$. Finally, it outputs an encapsulation key $\mathsf{k} \in \mathbb{G}_1$.
- $\mathsf{Decaps}_{\mathsf{rct}}(\mathsf{rct}_{\mathsf{id}'}, \mathsf{SK}_{\mathsf{id}'})$: On input a first-level ciphertext $\mathsf{rct}_{\mathsf{id}'} = \{\delta = (e(g_1, g_2)^t)^h, \mathsf{CT}_{\mathsf{id}'} = \{\mathsf{r}_1, \mathsf{r}_2\}\}$ and a private key $\mathsf{SK}_{\mathsf{id}'} = \{\mathsf{d}_1', \mathsf{d}_2'\}$, this algorithm first computes: $\mathcal{H}\left(\frac{e(\mathsf{d}_1', \mathsf{r}_1)}{e(\mathsf{d}_2', \mathsf{r}_2)}\right) = \mathcal{H}\left(\frac{e(g_2^{\alpha} \cdot \mathsf{eid}'^{r'}, g^{t'})}{e(g^{r'}, \mathsf{eid}'^{t'})}\right) = \mathcal{H}\left(\frac{e(g_1, g_2)^{t'} \cdot e(\mathsf{eid}'^{r'}, g^{t'})}{e(g^{r'}, \mathsf{eid}'^{t'})}\right) = \mathcal{H}\left(e(g_1, g_2)^{t'}\right) = h.$
  Finally, it outputs an encapsulation key $\mathsf{k} = (\delta)^{h^{-1}} = (e(g_1, g_2)^{th})^{h^{-1}} = e(g_1, g_2)^t$. Note that we use $r' \in \mathbb{Z}_p$ to represent the random number that use in the key generation algorithm for identity id'.

VII. Conclusions and Future Work

In this manuscript, we present a novel generic construction that can obtain a (single-hop unidirectional) predicate proxy key re-encapsulation mechanism from a linear predicate key encapsulation mechanism. Besides, by combining with a secure symmetric encryption, a (single-hop unidirectional) predicate proxy re-encryption mechanism is also obtained. Hence, the result provides a new solution for constructing a predicate proxy re-encryption that supports any predicate function, and solves the problem that the current predicate proxy re-encryption 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 bidirectional setting.

Acknowledgment

This research was supported by the Ministry of Science and Technology, Taiwan (ROC), under Project Numbers MOST 108-2218-E-004-001-, MOST 108-2218-E-004-002-MY2, MOST 109-2218-E-011-007-, and by Taiwan Information Security Center at National Sun Yat-sen University (TWISC@NSYSU).

References

[1] M. Blaze, G. Bleumer, and M. Strauss, "Divertible Protocols and Atomic Proxy Cryptography," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1998, pp. 127–144.
[2] S. S. Chow, J. Weng, Y. Yang, and R. H. Deng, "Efficient Unidirectional Proxy Re-encryption," in *International Conference on Cryptology in Africa*. Springer, 2010, pp. 316–332.
[3] M. Green and G. Ateniese, "Identity-based Proxy Re-encryption," in *International Conference on Applied Cryptography and Network Security*. Springer, 2007, pp. 288–306.
[4] C.-K. Chu and W.-G. Tzeng, "Identity-based Proxy Re-encryption without Random Oracles," in *International Conference on Information Security*. Springer, 2007, pp. 189–202.
[5] L. Wang, L. Wang, M. Mambo, and E. Okamoto, "New Identity-based Proxy Re-encryption Schemes to Prevent Collusion Attacks," in *International Conference on Pairing-Based Cryptography*. Springer, 2010, pp. 327–346.
[6] X. Liang, Z. Cao, H. Lin, and J. Shao, "Attribute based Proxy Re-encryption with Delegating Capabilities," in *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, 2009, pp. 276–286.
[7] S. Luo, J. Hu, and Z. Chen, "Ciphertext Policy Attribute-based Proxy Re-encryption," in *International Conference on Information and Communications Security*. Springer, 2010, pp. 401–415.
[8] K. Liang, L. Fang, W. Susilo, and D. S. Wong, "A Ciphertext-policy Attribute-based Proxy Re-encryption with Chosen-ciphertext Security," in *2013 5th International Conference on Intelligent Networking and Collaborative Systems*. IEEE, 2013, pp. 552–559.
[9] J. Katz, A. Sahai, and B. Waters, "Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2008, pp. 146–162.

Authorized licensed use limited to: National Cheng Chi University. Downloaded on December 09,2021 at 03:16:03 UTC from IEEE Xplore. Restrictions apply.

[10] M. Sepehri, A. Trombetta, M. Sepehri, and E. Damiani, "An Efficient Cryptography-Based Access Control Using Inner-Product Proxy Re-Encryption Scheme," in *Proceedings of the 13th International Conference on Availability, Reliability and Security*. Association for Computing Machinery, 2018.

[11] M. Backes, M. Gagné, and S. A. K. Thyagarajan, "Fully Secure Inner-product Proxy Re-encryption with Constant Size Ciphertext," in *Proceedings of the 3rd International Workshop on Security in Cloud Computing*, 2015, pp. 31–40.

[12] M. Sepehri, A. Trombetta, and 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, 2017.

[13] B. Waters, "Efficient Identity-based Encryption without Random Oracles," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2005, pp. 114–127.

[14] S. Agrawal and M. Chase, "A Study of Pair Encodings: Predicate Encryption in Prime Order Groups," in *Theory of Cryptography*, E. Kushilevitz and T. Malkin, Eds. Springer Berlin Heidelberg, 2016, pp. 259–288.

[15] N. Attrapadung, "Dual System Encryption via Doubly Selective Security: Framework, Fully Secure Functional Encryption for Regular Languages, and More," in *Advances in Cryptology – EUROCRYPT 2014*, P. Q. Nguyen and E. Oswald, Eds. Springer Berlin Heidelberg, 2014, pp. 557–577.

[16] H. Feng, J. Liu, Q. Wu, and W. Liu, "Predicate Fully Homomorphic Encryption: Achieving Fine-Grained Access Control over Manipulable Ciphertext," in *International Conference on Information Security and Cryptology*. Springer, 2017, pp. 278–298.

[17] R. Canetti and S. Hohenberger, "Chosen-ciphertext Secure Proxy Re-encryption," in *Proceedings of the 14th ACM conference on Computer and communications security*, 2007, pp. 185–194.

[18] R. H. Deng, J. Weng, S. Liu, and K. Chen, "Chosen-ciphertext Secure Proxy Re-encryption without Pairings," in *International Conference on Cryptology and Network Security*. Springer, 2008, pp. 1–17.

[19] B. Libert and D. Vergnaud, "Unidirectional Chosen-ciphertext Secure Proxy Re-encryption," in *International Workshop on Public Key Cryptography*. Springer, 2008, pp. 360–379.

[20] Y.-F. Tseng, Z.-Y. Liu, and R. Tso, "A Generic Construction of Predicate Proxy Key Re-encapsulation Mechanism," Cryptology ePrint Archive, Report 2020/713, 2020, https://eprint.iacr.org/2020/713.

Authorized licensed use limited to: National Cheng Chi University. Downloaded on December 09,2021 at 03:16:03 UTC from IEEE Xplore. Restrictions apply.