Universal Proxy Re-Encryption

Nico Döttling [1] Ryo Nishimaki [2]

[1] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
nico.doettling@cispa.saarland
[2] NTT Secure Platform Laboratories, Tokyo, Japan
ryo.nishimaki.zk@hco.ntt.co.jp

Abstract

We put forward the notion of universal proxy re-encryption (UPRE). A UPRE scheme enables us to convert a ciphertext under a (delegator) public key of *any existing public-key encryption (PKE) scheme* into another ciphertext under a (delegatee) public key of *any existing PKE scheme (possibly different from the delegator one)*. Such a conversion is executed by a third party called proxy that has a re-encryption key generated from the delegator's secret key and the delegatee public key. Proxy re-encryption is a related notion, but it can neither convert ciphertexts into ones of *possibly different PKE schemes* nor treat general PKE schemes.

Our contributions consist of three parts. One is a definitional work. We define the syntax and security of UPRE. Another is showing the (im)possibility of UPRE. We prove that the existence of UPRE implies the existence of average-case virtual black-box obfuscation for all re-encryption circuits. The other is presenting general constructions of UPRE schemes. More precisely, we present three UPRE schemes. One is a UPRE based on probabilistic indistinguishability obfuscation (PIO). It can re-encrypt ciphertexts polynomially many times. To circumvent our impossibility result, we define a notion of relaxed UPRE and show that it can be constructed from garbled circuits (GCs). It can re-encrypt ciphertexts polynomially many times. The relaxed variant means that decryption algorithms for re-encrypted ciphertext are slightly modified though we use only original delegatee secret keys for decryption. Our second construction of relaxed UPRE based on GCs satisfies a stronger security requirement. It can re-encrypt ciphertexts a constant number of times.

Keywords: universal proxy re-encrytion, public-key encryption, secret sharing.

Contents

1 Introduction

1.1 Background

Constructing a cryptographic system from scratch is a challenging task. We want to recycle existing secure cryptographic systems and public-key infrastructure (PKI) as possible when we design a new cryptosystem. Moreover, deploying a new cryptographic system is much more challenging. Many users are reluctant to migrate from a currently used cryptographic system to a new one, even if the new one offers better security and functionality since the work and financial cost of migration can be extremely high. Thus, we would like to explore a *universal* methodology to construct a new and easily deployable cryptographic system from existing cryptographic systems and PKI. Such a methodology also saves generating and certifying new public-keys.

As a particular example of cryptographic systems, we consider proxy re-encryption (PRE) [BBS98] in this study. PRE enables us to convert a ciphertext under public key pk_f (we call delegator public key and f denotes "from") into another ciphertext under public key pk_t (we call delegatee public key and t denotes "to") by using a re-encryption key $\mathsf{rk}_{f \to t}$ without decrypting the original ciphertext by sk_f (we call delegator secret key). A third party, called proxy, owns the re-encryption key $\mathsf{rk}_{f \to t}$ and executes the re-encryption procedure. PRE is a very useful cryptographic primitive and has numerous applications since it enables delegation. It can be used to achieve encrypted email forwarding [BBS98, Jak99], key escrow [ID03], encrypted file storage [AFGH05], secure publish-subscribe operation [PRSV17], and secure payment systems for credit cards [GSL19].

However, all known PRE schemes only support conversions from ciphertexts under a public key generated by *their key generation algorithm* into other ones under another key generated by *the same key generation algorithm with the same parameter*. They *cannot* convert ciphertexts into ones under another key generated by *another key generation algorithm of another encryption scheme*. Moreover, almost all known PRE schemes were constructed from scratch by using specific cryptographic assumptions such as the decisional Diffie-Hellman (DDH) assumption and the learning with errors (LWE) assumption. The formats of their keys and ciphertexts are fixed in advance at the setup and can never be changed. Only a few PRE schemes use public-key encryption (PKE) schemes generically [HKK$^+$12]. However, in such schemes, we cannot use a PKE scheme as it is (we need some additional conversion). Moreover, only delegatees (receivers of converted ciphertexts) can select any PKE scheme and delegators (senders of original ciphertexts) cannot. This is unsatisfactory from a practical point of view because we need to build a new system using a PRE scheme from scratch if we would like to use applications of PRE described above. When we use a PRE scheme, we cannot use existing and widely used public-key cryptosystems to achieve applications of PRE. Ideally, we would like to achieve a re-encryption mechanism that recycles existing PKE schemes without any modification and setup and does not rely on any particular construction of PKE schemes.

Universal Proxy Re-Encryption. To resolve the problems above, we put forward the concept of *universal proxy re-encryption (UPRE)* in this study. UPRE enables us to convert ciphertexts under a public key of a scheme Σ_f (delegator scheme) into ciphertexts under another public key of *another scheme* Σ_t (delegatee scheme). *We can select arbitrary secure PKE schemes for* Σ_f, Σ_t. For example, we can use Goldwasser-Micali PKE [GM84] as Σ_f and ElGamal PKE [ElG85] as Σ_t. If a delegator and delegatee have key pairs $(\mathsf{pk}_f, \mathsf{sk}_f)$ and $(\mathsf{pk}_t, \mathsf{sk}_t)$ of schemes Σ_f and Σ_t, respectively, then a re-encryption key generation algorithm of UPRE can output a re-encryption key $\mathsf{rk}_{f \to t}$ from $(\Sigma_f, \Sigma_t, \mathsf{sk}_f, \mathsf{pk}_t)$. A proxy can generate a re-encrypted ciphertext rct from $\mathsf{rk}_{f \to t}$ and $\mathsf{Enc}_f(\mathsf{pk}_f, m)$ where Enc_f is the encryption algorithm of Σ_f. Of course, the re-encrypted ciphertext rct can be correctly decrypted to m by using sk_t.

Ideally, a re-encrypted ciphertext should be decrypted by the original decryption algorithm of the delegatee scheme (i.e., $\mathsf{Dec}_t(\mathsf{sk}_t, \cdot)$). However, we can also consider a relaxed variant where a re-encrypted ciphertext can be decrypted via a slightly modified decryption algorithm *with the original delegatee decryption key* sk_t. We call this variant *relaxed UPRE*. We define both types of UPRE in this study. Here, we emphasize that the delegator uses only pk_f and Enc_f to encrypt a message and the delegatee uses only sk_t to decrypt a re-encrypted ciphertext (they do not need any additional keys) even if its decryption procedure is slightly modified. Our work is the first to explore such a universal methodology for proxy re-encryption.

UPRE enables us to build a re-encryption mechanism dynamically by using currently deployed cryptosystems. Users who have already used PKE schemes can convert ciphertexts into other ones by using a UPRE scheme. They do not need to setup a proxy re-encryption system from scratch. Therefore, UPRE offers more flexibility than standard PRE. Moreover, UPRE has applications that PRE does not have, e.g. the following. UPRE enables us to delegate migration of encryption systems to a third party such as cloud-servers with many computational resources when an encryption scheme with some parameter settings becomes obsolete, or a vulnerability is found in an encryption system. That is, we can outsource renewing encrypted storages to a third party.

UPRE can be seen as a generalized notion of PRE. Therefore, we can consider several analogies of the notions used in PRE. They are the notions of "direction" and "the number of hops". For directions, there are unidirectional and bidirectional, which means that a re-encryption key between pk_f and pk_t can be used for only one-way from f to t and both ways, respectively. For the number of hops, there are single-hop and multi-hop, which mean a re-encrypted ciphertext cannot be converted anymore and can be converted polynomially-many times, respectively. In particular, when only a constant number of conversions is possible, we call it constant-hop. In this study, we consider unidirectional single/constant/multi-hop but do not focus on bidirectional since a bidirectional re-encryption key is simulated by two unidirectional re-encryption keys.

The main question addressed in this work is which assumptions are necessary and sufficient to achieve UPRE. One might think it is not difficult to achieve a UPRE scheme by using indistinguishability obfuscation (IO) [BGI+12, GGH+16] or multilinear maps [GGH13, CLT13, GGH15].[1] This might not be false and, in fact, we present a construction based on IO as an initial step (we emphasize that formally proving security is not an easy task even if we use IO). Yet, the following question is natural.

Is it possible to achieve a UPRE scheme without IO and multilinear maps?

The most challenging task is resolving this question. We give both positive and negative answers to this question in this study.

1.2 Our Contributions

The main contributions of this study are the following.

1. We introduce the notion of UPRE and formally define its security.
2. We prove that the existence of UPRE implies the existence of an average-case virtual black-box obfuscator [HRsV11, HMLS10] for all re-encryption circuits (i.e., some type of *obfuscation is inherent in UPRE*).
3. We present a general construction of multi-hop UPRE for some class of PKE by using probabilistic IO (PIO).
4. We present a general construction of multi-hop relaxed UPRE for any PKE by using only garbled circuits (GC) and therefore need no additional assumptions.
5. By using our general constructions and known instantiations of tools above, we can obtain multi-hop (relaxed) UPRE schemes from IO, or generic standard assumptions.

We explain more details (tools, security levels and so on) of these contributions below.

For UPRE, we can consider a natural analog of security against chosen plaintext attacks (CPA) for PRE (PRE-CPA), where adversaries execute CPA attacks with oracles that give re-encryption keys and re-encrypted ciphertexts. However, we do not focus on the definition of CPA-security for UPRE (UPRE-CPA) because Cohen introduced a better security notion called security against honest re-encryption attacks (HRA) for PRE [Coh17][2]. Thus, we define security against honest re-encryption attacks for UPRE (UPRE-HRA), which implies UPRE-CPA, instead of CPA-security. Roughly speaking, in the setting of HRA, adversaries are allowed to obtain an honestly encrypted ciphertext *via an honest encryption oracle* and can convert it into a re-encrypted ciphertext under a key of a *corrupted* user via a re-encryption oracle. In PRE-CPA security, adversaries cannot obtain such a re-encrypted ciphertext because it is *not allowed* to obtain a re-encryption key query *from an honest user to a corrupted user* via the re-encryption key oracle to prevent trivial attacks[3]. Cohen observes that PRE-CPA security is not sufficient for many applications of PRE. Therefore, we define HRA-security for UPRE (in fact, we also define a selective variant). We also define security against corrupted-delegator re-encryption attacks (CRA) to consider the setting of migration of encryption system explained in Section 1.1. That is, even if a delegator is corrupted, once a ciphertext is re-encrypted for an honest delegatee, then the delegator cannot obtain information about a plaintext from the re-encrypted ciphertext.[4] We note that HRA secure UPRE schemes are CRA secure since re-encrypted ciphertexts

[1] A.k.a. "heavy hammers".

[2] Derler, Krenn, Lorünser, Ramacher, Slamanig, and Striecks also proposed a similar security notion in the forward secret setting as (fs)-RIND-CPA [DKL+18].

[3] Of course, a re-encryption query from an honest user to a corrupted user is also prohibited in PRE-CPA security.

[4] Note that the corrupted delegator does not have a ciphertext to be re-encrypted here.

are ciphertexts of a delegatee's PKE scheme in UPRE. In relaxed UPRE, HRA security does not imply CRA security in general. See Section 3 for details.

We show that it is hard to avoid using obfuscation to achieve (full blown) UPRE. That is, we prove that if we have a UPRE scheme (not relaxed one), then we can construct an average-case virtual black-box (ACVBB) obfuscator for all re-encryption circuits. The ACVBB obfuscation is the VBB obfuscation for a *randomly chosen* circuit in a family of circuits, which is suitable for obfuscating cryptographic functionalities such as re-encryption. This negative result is a strong motivation to consider *relaxed* UPRE.

We present three general constructions of UPRE. One is UPRE for some class of PKE based on PIO. PIO was introduced by Canetti, Lin, Tessaro, and Vaikuntanathan [CLTV15]. Another is relaxed UPRE for any PKE based on GC. The other is constant-hop and CRA-secure relaxed UPRE for any PKE based on GC. We emphasize that our relaxed UPRE is based on *generic* standard assumptions without relying on heavy tools. We look closer at what kind of (relaxed) UPRE is achieved below.

Our UPRE scheme based on PIO is a unidirectional multi-hop UPRE scheme. PIO is IO for randomized circuits. There are several security levels for PIO. Our UPRE schemes based on PIO can support a limited class of PKE. The required properties for PKE schemes depend on the security level of PIO. If we use a mild security level of PIO, then we can achieve UPRE by using sub-exponentially secure IO (sub-exp IO) for deterministic circuits and sub-exponentially secure OWF (sub-exp OWF). However, supported PKE schemes are trapdoor encryption schemes that satisfy a particular security level (See Section 5.1 for details). Such trapdoor encryption is achieved by several well-known CPA-secure PKE schemes such as ElGamal, Goldwasser-Micali PKE schemes. If we assume that there exists PIO with the strongest security for specific circuits (refer to [CLTV15]), then we can use any CPA-secure PKE scheme. The advantage of the scheme based on PIO is that it is a multi-hop UPRE scheme and conceptually simple. See Section 5 for more details.

Our relaxed UPRE scheme based on garbled circuits (GC) is a unidirectional multi-hop *relaxed* UPRE scheme for any PKE scheme. This is a significant contribution since GC is a light cryptographic tool. This relaxed UPRE scheme satisfies HRA-security. However, some meta information (all garbled circuits from the first delegator to the last delegatee) is directly preserved in all re-encrypted ciphertexts. Therefore, the number of hops cannot be hidden in the scheme based on GC. In particular, when a delegator is corrupted, we cannot prove that a re-encrypted ciphertext does not reveal information about the plaintext.

Our last UPRE scheme is a unidirectional constant-hop relaxed UPRE scheme for any PKE scheme based on GC. This scheme satisfies CRA-security unlike the multi-hop scheme above, but it can re-encrypt only constant times since its re-encryption procedure incurs polynomial blow-up.

In the GC-based schemes, we must use a slightly modified decryption algorithm (i.e., we achieve relaxed UPRE) though we can use the original delegatee decryption key as it is. While this is a small disadvantage of the GC-based constructions, we would like to emphasize that these are the first constructions of relaxed UPRE, achieved by the standard assumptions, and, in the light of our impossibility result, the best we can hope for without obfuscation.

1.3 Technical Overview

In this section, we give a high-level overview of our UPRE schemes and techniques. To achieve the re-encryption mechanism, we use a circuit with a hard-wired secret key of a delegator PKE scheme to generate a re-encryption key. This is because UPRE supports *general PKE schemes* and we need to decrypt ciphertexts once to re-encrypt them. However, such a circuit should not be directly revealed to a proxy to guarantee security. Therefore, we must hide information about the secret-key in a re-encryption key. That is, to use CPA security of the delegator PKE scheme, we must erase information about the secret key embedded in a re-encryption key in security proofs. This is the most notable issue to prove the security of UPRE. When we succeed in erasing secret keys from re-encryption keys in our reductions, we can directly use the CPA-security of delegators to prove the security of a UPRE scheme.

UPRE implies obfuscation for re-encryption. As we explained above, we must erase information about the secret key of the target PKE scheme and then use the CPA-security to prove the security of UPRE. At the same time, we must simulate re-encryption keys from the target delegator to some uncorrupted delegatees for UPRE adversaries. That is, we must simulate re-encryption keys *without the secret key of the target delegator* when we reduce the security of UPRE to that of PKE. In other words, if we can make the reduction, there must exist a simulator that generates a simulated re-encryption key without a delegator's secret-key that is indistinguishable from the real re-encryption key generated from the secret-key. This simulated re-encryption key generation algorithm is an obfuscator for all re-encryption circuits since we do not need any secret information for the simulation and we consider UPRE. We stress that, in UPRE, the format of re-encrypted ciphertexts is the same as that of delegatee's ciphertexts. This is an intuition for the proof that UPRE implies ACVBB obfuscation for all re-encryption circuits.

Based on IO IO is a promising tool to hide information about delegator secret keys since IO is a kind of compiler that outputs a functionally equivalent program that does not reveal information about the original program. We define a re-encryption circuit C_{re}, in which a delegator secret key sk_f and a delegatee public key pk_t are hard-wired in and which takes a delegator ciphertext ct_f as an input. The re-encryption circuit decrypts ct_f by using sk_f, obtains a plaintext m, and generates a ciphertext of m under pk_t. We can hide information about sk_f by using PIO (note that C_{re} is a randomized circuit). That is, a re-encryption key from delegator f to delegatee t is $pi\mathcal{O}(\mathsf{C}_{\mathsf{re}})$ where $pi\mathcal{O}$ is a PIO algorithm. A re-encrypted ciphertext is a fresh ciphertext under pk_t. Thus, we can achieve multi-hop UPRE. This construction is similar to the FHE scheme based on PIO presented by Canetti et al. [CLTV15]. However, we cannot directly use the result by Canetti et al. since the setting of unidirectional multi-hop UPRE is different from that of FHE.

The security proof proceeds as follows. To erase sk_f, we use a dummy re-encryption circuit that does not run the decryption algorithm of Σ_f with sk_f and just outputs a dummy ciphertext under pk_t (does not need plaintext m). We expect that adversaries cannot distinguish this change. This intuition is not false. However, to formally prove it, we cannot directly use the standard CPA-security of PKE since an obfuscated circuit of the re-encryption circuit generates ciphertexts under *hard-wired* pk_t. It means that we cannot use a target ciphertext of the CPA-security game and the common "punctured programming" approach unless the scheme has a kind of "puncturable" property for its secret key [CHN+16]. Therefore, we use trapdoor encryption introduced by Canetti et al. [CLTV15]. In trapdoor encryption, there are two modes for key generation. One is the standard key generation, and the other one is the trapdoor key generation, which does not output a secret key for decryption. The two modes are computationally indistinguishable. Ciphertexts under a trapdoor key are computationally/statistically/perfectly indistinguishable (See Section 5.1 for more details). Thus, we proceed as follows. First, we change the hard-wired public key pk_t into a trapdoor key tk_t. Second, we use the security of PIO. The indistinguishability under tk_t is used to satisfy the condition of PIO.

In the multi-hop setting, we can consider the relationships among keys as a directed acyclic graph (DAG). Each vertex is a user who has a key pair, and each edge means that a re-encryption key was generated between two vertices. To prove ciphertext indistinguishability under a target public-key, we repeat the two processes above from the farthest vertex connected to the target vertex to the target vertex. We gradually erase information about secret keys of vertices connected to the target vertex. At the final step, information about the target secret key is also deleted, and we can use security under the target public-key of the target PKE scheme.

Types of indistinguishability under trapdoor keys affect what kind of PIO can be used. The weakest indistinguishability under a trapdoor key, which is equivalent to the standard IND-CPA security, requires stronger security of PIO. If we use perfect indistinguishability under a trapdoor key, which is achieved by re-randomizable PKE schemes such as ElGamal PKE scheme, then we can use weaker PIO for circuits that are implied by sub-exp IO for circuits and sub-exp OWF. Finally, we can use doubly-probabilistic IO introduced by Agrikola, Couteau, and Hofheinz [ACH18] instead of PIO to achieve UPRE for IND-CPA PKE. Agrikola et al. prove that we can achieve doubly-probabilistic IO by using polynomially secure IO and exponential DDH assumption.

Based on GC The most challenging task in this work is achieving a relaxed UPRE scheme without obfuscation. Surprisingly, we can achieve a relaxed UPRE scheme for any CPA-secure PKE scheme by using GC in combination with a secret sharing scheme. The idea is that a proxy and a delegatee are different entities and can separately use shares of a decryption key. We generate *shares of a decryption key*, and use a garbled circuit where one of the shares is hardwired to hide information about the decryption key. Our re-encryption mechanism proceeds int the following two steps. First, we generate shares (s_1, s_2) of a delegator secret key sk_f by a secret sharing scheme. We encrypt share s_1 by using pk_t and obtain $\widetilde{\mathsf{ct}}_t \leftarrow \mathsf{Enc}(\mathsf{pk}_t, s_1)$. A re-encryption key from f to t is $\mathsf{rk}_{f \to t} := (s_2, \widetilde{\mathsf{ct}}_t)$. Roughly speaking, s_1 is hidden by the CPA-security of PKE, and s_2 does not reveal information about sk_f by the privacy property of secret sharing. We define a circuit $\mathsf{C}^{\mathsf{re}}_{\mathsf{de}}$ where s_2 and the delegator ciphertext ct_f are hard-wired. The circuit $\mathsf{C}^{\mathsf{re}}_{\mathsf{de}}$ takes as input s_1, reconstructs sk_f from (s_1, s_2), and computes $\mathsf{Dec}_f(\mathsf{sk}_f, \mathsf{ct}_f)$. Now, we garble $\mathsf{C}^{\mathsf{re}}_{\mathsf{de}}[s_2, \mathsf{ct}_f]$ and obtain a garbled circuit $\widetilde{\mathsf{C}}^{\mathsf{re}}_{\mathsf{de}}$ and labels $\left\{\mathsf{labels}_{i,b}\right\}_{i \in [|s_1|] b \in \{0,1\}}$. We set a re-encrypted ciphertext to $\mathsf{rct} := (\widetilde{\mathsf{ct}}_t, \widetilde{\mathsf{C}}^{\mathsf{re}}_{\mathsf{de}}, \left\{\mathsf{labels}_{i,b}\right\})$ (we omit $\left\{i \in [|s_1|], b \in \{0,1\}\right\}$ if it is clear from the context). The delegatee t can evaluate the garbled circuit and obtain decrypted value since the delegatee can obtain s_1 from $\widetilde{\mathsf{ct}}_t$. However, this apparently does not work since sending $\left\{\mathsf{labels}_{i,b}\right\}$ breaks the security of GC and sk_f is revealed.

Before we move to the second step, we introduce the notion of weak batch encryption, which is a non-succinct variant of batch encryption [BLSV18] and easily constructed from standard CPA-secure PKE. A batch key pair $(\hat{\mathsf{pk}}, \hat{\mathsf{sk}})$ is generated from a choice string $s \in \{0,1\}^{\lambda}$. We can encrypt a pair of vector messages $(\{m_{i,0}\}_{i\in[\lambda]}, \{m_{i,1}\}_{i\in[\lambda]})$ by using $\hat{\mathsf{pk}}$. We can obtain $\{m_{i,s[i]}\}_{i\in[\lambda]}$ from a batch ciphertext and $\hat{\mathsf{sk}}$. A batch public-key $\hat{\mathsf{pk}}$ does not reveal any information about s. Adversaries cannot obtain any information about $\{m_{i,1-s[i]}\}_{i\in[\lambda]}$ from a batch ciphertext even if $\hat{\mathsf{sk}}$ is given. By using 2λ pairs of a public-key and secret-key of PKE, we can achieve weak batch encryption (we select a key pair based on each bit of s). Note that we can recycle $\hat{\mathsf{pk}}$ for many vectors of messages. See Section 6.1 for details.

Now, we move to the second step. To send only $\{\mathsf{labels}_{i,s_1[i]}\}_{i\in|s_1|}$ to the delegatee t, we use weak batch encryption. That is, we let s_1 be choice bits of a batch key pair and $\{\mathsf{labels}_{i,b}\}$ be messages of batch encryption. To achieve a re-encryption mechanism with this idea, at the re-encryption key generation phase, we generate a batch key pair $(\hat{\mathsf{pk}}, \hat{\mathsf{sk}}) \leftarrow \mathsf{BatchGen}(s_1)$. Moreover, we encrypt the batch secret-key $\hat{\mathsf{sk}}$ under pk_t. That is, we set $\mathsf{rk}_{f\rightarrow t} := (\hat{\mathsf{pk}}, s_2, \mathsf{Enc}(\mathsf{pk}_t, \hat{\mathsf{sk}}))$. At the re-encryption phase, we generate not only the garbled circuit $\widetilde{\mathsf{C}^{\mathsf{re}}_{\mathsf{de}}}$ of $\mathsf{C}^{\mathsf{re}}_{\mathsf{de}}[s_2, \mathsf{ct}_f]$ and $\{\mathsf{labels}_{i,b}\}_{i,b}$ but also the batch ciphertext $\hat{\mathsf{ct}} \leftarrow \mathsf{BatchEnc}(\hat{\mathsf{pk}}, (\{\mathsf{labels}_{i,0}\}_i, \{\mathsf{labels}_{i,1}\}_i))$. That is, a re-encrypted ciphertext is $\mathsf{rct} := (\widetilde{\mathsf{ct}}_t, \hat{\mathsf{ct}}, \widetilde{\mathsf{C}^{\mathsf{re}}_{\mathsf{de}}})$.

The delegatee t can obtain the plaintext m as follows. It obtains $\hat{\mathsf{sk}} \leftarrow \mathsf{Dec}_t(\mathsf{sk}_t, \widetilde{\mathsf{ct}}_t)$ by its secret key sk_t, recover selected messages $\{\mathsf{labels}_{i,s_1[i]}\}_i \leftarrow \mathsf{BatchDec}(\hat{\mathsf{sk}}, \hat{\mathsf{ct}})$, and $m' \leftarrow \mathsf{Eval}(\widetilde{\mathsf{C}^{\mathsf{re}}_{\mathsf{de}}}, \{\mathsf{labels}_{i,s_1[i]}\}_i)$. By the functionality of GC, it holds that $m' = \mathsf{C}^{\mathsf{re}}_{\mathsf{de}}[s_2, \mathsf{ct}_f](s_1) = m$. Thus, this construction works as relaxed UPRE for any PKE scheme if there exists GC.

Intuitively, the re-encryption key $\mathsf{rk}_{f\rightarrow t}$ does not reveal information about sk_f since the CPA-security of PKE and the receiver privacy of weak batch encryption hides information about s_1. Adversaries cannot obtain any information about sk_f from the other share s_2 by the privacy property of the secret sharing scheme. That is, we can erase information about sk_f and can use the CPA-security of pk_f. Here, the choice s_1 is fixed at the re-encryption key generation phase and recycled in many re-encryption phases. However, this is not an issue since the security of weak batch encryption holds for many batch ciphertexts under the same batch key pair.

We explain only the single-hop case. However, we can easily extend the idea above to a multi-hop construction. See Section 6 for the detail.

In the construction above, a delegator can decrypt a re-encrypted ciphertext under $\mathsf{rk}_{f\rightarrow t}$ by using sk_f. That is, the construction above does not satisfy CRA-security. This is a problem when we use a relaxed UPRE scheme for migration of encryption systems explained in Section 1.1. However, we can easily solve this problem by encrypting a garbled circuit since a delegator cannot decrypt a re-encrypted ciphertext under $\mathsf{rk}_{f\rightarrow t}$ by using sk_f. Yet, this extension incurs polynomial blow-up of ciphertext size. Thus, we can apply the re-encryption procedure only constant times.

Summary of Our Results. We give a summary of our concrete instantiations in Table 1.

Table 1: Summary of our UPRE schemes. In "Type" column, rUPRE means relaxed UPRE. In "#Hop" column, const/multi means constant/multi-hop, respectively. In "Security" column, HRA and CRA means security against honest-re-encryption/corrupted-delegator-re-encryption attacks, respectively. In "Supported PKE" column, 0-hiding trapdoor means trapdoor encryption that satisfies 0-hiding security (see Section 5.1).

Instantiation	Type	#Hop	Security	Supported PKE	Assumptions
Ours in Sec. 5 + [CLTV15]	UPRE	multi	HRA & CRA	0-hiding trapdoor	sub-exp IO and OWF
Ours in Sec. 5 + [CLTV15]	UPRE	multi	HRA & CRA	any IND-CPA	di-PIO and OWF
Ours in Sec. 5 + [ACH18]	UPRE	multi	HRA & CRA	any IND-CPA	IO and exponential DDH
Ours in Sec. 6	rUPRE	multi	HRA	any IND-CPA	PKE
Ours in Sec. 7	rUPRE	const	HRA & CRA	any IND-CPA	PKE

1.4 Related Work

Encryption switching protocol (ESP), which was introduced by Couteau, Peters, and Pointcheval [CPP16], is a related notion. It is an interactive two-party computation that enables us to transform a ciphertext of a PKE scheme into a ciphertext of another PKE scheme and vice versa. It has a similar functionality to that of UPRE. However, they are incomparable in the following sense. In an ESP, parties must interactively communicate each other though there does not exists a proxy (and no re-encryption key). UPRE does not need interactive communication. Moreover, the proposed ESPs are not universal, that is, the protocols work only for specific PKE schemes. Thus, the purpose of ESPs is different from that of UPRE and they are incomparable.

There is a universal methodology to construct a new cryptographic system from existing *signature* schemes. Hohenberger, Koppula, and Waters introduce the notion of universal signature aggregator (USA) [HKW15], which enables us to aggregate signatures under different secret keys of *different* signature schemes. Standard aggregate signatures enable us to compress multiple signatures under different secret keys of *the same* scheme into one compact signature that is verified by a set of multiple verification keys [BGLS03]. Thus, USA is a generalization of aggregate signatures. Koppula et al. [HKW15] constructed selectively (resp. adaptively) secure USA scheme from sub-exp IO, sub-exp OWF, and additive homomorphic encryption (resp. IO, OWF, homomorphic encryption, and universal samplers [HJK$^+$16]) in the standard (resp. random oracle) model.

Reconfigurable cryptography was introduced by Hesse, Hofheinz, and Rupp [HHR16]. It makes updating PKI easier by using long-term keys, short-term keys, and common reference strings. Reconfigurable encryption can update keys, but cannot update ciphertexts.

There is a long series of works on proxy re-encryption. After the introduction of proxy cryptography by Blaze, Bleumer, and Strauss [BBS98], improved constructions [ID03, AFGH05], CCA-secure constructions [CH07, LV08, HKK$^+$12], key-private constructions [ABH09, ABPW13, NX15], obfuscation-based definition and constructions [HRsV11, CCV12, CCL$^+$14] have been proposed. Note that this is not an exhaustive list.

Organization. The main body of this paper consists of the following parts. In Section 2, we provide preliminaries and basic definitions. In Section 3, we introduce the syntax and security definitions of UPRE. In Section 4, we prove that UPRE implies ACVBB obfuscation for all re-encryption circuits. In Section 5, we present our UPRE scheme based on PIO and prove its security. In Section 6, we present our relaxed UPRE scheme based on GC, and prove its security. In Section 7, we present our CRA-secure relaxed UPRE scheme based on GC, and prove its security.

2 Preliminaries

We define some notations and introduce cryptographic primitives in this section.

2.1 Notations and Basic Concepts

In this paper, $x \leftarrow X$ denotes selecting an element from a finite set X uniformly at random, and $y \leftarrow \mathsf{A}(x)$ denotes assigning to y the output of a probabilistic or deterministic algorithm A on an input x. When we explicitly show that A uses randomness r, we write $y \leftarrow \mathsf{A}(x; r)$. For strings x and y, $x \| y$ denotes the concatenation of x and y. Let $[\ell]$ denote the set of integers $\{1, \cdots, \ell\}$, λ denote a security parameter, and $y := z$ denote that y is set, defined, or substituted by z. PPT stands for probabilistic polynomial time.

- A function $f : \mathbb{N} \rightarrow \mathbb{R}$ is a negligible function if for any constant c, there exists $\lambda_0 \in \mathbb{N}$ such that for any $\lambda > \lambda_0$, $f(\lambda) < \lambda^{-c}$. We write $f(\lambda) \leq \mathsf{negl}(\lambda)$ to denote $f(\lambda)$ being a negligible function.
- If $\mathcal{X}^{(b)} = \{X_\lambda^{(b)}\}_{\lambda \in \mathbb{N}}$ for $b \in \{0, 1\}$ are two ensembles of random variables indexed by $\lambda \in \mathbb{N}$, we say that $\mathcal{X}^{(0)}$ and $\mathcal{X}^{(1)}$ are computationally indistinguishable if for any PPT distinguisher $\mathcal{D}$, there exists a negligible function $\mathsf{negl}(\lambda)$, such that
$$\Delta := |\Pr[\mathcal{D}(X_\lambda^{(0)}) = 1] - \Pr[\mathcal{D}(X_\lambda^{(1)}) = 1]| \leq \mathsf{negl}(\lambda).$$
We write $\mathcal{X}^{(0)} \overset{\mathsf{c}}{\approx}_\delta \mathcal{X}^{(1)}$ and $\mathcal{X}^{(0)} \overset{\mathsf{c}}{\approx} \mathcal{X}^{(1)}$ to denote that the advantage Δ is bounded by δ and δ is negligible, respectively and call the former δ-indistinguishability.
- The statistical distance between $\mathcal{X}^{(0)}$ and $\mathcal{X}^{(1)}$ over a countable set S is defined as $\Delta_{\mathsf{s}}(\mathcal{X}^{(0)}, \mathcal{X}^{(1)}) := \frac{1}{2} \sum_{\alpha \in S} |\Pr[X_\lambda^{(0)} = \alpha] - \Pr[X_\lambda^{(1)} = \alpha]|$. We say that $\mathcal{X}^{(0)}$ and $\mathcal{X}^{(1)}$ are statistically indistinguishable (denoted by $\mathcal{X}^{(0)} \overset{\mathsf{s}}{\approx} \mathcal{X}^{(1)}$) if $\Delta_{\mathsf{s}}(\mathcal{X}^{(0)}, \mathcal{X}^{(1)}) \leq \mathsf{negl}(\lambda)$. We also say that $\mathcal{X}^{(0)}$ is ϵ-close to $\mathcal{X}^{(1)}$ if $\Delta_{\mathsf{s}}(\mathcal{X}^{(0)}, \mathcal{X}^{(1)}) = \epsilon$.

2.2 Basic Cryptographic Tools

Definition 2.1 (Public-key Encryption). *Let $\mathcal{M}$ be a message space. A PKE scheme for $\mathcal{M}$ is a tuple of algorithms* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *where:*

- $\mathsf{KeyGen}(1^\lambda)$ *takes as input the security parameter and outputs a public key* pk *and secret key* sk.
- $\mathsf{Enc}(\mathsf{pk}, m)$ *takes as input* pk *and a message* $m \in \mathcal{M}$ *and outputs a ciphertext* ct.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$ *takes as input* sk *and* ct, *and outputs some* $m' \in \mathcal{M}$, *or* $\perp$.

Correctness: *For any* $m \in \mathcal{M}$ *and* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, *we have that* $\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m$.

CPA-security: *We define the experiment* $\mathsf{Expt}^{\mathsf{pke}}_{\mathcal{A}}(1^\lambda, b)$ *between an adversary* $\mathcal{A}$ *and challenger as follows.*

1. *The challenger runs* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, *and gives* pk *to* $\mathcal{A}$.
2. *The following process can be repeated polynomially many times.*
   - $\mathcal{A}$ *sends two messages* m_0^*, m_1^* *as the challenge messages to the challenger.*
   - *The challenger generates ciphertext* $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b^*)$ *and sends* ct^* *to* $\mathcal{A}$.
3. *At some point,* $\mathcal{A}$ *outputs a guess* b' *for* b. *The experiment outputs* b'.

We say PKE *is CPA-secure if, for any PPT adversary* $\mathcal{A}$, *it holds that*

$$\mathsf{Adv}^{\mathsf{pke}}_{\mathcal{A}}(\lambda) := |\Pr[\mathsf{Expt}^{\mathsf{pke}}_{\mathcal{A}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Expt}^{\mathsf{pke}}_{\mathcal{A}}(1^\lambda, 1) = 1]| \le \mathsf{negl}(\lambda).$$

Note that we can achieve the CPA-security above by using the standard CPA-security of PKE, where $\mathcal{A}$ sends the challenge messages only once (with polynomial security loss) by using the standard hybrid argument.

Definition 2.2 (Pseudorandom functions). *For sets* $\mathcal{D}$ *and* $\mathcal{R}$, *let* $\{\mathsf{F}_{\mathsf{K}}(\cdot) : \mathcal{D} \to \mathcal{R} \mid \mathsf{K} \in \{0,1\}^\lambda\}$ *be a family of polynomially computable functions. We say that* F *is pseudorandom if for any PPT adversary* $\mathcal{A}$, *it holds that*

$$\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{F},\mathcal{A}}(\lambda) := |\Pr[\mathcal{A}^{\mathsf{F}_{\mathsf{K}}(\cdot)}(1^\lambda) = 1 \mid \mathsf{K} \leftarrow \{0,1\}^\lambda] - \Pr[\mathcal{A}^{\mathsf{R}(\cdot)}(1^\lambda) = 1 \mid \mathsf{R} \leftarrow \mathcal{F}_U]| \le \mathsf{negl}(\lambda) \enspace ,$$

where $\mathcal{F}_U$ *is the set of all functions from* $\mathcal{D}$ *to* $\mathcal{R}$.

Theorem 2.3 ([GGM86]). *If one-way functions exist, then for all efficiently computable functions* $n(\lambda)$ *and* $m(\lambda)$, *there exists a pseudorandom function that maps* $n(\lambda)$ *bits to* $m(\lambda)$ *bits (i.e.,* $\mathcal{D} := \{0,1\}^{n(\lambda)}$ *and* $\mathcal{R} := \{0,1\}^{m(\lambda)}$*).*

Definition 2.4 (Puncturable pseudorandom function). *For sets* $\mathcal{D}$ *and* $\mathcal{R}$, *a puncturable pseudorandom function* PPRF *consists of a tuple of algorithms* $(\mathsf{F}, \mathsf{Punc})$ *that satisfies the following two conditions.*

Functionality preserving under puncturing: *For all polynomial size subset* $\{x_i\}_{i\in[k]}$ *of* $\mathcal{D}$, *and for all* $x \in \mathcal{D} \setminus \{x_i\}_{i\in[k]}$, *we have* $\Pr[\mathsf{F}_{\mathsf{K}}(x) = \mathsf{F}_{\mathsf{K}^*}(x) : \mathsf{K} \leftarrow \{0,1\}^\lambda, \mathsf{K}^* \leftarrow \mathsf{Punc}(\mathsf{K}, \{x_i\}_{i\in[k]})] = 1$.

Pseudorandomness at punctured points: *For all polynomial size subset* $\{x_i\}_{i\in[k]}$ *of* $\mathcal{D}$, *and any PPT adversary* $\mathcal{A}$, *it holds that*

$$\Pr[\mathcal{A}(\mathsf{K}^*, \{\mathsf{F}_{\mathsf{K}}(x_i)\}_{i\in[k]}) = 1] - \Pr[\mathcal{A}(\mathsf{K}^*, \mathcal{U}^k) = 1] \le \mathsf{negl}(\lambda) \enspace ,$$

where $\mathsf{K} \leftarrow \{0,1\}^\lambda$, $\mathsf{K}^* \leftarrow \mathsf{Punc}(\mathsf{K}, \{x_i\}_{i\in[k]})$, *and* $\mathcal{U}$ *denotes the uniform distribution over* $\mathcal{R}$.

Theorem 2.5 ([GGM86, BW13, BGI14, KPTZ13]). *If one-way functions exist, then for all efficiently computable functions* $n(\lambda)$ *and* $m(\lambda)$, *there exists a puncturable pseudorandom function that maps* $n(\lambda)$ *bits to* $m(\lambda)$ *bits (i.e.,* $\mathcal{D} := \{0,1\}^{n(\lambda)}$ *and* $\mathcal{R} := \{0,1\}^{m(\lambda)}$*).*

Definition 2.6 (Garbling Scheme (Garbled Circuit)). *A grabling scheme* GC *is a two tuple* $(\mathsf{Grbl}, \mathsf{Eval})$ *of PPT algorithms.*

- *The garbling algorithm* Grbl, *given a security parameter* 1^λ *and a circuit* C *with* n*-bit input, outputs a garbled circuit* $\widetilde{\mathsf{C}}$, *together with* $2n$ *labels* $\{\mathsf{labels}_{k,b}\}_{k\in[n],b\in\{0,1\}}$.

- *The evaluation algorithm* Eval, *given a garbled circuit* $\widetilde{C}$ *and* n *labels* $\{\mathsf{labels}_k\}_{k\in[n]}$, *outputs* y.

Correctness: *We require* $\mathsf{Eval}(\widetilde{C}, \{\mathsf{labels}_{k,x_k}\}_{k\in[n]}) = C(x)$ *for every* $\lambda \in \mathbb{N}$, *a circuit* C *with* n*-bit input, and* $x \in \{0,1\}^n$, *where* $(\widetilde{C}, \{\mathsf{labels}_{k,b}\}_{k\in[n],b\in\{0,1\}}) \leftarrow \mathsf{Grbl}(1^\lambda, C)$ *and* x_k *is the* k*-th bit of* x *for every* $k \in [n]$.

Security: *Let* Sim *be a PPT algorithm. We define the following game* $\mathsf{Expt}^{\mathsf{gc}}_{\mathcal{A}}(1^\lambda, \beta)$ *between a challenger and an adversary* $\mathcal{A}$ *as follows.*

1. *The challenger sends the security parameter* 1^λ *to* $\mathcal{A}$.
2. $\mathcal{A}$ *sends a circuit* C *with* n*-bit input and an input* $x \in \{0,1\}^n$ *to the challenger.*
   - *If* $\beta = 0$, *then the challenger computes* $(\widetilde{C}, \{\mathsf{labels}_{k,b}\}_{k\in[n],b\in\{0,1\}}) \leftarrow \mathsf{Grbl}(1^\lambda, C)$ *and returns* $(\widetilde{C}, \{\mathsf{labels}_{k,x_k}\}_{k\in[n]})$ *to* $\mathcal{A}$.
   - *If* $\beta = 1$, *then it computes* $(\widetilde{C}, \{\mathsf{labels}_k\}_{k\in[n]}) \leftarrow \mathsf{Sim}(1^\lambda, 1^{|C|}, C(x))$, *and returns* $(\widetilde{C}, \{\mathsf{labels}_k\}_{k\in[n]})$ *to* $\mathcal{A}$.
3. $\mathcal{A}$ *outputs* $\beta' \in \{0,1\}$.

We say that a garbling scheme is selectively secure if there exists PPT Sim *such that for any PPT* $\mathcal{A}$, *we have*

$$|\Pr[\mathsf{Expt}^{\mathsf{gc}}_{\mathcal{A}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Expt}^{\mathsf{gc}}_{\mathcal{A}}(1^\lambda, 1) = 1]| \leq \mathsf{negl}(\lambda).$$

Definition 2.7 (Secret Sharing). *A* t*-out-of-*n *secret sharing scheme over message space* $\mathcal{M}$ *is a pair of algorithms* $(\mathsf{Share}, \mathsf{Reconstruct})$ *where:*

- $\mathsf{Share}(1^\lambda, m)$ *takes as input the security parameter and a message* $m \in \mathcal{M}$, *and outputs an* n*-tuple of shares* $(s_1, \ldots, s_n)$.
- $\mathsf{Reconstruct}(s_{i_1}, \ldots, s_{i_t})$ *takes as input* t *shares* $(s_{i_1}, \ldots, s_{i_t})$ *where* $i_k \in [n]$ *and* $k \in [t]$ *outputs a meesage* $m' \in \mathcal{M}$ *or* $\perp$.

Correctness: *For any* $m \in \mathcal{M}$ *and* $(i_1, \ldots, i_t) \subseteq [n]$ *of size* t, *we have that*

$$\Pr[\mathsf{Reconstruct}(s_{i_1}, \ldots, s_{i_t}) = m \mid (s_1, \ldots, s_n) \leftarrow \mathsf{Share}(m)] = 1.$$

Security: *For any* $m, m' \in \mathcal{M}$, $S \subseteq [n]$ *such that* $|S| < t$, *we have that*

$$\left\{\{s_i\}_{i\in S} \mid (s_1, \ldots, s_n) \leftarrow \mathsf{Share}(m)\right\} \overset{\mathsf{s}}{\approx} \left\{\{s'_i\}_{i\in S} \mid (s'_1, \ldots, s'_n) \leftarrow \mathsf{Share}(m')\right\}.$$

2.3 (Probabilistic) Indistinguishability Obfuscation

Definition 2.8 (Indistinguishability Obfuscator). *A PPT algorithm* $i\mathcal{O}$ *is an IO for a circuit class* $\{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ *if it satisfies the following two conditions.*

Functionality: *For any security parameter* $\lambda \in \mathbb{N}$, *circuit* $C \in \mathcal{C}_\lambda$, *and input* x, *we have that*

$$\Pr[C'(x) = C(x) \mid C' \leftarrow i\mathcal{O}(C)] = 1 \ .$$

Indistinguishability: *For any PPT distinguisher* $\mathcal{D}$ *and for any pair of circuits* $C_0, C_1 \in \mathcal{C}_\lambda$ *such that for any input* x, $C_0(x) = C_1(x)$ *and* $|C_0| = |C_1|$, *it holds that*

$$|\Pr[\mathcal{D}(i\mathcal{O}(C_0)) = 1] - \Pr[\mathcal{D}(i\mathcal{O}(C_1)) = 1]| \leq \mathsf{negl}(\lambda) \ .$$

We further say that $i\mathcal{O}$ *is sub-exponentially secure if for any PPT* $\mathcal{D}$ *the above advantage is smaller than* $2^{-\lambda^\epsilon}$ *for some* $0 < \epsilon < 1$.

Next, we consider a family of sets of randomized polynomial-size circuits, $\mathcal{C} := \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$. A circuit sampler for $\mathcal{C}$ is a distribution ensemble $\mathsf{Samp} := \{\mathsf{Samp}_\lambda\}_{\lambda\in\mathbb{N}}$, where the distribution of Samp_λ is (C_0, C_1, z) with $C_0, C_1 \in \mathcal{C}_\lambda$ such that C_0 and C_1 take inputs of the same length, and $\mathsf{z} \in \{0,1\}^{\mathrm{poly}(\lambda)}$. A class $\mathcal{S}$ of samplers for $\mathcal{C}$ is a set of circuit samplers for $\mathcal{C}$.

Definition 2.9 (PIO for a Class of Samplers [CLTV15]). *A PPT algorithm $\mathit{pi}\mathcal{O}$ is a probabilistic indistinguishability obfuscator for a class of samplers $\mathcal{S}$ over the randomized circuit family $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ if it satisfies the following.*

Alternative Correctness [DHRW16]: *For any $\lambda \in \mathbb{N}$, any $\mathsf{C} \in \mathcal{C}_\lambda$, any $\widehat{\mathsf{C}} \leftarrow \mathit{pi}\mathcal{O}(\mathsf{C})$ and any individual input x, the distribution of $\widehat{\mathsf{C}}(x)$ and $\mathsf{C}(x)$ are identical.*

Security with respect to $\mathcal{S}$: *We define the following experiments $\mathsf{Expt}^{\mathsf{pio}}_{\mathcal{D}}(1^\lambda, b)$ between a challenger and a distinguisher $\mathcal{D}$ as follows.*

1. *The challenger samples $(\mathsf{C}_0, \mathsf{C}_1, \mathsf{z}) \leftarrow \mathsf{Samp}_\lambda$.*
2. *The challenger computes $\widehat{\mathsf{C}}_b \leftarrow \mathit{pi}\mathcal{O}(\mathsf{C}_b)$ and sends $(1^\lambda, \mathsf{C}_0, \mathsf{C}_1, \widehat{\mathsf{C}}_b, \mathsf{z})$ to $\mathcal{D}$.*
3. *$\mathcal{D}$ outputs a guess $b' \in \{0,1\}$. The experiment outputs b'.*

We say that $\mathit{pi}\mathcal{O}$ is secure PIO for $\mathcal{S}$ if for any sampler $\mathsf{Samp} = \{\mathsf{Samp}_\lambda\}_{\lambda \in \mathbb{N}} \in \mathcal{S}$, and for any PPT $\mathcal{D}$, it holds that

$$|\Pr[\mathsf{Expt}^{\mathsf{pio}}_{\mathcal{D}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Expt}^{\mathsf{pio}}_{\mathcal{D}}(1^\lambda, 1) = 1]| \leq \mathsf{negl}(\lambda).$$

As noted by Dodis et al. [DHRW16], the PIO construction by Canetti et al. [CLTV15] can be easily modified to satisfy the alternative correctness above, so we use it. Canetti et al. [CLTV15] introduced a few types of samplers. We review static-input X-indistinguishable and dynamic-input indistinguishable samplers.

Definition 2.10 (Static-input X-Indistinguishable-Samplers). *Let $X(\lambda)$ be a function bounded by 2^λ. The class $\mathcal{S}^{\mathsf{X\text{-}ind}}$ of static-input X-IND-samplers for a circuit family $\mathcal{C}$ contains all circuit samplers $\mathsf{Samp} = \{\mathsf{Samp}_\lambda\}_{\lambda \in \mathbb{N}}$ for $\mathcal{C}$ satisfying the following. For any $\lambda \in \mathbb{N}$, there exists a set $\mathcal{X} = \mathcal{X}_\lambda \subseteq \{0,1\}^*$ of size at most $X(\lambda)$ such that the following two conditions hold.*

X differing inputs: *For any input $x' \notin \mathcal{X}$, for any random coin r, it holds that*

$$\Pr[\mathsf{C}_0(x'; r) = \mathsf{C}_1(x'; r) \mid (\mathsf{C}_0, \mathsf{C}_1, \mathsf{z}) \leftarrow \mathsf{Samp}_\lambda] > 1 - \mathsf{negl}(\lambda).$$

X-indistinguishability: *For any PPT $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{si\text{-}ind}}_{\mathcal{A},\mathsf{Samp}}(\lambda) \leq \mathsf{negl}(\lambda) \cdot X^{-1}$ holds, where $\mathsf{Adv}^{\mathsf{si\text{-}ind}}_{\mathcal{A},\mathsf{Samp}}(\lambda)$ is defined as below.*

$$\mathsf{Adv}^{\mathsf{si\text{-}ind}}_{\mathcal{A},\mathsf{Samp}}(\lambda) := |\Pr[\mathsf{Exp}^{\mathsf{si\text{-}ind}}_{\mathcal{A},\mathsf{Samp}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Exp}^{\mathsf{si\text{-}ind}}_{\mathcal{A},\mathsf{Samp}}(1^\lambda, 1) = 1]|,$$

where experiments $\mathsf{Exp}^{\mathsf{si\text{-}ind}}_{\mathcal{A},\mathsf{Samp}}(1^\lambda, b)$ between a challenger and an adversary $\mathcal{A}$ are as follows.

1. *The adversary $\mathcal{A}$ sends x to the challenger.*
2. *The challenger samples $(\mathsf{C}_0, \mathsf{C}_1, \mathsf{z}) \leftarrow \mathsf{Samp}_\lambda$.*
3. *The challenger computes $y \leftarrow \mathsf{C}_b(x)$ and sends $(\mathsf{C}_0, \mathsf{C}_1, \mathsf{z}, y)$ to $\mathcal{A}$.*
4. *$\mathcal{A}$ outputs a guess $b' \in \{0,1\}$. The experiment outputs b'.*

Definition 2.11 (X-IND PIO for Randomized Circuits). *Let $X(\lambda)$ be any function bounded by 2^λ. A PPT algorithm $\mathit{pi}\mathcal{O}$ (X-$\mathit{pi}\mathcal{O}$) is an X-PIO for randomized circuits if it is a PIO for the class of X-IND samplers $\mathcal{S}^{\mathsf{X\text{-}ind}}$ over $\mathcal{C}$ that includes all randomized circuits of size at most λ.*

Theorem 2.12 ([CLTV15, DHRW16]). *If there exists sub-exponentially secure IO for circuits and sub-exponentially secure puncturable PRF, then there exists an X-IND PIO with alternative correctness for randomized circuits.*

Definition 2.13 (Dynamic-input Indistinguishable Sampler). *We define the experiments $\mathsf{Exp}^{\mathsf{di\text{-}ind}}_{\mathcal{A},\mathsf{Samp}}(1^\lambda, b)$ between a challenger and an adversary $\mathcal{A}$ as follows.*

1. *The challenger samples $(\mathsf{C}_0, \mathsf{C}_1, \mathsf{z}) \leftarrow \mathsf{Samp}_\lambda$ and sends it to $\mathcal{A}$.*

2. *The adversary $\mathcal{A}$ outputs x and sends it to the challenger.*

3. *The challenger computes $y \leftarrow \mathsf{C}_b(x)$ and sends $(\mathsf{C}_0, \mathsf{C}_1, \mathsf{z}, y)$ to $\mathcal{A}$.*

4. *$\mathcal{A}$ outputs a guess $b' \in \{0,1\}$. The experiment outputs b'.*

The class $\mathcal{S}^{\mathsf{di\text{-}ind}}$ of dynamic-input indistinguishable sampler for a circuit family $\mathcal{C}$ contains all circuit samplers $\mathsf{Samp} = \{\mathsf{Samp}_\lambda\}_{\lambda \in \mathbb{N}}$ *for $\mathcal{C}$ satisfies the following. If for any PPT $\mathcal{A}$, it holds that*

$$\mathsf{Adv}^{\mathsf{di\text{-}ind}}_{\mathcal{A},\mathsf{Samp}}(\lambda) := |\Pr[\mathsf{Exp}^{\mathsf{di\text{-}ind}}_{\mathcal{A},\mathsf{Samp}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Exp}^{\mathsf{di\text{-}ind}}_{\mathcal{A},\mathsf{Samp}}(1^\lambda, 1) = 1]| \leq \mathsf{negl}(\lambda).$$

Definition 2.14 (Dynamic-input PIO for Randomized Circuits). *A PPT algorithm $pi\mathcal{O}$ (di-$pi\mathcal{O}$) is a dynamic-input PIO for randomized circuits if it is a PIO for the class of dynamic-input indistinguishable samplers $\mathcal{S}^{\mathsf{di\text{-}ind}}$ over $\mathcal{C}$ that includes all randomized circuits of size at most λ.*

Canetti et al. [CLTV15] wrote that a construction of dynamic-input PIO for *specific* classes of samplers is possible as in the case of differing-input obfuscation [ABG$^+$13, BCP14] for specific circuits.

3 Definition of Universal Proxy Re-Encryption

In this section, we present the definitions of universal proxy re-encryption (UPRE). In particular, we present the definition of UPRE for PKE and its security notions. A UPRE scheme enables us to convert ciphertexts of a PKE scheme Σ_f into ciphertexts of a (possibly) different PKE scheme Σ_t. A UPRE scheme does not need a setup for a system. That is, it can use existing PKE schemes with different parameters. UPRE can be seen as a generalization proxy re-encryption [BBS98]. Therefore, we borrow many terms of proxy re-encryption [AFGH05, CH07].

Notations. We consider multiple PKE schemes and key pairs, so we assume that every known PKE scheme is named by a number in $[N]$ (say, 1 is for Goldwasser-Micali PKE, 2 is for ElGamal PKE etc). We also put a number in $[U]$ for a generated key pair. When we write $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}_{\sigma_i}(1^\lambda)$, we mean that i-th key pair is generated by PKE scheme $\Sigma_{\sigma_i} = (\mathsf{Gen}_{\sigma_i}, \mathsf{Enc}_{\sigma_i}, \mathsf{Dec}_{\sigma_i})$ where $\sigma_i \in [N]$. In this paper, when we emphasize which user is a delegator or delegatee, we denote delegator and delegatee key pairs by $(\mathsf{pk}_f, \mathsf{sk}_f)$ and $(\mathsf{pk}_t, \mathsf{sk}_t)$, respectively ($f$ and t mean "from" and "to", respectively). That is, a ciphertext under pk_f will be converted into a ciphertext pk_t. We assume that in the description of Σ_{σ_i}, ciphertext space $\mathcal{C}_{\sigma_i}$ and message space $\mathcal{M}_{\sigma_i}$ are also included.

3.1 Unidirectional UPRE

Definition 3.1 (Universal Proxy Re-Encryption for PKE: Syntax). *A universal re-encryption scheme* UPRE *consists of two algorithms* $(\mathsf{ReKeyGen}, \mathsf{ReEnc})$.

- $\mathsf{ReKeyGen}(1^\lambda, \Sigma_{\sigma_f}, \Sigma_{\sigma_t}, \mathsf{sk}_f, \mathsf{pk}_t)$ *takes the security parameter, a pair of PKE scheme $(\Sigma_{\sigma_f}, \Sigma_{\sigma_t})$, a secret-key sk_f of Σ_{σ_f}, and a public-key pk_t of Σ_{σ_t} and outputs a re-encryption key $\mathsf{rk}_{f \to t}$ for ciphertexts under pk_f. The security parameter is often omitted.*
- $\mathsf{ReEnc}(\Sigma_{\sigma_f}, \Sigma_{\sigma_t}, \mathsf{rk}_{f \to t}, \mathsf{ct}_f)$ *takes a pair of PKE schemes $(\Sigma_{\sigma_f}, \Sigma_{\sigma_t})$, a re-encryption key $\mathsf{rk}_{f \to t}$, and a ciphertext ct_f under pk_f of Σ_{σ_f}, and outputs a re-encrypted ciphertext ct_t under pk_t.*

Definition 3.2 (Relaxed Universal Proxy Re-Encryption for PKE: Syntax). *A relaxed universal re-encryption scheme* UPRE *consists of three algorithms* $(\mathsf{ReKeyGen}, \mathsf{ReEnc}, \mathsf{mDec})$.

- $\mathsf{ReKeyGen}(1^\lambda, \Sigma_{\sigma_f}, \Sigma_{\sigma_t}, \mathsf{sk}_f, \mathsf{pk}_t)$ *is the same as in Definition 3.1.*
- $\mathsf{ReEnc}(\Sigma_{\sigma_f}, \Sigma_{\sigma_t}, \mathsf{rk}_{f \to t}, \mathsf{ct}_f)$ *takes a pair of PKE schemes $(\Sigma_{\sigma_f}, \Sigma_{\sigma_t})$, a re-encryption key $\mathsf{rk}_{f \to t}$, and a ciphertext ct_f under pk_f of Σ_{σ_f}, and outputs a re-encrypted ciphertext rct. We implicitly assume that rct includes index ℓ which indicates how many times ReEnc was applied so far. When we write $\mathsf{rct}^{(\ell)}$, it means that $\mathsf{rct}^{(\ell)}$ was obtained by applying ReEnc ℓ times.*
- $\mathsf{mDec}(\Sigma_{\sigma_t}, \mathsf{sk}_t, \mathsf{rct}^{(\ell)}, \ell)$ *takes a PKE scheme Σ_{σ_t}, a secret key sk_t, a re-encrypted ciphertext $\mathsf{rct}^{(\ell)}$ under $\mathsf{rk}_{f \to t}$, and index ℓ and outputs a message m. When $\ell = 1$, we omit the index.*

The difference between UPRE and relaxed UPRE is that we can use the decryption algorithm of Σ_{σ_t} as it is in UPRE. In relaxed UPRE, we need use a modified decryption algorithm though what we need for decryption is the original secret key sk_t. Note that re-encrypted ciphertext space $\mathcal{C}_{\sigma_f \to \sigma_t}$ potentially depends on $\mathcal{C}_{\sigma_f}$ and $\mathcal{C}_{\sigma_t}$ and possibly $\mathsf{rct} \notin \mathcal{C}_{\sigma_t}$ happens.

Hereafter, we focus only on the relaxed notion since we can easily replace $\mathsf{mDec}(\Sigma_{\sigma_t}, \mathsf{sk}_t, \mathsf{rct}^{(\ell)}, \ell)$ with $\mathsf{Dec}(\mathsf{sk}_t, \mathsf{ct}_t)$.

On Message Space. For simplicity, we consider messages in $\mathcal{M}_{\sigma_1} \cap \cdots \cap \mathcal{M}_{\sigma_N}$ where N is the number of considered PKE scheme in security games (described later). We can consider $\{0,1\}^\ell$ as a message space where ℓ is a polynomial of a security parameter and UPRE for such a message space by considering bit-by-bit encryption for all PKE scheme. However, this is cumbersome. Thus, hereafter, we consider messages in the intersection of all message spaces though we do not explicitly mention.

Bidirectional UPRE. We can consider bidirectional UPRE, where a re-encryption key generated from key pairs $(\mathsf{pk}_f, \mathsf{sk}_f)$ and $(\mathsf{pk}_t, \mathsf{sk}_t)$ can convert ciphertexts under pk_f (resp. pk_t) into ciphertexts that can be decrypted by sk_t (resp. sk_f). Unidirectional UPRE is stronger than bidirectional UPRE since unidirectional one can support bidirectional one by generating two re-encryption keys $\mathsf{rk}_{f \to t}$ and $\mathsf{rk}_{t \to f}$. Thus, we focus on unidirectional UPRE in this study.

Functionality and Security. We introduce the correctness and a security notion of UPRE that we call security against *honest re-encryption attacks (HRA)* for UPRE. This notion is based on security against HRA of PRE introduced by Cohen [Coh17]. Correctness is easy to understand. First, we consider *single-hop* UPRE, where if a ciphertext is converted into another ciphertext, then we cannot convert the re-encrypted one anymore.

Definition 3.3 (UPRE for PKE: Single-Hop Correctness). *A relaxed UPRE scheme* UPRE *for PKE is correct if for all pairs of PKE schemes* $(\Sigma_{\sigma_f}, \Sigma_{\sigma_t})$, $(\mathsf{pk}_f, \mathsf{sk}_f) \leftarrow \mathsf{Gen}_{\sigma_f}(1^{\lambda_f})$, $(\mathsf{pk}_t, \mathsf{sk}_t) \leftarrow \mathsf{Gen}_{\sigma_t}(1^{\lambda_t})$, $m \in \mathcal{M}_{\sigma_f} \cap \mathcal{M}_{\sigma_t}$, $\mathsf{ct}_f \leftarrow \mathsf{Enc}_{\sigma_f}(\mathsf{pk}_f, m)$, *it holds that*

$$\Pr[\mathsf{mDec}(\Sigma_{\sigma_t}, \mathsf{sk}_t, \mathsf{ReEnc}(\Sigma', \mathsf{ReKeyGen}(\Sigma', \mathsf{sk}_f, \mathsf{pk}_t), \mathsf{ct}_f)) = m] = 1,$$

where $\Sigma' := (\Sigma_{\sigma_f}, \Sigma_{\sigma_t})$. *In the case of UPRE,* $\mathsf{mDec}(\Sigma_{\sigma_t}, \cdot, \cdot) = \mathsf{Dec}_{\sigma_t}(\cdot, \cdot)$.

Before we present the definition of the HRA security for UPRE, we give an informal explanation about it. Readers who are familiar with PRE-HRA security [Coh17] may be able to skip explanations below and jump into the formal definition. Readers who are familiar with PRE-CPA security [ABH09, Coh17] may be able to skip explanations below except "Honest encryption and re-encryption query" part.

Challenge query: Basically, we consider a natural extension of the CPA security of PKE. The adversary selects a target public-key pk_{i^*} indexed by i^* and tries to distinguish whether a target ciphertext ct_{i^*} is an encryption of m_0 or m_1 that it selects. This will be modeled by the challenge oracle $\mathcal{O}_{\mathsf{cha}}$.

Key query: The adversary can be given public keys pk_i or key pairs $(\mathsf{pk}_i, \mathsf{sk}_i)$ by specifying a user and a PKE scheme at the setup phase since we consider multiple keys and schemes. When a secret key is given, it means its owner is corrupted.

Re-encryption key query: The most notable feature is that the adversary is given re-encryption keys by the re-encryption key oracle $\mathcal{O}_{\mathsf{rekey}}$. If the adversary specifies existing indices of keys, say (i, j), then it is given a corresponding re-encryption key from i to j. Here, we must restrict queries for some indices to prevent trivial attacks. If j is a corrupted user and i is the target user (queried to $\mathcal{O}_{\mathsf{cha}}$), then the adversary trivially wins the security game by converting the target ciphertext and decrypting with the corrupted key sk_j. Therefore, such queries must be prohibited.

Honest encryption and re-encryption query: If the adversary specifies keys and a ciphertext to the re-encryption oracle $\mathcal{O}_{\mathsf{reenc}}$, then it is given a re-encrypted ciphertext generated from queried values. One might think this oracle is redundant since it is simulatable by $\mathcal{O}_{\mathsf{rekey}}$. However, there is a subtle issue here since a re-encryption key query with a corrupted delegatee is prohibited as explained above. As Cohen observed [Coh17] in the setting of PRE, simply prohibiting such a query is not sufficient and considering re-encryption queries is meaningful.

Re-encrypted ciphertexts may leak information about a delegator key pair and help to attack a delegator ciphertext. As Cohen observed [Coh17], if a re-encryption key is $\mathsf{Enc}(\mathsf{pk}_t, \mathsf{sk}_f)$ and *it is included in a re-encrypted ciphertext*, then the delegatee easily breaks security. This is unsatisfactory when we consider applications of PRE and UPRE. However, in the setting of PRE, such a construction is secure under the standard CPA-security model since it prohibits queries (i, j) (resp. (i, j, ct_i)) to the re-encryption key generation (resp. re-encryption) oracle [Coh17]. Thus, we introduce the notion of derivative and the honest encryption oracle $\mathcal{O}_{\mathsf{enc}}$ in UPRE as Cohen did.

We say that a (re-encrypted) ciphertext is a derivative if it is the target ciphertext generated by the challenge oracle or a re-encrypted ciphertext from the target ciphertext. This is managed by a set Drv. The honest encryption oracle allows the adversary to obtain a re-encrypted ciphertext under a corrupted key from honest encryption. The re-encryption oracle does not accept queries whose delegatee is a corrupted user j and ciphertext is a derivative to prevent trivial attacks. Moreover, the re-encryption oracle does not accept ciphertexts that are not generated via the honest encryption oracle.

Definition 3.4 (Derivative). *We say that a (re-encrypted) ciphertext is a derivative when the (re-encrypted) ciphertext is a target ciphertext itself or obtained from a target ciphertext given by $\mathcal{O}_{\mathsf{cha}}$ by applying re-encryption.*

Definition 3.5 (UPRE for PKE: Single-Hop HRA Security). *We define the experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{upre\text{-}hra}}(1^{\lambda}, b)$ between an adversary $\mathcal{A}$ and a challenger. The experiment consists of three phases.*
Phase 1 (Setup): *This is the setup phase. All security parameters are chosen by the challenger.*

- *The challenger initializes $\#\mathsf{Keys} := 0, \mathsf{HList} := \emptyset, \mathsf{CList} := \emptyset$, $\#\mathsf{CT} := 0$, $\mathsf{KeyCTList} := \emptyset$, $\mathsf{Drv} := \emptyset$. Note that we assume that all indices are recorded with keys and corresponding schemes though we do not explicitly write for simplicity.*
- *For an honest key query (i, σ_i), the challenger generates uncorrupted keys $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}_{\sigma_i}(1^{\lambda_i})$, sends $(\Sigma_{\sigma_i}, \mathsf{pk}_i)$ to $\mathcal{A}$, and sets $\mathsf{HList} := \mathsf{HList} \cup i$ and $\#\mathsf{Keys} := \#\mathsf{Keys} + 1$.*
- *For a corrupted key query (i, σ_i), the challenger generates corrupted keys $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}_{\sigma_i}(1^{\lambda_i})$, sends $(\Sigma_{\sigma_i}, \mathsf{pk}_i, \mathsf{sk}_i)$ to $\mathcal{A}$, and sets $\mathsf{CList} := \mathsf{CList} \cup i$ and $\#\mathsf{Keys} := \#\mathsf{Keys} + 1$.*

Let $\mathcal{M}_U$ be the intersection of all message spaces defined by $\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_{\#\mathsf{Keys}}}$.
Phase 2 (Oracle query): *This is the oracle query phase.*

$\mathcal{O}_{\mathsf{enc}}(i, m)$**:** *For an honest encryption query (i, m) where $i \leq \#\mathsf{Keys}$, the challenger generates $\mathsf{ct}_i \leftarrow \mathsf{Enc}_{\sigma_i}(\mathsf{pk}_i, m)$, sets $\#\mathsf{CT} := \#\mathsf{CT} + 1$, records $(\mathsf{ct}_i, \Sigma_{\sigma_i}, i, \#\mathsf{CT})$ in $\mathsf{KeyCTList}$, and gives $(\mathsf{ct}_i, \#\mathsf{CT})$ to $\mathcal{A}$.*

$\mathcal{O}_{\mathsf{rekey}}(i, j)$**:** *For a re-encryption key query (i, j) where $i, j \leq \#\mathsf{Keys}$, the challenger outputs $\perp$ if $i = j$ or $i \in \mathsf{HList} \wedge j \in \mathsf{CList}$. If a re-encryption key $\mathsf{rk}_{i \to j}$ for (i, j) is already stored, the challenger just retrieves and returns it. Otherwise, the challenger generates $\mathsf{rk}_{i \to j} \leftarrow \mathsf{ReKeyGen}(\Sigma_{\sigma_i}, \Sigma_{\sigma_j}, \mathsf{sk}_i, \mathsf{pk}_j)$ and gives $\mathsf{rk}_{i \to j}$ to $\mathcal{A}$ and stores it.*

$\mathcal{O}_{\mathsf{reenc}}(i, j, k)$**:** *For a re-encryption query (i, j, k) where $i, j \leq \#\mathsf{Keys}$ and $k \leq \#\mathsf{CT}$, the challenger does the following.*

1. *If $j \in \mathsf{CList} \wedge k \in \mathsf{Drv}$, then returns $\perp$.*
2. *If there is no value $(*, *, i, k)$ in $\mathsf{KeyCTList}$, returns $\perp$.*
3. *Otherwise, retrieves $\mathsf{rk}_{i \to j}$ for (i, j) (if it does not exists, generates $\mathsf{rk}_{i \to j} \leftarrow \mathsf{ReKeyGen}(\Sigma_{\sigma_i}, \Sigma_{\sigma_j}, \mathsf{sk}_i, \mathsf{pk}_j)$ and stores it), generates $\mathsf{rct} \leftarrow \mathsf{ReEnc}(\Sigma_{\sigma_i}, \Sigma_{\sigma_j}, \mathsf{rk}_{i \to j}, \mathsf{ct}_i)$ from ct_i in $\mathsf{KeyCTList}$, sets $\#\mathsf{CT} := \#\mathsf{CT} + 1$, records $(\mathsf{rct}, \Sigma_{\sigma_j}, j, \#\mathsf{CT})$ in $\mathsf{KeyCTList}$, and gives $(\mathsf{rct}, \#\mathsf{CT})$ to $\mathcal{A}$.*

$\mathcal{O}_{\mathsf{cha}}(i^*, m_0, m_1)$**:** *This oracle is invoked only once. For a challenge query (i^*, m_0, m_1) where $i^* \in \mathsf{HList}$ and $m_0, m_1, \in \mathcal{M}_U$ (defined at the end of Phase 1), the challenger generates $\mathsf{ct}^* \leftarrow \mathsf{Enc}_{\sigma_{i^*}}(pk_{i^*}, m_b)$, gives it to $\mathcal{A}$, and sets $\#\mathsf{CT} := \#\mathsf{CT} + 1$, $\mathsf{Drv} := \mathsf{Drv} \cup \{\#\mathsf{CT}\}$, $\mathsf{KeyCTList} := \mathsf{KeyCTList} \cup \{(\mathsf{ct}^*, \Sigma_{\sigma_{i^*}}, i^*, \#\mathsf{CT})\}$.*

Phase 3 (Decision) : *This is the decision phase. $\mathcal{A}$ outputs a guess b' for b. The experiment outputs b'.*
We say the UPRE *is single-hop UPRE-HRA secure if, for any PPT $\mathcal{A}$, it holds that*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{upre\text{-}hra}}(\lambda) := |\Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{upre\text{-}hra}}(1^{\lambda}, 0) = 1] - \Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{upre\text{-}hra}}(1^{\lambda}, 1) = 1]| \leq \mathsf{negl}(\lambda).$$

Discussion on Definition 3.5. (1) On security parameter: We can simply set $\forall i\ \lambda_i := \lambda$. Some λ_j may be longer than other λ_i (say, $\lambda_j = \mathrm{poly}(\lambda_i)$). (2) On adaptive corruption: The adversary is not allowed to adaptively corrupt users during the experiment. This is because, in general, it is difficult to achieve security against adaptive corruption. In particular, in our setting, $\mathcal{O}_{\mathsf{rekey}}$ cannot decide whether it should return $\perp$ or a valid re-encryption key if j may be corrupted later. This static security is standard in the PRE setting [AFGH05, CH07, LV08, ABH09]. One exception is the work by Fuchsbauer, Kamath, Klein, and Pietrzak [FKKP18]. The honest and corrupted key generation queries could be moved to the oracle query phase, but it does not incur a significant difference. Thus, we select a simpler model as most works on re-encryption did [AFGH05, LV08, ABH09, Coh17].

3.2 Unidirectional Multi-Hop UPRE

In this section, we introduce multi-hop UPRE, which is an extension of single-hop UPRE, where a re-encrypted ciphertext rct generated by $\mathsf{rk}_{f\to t}$ could be re-encrypted many times. Let $L = L(\lambda)$ be the maximum number of hops that a UPRE scheme can support.

Definition 3.6 (UPRE for PKE: L-hop Correctness). *A multi-hop UPRE scheme* mUPRE *for PKE is L-hop correct if for all PKE schemes* $(\Sigma_{\sigma_0}, \Sigma_{\sigma_1}, \ldots, \Sigma_{\sigma_L})$ *that satisfy correctness,* $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}_{\sigma_i}(1^{\lambda_i})$ *(for all $i = 0, \ldots, L$),* $m \in \mathcal{M}_{\sigma_0} \cap \cdots \cap \mathcal{M}_{\sigma_L}$, $\mathsf{ct}_0 \leftarrow \mathsf{Enc}_{\sigma_0}(\mathsf{pk}_0, m)$, *it holds that*

$$\Pr[\mathsf{mDec}(\Sigma_{\sigma_j}, \mathsf{sk}_j, \mathsf{rct}^{(j)}, j) = m] = 1$$

where $\mathsf{rct}^{(j)} \leftarrow \mathsf{ReEnc}(\Sigma'_j, \mathsf{ReKeyGen}(\Sigma'_j, \mathsf{sk}_{j-1}, \mathsf{pk}_j), \mathsf{rct}^{(j-1)})$, $\mathsf{rct}^{(0)} = \mathsf{ct}_0$, $\Sigma'_j := (\Sigma_{\sigma_{j-1}}, \Sigma_{\sigma_j})$ *and* $j \in [1, L]$.

The reason why mDec is indexed by j is that the decryption procedure for j-times re-encrypted ciphertexts might be different. See Section 6 as a concrete example.

The security notion of multi-hop UPRE is similar to that of single-hop one, but slightly more complex since we consider many intermediate keys from a delegator to a delegatee. In particular, we use a directed acyclic graph (DAG) to reflect the relationships among keys. A user is modeled as a vertex in a graph and if there exists a re-encryption key from vertex (user) i to vertex (user) j, then a directed edge (i, j) is assigned between the vertices (note that edge (i, j) is not equal to (j, i) since we consider DAGs). That is, a DAG $G = (V, E)$ denotes that V is a set of users and E is a set of index pairs whose re-encryption key was issued. We do not consider cyclic graphs in this study since it incurs an issue of circular security in our constructions[5].

We introduce the notion of *admissible edges* to exclude trivial attacks by using oracles. Roughly speaking, an admissible edge means that ciphertexts under a target public key will not be converted into ciphertexts under *corrupted* public keys in CList.

Definition 3.7 (Admissible edge). *We say that (i, j) is an admissible edge with respect to $G = (V, E)$ if, in $E \cup (i, j)$, there does not exist a path from any vertex $i^* \in$ HList (honest user set fixed at the setup phase) to $j^* \in$ CList such that the path includes edge (i, j) as an intermediate edge (this includes the case $j = j^*$). That is, no edge sets* $\{(i^*, i'_x), (i'_x, i'_{x+1}), \ldots, (i'_{y-1}, i'_y), (i'_y, i), (j, j'_z), (j'_z, j'_{z+1}), \ldots, (j'_{w-1}, j'_w), (j'_w, j^*)\}$ *in E.*

We also introduce the notion of the *selective graph model* as a weaker attack model. In the selective graph model, the adversary must commit a DAG $G^* = (V^*, E^*)$ at the beginning of an experiment. To formally define this model, we define a *deviating edge with respect to* $G^* = (V^*, E^*)$.

Definition 3.8 (deviating edge). *We say that (i, j) is a deviating edge with respect to G^* in the selective graph model if $i \in V^* \wedge j \notin V^*$ or $j \in V^* \wedge i \notin V^*$.*

In the selective graph model, the adversary must select $i^* \in V^*$ as the target vertex that will be queried to $\mathcal{O}_{\mathsf{cha}}$. Moreover, the adversary is not given re-encryption keys from $\mathcal{O}_{\mathsf{rekey}}$ if queried (i, j) is a deviating edge. That is, the structure of DAG that is connected to the target vertex must be fixed in advance. This DAG also describes which re-encryption keys are queried since it fixes the edge set. The structure of DAG outside of G^* is dynamically determined according to queries to $\mathcal{O}_{\mathsf{rekey}}$. We call selective multi-hop HRA security if we consider the selective graph attack model. The description of the multi-hop selective HRA security overlaps most of that of the multi-hop HRA security, so we use $\boxed{\text{boxes}}$ to show that the conditions in boxes are only for the selective security.

Definition 3.9 (UPRE for PKE: Multi-Hop (selective) HRA Security). *We define the experiment* $\mathsf{Exp}^{\mathsf{m\text{-}upre\text{-}hra}}_{\mathcal{A}}(1^\lambda, b)$ *between an adversary $\mathcal{A}$ and a challenger. The experiment consists of three phases.*
Phase 1 (Setup): *This is the setup phase. All security parameters are chosen by the challenger.*

- *The challenger initializes* $\#\mathsf{Keys} := 0, \mathsf{HList} := \emptyset, \mathsf{CList} := \emptyset, \#\mathsf{CT} := 0, \mathsf{KeyCTList} := \emptyset, \mathsf{Drv} := \emptyset, V := \emptyset, E := \emptyset$.
- $\boxed{\text{This item is only for the selective graph model.}}$ *At the beginning of this phase, $\mathcal{A}$ must commit a DAG $G^* = (V^*, E^*)$. We assume that $\mathcal{A}$ selects σ_i for all $i \in V^*$. The challenger generates uncorrupted keys* $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}_{\sigma_i}(1^{\lambda_i})$ *for all $i \in V^*$. The challenger sends* $(\Sigma_{\sigma_i}, \mathsf{pk}_i)$ *for all $i \in V^*$ to $\mathcal{A}$. The challenger sets* $\mathsf{HList} := \mathsf{HList} \cup V^*$, $\#\mathsf{Keys} := \#\mathsf{Keys} + |V^*|$, $V := V^*$, *and* $E := E^*$.

[5] The circular security issue arises in constructions that use general PKE schemes. If there exists a cycle, any re-encryption key in the cycle depends on all secret keys in the cycle. Thus, we have no way to erase secret keys in security proofs. This does not happen in concrete constructions based on some hard problems such as the DDH.

- *For an honest key generation query (i,σ_i), the challenger generates uncorrupted keys $(\mathsf{pk}_i,\mathsf{sk}_i) \leftarrow \mathsf{Gen}_{\sigma_i}(1^{\lambda_i})$, sends $(\Sigma_{\sigma_i},\mathsf{pk}_i)$ to $\mathcal{A}$, and sets $\mathsf{HList} := \mathsf{HList} \cup i$, $\#\mathsf{Keys} := \#\mathsf{Keys} + 1$, and $V := V \cup \{i\}$.*

- *For a corrupted key generation query (i,σ), the challenger generates corrupted keys $(\mathsf{pk}_i,\mathsf{sk}_i) \leftarrow \mathsf{Gen}_{\sigma_i}(1^{\lambda_i})$, sends $(\Sigma_{\sigma_i},\mathsf{pk}_i,\mathsf{sk}_i)$ to $\mathcal{A}$, and sets $\mathsf{CList} := \mathsf{CList} \cup i$, $\#\mathsf{Keys} := \#\mathsf{Keys} + 1$, and $V := V \cup \{i\}$.*

- *The challenger maintains graph $G := (V,E)$ during the experiment. Note that we assume that all keys and schemes are recorded with vertices and edges though we do not explicitly write for simplicity.*

Phase 2 (Oracle query): *This is the oracle query phase.*

$\mathcal{O}_{\mathsf{enc}}(i,m)$**:** *For an honest encryption query (i,m) where $i \le \#\mathsf{Keys}$, the challenger generates $\mathsf{ct}_i \leftarrow \mathsf{Enc}_{\sigma_i}(\mathsf{pk}_i,m)$, sets $\#\mathsf{CT} := \#\mathsf{CT} + 1$, record $(\mathsf{ct}_i,\Sigma_{\sigma_i},i,\#\mathsf{CT})$ in $\mathsf{KeyCTList}$, and gives $(\mathsf{ct}_i,\#\mathsf{CT})$ to $\mathcal{A}$.*

$\mathcal{O}_{\mathsf{rekey}}(i,j)$**:** *For a re-encryption key query (i,j) where $i,j \le \#\mathsf{Keys}$, the challenger does the following.*

1. *If $\boxed{(i,j) \text{ is a deviating edge with respect to } G^*,}$ $i = j$, or (i,j) is not an admissible edge with respect to $G = (V,E)$, then output $\bot$. See Definitions 3.7 and 3.8 for admissible edge and deviating edge.*
2. *Otherwise, the challenger generates $\mathsf{rk}_{i\to j} \leftarrow \mathsf{ReKeyGen}(\Sigma_{\sigma_i},\Sigma_{\sigma_j},\mathsf{sk}_i,\mathsf{pk}_j)$ and updates $E := E \cup (i,j)$ (if $\mathsf{rk}_{i\to j}$ is already recorded, then the challenger just retrieves it) and gives $\mathsf{rk}_{i\to j}$ to $\mathcal{A}$.*

$\mathcal{O}_{\mathsf{reenc}}(i,j,k)$**:** *For a re-encryption query (i,j,k) where $i,j \le \#\mathsf{Keys}$ and $k \le \#\mathsf{CT}$, the challenger does the following.*

1. *If (i,j) is not an admissible with respect to $G = (V,E)$ and $k \in \mathsf{Drv}$, then returns $\bot$.*
2. *If there is no $(*,*,i,k)$ in $\mathsf{KeyCTList}$, then outputs $\bot$.*
3. *Otherwise, retrieves $\mathsf{rk}_{i\to j}$for (i,j) (if it does not exists, generates $\mathsf{rk}_{i\to j} \leftarrow \mathsf{ReKeyGen}(\Sigma_{\sigma_i},\Sigma_{\sigma_j},\mathsf{sk}_i,\mathsf{pk}_j)$ and stores it), generates $\mathsf{rct}_j \leftarrow \mathsf{ReEnc}(\Sigma_{\sigma_i},\Sigma_{\sigma_j},\mathsf{rk}_{i\to j},\mathsf{rct}_i)$ from rct_i in $\mathsf{KeyCTList}$, sets $\#\mathsf{CT} := \#\mathsf{CT} + 1$, records $(\mathsf{rct}_j,\Sigma_{\sigma_j},j,\#\mathsf{CT})$ in $\mathsf{KeyCTList}$, and gives $(\#\mathsf{CT},\mathsf{rct}_j)$ to $\mathcal{A}$. If $k \in \mathsf{Drv}$, then also sets $\mathsf{Drv} := \mathsf{Drv} \cup \{\#\mathsf{CT}\}$.*

$\mathcal{O}_{\mathsf{cha}}(i^*,m_0,m_1)$**:** *This oracle is invoked only once. For a challenge query (i^*,m_0,m_1) where $i^* \in \mathsf{HList}$ and $m_0,m_1, \in \mathcal{M}_U$ (same as defined in Definition 3.5)$\boxed{\text{, and } i^* \in V^*}$, the challenger generates $\mathsf{ct}^* \leftarrow \mathsf{Enc}_{\sigma_{i^*}}(pk_{i^*},m_b)$ and gives it to $\mathcal{A}$. The challenger also sets $\#\mathsf{CT} := \#\mathsf{CT} + 1$, $\mathsf{Drv} := \mathsf{Drv} \cup \{\#\mathsf{CT}\}$, $\mathsf{KeyCTList} := \mathsf{KeyCTList} \cup \left\{(\mathsf{ct}^*,\Sigma_{\sigma_{i^*}},i^*,\#\mathsf{CT})\right\}$.*

Phase 3 (Decision) : *This is the decision phase. $\mathcal{A}$ outputs a guess b' for b. The experiment outputs b'.*

We say the UPRE is multi-hop $\boxed{\text{selectively}}$ UPRE-HRA secure if, for any PPT $\mathcal{A}$, it holds that

$$\mathsf{Adv}^{\text{m-upre-hra}}_{\mathcal{A}}(\lambda) := |\Pr[\mathsf{Exp}^{\text{m-upre-hra}}_{\mathcal{A}}(1^\lambda,0) = 1] - \Pr[\mathsf{Exp}^{\text{m-upre-hra}}_{\mathcal{A}}(1^\lambda,1) = 1]| \le \mathsf{negl}(\lambda).$$

We use $\boxed{\mathsf{Adv}^{\text{ms-upre-hra}}_{\mathcal{A}}(\lambda) \text{ and } \mathsf{Exp}^{\text{ms-upre-hra}}_{\mathcal{A}}(1^\lambda,b)}$ for the selective security.

In fact, the single-hop HRA security is a special case of the multi-hop HRA security. However, we separately write them since the single-hop one is easier to understand. We can also consider single-hop selective HRA security. We use $\mathsf{Adv}^{\text{s-upre-hra}}_{\mathcal{A}}(\lambda)$ and $\mathsf{Exp}^{\text{s-upre-hra}}_{\mathcal{A}}(1^\lambda,b)$ for their experiment and advantage.

UPRE-CPA Security. We can easily consider the CPA-security of UPRE. We can obtain the security experiment of the CPA-security if we employ the following items in the experiment of the HRA security.

1. The honest encryption oracle $\mathcal{O}_{\mathsf{enc}}$ is not used.
2. Neither the set Drv nor number $\#\mathsf{CT}$ is used.
3. The condition that $\mathcal{O}_{\mathsf{reenc}}$ outputs $\bot$ for a query (i,j) such that $i \in \mathsf{HList} \land j \in \mathsf{CList}$ (or (i,j) is not an admissible edge) is used instead of the first and second conditions of $\mathcal{O}_{\mathsf{reenc}}$ in the experiment of the HRA security.

3.3 Security against Corrupted-Delegator Re-Encryption Attacks

Re-encrypted ciphertexts of relaxed UPRE schemes might include values that leak information about a plaintext to a delegator (that is, an entity that has a secret key for the original ciphertext). This is an important issue to use UPRE in migration of encryption systems explained in Section 1.1. We will see a concrete example in Section 6. To capture attacks on re-encrypted ciphertext by corrupted delegator, we define a new security notion for UPRE (and PRE), security against corrupted-delegator re-encryption attacks (CRA). We write the definition of the UPRE case. The PRE case is similarly defined as PRE-CRA security. We can also similarly define a single-hop variant. Note that this is meaningful for relaxed UPRE since, in UPRE, a re-encrypted ciphertext is a ciphertext of a delegatee.

Definition 3.10 ((Selective) UPRE-CRA security). *The experiment* $\mathsf{Exp}^{\text{m-upre-cra}}_{\mathcal{A}}(1^\lambda, b)$ *of this security notion is the same as that of multi-hop UPRE-HRA security except that the challenge oracle* $\mathcal{O}_{\mathsf{cha}}$ *is modified as follows. Contents in* $\boxed{\textit{boxes}}$ *are only applied to the selective security.*

$\mathcal{O}_{\mathsf{cha}}(i_{\mathsf{c}}, i^*, m_0, m_1)$**:** *This oracle is invoked only once. For a challenge query* $(i_{\mathsf{c}}, i^*, m_0, m_1)$ *where* $i_{\mathsf{c}} \in \mathsf{CList} \wedge i^* \in \mathsf{HList}$ *and* $m_0, m_1, \in \mathcal{M}_U$ *(same as defined in Definition 3.5)*$\boxed{\textit{, and } i^* \in V^*}$*, the challenger does the following.*

1. *Generates* $\mathsf{ct}_{i_{\mathsf{c}}} \leftarrow \mathsf{Enc}_{\sigma_{i_{\mathsf{c}}}}(pk_{i_{\mathsf{c}}}, m_b)$.
2. *Retrieves* $\mathsf{rk}_{i_{\mathsf{c}} \to i^*} = \mathsf{ReKeyGen}(\Sigma_{\sigma_{i_{\mathsf{c}}}}, \Sigma_{\sigma_{i^*}}, \mathsf{sk}_{i_{\mathsf{c}}}, \mathsf{pk}_{i^*})$ *(if there does not exists, generates it).*
3. *Generates* $\mathsf{rct}^* \leftarrow \mathsf{ReEnc}(\Sigma_{\sigma_{i_{\mathsf{c}}}}, \Sigma_{\sigma_{i^*}}, \mathsf{rk}_{i_{\mathsf{c}} \to i^*}, \mathsf{ct}_{i_{\mathsf{c}}})$ *and gives* $(\mathsf{rct}^*, \mathsf{rk}_{i_{\mathsf{c}} \to i^*})$ *to* $\mathcal{A}$.

The challenger also sets $\#\mathsf{CT} := \#\mathsf{CT} + 1$, $\mathsf{Drv} := \mathsf{Drv} \cup \{\#\mathsf{CT}\}$, $\mathsf{KeyCTList} := \mathsf{KeyCTList} \cup \left\{(\mathsf{ct}^*, \Sigma_{\sigma_{i^*}}, i^*, \#\mathsf{CT})\right\}$.

We say the UPRE *is multi-hop* $\boxed{\textit{selectively}}$ *UPRE-CRA secure if, for any PPT* $\mathcal{A}$*, it holds that*

$$\mathsf{Adv}^{\text{m-upre-cra}}_{\mathcal{A}}(\lambda) := |\Pr[\mathsf{Exp}^{\text{m-upre-cra}}_{\mathcal{A}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Exp}^{\text{m-upre-cra}}_{\mathcal{A}}(1^\lambda, 1) = 1]| \leq \mathsf{negl}(\lambda).$$

We use $\boxed{\mathsf{Adv}^{\text{ms-upre-cra}}_{\mathcal{A}}(\lambda) \textit{ and } \mathsf{Exp}^{\text{ms-upre-cra}}_{\mathcal{A}}(1^\lambda, b)}$ *for the selective security.*

This definition means that adversaries that have secret key $\mathsf{sk}_{i_{\mathsf{c}}}$ cannot break the security of the re-encrypted ciphertext rct^* generated from the ciphertext $\mathsf{ct}_{i_{\mathsf{c}}}$ under $\mathsf{pk}_{i_{\mathsf{c}}}$ if they are not given the original ciphertext $\mathsf{ct}_{i_{\mathsf{c}}}$ (even if re-encryption key $\mathsf{rk}_{i_{\mathsf{c}} \to i^*}$ is given). The fact that $\mathsf{ct}_{i_{\mathsf{c}}}$ is not given to $\mathcal{A}$ guarantees that $\mathcal{A}$ cannot trivially break the security.

3.4 On Re-Encryption Simulatability

Cohen introduced the notion of re-encryption simulatability for PRE to prove PRE-HRA security in a modular way [Coh17]. He proved that if a PRE scheme is PRE-CPA secure and satisfies re-encryption simulatability[6], then the scheme is PRE-HRA secure. See Definition A.1 in Appendix A for the definition.

The re-encryption simulatability is sufficient to prove PRE-HRA security (if a PRE is PRE-CPA secure scheme) and useful. Thus, one might think it is better to use re-encryption simulatability for UPRE. However, it is a slightly stronger security notation. Our relaxed UPRE schemes in Sections 6 and 7 are *UPRE-HRA secure*, yet *does not* satisfy re-encryption simulatability. Thus, we do not use re-encryption simulatability to prove UPRE-HRA security in this study[7]. See Appendix A.1 for the reason why our schemes in Sections 6 and 7 does not satisfies re-encryption simulatability.

3.5 UPRE for More Advanced Encryption

We give the basic definitions of UPRE for PKE in Sections 3.1 and 3.2. We can consider more definitions for advanced encryption since UPRE is a general concept.

[6] Note that Cohen *does not* use key-privacy of PRE [ABH09] to prove PRE-HRA security.

[7] For our UPRE scheme in Section 5, we might be able to use re-encryption simulatability to prove UPRE-HRA security since Our UPRE scheme in Section 5 satisfies re-encryption simulatability for UPRE defined in Appendix A. Moreover, we define a weaker variant of re-encryption simulatability for UPRE (and PRE) that still implies HRA security in Appendix A.2. However, such a definition is not simple, and proofs are not simplified. Proving such a weak re-encryption simulatability takes almost the same efforts to prove HRA security directly. Thus, we do not use re-encryption simulatability in the main body.

CCA-security. First, we can consider CCA-security of UPRE for PKE. The definition of CCA-security of UPRE for PKE could be defined in a similar way to that of PRE [CH07, LV08, HKK$^+$12] though it will be more complex. We leave giving a formal definition of CCA-security and concrete constructions as an open problem since they are not in the scope of this paper. The focus of this study is that we initiate the study of UPRE, present the basic definition, and construct concrete schemes from well-known cryptographic assumptions.

Beyond PKE. We can also consider not only UPRE for PKE but also UPRE for identity-based encryption (IBE), attribute-based encryption (ABE), and functional encryption (FE). Moreover, we can even consider UPRE from a primitive to another primitive such as from IBE to FE. It is easier to consider UPRE between the same primitive since additional inputs to encryption algorithms such as an attribute in a delegator ciphertext can be recycled in a re-encrypted ciphertext. Defining UPRE between different primitives is much challenging since we have issues about how to set such additional inputs at re-encryption phase and define security between different primitives. We leave these as open problems since they are not in the scope of this paper.

4 On the (Im)possibility of Universal Proxy Re-Encryption

In this section, we prove that the existence of UPRE implies that of ACVBB obfuscation for all re-encryption circuits.

4.1 Average-Case Virtual Black-Box Obfuscation for All Re-Encryption

First, we review the definition of ACVBB obfuscation.

Definition 4.1 (Average-Case Virtual Black-Box Secure Obfuscation [HRsV11]). *A PPT algorithm $\mathcal{O}$ is an average-case virtual black-box (ACVBB) secure obfuscator for the family $\mathcal{C} = \{\mathcal{C}_\lambda\}$ if it satisfies the following properties.*

Preserving Functionality: *There exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for any input length λ, for any $C \in \mathcal{C}_\lambda$, it holds that*

$$\Pr_{r \leftarrow \mathcal{R}_{\mathcal{O}}}[\forall x \in \{0,1\}^\lambda : \Delta(\mathcal{O}(C)(x), C(x)) \leq \mathsf{negl}(\lambda)] > 1 - \mathsf{negl}(\lambda).$$

Polynomial Slowdown: *There exists a polynomial $p(\lambda)$ such that for sufficiently large input length λ, for any $C \in \mathcal{C}_\lambda$, it holds that $|\mathcal{O}(C)| \leq p(|C|)$.*

Average-Case Secure Virtual Black-Box: *For any PPT adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ and a negligible function* $\mathsf{negl}(\lambda)$ *such that for any PPT distinguisher $\mathcal{D}$, for sufficiently long input length λ and for any polynomial-size auxiliary input* z, *it holds that*

$$|\Pr[\mathcal{D}^C(\mathcal{A}(\mathcal{O}(C),\mathsf{z}),\mathsf{z}) = 1 \mid C \leftarrow \mathcal{C}_\lambda] - \Pr[\mathcal{D}^C(\mathcal{S}^C(1^\lambda,\mathsf{z}),\mathsf{z}) = 1 \mid C \leftarrow \mathcal{C}_\lambda]| \leq \mathsf{negl}(\lambda).$$

Next, we define obfuscation for all re-encryption circuits. That is, an obfuscator can treat all re-encryption circuits that are specified by two key pairs of any PKE scheme.

Definition 4.2 (ACVBB Obfuscator for All Re-Encryption Circuits). *Let $\mathcal{RC}_\lambda$ be the family of re-encryption circuits defined as follows.*

$$\mathcal{RC}_\lambda := \{\mathsf{C}_{\mathsf{re}}[\Sigma_{\sigma_f}, \Sigma_{\sigma_t}, \mathsf{sk}_f, \mathsf{pk}_f, \mathsf{pk}_t] \mid (\mathsf{pk}_f, \mathsf{sk}_f) \leftarrow \mathsf{Gen}_{\sigma_f}(1^\lambda), (\mathsf{pk}_t, \mathsf{sk}_t) \leftarrow \mathsf{Gen}_{\sigma_t}(1^\lambda)\},$$

where $\Sigma_{\sigma_f} = (\mathsf{Gen}_{\sigma_f}, \mathsf{Enc}_{\sigma_f}, \mathsf{Dec}_{\sigma_f})$ and $\Sigma_{\sigma_t} = (\mathsf{Gen}_{\sigma_t}, \mathsf{Enc}_{\sigma_t}, \mathsf{Dec}_{\sigma_t})$ are any PKE schemes. We say that a PPT algorithm $\mathcal{O}$ is an ACVBB secure obfuscator for all re-encryption circuits if it is ACVBB secure obfuscator for $\mathcal{RC} = \{\mathcal{RC}_\lambda\}$.

Hereafter, we overload the notation $\Sigma_{\sigma_i} = (\mathsf{Gen}_{\sigma_i}, \mathsf{Enc}_{\sigma_i}, \mathsf{Dec}_{\sigma_i})$ by $\Sigma_i = (\mathsf{Gen}_i, \mathsf{Enc}_i, \mathsf{Dec}_i)$ for ease of notation. That is, we think Σ_i is a scheme used by user i and it may happen $\Sigma_i = \Sigma_j$ since these were originally Σ_{σ_i} and Σ_{σ_j} $(\sigma_i, \sigma_j \in [N])$.

Re-Encryption Circuit $\mathsf{C}_{\mathsf{re}}[\Sigma_{\sigma_f}, \Sigma_{\sigma_t}, \mathsf{sk}_f, \mathsf{pk}_f, \mathsf{pk}_t](x)$

Hardwired: Σ_{σ_f}, Σ_{σ_t}, sk_f, pk_f pk_t.

Input: x

1. If the input is keys, then outputs $(\Sigma_{\sigma_f}, \Sigma_{\sigma_t}, \mathsf{pk}_f, \mathsf{pk}_t)$.
2. If $x \notin \mathcal{C}_f$, then outputs $\bot$
3. Otherwise, compute $m \leftarrow \mathsf{Dec}_{\sigma_f}(\mathsf{sk}_f, x)$ and generate and return $\mathsf{ct}_t \leftarrow \mathsf{Enc}_{\sigma_t}(\mathsf{pk}_t, m)$.

Figure 1: The description of C_{re}

4.2 Universal Proxy Re-Encryption Implies Obfuscation for All Re-Encryption Circuits

In this section, we prove the following.

Theorem 4.3. *If* UPRE *is a (multi-hop) UPRE-CPA secure UPRE scheme, then there exists an ACVBB secure obfuscator for all re-encryption circuits.*

To prove this theorem, we introduce a notion of re-encryption key simulatability.

Definition 4.4 (Re-Encryption Key Simulatability for UPRE). *Let* $\mathsf{ReKeySim}$ *be a PPT simulator. We define the following experiments* $\mathsf{Exp}^{\mathsf{re\text{-}key\text{-}sim}}_{\mathcal{D}}(1^\lambda, b)$ *between a challenger and a distinguisher* $\mathcal{D}$ *as follows.*

1. *The challenger generates* $(\mathsf{pk}_f, \mathsf{sk}_f) \leftarrow \mathsf{Gen}_f(1^{\lambda_f})$ *and* $(\mathsf{pk}_t, \mathsf{sk}_t) \leftarrow \mathsf{Gen}_t(1^{\lambda_t})$.
2. *If* $b = 0$, *the challenger generates* $\mathsf{rk}_{f \to t} \leftarrow \mathsf{ReKeyGen}(1^\lambda, \Sigma_f, \Sigma_t, \mathsf{sk}_f, \mathsf{pk}_t)$ *and sends* $(\Sigma_f, \Sigma_t, \mathsf{pk}_f, \mathsf{pk}_t, \mathsf{rk}_{f \to t})$.
3. *If* $b = 1$, *the challenger generates* $\widetilde{\mathsf{rk}}_{f \to t} \leftarrow \mathsf{ReKeySim}(1^\lambda, \Sigma_f, \Sigma_t, \mathsf{pk}_f, \mathsf{pk}_t)$ *and sends* $(\Sigma_f, \Sigma_t, \mathsf{pk}_f, \mathsf{pk}_t, \widetilde{\mathsf{rk}}_{f \to t})$.
4. $\mathcal{D}$ *outputs* $b' \in \{0, 1\}$. *The experiment outputs* b'.

We say that UPRE *is re-encryption key simulatable if there exists a simulator* $\mathsf{ReKeySim}$, *for any PPT* $\mathcal{D}$, *it holds that*

$$|\Pr[\mathsf{Exp}^{\mathsf{re\text{-}key\text{-}sim}}_{\mathcal{D}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Exp}^{\mathsf{re\text{-}key\text{-}sim}}_{\mathcal{D}}(1^\lambda, 1) = 1]| \leq \mathsf{negl}(\lambda).$$

Remark 4.5. The re-encryption key simulatability above is a simple one since we do not consider any oracle that are defined in the security of UPRE (such as $\mathcal{O}_{\mathsf{rekey}}$, $\mathcal{O}_{\mathsf{reenc}}$). This is not an issue since we discuss the minimum requirement for UPRE here. We will argue that UPRE must satisfy at least the weaker re-encryption key simulatability above.

Remark 4.6. Definition 4.4 implicitly requires that $\mathsf{ReEnc}(\widetilde{\mathsf{rk}}_{f \to t}, \mathsf{ct}_f)$ outputs a ciphertext $\widetilde{\mathsf{ct}}_t \in \mathcal{C}_t$ and $\widetilde{\mathsf{ct}}_t \overset{\mathsf{c}}{\approx} \mathsf{ct}_t \leftarrow \mathsf{ReEnc}(\mathsf{rk}_{f \to t}, \mathsf{ct}_f)$ since otherwise re-encryption simulatability is trivially broken. Note that a re-encrypted ciphertext is exactly in the ciphertext space of a delegatee's PKE scheme since we consider UPRE.

We will prove the following. If a UPRE scheme $\mathsf{UPRE} = (\mathsf{ReKeyGen}, \mathsf{ReEnc})$ is UPRE-CPA secure, then UPRE must satisfy re-encryption key simulatability. That is, there exists a simulator that can simulate re-encryption keys for any PKE scheme without secret keys. Unless there exists such a simulator, we cannot prove the UPRE-CPA security. This is because in security proofs for UPRE-CPA, we must use IND-CPA security of Σ_{σ_f}. This means that:

- In the reduction, we must erase information about sk_f at some point.
- At the same time, we must simulate re-encryption keys to use an adversary of the UPRE-CPA security game.

Thus, there must exist a simulator $\mathsf{ReKeySim}$ that can simulate re-encryption keys without secret keys.

Lemma 4.7 (On the existence of re-encryption key simulator for UPRE). *If* $\mathsf{UPRE} = (\mathsf{ReKeyGen}, \mathsf{ReEnc})$ *is (multi-hop) UPRE-CPA secure for IND-CPA secure PKE schemes, then there exists a re-encryption key simulator* $\mathsf{ReKeySim}$ *that satisfies re-encryption key simulatability in Definition 4.4.*

Proof. Recall that UPRE can treat all PKE schemes and a re-encrypted ciphertext is exactly in the ciphertext space of a delegatee's PKE scheme. In the security experiment of UPRE, suppose that the adversary $\mathcal{A}$ sends a challenge query (i^*, m_0, m_1) to the challenge oracle $\mathcal{O}_{\mathsf{cha}}$ such that $i^* \in \mathsf{HList}$ and the target key pk_{i^*}. Here, pk_{i^*} could be any public-key of any IND-CPA secure PKE scheme and we cannot rely on any specific property of concrete PKE schemes in the setting of UPRE. A target ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Enc}_{i^*}(pk_{i^*}, m_b)$ is given to $\mathcal{A}$. Therefore, we must use the IND-CPA security of the target $\Sigma_{i^*} = (\mathsf{Gen}_{i^*}, \mathsf{Enc}_{i^*}, \mathsf{Dec}_{i^*})$ in a black-box way. This means that there must exist a reduction from breaking UPRE-CPA security to breaking IND-CPA security. In other words, we must construct an adversary $\mathcal{B}$ for PKE Σ_{i^*} that breaks IND-CPA security by using an adversary $\mathcal{A}$ for UPRE UPRE that breaks UPRE-CPA.

We can consider several hybrid experiments to prove UPRE-CPA of UPRE. Let $\mathsf{Hyb}^v_{\mathcal{A}}(b)$ be a hybrid experiment where the target ciphertext of UPRE is $\mathsf{ct}^* \leftarrow \mathsf{Enc}_{i^*}(pk_{i^*}, m_b)$ and other parts might be modified from the original experiment of UPRE-CPA. To prove UPRE-CPA security of UPRE, we must prove $\mathsf{Hyb}^v_{\mathcal{A}}(0) \overset{\mathsf{c}}{\approx} \mathsf{Hyb}^v_{\mathcal{A}}(1)$ by using IND-CPA security of Σ_{i^*}. In the reduction, an adversary of Σ_{i^*} denoted by $\mathcal{B}$ does not have sk_{i^*}. However, $\mathcal{B}$ must simulate the re-encryption key oracle $\mathcal{O}_{\mathsf{rekey}}$ to run $\mathcal{A}$. $\mathcal{A}$ is allowed to send query (i^*, j) such that (i^*, j) is an admissible edge. That is, $\mathcal{B}$ must simulate $\mathsf{rk}_{i^* \to j} \leftarrow \mathsf{ReKeyGen}(\Sigma_{i^*}, \Sigma_j, \mathsf{sk}_{i^*}, \mathsf{pk}_j)$ without sk_{i^*} to answer the query (i^*, j) from $\mathcal{A}$. If UPRE is UPRE-CPA secure, then there must exist the reduction above. In other words, $\mathcal{B}$ can simulate $\mathsf{rk}_{i^* \to j}$ that $\mathcal{A}$ cannot distinguish from $\mathsf{ReKeyGen}(\Sigma_{i^*}, \Sigma_j, \mathsf{sk}_{i^*}, \mathsf{pk}_j)$ without sk_{i^*}. Therefore, there must exist a simulator that can generate $\widetilde{\mathsf{rk}}_{i^* \to j}$ by using only publicly available values $(\Sigma_{i^*}, \Sigma_j, \mathsf{pk}_{i^*}, \mathsf{pk}_j)$ such that $\widetilde{\mathsf{rk}}_{i^* \to j}$ is computationally indistinguishable from $\mathsf{ReKeyGen}(\Sigma_{i^*}, \Sigma_j, \mathsf{sk}_{i^*}, \mathsf{pk}_j)$. That is, there exists an algorithm that takes as input $(\Sigma_{i^*}, \Sigma_j, \mathsf{pk}_{i^*}, \mathsf{pk}_j)$ and outputs $\widetilde{\mathsf{rk}}_{i^* \to j}$ that is computationally indistinguishable from $\mathsf{ReKeyGen}(\Sigma_{i^*}, \Sigma_j, \mathsf{sk}_{i^*}, \mathsf{pk}_j)$. This algorithm is the re-encryption key simulator $\mathsf{ReKeySim}$ since Σ_{i^*} and Σ_j could be any PKE scheme. ■

Lemma 4.8 (Re-encryption key simulatability implies ACVBB obfuscator for all re-encryption circuits). *If a UPRE scheme* $\mathsf{UPRE} = (\mathsf{ReKeyGen}, \mathsf{ReEnc})$ *satisfies re-encryption key simulatability for UPRE and* Σ_f *and* Σ_t *are IND-CPA secure PKE, then we can construct an ACVBB obfuscator for all re-encryption circuits.*

Proof. Our ACVBB obfuscator $\mathcal{O}$ is as follows.

$\underline{\mathcal{O}(\mathsf{C}_{\mathsf{re}}[\Sigma_f, \Sigma_t, \mathsf{sk}_f, \mathsf{pk}_f, \mathsf{pk}_t])}$:

- It takes $(\Sigma_f, \Sigma_t, \mathsf{sk}_f, \mathsf{pk}_f, \mathsf{pk}_t)$ as input and generates $\mathsf{rk}_{f \to t} \leftarrow \mathsf{ReKeyGen}(\Sigma_f, \Sigma_t, \mathsf{sk}_f, \mathsf{pk}_t)$.
- It generates a circuit $\mathcal{RE}[\Sigma_f, \Sigma_t, \mathsf{pk}_f, \mathsf{pk}_t, \mathsf{rk}_{f \to t}]$ described in Figure 2.
- It outputs $\overline{\mathcal{RE}} := \mathcal{RE}[\Sigma_f, \Sigma_t, \mathsf{pk}_f, \mathsf{pk}_t, \mathsf{rk}_{f \to t}]$ as an obfuscated circuit of $\mathsf{C}_{\mathsf{re}}[\Sigma_f, \Sigma_t, \mathsf{sk}_f, \mathsf{pk}_f, \mathsf{pk}_t]$.

Re-Encryption Circuit $\mathcal{RE}[\Sigma_f, \Sigma_t, \mathsf{pk}_f, \mathsf{pk}_t, \mathsf{rk}_{f \to t}](x)$

Hardwired: $\Sigma_f, \Sigma_t, \mathsf{pk}_f, \mathsf{pk}_t, \mathsf{rk}_{f \to t}$.

Input: x

1. If the input is keys, then outputs $(\Sigma_f, \Sigma_t, \mathsf{pk}_f, \mathsf{pk}_t)$.
2. If $x \notin \mathcal{C}_f$, then outputs $\perp$.
3. Otherwise, compute and return $\mathsf{ct}_t \leftarrow \mathsf{ReEnc}(\mathsf{rk}_{f \to t}, \mathsf{ct}_f)$.

Figure 2: The description of $\mathcal{RE}$

To prove the lemma, we show that for any PPT adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ and a negligible function $\mathsf{negl}(\lambda)$ such that for any PPT distinguisher $\mathcal{D}$, for sufficiently long input length λ and for any polynomial-size auxiliary input z, it holds that the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{C}_{\mathsf{re}}}(\lambda, \mathsf{z}) := |\Pr[\mathcal{D}^{\mathsf{C}_{\mathsf{re}}}(\mathcal{A}(\mathcal{O}(\mathsf{C}_{\mathsf{re}}), \mathsf{z}), \mathsf{z}) = 1 \mid \mathsf{C}_{\mathsf{re}} \leftarrow \mathcal{RC}_\lambda] - \Pr[\mathcal{D}^{\mathsf{C}_{\mathsf{re}}}(\mathcal{S}(1^\lambda, \mathsf{z}), \mathsf{z}) = 1 \mid \mathsf{C}_{\mathsf{re}} \leftarrow \mathcal{RC}_\lambda]|,$$

where $\mathsf{C}_{\mathsf{re}}[\Sigma_f, \Sigma_t, \mathsf{sk}_f, \mathsf{pk}_f, \mathsf{pk}_t]$ is described in Figure 1 and we omit hard-wired values $(\Sigma_f, \Sigma_t, \mathsf{sk}_f, \mathsf{pk}_f, \mathsf{pk}_t)$ for notational simplicity. The simulator $\mathcal{S}$ proceeds as follows.

$\underline{\mathcal{S}^{\mathsf{C}_{\mathsf{re}}}(1^\lambda, \mathsf{z})}$:

- It sends $\mathtt{keys}$ to $\mathsf{C_{re}}$ and receives $(\Sigma_f, \Sigma_t, \mathsf{pk}_f, \mathsf{pk}_t)$.
- It runs $\widetilde{\mathsf{rk}}_{f \to t} \leftarrow \mathsf{ReKeySim}(\Sigma_f, \Sigma_t, \mathsf{pk}_f, \mathsf{pk}_t)$.
- It constructs and outputs $\widetilde{\mathcal{RE}} := \mathcal{RE}[\Sigma_f, \Sigma_t, \mathsf{pk}_f, \mathsf{pk}_t, \widetilde{\mathsf{rk}}_{f \to t}]$ as a simulated obfuscated circuit of $\mathsf{C_{re}}$.

To prove $\mathsf{Adv}^{\mathsf{C_{re}}}(\lambda, \mathsf{z}) \leq \mathsf{negl}(\lambda)$, we introduce an oracle and advantage as follows. Let $\mathcal{J}$ be an oracle defined in Figure 3.

Dummy Re-Encryption Oracle $\mathcal{J}[\Sigma_f, \Sigma_t, \mathsf{pk}_f, \mathsf{pk}_t](x)$

Hardwired: Σ_f, Σ_t, pk_f, pk_t.
Input: x

1. If the input is $\mathtt{keys}$, then outputs $(\Sigma_f, \Sigma_t, \mathsf{pk}_f, \mathsf{pk}_t)$.
2. If $x \notin \mathcal{C}_f$, then outputs $\perp$.
3. Otherwise, compute and return $\mathsf{ct}_t \leftarrow \mathsf{Enc}_t(\mathsf{pk}_t, 0^\ell)$. ($\ell$ is the plaintext length of Σ_t.)

Figure 3: The description of $\mathcal{J}$

Next, we define

$$\mathsf{Adv}^{\mathcal{J}}(\lambda, \mathsf{z}) := |\Pr[\mathcal{D}^{\mathcal{J}}(\mathcal{A}(\mathcal{O}(\mathsf{C_{re}}), \mathsf{z}), \mathsf{z}) = 1 \mid \mathsf{C_{re}} \leftarrow \mathcal{RC}_\lambda] - \Pr[\mathcal{D}^{\mathcal{J}}(\mathcal{S}(1^\lambda, \mathsf{z}), \mathsf{z}) = 1 \mid \mathsf{C_{re}} \leftarrow \mathcal{RC}_\lambda]|.$$

Note that we omit hard-wired values $(\Sigma_f, \Sigma_t, \mathsf{pk}_f, \mathsf{pk}_t)$ from $\mathcal{J}$ for notational simplicity.

Lemma 4.9. *If* $\mathsf{UPRE} = (\mathsf{ReKeyGen}, \mathsf{ReEnc})$ *satisfies re-encryption key simulatability for UPRE, then* $\mathsf{Adv}^{\mathcal{J}}(\lambda, \mathsf{z}) \leq \mathsf{negl}(\lambda)$.

Lemma 4.10. *The advantage* $\frac{1}{2}|\mathsf{Adv}^{\mathsf{C_{re}}}(\lambda, \mathsf{z}) - \mathsf{Adv}^{\mathcal{J}}(\lambda, \mathsf{z})|$ *is bounded by the IND-CPA security advantage of* Σ_t, $\mathsf{Adv}^{\mathsf{pke}}_{\mathcal{A},\Sigma_t}(\lambda)$. *That is,*

$$\frac{1}{2}|\mathsf{Adv}^{\mathsf{C_{re}}}(\lambda, \mathsf{z}) - \mathsf{Adv}^{\mathcal{J}}(\lambda, \mathsf{z})| \leq \mathsf{Adv}^{\mathsf{pke}}_{\mathcal{A},\Sigma_t}(\lambda).$$

By the IND-CPA security of Σ_t, $\mathsf{Adv}^{\mathsf{pke}}_{\mathcal{A},\Sigma_t}(\lambda)$ is negligible. Therefore, if we complete the proofs of Lemmata 4.9 and 4.10, then we complete the proof of Lemma 4.8 since those imply $\mathsf{Adv}^{\mathsf{C_{re}}}(\lambda, \mathsf{z})$ is negligible. ■

Proof of Lemma 4.9. Now, we prove that if there exists a distinguisher $\mathcal{D}^{\mathcal{J}}$ that distinguishes $(\mathcal{A}(\mathcal{O}(\mathsf{C_{re}}), \mathsf{z}), \mathsf{z})$ from $(\mathcal{S}(1^\lambda, \mathsf{z}), \mathsf{z})$, then there exists a distinguisher $\mathcal{D}_{\mathsf{UPRE}}$ that distinguishes $\mathsf{rk}_{f \to t} \leftarrow \mathsf{ReKeyGen}(\Sigma_f, \Sigma_t, \mathsf{sk}_f, \mathsf{pk}_t)$ from $\widetilde{\mathsf{rk}}_{f \to t} \leftarrow \mathsf{ReKeySim}(1^\lambda, \Sigma_{\sigma_f}, \Sigma_{\sigma_t}, \mathsf{pk}_f, \mathsf{pk}_t)$. We construct the distinguisher $\mathcal{D}_{\mathsf{UPRE}}$ by using $\mathcal{D}$ as follows.

$\underline{\mathcal{D}_{\mathsf{UPRE}}(\Sigma_f, \Sigma_t, \mathsf{pk}_f, \mathsf{pk}_t, \mathsf{rk}^*_{f \to t})}$:

- It receives $(\Sigma_f, \Sigma_t, \mathsf{pk}_f, \mathsf{pk}_t, \mathsf{rk}^*_{f \to t})$ as inputs from the challenger.
- It constructs a re-encryption circuit $\mathcal{RE}[\Sigma_f, \Sigma_t, \mathsf{pk}_f, \mathsf{pk}_t, \mathsf{rk}^*_{f \to t}]$.
- It simulate the dummy oracle $\mathcal{J}$ for $\mathcal{D}$ as follows.
  - If $\mathcal{D}$ sends $\mathtt{keys}$, then $\mathcal{D}_{\mathsf{UPRE}}$ returns $(\Sigma_f, \Sigma_t, \mathsf{pk}_f, \mathsf{pk}_t)$.
  - If $\mathcal{D}$ sends ct_f, then $\mathcal{D}_{\mathsf{UPRE}}$ computes and returns $\mathsf{ct}'_t \leftarrow \mathsf{Enc}_t(\mathsf{pk}_t, 0^\ell)$.
- It outputs the output of $\mathcal{D}^{\mathcal{J}}$.

Apparently, $\mathcal{D}_{\mathsf{UPRE}}$ perfectly simulates $\mathcal{J}$. If $\mathsf{rk}^*_{f \to t}$ is generated by $\mathsf{ReKeyGen}(1^\lambda, \Sigma_{\sigma_f}, \Sigma_{\sigma_t}, \mathsf{sk}_f, \mathsf{pk}_t)$, then $\mathcal{D}_{\mathsf{UPRE}}$ perfectly simulates $\mathcal{O}(\mathsf{C_{re}})$. If $\mathsf{rk}^*_{f \to t}$ is generated by $\mathsf{ReKeySim}(1^\lambda, \Sigma_{\sigma_f}, \Sigma_{\sigma_t}, \mathsf{pk}_f, \mathsf{pk}_t)$, then then $\mathcal{D}_{\mathsf{UPRE}}$ perfectly simulates $\mathcal{S}(1^\lambda, \mathsf{z})$. Thus, if $\mathcal{D}^{\mathcal{J}}$ can distinguish, then $\mathcal{D}_{\mathsf{UPRE}}$ can break re-encryption key simulatability. This completes the proof. ■

Proof of Lemma 4.10. We construct an adversary $\mathcal{B}$ that breaks IND-CPA security of Σ_t by using a distinguisher $\mathcal{D}$ for $\mathsf{Adv}^{\mathsf{C}_{\mathsf{re}}}$ or $\mathsf{Adv}^{\mathcal{J}}$. $\mathcal{B}$ takes as inputs, a public key pk_t and an auxiliary input z, and proceeds as follows.

$\underline{\mathcal{B}(\mathsf{pk}_t, \mathsf{z})}$:

- generates $(\mathsf{pk}_f, \mathsf{sk}_f) \leftarrow \mathsf{Gen}_f(1^\lambda)$.
- generates $\mathsf{rk}_{f\to t} \leftarrow \mathsf{ReKeyGen}(1^\lambda, \Sigma_{\sigma_f}, \Sigma_{\sigma_t}, \mathsf{sk}_f, \mathsf{pk}_t)$ and $\widetilde{\mathsf{rk}}_{f\to t} \leftarrow \mathsf{ReKeySim}(1^\lambda, \Sigma_{\sigma_f}, \Sigma_{\sigma_t}, \mathsf{pk}_f, \mathsf{pk}_t)$.
- constructs $\overline{\mathcal{RE}} := \mathcal{RE}[\Sigma_{\sigma_f}, \Sigma_{\sigma_t}, \mathsf{pk}_f, \mathsf{pk}_t, \mathsf{rk}_{f\to t}]$ and $\widetilde{\mathcal{RE}} := \mathcal{RE}[\Sigma_{\sigma_f}, \Sigma_{\sigma_t}, \mathsf{pk}_f, \mathsf{pk}_t, \widetilde{\mathsf{rk}}_{f\to t}]$.
- chooses $b \leftarrow \{0,1\}$ that indicates which of $\mathsf{rk}_{f\to t}$ and $\widetilde{\mathsf{rk}}_{f\to t}$ is used to run $\mathcal{D}$.
- If $b = 1$, then $\mathcal{B}$ runs $\mathcal{D}^{\mathcal{O}}(\Sigma_{\sigma_f}, \Sigma_{\sigma_t}, \mathsf{pk}_f, \mathsf{pk}_t, \overline{\mathcal{RE}}, \mathsf{z})$ where $\mathcal{O}$ is defined below.
- If $b = 0$, then $\mathcal{B}$ runs $\mathcal{D}^{\mathcal{O}}(\Sigma_{\sigma_f}, \Sigma_{\sigma_t}, \mathsf{pk}_f, \mathsf{pk}_t, \widetilde{\mathcal{RE}}, \mathsf{z})$.

For notational convenience, we let $\mathsf{Real} := (\Sigma_{\sigma_f}, \Sigma_{\sigma_t}, \mathsf{pk}_f, \mathsf{pk}_t, \overline{\mathcal{RE}}, \mathsf{z})$ and $\mathsf{Sim} := (\Sigma_{\sigma_f}, \Sigma_{\sigma_t}, \mathsf{pk}_f, \mathsf{pk}_t, \widetilde{\mathcal{RE}}, \mathsf{z})$.

$\underline{\text{Simulation of } \mathcal{O} \text{ by } \mathcal{B}}$:

- If $\mathcal{D}$ sends $\mathtt{keys}$, then $\mathcal{B}$ returns $(\Sigma_{\sigma_f}, \Sigma_{\sigma_t}, \mathsf{pk}_f, \mathsf{pk}_t)$.
- If $\mathcal{D}$ sends $x \notin \mathcal{C}_f$, then $\mathcal{B}$ outputs $\perp$.
- If $\mathcal{D}$ sends $\mathsf{ct}_f \in \mathcal{C}_f$, then $\mathcal{B}$ does the following.
  1. computes $m \leftarrow \mathsf{Dec}_f(\mathsf{sk}_f, \mathsf{ct}_f)$.
  2. sends $(m, 0^\ell)$ as the target message pair to the challenger of IND-CPA security of Σ_t.
  3. receives ct_t^* from the challenger and returns ct_t^* to $\mathcal{D}$.

  At some point, $\mathcal{D}$ outputs $b' \in \{0,1\}$. If $b' = b$, then $\mathcal{B}$ outputs 1 (this means ct_t^* is an encryption of m), outputs 0 otherwise (this means ct_t^* is an encryption of 0^ℓ).

Apparently, if ct_t^* is an output of the left oracle of IND-CPA (i.e., encryption of m), then $\mathcal{B}$ perfectly simulates C_{re}. If ct_t^* is an output of the right oracle of IND-CPA (i.e., encryption of 0^ℓ), then $\mathcal{B}$ perfectly simulates $\mathcal{J}$.

If ct_t^* is an encryption m, then $\Pr[\mathcal{B}(\mathsf{pk}_t, \mathsf{z}) = 1] = \frac{1}{2} + \frac{1}{2}\mathsf{Adv}^{\mathsf{C}_{\mathsf{re}}}(\lambda, \mathsf{z})$. This is because

$$\begin{aligned}
\Pr[\mathcal{B}(\mathsf{pk}_t, \mathsf{z}) = 1 \mid \mathsf{ct}_t^* \leftarrow \mathsf{Enc}_t(\mathsf{pk}_t, m)] &= \frac{1}{2}(\Pr[\mathcal{D}^{\mathsf{C}_{\mathsf{re}}}(\mathsf{Real}) = 1] + \Pr[\mathcal{D}^{\mathsf{C}_{\mathsf{re}}}(\mathsf{Sim}) = 0]) \\
&= \frac{1}{2}(\Pr[\mathcal{D}^{\mathsf{C}_{\mathsf{re}}}(\mathsf{Real}) = 1] + 1 - \Pr[\mathcal{D}^{\mathsf{C}_{\mathsf{re}}}(\mathsf{Sim}) = 1]) \\
&= \frac{1}{2} + \frac{1}{2}(\Pr[\mathcal{D}^{\mathsf{C}_{\mathsf{re}}}(\mathsf{Real}) = 1] - \Pr[\mathcal{D}^{\mathsf{C}_{\mathsf{re}}}(\mathsf{Sim}) = 1]) \\
&= \frac{1}{2} + \frac{1}{2}\mathsf{Adv}^{\mathsf{C}_{\mathsf{re}}}(\lambda, \mathsf{z}).
\end{aligned}$$

If ct_t^* is an encryption 0^ℓ, then $\Pr[\mathcal{B}(\mathsf{pk}_t, \mathsf{z}) = 1] = \frac{1}{2} + \frac{1}{2}\mathsf{Adv}^{\mathcal{J}}(\lambda, \mathsf{z})$. This is because

$$\begin{aligned}
\Pr[\mathcal{B}(\mathsf{pk}_t, \mathsf{z}) = 1 \mid \mathsf{ct}_t^* \leftarrow \mathsf{Enc}_t(\mathsf{pk}_t, 0^\ell)] &= \frac{1}{2}(\Pr[\mathcal{D}^{\mathcal{J}}(\mathsf{Real}) = 1] + \Pr[\mathcal{D}^{\mathcal{J}}(\mathsf{Sim}) = 0]) \\
&= \frac{1}{2}(\Pr[\mathcal{D}^{\mathcal{J}}(\mathsf{Real}) = 1] + 1 - \Pr[\mathcal{D}^{\mathcal{J}}(\mathsf{Sim}) = 1]) \\
&= \frac{1}{2} + \frac{1}{2}(\Pr[\mathcal{D}^{\mathcal{J}}(\mathsf{Real}) = 1] - \Pr[\mathcal{D}^{\mathcal{J}}(\mathsf{Sim}) = 1]) \\
&= \frac{1}{2} + \frac{1}{2}\mathsf{Adv}^{\mathcal{J}}(\lambda, \mathsf{z}).
\end{aligned}$$

Thus,

$$\begin{aligned}\mathsf{Adv}^{\mathsf{pke}}_{\mathcal{A},\Sigma_t}(\lambda) &= |\Pr[\mathcal{B}(\mathsf{pk}_t, \mathsf{z}) = 1 \mid \mathsf{ct}^*_t \leftarrow \mathsf{Enc}_t(\mathsf{pk}_t, m)] - \Pr[\mathcal{B}(\mathsf{pk}_t, \mathsf{z}) = 1 \mid \mathsf{ct}^*_t \leftarrow \mathsf{Enc}_t(\mathsf{pk}_t, 0^{\ell})]| \\ &= \frac{1}{2}|\mathsf{Adv}^{\mathsf{C}_{\mathsf{re}}}(\lambda, \mathsf{z}) - \mathsf{Adv}^{\mathcal{J}}(\lambda, \mathsf{z})|.\end{aligned}$$

This complete the proof. ■

By Lemmata 4.9 and 4.10, we complete the proof of Lemma 4.8. Therefore, from Lemmata 4.7 and 4.8, we obtain Theorem 4.3.

5 Multi-Hop Construction based on Indistinguishability Obfuscation

In this section, we present a UPRE scheme for PKE based on PIO as the first step. To prove the security of our UPRE scheme by using sub-exponentially secure IO, we need to assume that PKE schemes are (0-hiding) trapdoor encryption (explained in Section 5.1). Several well-known CPA-secure PKE schemes could be transformed into (0-hiding) trapdoor encryption [ElG85, Pai99, GM84, DJ01]. If we use a stronger obfuscation, called dynamic-input PIO for randomized circuits [CLTV15], then we can use any standard CPA-secure PKE scheme. There is a possibility to construct dynamic-input PIO for specific dynamic-input indistinguishable samplers [CLTV15].

We can describe our UPRE scheme based on PIO in a unified way by the language of trapdoor encryption as Canetti et al. did [CLTV15].

5.1 Trapdoor Encryption

Before we proceed to present our UPRE scheme and prove the security, we present the notion of trapdoor encryption.

Definition 5.1 (Trapdoor Encryption [CLTV15]). *We say that* $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{TrapGen})$ *with message space* $\mathcal{M}$ *is a trapdoor encryption scheme if* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *with message space* $\mathcal{M}$ *is a CPA-secure PKE scheme and the trapdoor key generation algorithm* $\mathsf{TrapGen}$ *satisfies the following.*

Trapdoor Public Key Indistinguishability: *It holds that*

$$\left\{\mathsf{pk} \mid (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)\right\}_{\lambda \in \mathbb{N}} \overset{\mathsf{c}}{\approx} \left\{\mathsf{tpk} \mid \mathsf{tpk} \leftarrow \mathsf{TrapGen}(1^\lambda)\right\}_{\lambda \in \mathbb{N}}.$$

Computational/Statistical/δ-Hiding: *We define ensembles of random variables,* $\mathsf{view}_{\mathsf{tpke}}(b)$*, which are all view from an adversary* $\mathcal{A}$ *during experiments between* $\mathcal{A}$ *and challenger defined as follows.*

1. *The challenger runs* $\mathsf{tpk} \leftarrow \mathsf{TrapGen}(1^\lambda)$ *and gives* tpk *to* $\mathcal{A}$.
2. $\mathcal{A}$ *sends two messages* $m_0, m_1 \in \mathcal{M}$ *as the challenge messages to the challenger.*
3. *The challenger generates ciphertext* $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{tpk}, m_b)$ *and sends* ct^* *to* $\mathcal{A}$.
4. *We set* $\mathsf{view}_{\mathsf{tpke}}(b) := (\mathsf{tpk}, m_0, m_1, \mathsf{ct}^*)$.

We say Σ *is computational/statistical/*δ*-hiding if, for any PPT/unbounded/PPT adversary* $\mathcal{A}$*, it holds that*

$$\mathsf{view}_{\mathsf{tpke}}(0) \overset{\mathsf{x}}{\approx} \mathsf{view}_{\mathsf{tpke}}(1),$$

where $\overset{\mathsf{x}}{\approx}$ *is* $\overset{\mathsf{c}}{\approx}$ / $\overset{\mathsf{s}}{\approx}$ / $\overset{\mathsf{c}}{\approx}_\delta$*, respectively.*

In particular, 0-hiding is important for our constructions. It is easy to see that a standard CPA-secure PKE is trapdoor encryption with computational-hiding [CLTV15].

Theorem 5.2 ([CLTV15]). *Any IND-CPA secure PKE schemes are computational-hiding trapdoor encryption.*

Definition 5.3 (δ-Rerandomizable Encryption [CLTV15]). *We say that* $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{reRand})$ *is a* δ*-rerandomizable encryption scheme if* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is a CPA-secure PKE scheme and the additional algorithm* reRand *satisfies the following.*

δ-Rerandomizability: *We define the following experiments $\mathsf{Expt}^{\mathsf{rerand}}_{\mathcal{A}}(1^\lambda, b)$ between a challenger and an adversary $\mathcal{A}$ as follows.*

1. *The challenger chooses a bit $b \leftarrow \{0,1\}$, generates $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$, and sends pk to $\mathcal{A}$.*
2. *$\mathcal{A}$ sends $m \in \mathcal{M}$ where $\mathcal{M}$ is the message space of Σ to the challenger.*
3. *The challenger generates $\mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$ and $\mathsf{ct}_1 \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$.*
4. *If $b = 0$, the challenger computes $\widehat{\mathsf{ct}} \leftarrow \mathsf{reRand}(\mathsf{pk}, c_0)$. Otherwise, the challenger computes $\widehat{\mathsf{ct}} \leftarrow \mathsf{reRand}(\mathsf{pk}, c_1)$.*
5. *The challenger returns $(\mathsf{ct}_0, \mathsf{ct}_1, \widehat{\mathsf{ct}})$ to $\mathcal{A}$.*
6. *$\mathcal{A}$ outputs a guess $b' \in \{0,1\}$. The experiment outputs b'.*

We say that FSS is δ-rerandomizable if for any PPT $\mathcal{A}$, it holds that

$$|\Pr[\mathsf{Expt}^{\mathsf{rerand}}_{\mathcal{A}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Expt}^{\mathsf{rerand}}_{\mathcal{A}}(1^\lambda, 1) = 1]| \leq \delta(\lambda).$$

Re-randomizable encryption can be transformed into trapdoor encryption [CLTV15]. This transformation only changes the format of public-keys. *It does not change the format of ciphertexts at all*. Therefore, decryption procedure in the transformed scheme is completely the same as the original one. This is important for construction of UPRE based on PIO since we would like to use a PKE scheme as it is. We review the theorem and construction by Canetti et al. [CLTV15].

Theorem 5.4 ([CLTV15]). *If there exists δ-rerandomizable encryption, then $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{TrapGen})$ described below is δ-hiding trapdoor encryption scheme whose message space is $\{0,1\}$.*

Let $\Sigma' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}', \mathsf{reRand})$ be a δ-rerandomizable encryption scheme.

$\mathsf{Gen}(1^\lambda)$**:** generates $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{Gen}'(1^\lambda)$ and $\mathsf{ct}_b \leftarrow \mathsf{Enc}'(\mathsf{pk}', b)$ for $b = 0, 1$ and outputs $(\mathsf{pk}, \mathsf{sk}) := ((\mathsf{pk}', \mathsf{ct}_0, \mathsf{ct}_1), \mathsf{sk}')$.

$\mathsf{Enc}(\mathsf{pk}, b)$**:** parses $\mathsf{pk} = (\mathsf{pk}', \mathsf{ct}_0, \mathsf{ct}_1)$ and outputs $\mathsf{ct} \leftarrow \mathsf{reRand}(\mathsf{pk}', \mathsf{ct}_b)$.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$**:** outputs $b' \leftarrow \mathsf{Dec}'(\mathsf{sk}', \mathsf{ct})$.

$\mathsf{TrapGen}(1^\lambda)$**:** generates $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{Gen}'(1^\lambda)$ and $\mathsf{ct}_b \leftarrow \mathsf{Enc}'(\mathsf{pk}', 0)$ for $b = 0, 1$ and outputs $\mathsf{tpk} := (\mathsf{pk}', \mathsf{ct}_0, \mathsf{ct}_1)$.

Theorem 5.5 ([CLTV15]). *Goldwasser-Micali [GM84], ElGamal [ElG85], Paillier [Pai99], and Damgård-Jurik PKE [DJ01] schemes can be transformed into 0-hiding trapdoor encryption schemes by the transformation described in Theorem 5.4 in Section 5.1.*

5.2 Our Multi-Hop Scheme from PIO

Now, we present our UPRE scheme based on PIO. In fact, the scheme is a modification of fully homomorphic encryption scheme from PIO and trapdoor encryption by Canetti et al. [CLTV15]. The scheme is simple and easy to understand. Hereafter, we overload the notation $\Sigma_{\sigma_i} = (\mathsf{Gen}_{\sigma_i}, \mathsf{Enc}_{\sigma_i}, \mathsf{Dec}_{\sigma_i})$ by $\Sigma_i = (\mathsf{Gen}_i, \mathsf{Enc}_i, \mathsf{Dec}_i)$ for ease of notation as in Section 4. Our scheme $\mathsf{UPRE}_{\mathsf{pio}}$ is as follows.

- $\mathsf{ReKeyGen}(\Sigma_f, \Sigma_t, \mathsf{sk}_f, \mathsf{pk}_t)$:
  - Define a probabilistic circuit $\mathsf{C}^{\mathsf{pio}}_{\mathsf{re}}$ described in Figure 4.
  - Output $\mathsf{rk}_{f \to t} := pi\mathcal{O}(\mathsf{C}^{\mathsf{pio}}_{\mathsf{re}})$.
- $\mathsf{ReEnc}(\Sigma_f, \Sigma_t, \mathsf{rk}_{f \to t}, \mathsf{ct}_f)$:
  - Parse $\mathsf{rk}_{f \to t} = pi\mathcal{O}(\mathsf{C}^{\mathsf{pio}}_{\mathsf{re}})$.
  - Output $\mathsf{rct} := pi\mathcal{O}(\mathsf{C}^{\mathsf{pio}}_{\mathsf{re}})(\mathsf{ct}_f)$.

Re-Encryption Function $\mathsf{C}_{\mathsf{re}}^{\mathsf{pio}}[\Sigma_f, \Sigma_t, \mathsf{sk}_f, \mathsf{pk}_t](\mathsf{ct}_f)$

Hardwired: Σ_f, Σ_t, sk_f, pk_t.
Input: A ciphertext $\mathsf{ct}_f \in \mathcal{C}_f$.
Padding: This circuit is padded to size $\mathsf{pad}_\mathsf{T} := \mathsf{pad}_\mathsf{T}(\lambda, \lambda_f, \lambda_t)$, which is determined in analysis (we may omit λ_f and λ_t).

1. Compute $m \leftarrow \mathsf{Dec}_f(\mathsf{sk}_f, \mathsf{ct}_f)$.
2. Generate and return $\mathsf{ct}_t \leftarrow \mathsf{Enc}_t(\mathsf{pk}_t, m)$.

Figure 4: The description of $\mathsf{C}_{\mathsf{re}}^{\mathsf{pio}}$

Dummy Re-Encryption Function $\mathsf{dC}_{\mathsf{re}}^{\mathsf{pio}}[\Sigma_t, \mathsf{tpk}_t](\mathsf{ct}_f)$

Hardwired: Σ_t, tpk_t.
Input: A ciphertext $\mathsf{ct}_f \in \mathcal{C}_f$.
Padding: This circuit is padded to size $\mathsf{pad}_\mathsf{T} := \mathsf{pad}_\mathsf{T}(\lambda, \lambda_f, \lambda_t)$, which is determined in analysis (we may omit λ_f and λ_t).

1. Generate and return $\mathsf{ct}_t \leftarrow \mathsf{Enc}_t(\mathsf{tpk}_t, 0^{\ell_t})$.

Figure 5: The description of $\mathsf{dC}_{\mathsf{re}}^{\mathsf{pio}}$

Correctness. From the definition of $\mathsf{C}_{\mathsf{re}}^{\mathsf{pio}}$, for $\mathsf{ct}_f \leftarrow \mathsf{Enc}_f(\mathsf{pk}_f, m)$, it holds that $\mathsf{C}_{\mathsf{re}}^{\mathsf{pio}}(\mathsf{ct}_f) = \mathsf{Enc}_t(\mathsf{pk}_t, m) = \mathsf{ct}_t$. From the alternative correctness of $pi\mathcal{O}$ (See Definition 2.9) and the correctness of Σ_f, it holds that

$$\mathsf{rct} = pi\mathcal{O}(\mathsf{C}_{\mathsf{re}}^{\mathsf{pio}})(\mathsf{ct}_f) = \mathsf{ct}_t.$$

Therefore, it holds that

$$\mathsf{Dec}(\Sigma_j, \mathsf{sk}_j, \mathsf{ReEnc}(\Sigma'_j, \mathsf{ReKeyGen}(\Sigma'_j, \mathsf{sk}_{j-1}, \mathsf{pk}_j), \mathsf{rct}_{j-1})) = \mathsf{Dec}(\Sigma_{j-1}, \mathsf{sk}_{j-1}, \mathsf{rct}_{j-1}),$$

where $\Sigma'_j = (\Sigma_{j-1}, \Sigma_j)$ and the correctness holds. Note that a re-encrypted ciphertext under delegatee public-key pk_j is exactly in $\mathcal{C}_j$.

Padding Parameter. To use PIO, we need pad the size of circuits to be obfuscated. We set $\mathsf{pad}_\mathsf{T}(\lambda, \lambda_f, \lambda_t) := \max\{|\mathsf{C}_{\mathsf{re}}^{\mathsf{pio}}|, |\mathsf{dC}_{\mathsf{re}}^{\mathsf{pio}}|\}$, which is polynomial of $(\lambda, \lambda_f, \lambda_t)$ since an input of $\mathsf{C}_{\mathsf{re}}^{\mathsf{pio}}, \mathsf{dC}_{\mathsf{re}}^{\mathsf{pio}}$ is ciphertext under pk_f generated by $\mathsf{Gen}_f(1^{\lambda_f})$ and all hard wired values are keys of Σ_f, Σ_t. We can think λ_f and λ_t are polynomials in λ, so we may omit λ_f and λ_t hereafter.

5.3 Security Proof

Theorem 5.6 (UPRE-HRA security). *If there exists PIO for the class of sampler $\mathcal{S}^{\Sigma_i,\Sigma_j}$ defined below and both Σ_f and Σ_t are trapdoor encryption schemes, then $\mathsf{UPRE}_{\mathsf{pio}}$ is selectively UPRE-HRA secure. More specifically, if $pi\mathcal{O}$ is a PIO for the class of dynamic-input sampler $\mathcal{S}^{\Sigma_i,\Sigma_j}$ and both Σ_f and Σ_t are IND-CPA secure PKE schemes, or if $pi\mathcal{O}$ is a PIO for the class of X-IND sampler $\mathcal{S}^{\Sigma_i,\Sigma_j}$ and both Σ_f and Σ_t are 0-hiding trapdoor encryption schemes, then $\mathsf{UPRE}_{\mathsf{pio}}$ is selectively UPRE-HRA secure.*

Before we proceed to prove Theorem 5.6, we define the class of sampler $\mathcal{S}^{\Sigma_i,\Sigma_j}$ defined by trapdoor encryption schemes Σ_i, Σ_j as follows.

Sampler $\mathsf{Samp}^{\mathsf{SK}}$: The distribution $\mathsf{Samp}^{\mathsf{SK}}$ samples a trapdoor public key $\mathsf{tpk}_j \leftarrow \mathsf{TrapGen}_j(1^{\lambda_j})$ and outputs circuits $C_0 := \mathsf{C}_{\mathsf{re}}^{\mathsf{pio}}[\Sigma_i, \Sigma_j, \mathsf{sk}_i, \mathsf{tpk}_j]$ and $C_1 := \mathsf{dC}_{\mathsf{re}}^{\mathsf{pio}}[\Sigma_j, \mathsf{tpk}_j]$, and $z := \mathsf{tpk}_j$, where $\mathsf{SK} = \{\mathsf{sk}_{\lambda_i}\}$ is a sequence of strings of length $\rho_i(\lambda_i)$ and $\mathsf{sk} := \mathsf{sk}_{\lambda_i}$.

Class $\mathcal{S}^{\Sigma_i,\Sigma_j}$: Let $\mathcal{S}^{\Sigma_f,\Sigma_t}$ be the class of samplers with distribution $\mathsf{Samp}^{\mathsf{SK}}$ for all sequence of strings SK of length $\rho_j(\lambda_j)$.

Now, we proceed to prove Theorem 5.6.

Proof. We define a sequence of hybrid experiments $\mathsf{Hyb}^{x}_{\mathcal{A}}(b)$. We emphasize differences among hybrid experiments by using red underlines. Hereafter, $\mathsf{Hyb}^{x}_{\mathcal{A}}(b) \approx \mathsf{Hyb}^{y}_{\mathcal{A}}(b)$ denotes $|\Pr[\mathsf{Hyb}^{x}_{\mathcal{A}}(b) = 1] - \Pr[\mathsf{Hyb}^{y}_{\mathcal{A}}(b) = 1]| \leq \mathsf{negl}(\lambda)$.

$\mathsf{Hyb}^{0}_{\mathcal{A}}(b)$**:** The first experiment is the original security experiment for b, $\mathsf{Exp}^{\mathsf{ms\text{-}upre\text{-}hra}}_{\mathcal{A}}(1^{\lambda}, b)$. That is, it holds that $\mathsf{Hyb}^{0}_{\mathcal{A}}(b) = \mathsf{Exp}^{\mathsf{ms\text{-}upre\text{-}hra}}_{\mathcal{A}}(1^{\lambda}, b)$. Note that in the successive experiments, we can easily simulate all keys outside of G^* since vertices in $V \setminus V^*$ are not connected to the target vertex and simulators can generate keys for them by itself.

$\mathsf{Hyb}^{0'}_{\mathcal{A}}(b)$**:** This experiment is the same as $\mathsf{Hyb}^{0}_{\mathcal{A}}(b)$ except that we guess the target vertex i^* that will be queried to challenge oracle $\mathcal{O}_{\mathsf{cha}}$ and abort if the guess is incorrect. The guess is correct with probability $1/|V^*|$, so $\Pr[\mathsf{Hyb}^{0'}_{\mathcal{A}}(b) = 1] = \frac{1}{|V^*|} \cdot \Pr[\mathsf{Hyb}^{0}_{\mathcal{A}}(b) = 1]$.

$\mathsf{Hyb}^{1}_{\mathcal{A}}(b)$**:** This experiment is the same as $\mathsf{Hyb}^{0'}_{\mathcal{A}}$ except that

1. we record not only $(\mathsf{rct}_i, \Sigma_i, i, \#\mathsf{CT})$ but also m in $\mathsf{KeyCTList}$ for encryption query (i, m) and
2. for re-encryption query (i', j, k) such that (i', j) is not an admissible edge with respect to $G = (V, E)$ and $k \notin \mathsf{Drv}$, the re-encrypted ciphertext is differently generated as follows. First, we retrieve $(\mathsf{rct}_{i'}, \Sigma_{\sigma_{i'}}, i', k, m)$ from $\mathsf{KeyCTList}$ (if there is not such an entry, just outputs $\bot$). Then, we compute $\mathsf{rct}_j \leftarrow \mathsf{Enc}_j(\mathsf{pk}_j, m)$ instead of $\mathsf{rk}_{i' \to j} \leftarrow \mathsf{ReKeyGen}(\Sigma_{i'}, \Sigma_j, \mathsf{sk}_{i'}, \mathsf{pk}_j)$ and $\mathsf{rct} \leftarrow \mathsf{ReEnc}(\Sigma_{i'}, \Sigma_j, \mathsf{rk}_{i' \to j}, \mathsf{rct}_{i'})$.

That is, we do not need $\mathsf{sk}_{i'}$ to generate rct_j. By the alternative correctness of $pi\mathcal{O}$ in Definition 2.9, this perfectly simulates $\mathsf{Hyb}^{0'}_{\mathcal{A}}(b)$ since an output of $\mathsf{C}^{\mathsf{pio}}_{\mathsf{re}}$ for input $\mathsf{rct}_{i'} = \mathsf{Enc}_{i'}(\mathsf{pk}_{i'}, m)$ is just a fresh ciphertext of m under pk_j.

Process for removing sk_{i^*} of the target vertex: Now, we focus on vertices in V^* connected via admissible edges. To use the security of Σ_{i^*}, we need remove information about sk_{i^*} from all re-encryption keys in $G^* = (V^*, E^*)$ possibly connected to i^*. For all (honest) vertex $j \in V^*$ that have incoming edge (i, j) such that $i \in V^*$ and do not have outgoing edge (j, j') for some j', we repeat the processes below for $v = 1, \ldots, Q$ where Q is the total number of admissible edges connected to target vertex i^*. We let Dlist be the list of vertices whose public key is replaced with a dummy key tpk. We initialize $\mathsf{Dlist} := \emptyset$ and maintain Dlist during the repeated processes below.

$\mathsf{Hyb}^{2,v}_{\mathcal{A}}(b)$**:** First, at this point, honest vertex j does not have any outgoing edge. This experiment is the same as $\mathsf{Hyb}^{3,(v-1)}_{\mathcal{A}}(b)$ except that for honest key generation query (j, Σ_j), the challenger generates $\mathsf{tpk}_j \leftarrow \mathsf{TrapGen}_j(1^{\lambda_j})$ and for all query (i, j) to $\mathcal{O}_{\mathsf{rekey}}$ such that $i \in V^*$, the challenger uses tpk_j to generate $\mathsf{rk}_{i \to j} = pi\mathcal{O}(\mathsf{C}^{\mathsf{pio}}_{\mathsf{re}}[\Sigma_i, \Sigma_j, \mathsf{sk}_i, \underline{\mathsf{tpk}_j}])$ instead of pk_j and $\mathsf{rk}_{i \to j} = pi\mathcal{O}(\mathsf{C}^{\mathsf{pio}}_{\mathsf{re}}[\Sigma_i, \Sigma_j, \mathsf{sk}_i, \underline{\mathsf{pk}_j}])$. We renew $\mathsf{Dlist} := \mathsf{Dlist} \cup \{j\}$. In Lemma 5.7, we prove that $\mathsf{Hyb}^{3,v-1}_{\mathcal{A}}(b) \overset{\mathsf{c}}{\approx} \mathsf{Hyb}^{2,v}_{\mathcal{A}}(b)$ holds due to the trapdoor public key indistinguishability property in Definition 5.1. Apparently, it holds that $\mathsf{Hyb}^{3,0}_{\mathcal{A}}(b) = \mathsf{Hyb}^{1}_{\mathcal{A}}(b)$.

$\mathsf{Hyb}^{3,v}_{\mathcal{A}}(b)$**:** This experiment is the same as $\mathsf{Hyb}^{2,v}_{\mathcal{A}}(b)$ except that for all query (i, j) to $\mathcal{O}_{\mathsf{rekey}}$ such that $i \in V^*$, the challenger generates uses $\mathsf{dC}^{\mathsf{pio}}_{\mathsf{re}}$ instead of $\mathsf{C}^{\mathsf{pio}}_{\mathsf{re}}$. That is, $\mathsf{rk}_{i \to j} = pi\mathcal{O}(\underline{\mathsf{dC}^{\mathsf{pio}}_{\mathsf{re}}[\Sigma_j, \mathsf{tpk}_j]})$ instead of $\mathsf{rk}_{i \to j} = pi\mathcal{O}(\underline{\mathsf{C}^{\mathsf{pio}}_{\mathsf{re}}[\Sigma_i, \Sigma_j, \mathsf{sk}_i, \mathsf{tpk}_j]})$. In Lemma 5.8, we prove that $\mathsf{Hyb}^{2,v}_{\mathcal{A}}(b) \overset{\mathsf{c}}{\approx} \mathsf{Hyb}^{3,v}_{\mathcal{A}}(b)$ holds due to the security of PIO with respect to $\mathcal{S}^{\Sigma_i, \Sigma_j}$. The sampler $\mathsf{Samp}^{\mathsf{SK}}$ generates the following distributions.

$$(\mathsf{C}_0 = \mathsf{hybC}_{\mathsf{re}}[\Sigma_i, \Sigma_j, \mathsf{sk}_i, \mathsf{tpk}_j], \mathsf{C}_1 = \mathsf{dC}^{\mathsf{pio}}_{\mathsf{re}}[\Sigma_j, \mathsf{tpk}_j], \mathsf{z} = \mathsf{tpk}_j) \leftarrow \mathsf{Samp}^{\mathsf{SK}}.$$

In $\mathsf{Hyb}^{3,Q}_{\mathcal{A}}(b)$, sk_{i^*} is neither written in any re-encryption key nor used to generate a re-encrypted ciphertext. Thus, we can use the security of Σ_{i^*}. In Lemma 5.9, we prove that $\mathsf{Hyb}^{3,Q}_{\mathcal{A}}(0) \overset{\mathsf{c}}{\approx} \mathsf{Hyb}^{3,Q}_{\mathcal{A}}(1)$ holds due to the CPA-security of Σ_{i^*}. Therefore, it holds that $\mathsf{Hyb}^{0}_{\mathcal{A}}(0) \overset{\mathsf{c}}{\approx} \mathsf{Hyb}^{0}_{\mathcal{A}}(1)$ by Lemmata 5.7 to 5.9 since Q and $|V^*|$ are polynomials. ■

Lemma 5.7. *If Σ_j is a trapdoor encryption scheme, then it holds $\mathsf{Hyb}^{3,v-1}_{\mathcal{A}}(b) \overset{\mathsf{c}}{\approx} \mathsf{Hyb}^{2,v}_{\mathcal{A}}(b)$.*

Proof. We construct $\mathcal{B}$ for the trapdoor public key indistinguishability property of Σ_j. First, $\mathcal{B}$ is given a target key ek_j ($\mathsf{ek}_j = \mathsf{pk}_j$ or $\mathsf{ek}_j = \mathsf{tpk}_j$). To use $\mathcal{A}$, $\mathcal{B}$ generates key pairs $(\mathsf{pk}_{i'}, \mathsf{sk}_{i'})$ for all $i' \in \mathsf{HList} \setminus (\mathsf{Dlist} \cup \{j\})$ and $i' \in \mathsf{CList}$. Note that $\mathsf{Dlist} \subseteq V^*$ by definition. For all $i' \in \mathsf{Dlist}$, $\mathcal{B}$ generates $\mathsf{tpk}_{i'} \leftarrow \mathsf{TrapGen}_{i'}(1^{\lambda_{i'}})$. For honest key generation query (j, Σ_j), $\mathcal{B}$ sets ek_j as the public-key of vertex j. For all queries (i,j) to $\mathcal{O}_{\mathsf{rekey}}$ such that $i \in V^*$, $\mathcal{B}$ generates $\mathsf{rk}_{i \to j} = pi\mathcal{O}(\mathsf{C}^{\mathsf{pio}}_{\mathsf{re}}[\Sigma_i, \Sigma_j, \mathsf{sk}_i, \mathsf{ek}_j])$. If $\mathsf{ek}_j = \mathsf{pk}_j$, then the view is totally the same as $\mathsf{Hyb}^{3,v-1}_{\mathcal{A}}(b)$. If $\mathsf{ek}_j = \mathsf{tpk}_j$, then the view is totally the same as $\mathsf{Hyb}^{2,v}_{\mathcal{A}}(b)$. Therefore, if $\mathcal{A}$ can distinguish two experiments, $\mathcal{B}$ can break the trapdoor key indistinguishability property of Σ_j. ■

Lemma 5.8. *If $pi\mathcal{O}$ is a PIO for the sampler $\mathcal{S}^{\Sigma_i,\Sigma_j}$, then it holds that* $\mathsf{Hyb}^{2,v}_{\mathcal{A}}(b) \overset{\mathsf{c}}{\approx} \mathsf{Hyb}^{3,v}_{\mathcal{A}}(b)$.

Proof. We construct a distinguisher $\mathcal{D}$ of PIO. To use $\mathcal{A}$ of UPRE, $\mathcal{D}$ generates key pairs $(\mathsf{pk}_{i'}, \mathsf{sk}_{i'})$ for all $i' \in \mathsf{HList} \setminus \mathsf{Dlist}$ and $i' \in \mathsf{CList}$. For all $i' \in \mathsf{Dlist}$, $\mathcal{D}$ generates $\mathsf{tpk}_{i'} \leftarrow \mathsf{TrapGen}_{i'}(1^{\lambda_{i'}})$. At this point, we do not need $\mathsf{sk}_{i'}$ for $i' \in \mathsf{Dlist}$. Therefore, $\mathcal{B}$ can simulate all oracles. However, to use $\mathcal{A}$, $\mathcal{B}$ simulates $\mathcal{O}_{\mathsf{rekey}}$ in a slightly different way. The simulation for query (i,j) such that (i,j) is an admissible but not deviating edge is different. When $\mathcal{B}$ receives a re-encryption key query for such (i,j), $\mathcal{B}$ uses the challenger of PIO. The challenger samples $(C_0, C_1, \mathsf{z}) \leftarrow \mathsf{Samp}^{\mathsf{sk}_i}$ where $C_0 = \mathsf{C}^{\mathsf{pio}}_{\mathsf{re}}[\Sigma_i, \Sigma_j, \mathsf{sk}_i, \mathsf{tpk}_j]$, $C_1 = \mathsf{dC}^{\mathsf{pio}}_{\mathsf{re}}[\Sigma_j, \mathsf{tpk}_j]$, and $\mathsf{z} = \mathsf{tpk}_j$ and generates $\widehat{C}$ (obfuscated circuit of C_0 or C_1). When $\mathcal{B}$ is given $\widehat{C}$ from the challenger of PIO, $\mathcal{B}$ sends $\mathsf{rk}_{i \to j} := \widehat{C}$ to $\mathcal{A}$. This completes the simulation. If $\mathcal{B}$ is given $\widehat{C} = pi\mathcal{O}(\mathsf{C}^{\mathsf{pio}}_{\mathsf{re}}[\Sigma_i, \Sigma_j, \mathsf{sk}_i, \mathsf{tpk}_j])$, then the view is totally the same as $\mathsf{Hyb}^{2,v}_{\mathcal{A}}(b)$. If $\mathcal{B}$ is given $\widehat{C} = pi\mathcal{O}(\mathsf{dC}^{\mathsf{pio}}_{\mathsf{re}}[\Sigma_j, \mathsf{tpk}_j])$, then the view is totally the same as $\mathsf{Hyb}^{3,v}_{\mathcal{A}}(b)$. Therefore, if $\mathcal{A}$ can distinguish two experiments, $\mathcal{B}$ can break the security of PIO for sampler $\mathsf{Samp}^{\mathsf{sk}_i}$. ■

Lemma 5.9. *If Σ_{i^*} is a trapdoor encryption scheme, then it holds* $\mathsf{Hyb}^{3,Q}_{\mathcal{A}}(0) \overset{\mathsf{c}}{\approx} \mathsf{Hyb}^{3,Q}_{\mathcal{A}}(1)$.

Proof. We construct $\mathcal{B}$ for (computational/statistical/δ-) hiding of Σ_{i^*}. First, $\mathcal{B}$ is given a target key pk_{i^*}. To use $\mathcal{A}$, $\mathcal{B}$ generates all keys except for vertex i^*. When (i^*, Σ_{i^*}) is queried as an uncorrupted key generation query, $\mathcal{B}$ sets pk_{i^*} as the public-key for user i^*. At this point, $\mathcal{B}$ does not need sk_{i^*} since it is neither written in any re-encryption key nor used in $\mathcal{O}_{\mathsf{reenc}}$. For the challenge query (i^*, m_0, m_1) to $\mathcal{O}_{\mathsf{cha}}$, $\mathcal{B}$ passes (m_0, m_1) to its challenger of Σ_{i^*}. When $\mathcal{B}$ receives $\mathsf{ct}^*_{i^*}$, then passes it to $\mathcal{A}$ as the target ciphertext. If $\mathsf{ct}^*_{i^*} \leftarrow \mathsf{Enc}_{i^*}(\mathsf{pk}_{i^*}, m_0)$, then the view is totally the same as $\mathsf{Hyb}^{3,Q}_{\mathcal{A}}(0)$. If $\mathsf{ct}^*_{i^*} \leftarrow \mathsf{Enc}_{i^*}(\mathsf{pk}_{i^*}, m_1)$, then the view is totally the same as $\mathsf{Hyb}^{3,Q}_{\mathcal{A}}(1)$. Therefore, if $\mathcal{A}$ can distinguish two experiments, $\mathcal{B}$ can break the hiding property of Σ_{i^*}. ■

5.4 Instantiation of UPRE scheme based on PIO

Instantiation by 0-hiding trapdoor encryption and IO. To instantiate by sub-exponentially secure IO and OWF, we should prove that $\mathcal{S}^{\Sigma_i,\Sigma_j}$ is a static-input X-indistinguishable sampler for $\mathcal{X} := \mathcal{C}_i$ if Σ_i and Σ_j are δ-hiding trapdoor encryption schemes.

We let $\gamma(\lambda) := \log |\mathcal{C}_i| := \log X(\mathsf{pad}_{\mathsf{T}}(\lambda))$ and set $\delta := \mathsf{negl}(\lambda) \cdot 2^{-\gamma(\lambda)}$. It is easy to see that X differing inputs holds since $\mathcal{X} = \mathcal{C}_i$ is the whole domain of circuits $C_0 = \mathsf{C}^{\mathsf{pio}}_{\mathsf{re}}[\Sigma_i, \Sigma_j, \mathsf{sk}_i, \mathsf{tpk}_j]$ and $C_1 = \mathsf{dC}^{\mathsf{pio}}_{\mathsf{re}}[\Sigma_j, \mathsf{tpk}_j]$. It is also easy to see that X-indistinguishability holds since outputs of $\mathsf{C}^{\mathsf{pio}}_{\mathsf{re}}[\Sigma_i, \Sigma_j, \mathsf{sk}_i, \mathsf{tpk}_j](\mathsf{ct}_i)$ and $\mathsf{dC}^{\mathsf{pio}}_{\mathsf{re}}[\Sigma_j, \mathsf{tpk}_j](\mathsf{ct}_i)$ are $\mathsf{Enc}_j(\mathsf{tpk}_j, m)$ and $\mathsf{Enc}_j(\mathsf{tpk}_j, 0^{\ell_j})$, respectively and these are $\mathsf{negl}(\lambda) \cdot 2^{-\gamma(\lambda)}$-indistinguishable due to the δ-hiding property. This means the outputs of C_0 and C_1 are $\mathsf{negl}(\lambda) \cdot X^{-1}$-indistinguishable since $X(\mathsf{pad}_{\mathsf{T}}(\lambda)) = 2^{\gamma(\lambda)}$. This parameter setting of δ is achievable by 0-hiding trapdoor encryption, which is instantiated by well-known IND-CPA PKE schemes such as ElGamal PKE (see Section 5.1). Note that as observed in Theorem 5.4, the message space of this instantiation is $\{0, 1\}$.

Corollary 5.10. *If there exists sub-exponentially secure IO and sub-exponentially secure OWF, then* $\mathsf{UPRE}_{\mathsf{pio}}$ *is a multi-hop selectively UPRE-HRA secure UPRE scheme for* 0*-hiding trapdoor encryption schemes.*

Instantiation by IND-CPA PKE and dynamic-input PIO. If we can use a dynamic-input PIO (see Definitions 2.9 and 2.13 for the definition), then we can set $\delta = \mathsf{negl}(\lambda)$ in the analysis above and it is achievable by standard IND-CPA PKE schemes (that is, trapdoor encryption with computational hiding). A dynamic-input PIO for the class of sampler $\mathcal{S}^{\Sigma_i,\Sigma_j}$ might exist (no impossibility result on PIO for a specific class) though it is not proved [CLTV15]. The PIO construction by Canetti et al. [CLTV15] is a candidate of dynamic-input PIO for the class of sampler $\mathcal{S}^{\Sigma_i,\Sigma_j}$.

Conjecture 5.11. A dynamic-input PIO for the class of sampler $\mathcal{S}^{\Sigma_i,\Sigma_j}$ exists.

Corollary 5.12. *If there exists dynamic-input PIO for the class of sampler $\mathcal{S}^{\Sigma_i,\Sigma_j}$ where Σ_i, Σ_j are IND-CPA PKE schemes, then $\mathsf{UPRE_{pio}}$ is a multi-hop selectively UPRE-HRA secure UPRE scheme for any IND-CPA PKE scheme.*

By using Theorem 4.3, we immediately obtain the following corollary.

Corollary 5.13. *If there exists dynamic-input PIO for the class of sampler $\mathcal{S}^{\Sigma_i,\Sigma_j}$ where Σ_i, Σ_j are IND-CPA PKE schemes, then there exists ACVBB obfuscator for all re-encryption circuits in the sense of Definition 4.2.*

Instantiation by IND-CPA PKE, doubly-probabilistic IO, and exponential DDH Agrikola, Couteau, and Hofheinz [ACH18] introduced the notion of doubly-probabilistic IO (DPIO), which is an extended notion of PIO. They prove that in most applications of PIO (fully homomorphic encryption, spooky encryption, function secret sharing etc.) we can replace PIO with DPIO. They also prove that we can achieve DPIO by using polynomially secure IO and exponential DDH assumption. It is easy to see that we can replace the PIO in $\mathsf{UPRE_{pio}}$ with DPIO.[8] Therefore, we can obtain the following theorem.

Theorem 5.14. *If there exists polynomially secure IO and exponential DDH assumption is true, then $\mathsf{UPRE_{pio}}$ is a multi-hop selectively UPRE-HRA secure UPRE scheme for IND-CPA encryption schemes.*

We can prove this theorem by combining the proof techniques of Theorem 5.6 and that of fully homomorphic encryption based on DPIO by Agrikola et al. [ACH18]. We omit the detail in this manuscript.

6 Multi-Hop Construction based on Garbled Circuits

In this section, we provide a UPRE scheme using garbled circuits. The main idea of the construction provided here is that the re-encryptor delegates decryption to the target node via garbled circuits. To achieve UPRE, we use weak batch encryption schemes, which are constructed from standard IND-CPA secure PKE schemes.

6.1 Weak Batch Encryption

Definition 6.1 (Weak Batch Encryption). *Let $\mathcal{M}$ be a message space. A weak batch encryption scheme is a tuple of algorithms $(\mathsf{BatchGen}, \mathsf{BatchEnc}, \mathsf{BatchDec})$ where*

- $\mathsf{BatchGen}(1^\lambda, s)$ *takes as input the security parameter and selection bits $s \in \{0,1\}^\lambda$, and outputs a pair $(\hat{\mathsf{pk}}, \hat{\mathsf{sk}})$ of public and secret keys.*
- $\mathsf{BatchEnc}(\hat{\mathsf{pk}}, \{(m_{i,0}, m_{i,1})\}_{i\in[\lambda]})$ *takes as input a public key $\hat{\mathsf{pk}}$ and λ-pairs of messages $\{(m_{i,0}, m_{i,1})\}_{i\in[\lambda]}$ where $m_{i,b} \in \mathcal{M}$, and outputs a ciphertext $\hat{\mathsf{ct}}$.*
- $\mathsf{BatchDec}(\hat{\mathsf{sk}}, \hat{\mathsf{ct}})$ *takes as input a secret key $\hat{\mathsf{sk}}$ and a ciphertext message $\hat{\mathsf{ct}}$, and outputs $\{m'_i\}_{i\in[\lambda]}$, or $\bot$.*

Correctness: *For any λ, $s \in \{0,1\}^\lambda$, $m_{i,b} \in \mathcal{M}$, we have that*

$$\Pr\left[\forall i\; m'_i = m_{i,s[i]} \;\middle|\; \begin{array}{rl} (\hat{\mathsf{pk}}, \hat{\mathsf{sk}}) & \leftarrow \mathsf{BatchGen}(1^\lambda, s), \\ \hat{\mathsf{ct}} & \leftarrow \mathsf{BatchEnc}(\hat{\mathsf{pk}}, \{(m_{i,0}, m_{i,1})\}_{i\in[\lambda]}), \\ \{m'_i\}_{i\in[\lambda]} & \leftarrow \mathsf{BatchDec}(\hat{\mathsf{sk}}, \hat{\mathsf{ct}})] = 1 \end{array}\right]$$

where $s[i]$ denotes i-th bit of s.

Receiver Privacy: *We require that public keys $\hat{\mathsf{pk}}$ are independent of the selection bits $s \in \{0,1\}^\lambda$ used to generate $\hat{\mathsf{pk}}$. That is, for all s_1, s_2 it holds that*

$$\hat{\mathsf{pk}}_1 \equiv \hat{\mathsf{pk}}_2$$

where $(\hat{\mathsf{pk}}_1, \hat{\mathsf{sk}}_1) \leftarrow \mathsf{BatchGen}(1^\lambda, s_1)$ and $(\hat{\mathsf{pk}}_2, \hat{\mathsf{sk}}_2) \leftarrow \mathsf{BatchGen}(1^\lambda, s_2)$ and $\equiv$ means the statistical distance is equal to 0.

[8] This is because the design idea of $\mathsf{UPRE_{pio}}$ is based on fully homomorphic encryption scheme based on PIO and Agrikola et al. achieve fully homomorphic encryption from DPIO.

Sender Privacy against Semi-Honest Receiver: *We define the experiment* $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{wbe\text{-}cpa}}(1^\lambda,\beta)$ *between an adversary* $\mathcal{A}$ *and challenger as follows.*

1. $\mathcal{A}$ *chooses* $s \in \{0,1\}^\lambda$ *and* $\{(m_{i,0}, m_{i,1})\}_{i\in[\lambda]}$ *and send them to the challenger.*
2. *The challenger computes* $(\hat{\mathsf{pk}}, \hat{\mathsf{sk}}) \leftarrow \mathsf{BatchGen}(1^\lambda, s)$ *and:*
   - *If* $\beta = 0$, *the challenger computes* $\hat{\mathsf{ct}}^* \leftarrow \mathsf{BatchEnc}(\hat{\mathsf{pk}}, \{(m_{i,0}, m_{i,1})\})$.
   - *Else if* $\beta = 1$, *the challenger computes* $\hat{\mathsf{ct}}^* \leftarrow \mathsf{BatchEnc}(\hat{\mathsf{pk}}, \left\{(m_{i,s[i]}, m_{i,s[i]})\right\})$.
3. *The challenger sends* $(\hat{\mathsf{pk}}, \hat{\mathsf{sk}}, \hat{\mathsf{ct}}^*)$ *to* $\mathcal{A}$.
4. $\mathcal{A}$ *outputs a guess* β' *for* β. *The experiment outputs* β'.

We say $(\mathsf{BatchGen}, \mathsf{BatchEnc}, \mathsf{BatchDec})$ *is WBE-CPA secure against semi-honest receiver if for any PPT adversary* $\mathcal{A}$, *it holds that*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{wbe\text{-}cpa}}(\lambda) := |\Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{wbe\text{-}cpa}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{wbe\text{-}cpa}}(1^\lambda, 1) = 1]| \leq \mathsf{negl}(\lambda).$$

We can consider a multi-challenge variant. That is, $\mathcal{A}$ *can send* $\{(m_{i,0}^{(j)}, m_{i,1}^{(j)})\}_{i\in[\lambda]}$ *and obtain many target ciphertexts after* $(\hat{\mathsf{pk}}, \hat{\mathsf{sk}})$ *is given for* $j = 1, \ldots, \mathrm{poly}(\lambda)$.

The difference between weak batch encryption and batch encryption proposed by Brakerski, Lombardi, Segev, and Vaikuntanathan [BLSV18] is that there is no efficiency requirement on the size of the batch public-key $\hat{\mathsf{pk}}$. Thus, it is easy to achieve weak batch encryption.

Theorem 6.2 (Weak Batch Encryption from IND-CPA PKE). *If there exists IND-CPA secure PKE, then there exists weak batch encryption.*

Proof. Let $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an IND-CPA secure PKE scheme.

$\mathsf{BatchGen}(1^\lambda, s)$**:** It generates $(\mathsf{pk}_{i,b}, \mathsf{sk}_{i,b}) \leftarrow \mathsf{Gen}(1^\lambda)$ for all $i \in [\lambda]$ and $b \in \{0,1\}$ and outputs $\hat{\mathsf{pk}} := \{\mathsf{pk}_{i,b}\}_{i\in[\lambda], b\in\{0,1\}}$ and $\hat{sk} := \left\{sk_{i,s[i]}\right\}_{i\in[\lambda]}$.

$\mathsf{BatchEnc}(\hat{\mathsf{pk}}, \{(m_{i,0}, m_{i,1})\}_{i\in[\lambda]})$**:** It generates $\mathsf{ct}_{i,b} \leftarrow \mathsf{Enc}(\mathsf{pk}_{i,b}, m_{i,b})$ for all $i \in [\lambda]$ and $b \in \{0,1\}$. It outputs $\hat{\mathsf{ct}} := \{\mathsf{ct}_{i,b}\}_{i\in[\lambda], b\in\{0,1\}}$.

$\mathsf{BatchDec}(\hat{\mathsf{sk}}, \hat{\mathsf{ct}})$**:** It parses $\hat{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_\lambda)$ and $\hat{\mathsf{ct}} = \{\mathsf{ct}_{i,b}\}_{i\in[\lambda], b\in\{0,1\}}$. It computes $m_i' \leftarrow \mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct}_{i,b})$ for $b \in \{0,1\}$ and sets $m_i := m_{i,b}'$ if $m_{i,b}' \neq \bot$. It outputs $\{m_i\}_{i\in[\lambda]}$.

The receiver privacy trivially holds since $\hat{\mathsf{pk}}$ does not include any information about s. The sender privacy follows from the IND-CPA security of Σ and the standard hybrid argument because $\{sk_{i,1-s[i]}\}_{i\in[\lambda]}$ are never used. Moreover, it is easy to see that the scheme satisfies the multi-challenge version by the standard hybrid argument. ■

6.2 Our Multi-Hop Scheme from GC

Our scheme $\mathsf{UPRE}_{\mathsf{gc}}$ is based on a garbling scheme $(\mathsf{Garble}, \mathsf{Eval})$, a weak batch-encryption scheme $(\mathsf{BatchGen}, \mathsf{BatchEnc}, \mathsf{BatchDec})$ and a 2-player secret-sharing scheme $(\mathsf{Share}, \mathsf{Reconstruct})$. As in Section 5, we overload the notation $\Sigma_{\sigma_i} = (\mathsf{Gen}_{\sigma_i}, \mathsf{Enc}_{\sigma_i}, \mathsf{Dec}_{\sigma_i})$ by $\Sigma_i = (\mathsf{Gen}_i, \mathsf{Enc}_i, \mathsf{Dec}_i)$ for ease of notation. Moreover, we sometimes write labels instead of $\{\mathsf{labels}_{k,b}\}_{k\in[n], b\in\{0,1\}}$ if it is clear from the context for ease of notation. We also denote by labels_s labels selected by s, that is, $\{\mathsf{labels}_{i,s_i}\}_{i\in[\lambda]}$. Moreover, $\widetilde{\mathsf{labels}}$ basically denotes selected labels output by $\mathsf{BatchDec}$.

- $\mathsf{ReKeyGen}(1^\lambda, \Sigma_f, \Sigma_t, \mathsf{sk}_f, \mathsf{pk}_t)$:
  - Compute $(s_1, s_2) \leftarrow \mathsf{Share}(\mathsf{sk}_f)$
  - $(\hat{\mathsf{pk}}, \hat{\mathsf{sk}}) \leftarrow \mathsf{BatchGen}(1^\lambda, s_1)$
  - Compute $\widetilde{\mathsf{ct}}_t \leftarrow \mathsf{Enc}_t(\mathsf{pk}_t, \hat{\mathsf{sk}})$

  - Output $\mathsf{rk}_{f\to t} := (\hat{\mathsf{pk}}, s_2, \tilde{\mathsf{ct}}_t)$.

- $\mathsf{ReEnc}(\Sigma_f, \Sigma_t, \mathsf{rk}_{f\to t}, \mathsf{ct}_f)$:
  - Parse $\mathsf{rk}_{f\to t} = (\hat{\mathsf{pk}}, s_2, \tilde{\mathsf{ct}}_t)$.
  - Parse $\mathsf{ct}_f = (\hat{\mathsf{ct}}', \tilde{\mathsf{ct}}_f, \tilde{\mathsf{C}}_{i-1}, \ldots, \tilde{\mathsf{C}}_1)$.
  - If $i = 1$ set $\mathsf{C} \leftarrow \mathsf{P}[s_2, \mathsf{ct}_f]$; Else if $i > 1$ set $\mathsf{C} \leftarrow \mathsf{Q}[s_2, \hat{\mathsf{ct}}', \tilde{\mathsf{ct}}_f]$
  - Compute $(\tilde{\mathsf{C}}_i, \mathsf{labels}) \leftarrow \mathsf{Garble}(\mathsf{C})$.
  - Compute $\hat{\mathsf{ct}} \leftarrow \mathsf{BatchEnc}(\hat{\mathsf{pk}}, \mathsf{labels})$
  - Output $(\hat{\mathsf{ct}}, \tilde{\mathsf{ct}}_t, \tilde{\mathsf{C}}_i, \ldots, \tilde{\mathsf{C}}_1)$

- $\mathsf{mDec}(\Sigma_t, \mathsf{sk}_t, \mathsf{rct}, i)$: Parse $\mathsf{rct} = (\hat{\mathsf{ct}}, \tilde{\mathsf{ct}}_t, \tilde{\mathsf{C}}_i, \ldots, \tilde{\mathsf{C}}_1)$.
  - Compute $\hat{\mathsf{sk}}' \leftarrow \mathsf{Dec}(\mathsf{sk}_t, \tilde{\mathsf{ct}}_t)$.
  - Compute $\widetilde{\mathsf{labels}}_i \leftarrow \mathsf{BatchDec}(\hat{\mathsf{sk}}', \hat{\mathsf{ct}})$
  - For $j = i, \ldots, 2$ do: Compute $\widetilde{\mathsf{labels}}_{j-1} \leftarrow \mathsf{Eval}(\tilde{\mathsf{C}}_j, \widetilde{\mathsf{labels}}_j)$.
  - Compute and output $m' \leftarrow \mathsf{Eval}(\tilde{\mathsf{C}}_1, \widetilde{\mathsf{labels}}_1)$.

First Level Re-Encryption Circuit $\mathsf{P}[s_2, \mathsf{ct}_f](s_1)$

Hardwired: s_2, ct_f.
Input: A share s_1.

- Compute $\mathsf{sk}'_f \leftarrow \mathsf{Reconstruct}(s_1, s_2)$.
- Compute and output $m' \leftarrow \mathsf{Dec}_f(\mathsf{sk}'_f, \mathsf{ct}_f)$.

Figure 6: The description of the first level re-encryption circuit P

Higher Level Re-Encryption Circuit $\mathsf{Q}[s_2, \hat{\mathsf{ct}}', \tilde{\mathsf{ct}}_f](s_1)$

Hardwired: s_2, $\hat{\mathsf{ct}}'$, $\tilde{\mathsf{ct}}_f$.
Input: A share s_1.

- Compute $\mathsf{sk}'_f \leftarrow \mathsf{Reconstruct}(s_1, s_2)$.
- Compute $\hat{\mathsf{sk}}' \leftarrow \mathsf{Dec}(\mathsf{sk}'_f, \tilde{\mathsf{ct}}_f)$.
- Compute and output $\widetilde{\mathsf{labels}} \leftarrow \mathsf{BatchDec}(\hat{\mathsf{sk}}', \hat{\mathsf{ct}}')$.

Figure 7: The description of the higher level re-encryption circuit Q

Correctness. We now turn to the correctness of $(\mathsf{ReKeyGen}, \mathsf{ReEnc}, \mathsf{mDec})$. We will show correctness via induction.

We will first show correctness for level 1 ciphertexts. Let thus $\mathsf{rct} = (\hat{\mathsf{ct}}, \tilde{\mathsf{ct}}_t, \tilde{\mathsf{C}}_1)$ be a level 1 ciphertext, where $(\tilde{\mathsf{C}}_1, \mathsf{labels}) \leftarrow \mathsf{Garble}(\mathsf{P}[s_2, \mathsf{ct}_f])$, $\hat{\mathsf{ct}} = \mathsf{BatchEnc}(\hat{\mathsf{pk}}, \mathsf{labels})$ and $\tilde{\mathsf{ct}}_t = \mathsf{Enc}_t(\mathsf{pk}_t, \hat{\mathsf{sk}})$. Consider the computation of $\mathsf{mDec}(\Sigma_t, \mathsf{sk}_t, \mathsf{rct})$. By the correctness of Σ_t it holds that $\hat{\mathsf{sk}}' = \mathsf{Dec}(\mathsf{sk}_t, \tilde{\mathsf{ct}}_t) = \hat{\mathsf{sk}}$. Next, by the correctness of the batch public key encryption $(\mathsf{BatchGen}, \mathsf{BatchEnc}, \mathsf{BatchDec})$ it holds that that $\widetilde{\mathsf{labels}} = \mathsf{BatchDec}(\hat{\mathsf{sk}}, \hat{\mathsf{ct}}) = \mathsf{labels}_{s_1}$. Thus, by the correctness of the garbling scheme $(\mathsf{Garble}, \mathsf{Eval})$ it holds that $\mathsf{Eval}(\tilde{\mathsf{C}}_1, \widetilde{\mathsf{labels}}) = \mathsf{Eval}(\tilde{\mathsf{C}}_1, \mathsf{labels}_{s_1}) = \mathsf{P}[s_2, \mathsf{ct}_f](s_1)$. By the definition of P, $\mathsf{P}[s_2, \mathsf{ct}_f](s_1)$ computes $\mathsf{sk}_f \leftarrow \mathsf{Reconstruct}(s_1, s_2)$ and outputs $m' \leftarrow \mathsf{Dec}_f(\mathsf{sk}'_f, \mathsf{ct}_f)$. Thus, by the correctness of $(\mathsf{Share}, \mathsf{Reconstruct})$ it holds that $\mathsf{sk}'_f = \mathsf{sk}_f$ and finally by the correctness of Σ_f we get that $m' = m$.

Now assume that decryption is correct for level $(i-1)$ ciphertexts and consider a ciphertext $\mathsf{rct} = (\hat{\mathsf{ct}}, \tilde{\mathsf{ct}}_t, \tilde{\mathsf{C}}_i, \ldots, \tilde{\mathsf{C}}_1)$ at level $i > 1$. As before, it holds that $(\tilde{\mathsf{C}}_i, \mathsf{labels}) \leftarrow \mathsf{Garble}(\mathsf{Q}[s_2, \hat{\mathsf{ct}}', \tilde{\mathsf{ct}}_f])$, $\hat{\mathsf{ct}} =$

$\mathsf{BatchEnc}(\hat{\mathsf{pk}}, \mathsf{labels})$ and $\widetilde{\mathsf{ct}}_t = \mathsf{Enc}_t(\mathsf{pk}_t, \hat{\mathsf{sk}})$. Again consider the computation of $\mathsf{mDec}(\Sigma_t, \mathsf{sk}_t, \mathsf{rct})$. By the correctness of Σ_t it holds that $\hat{\mathsf{sk}}' = \mathsf{Dec}(\mathsf{sk}_t, \widetilde{\mathsf{ct}}_t) = \hat{\mathsf{sk}}$. Next, by the correctness of the batch public key encryption scheme $(\mathsf{BatchGen}, \mathsf{BatchEnc}, \mathsf{BatchDec})$ it holds that that $\widetilde{\mathsf{labels}} = \mathsf{BatchDec}(\hat{\mathsf{sk}}, \hat{\mathsf{ct}}) = \mathsf{labels}_{s_1}$. Thus, by the correctness of the garbling scheme $(\mathsf{Garble}, \mathsf{Eval})$ it holds that $\mathsf{Eval}(\tilde{\mathsf{C}}_i, \widetilde{\mathsf{labels}}_i) = \mathsf{Eval}(\tilde{\mathsf{C}}_i, \widetilde{\mathsf{labels}}_{s_1}) = \mathsf{Q}[s_2, \hat{\mathsf{ct}}', \widetilde{\mathsf{ct}}_f](s_1)$.

Notice now that we can substitute $\mathsf{Q}[s_2, \hat{\mathsf{ct}}', \widetilde{\mathsf{ct}}_f](s_1)$ by

- Compute $\mathsf{sk}'_f \leftarrow \mathsf{Reconstruct}(s_1, s_2)$.
- Compute $\hat{\mathsf{sk}} \leftarrow \mathsf{Dec}(\mathsf{sk}_f, \widetilde{\mathsf{ct}}_f)$.
- Compute $\widetilde{\mathsf{labels}} \leftarrow \mathsf{BatchDec}(\hat{\mathsf{sk}}, \hat{\mathsf{ct}}')$.

By the correctness of $(\mathsf{Share}, \mathsf{Reconstruct})$ it holds that $\mathsf{sk}'_f = \mathsf{Reconstruct}(s_1, s_2) = \mathsf{sk}_f$. By inspection we see that the remaining steps of the computation are identical to the decryption of a level $(i-1)$ ciphertext. The induction hypothesis provides that decryption is correct for level $(i-1)$ ciphertexts and we are done.

6.3 Security Proof

Theorem 6.3 (UPRE-HRA security). *Assume that* $\mathsf{gc} = (\mathsf{Garble}, \mathsf{Eval})$ *is a selectively secure garbling scheme,* $(\mathsf{Share}, \mathsf{Reconstruct})$ *is a 2-out-of-2 secret sharing scheme and* $(\mathsf{BatchGen}, \mathsf{BatchEnc}, \mathsf{BatchDec})$ *is a weak batch encryption scheme in the sense of Definition 6.1, and both* Σ_f *and* Σ_t *are IND-CPA secure PKE, then* $\mathsf{UPRE}_{\mathsf{gc}}$ *is selectively UPRE-HRA secure.*

Proof. We define a sequence of hybrid experiments $\mathsf{Hyb}^x_{\mathcal{A}}(b)$. We emphasize differences among hybrid experiments by using red underlines. Hereafter, $\mathsf{Hyb}^x_{\mathcal{A}}(b) \approx \mathsf{Hyb}^y_{\mathcal{A}}(b)$ denotes $|\Pr[\mathsf{Hyb}^x_{\mathcal{A}}(b) = 1] - \Pr[\mathsf{Hyb}^y_{\mathcal{A}}(b) = 1]| \leq \mathsf{negl}(\lambda)$.

Say that a ciphertext ct is a level i re-encryption, if ct is of the form $\mathsf{ct} = (\hat{\mathsf{ct}}, \widetilde{\mathsf{ct}}_t, \tilde{\mathsf{C}}_i, \ldots, \tilde{\mathsf{C}}_1)$, i.e. ct is the result of i re-encryptions.

$\mathsf{Hyb}^0_{\mathcal{A}}(b)$**:** The first experiment is the original security experiment for b, $\mathsf{Exp}^{\mathsf{ms\text{-}upre\text{-}hra}}_{\mathcal{A}}(1^\lambda, b)$. That is, it holds that $\mathsf{Hyb}^0_{\mathcal{A}}(b) = \mathsf{Exp}^{\mathsf{ms\text{-}upre\text{-}hra}}_{\mathcal{A}}(1^\lambda, b)$. Note that in the successive experiments, we can easily simulate all keys outside of G^* since vertices in $V \setminus V^*$ are not connected to the target vertex and simulators can generate keys for them by itself.

$\mathsf{Hyb}^{0'}_{\mathcal{A}}(b)$**:** This experiment is the same as $\mathsf{Hyb}^0_{\mathcal{A}}(b)$ except that we guess the target vertex i^* that will be queried to challenge oracle $\mathcal{O}_{\mathsf{cha}}$ and abort if the guess is incorrect. The guess is correct with probability $1/|V^*|$, so $\Pr[\mathsf{Hyb}^{0'}_{\mathcal{A}}(b) = 1] = \frac{1}{|V^*|} \cdot \Pr[\mathsf{Hyb}^0_{\mathcal{A}}(b) = 1]$.

$\mathsf{Hyb}^1_{\mathcal{A}}(b)$**:** In this hybrid we record not only $(\mathsf{rct}_i, \Sigma_i, i, \#\mathsf{CT})$ but also m in $\mathsf{KeyCTList}$ for encryption query (i, m).

Moreover, for each re-encryption query, store the value $\widetilde{\mathsf{labels}} = \mathsf{labels}_{s_1}$.

The modification between $\mathsf{Hyb}^{0'}_{\mathcal{A}}(b)$ and $\mathsf{Hyb}^1_{\mathcal{A}}(b)$ is merely syntactic, thus it holds that $\Pr[\mathsf{Hyb}^{0'}_{\mathcal{A}}(b) = 1] = \Pr[\mathsf{Hyb}^1_{\mathcal{A}}(b) = 1]$.

We will now replace re-encrypted ciphertexts by simulated re-encrypted ciphertexts. For re-encryption query $(\widehat{i}, j, k)$ such that $(\widehat{i}, j)$ is not an admissible edge with respect to $G = (V, E)$ and $k \notin \mathsf{Drv}$, the re-encrypted ciphertext is differently generated by a modified re-encryption procedure. We can assume $\widehat{i}$ is honest since we do not need guarantee anything if $\widehat{i}$ is not honest. The goal of the processes below is erasing secret keys of honest vertices queried by re-encryption queries. Note that $\widehat{i} = i^*$ is possible due to the restriction $k \notin \mathsf{Drv}$ though $(\widehat{i}, j)$ is not admissible. We repeat the processes below for $u = 1, \ldots, Q_{\mathsf{reenc}}$ where Q_{reenc} is the total number of tuples $(\widehat{i}, j, k)$ such that $(\widehat{i}, j)$ is not an admissible edge with respect to $G = (V, E)$ and $k \notin \mathsf{Drv}$. Without loss of generality, we can assume that each $\widehat{i}$ is different for each such re-encryption query.[9] The changes in experiments below are for re-encryption query for u-th tuple $(\widehat{i}, j, k)$ such that $(\widehat{i}, j)$ is not an admissible edge with respect to $G = (V, E)$ and $k \notin \mathsf{Drv}$.

[9] If there exists $(\widehat{i}, j_1, k_1)$ and $(\widehat{i}, j_2, k_2)$ such that $(\widehat{i}, j_1)$ and $(\widehat{i}, j_2)$ are not admissible and $k_1, k_2 \notin \mathsf{Drv}$, then we can use the same simulation process described in hybrid experiments for those queries.

$\mathsf{Hyb}^{1,u,1}_{\mathcal{A}}(b)$**:** This is the same as $\mathsf{Hyb}^{1,(u-1),3}_{\mathcal{A}}$ except that: Retrieve s_1 of $\widehat{i}$,

- Parse $\mathsf{rk}_{\widehat{i}\to j} = (\hat{\mathsf{pk}}, s_2, \widetilde{\mathsf{ct}}_j)$ and $\mathsf{ct}_{\widehat{i}} = (\hat{\mathsf{ct}}', \widetilde{\mathsf{ct}}_{\widehat{i}}, \tilde{\mathsf{C}}_{\iota-1}, \dots, \tilde{\mathsf{C}}_1)$.
- If $\iota = 1$ set $\mathsf{C} \leftarrow \mathsf{P}[s_2, \mathsf{ct}_{\widehat{i}}]$; Else if $\iota > 1$ set $\mathsf{C} \leftarrow \mathsf{Q}[s_2, \hat{\mathsf{ct}}', \widetilde{\mathsf{ct}}_{\widehat{i}}]$
- Compute $(\tilde{\mathsf{C}}_\iota, \mathsf{labels}) \leftarrow \mathsf{Garble}(\mathsf{C})$.
- Compute $\underline{\mathsf{labels}^* \leftarrow \left\{(\mathsf{labels}_{i,s_1[i]}, \mathsf{labels}_{i,s_1[i]})\right\}_{i\in[\lambda]}}$
- Compute $\underline{\hat{\mathsf{ct}} \leftarrow \mathsf{BatchEnc}(\hat{\mathsf{pk}}, \mathsf{labels}^*)}$.
- Output $(\hat{\mathsf{ct}}, \widetilde{\mathsf{ct}}_j, \tilde{\mathsf{C}}_\iota, \dots, \tilde{\mathsf{C}}_1)$

That is, we compute $\hat{\mathsf{ct}}$ via $\mathsf{BatchEnc}(\hat{\mathsf{pk}}, \mathsf{labels}^*)$ instead of $\mathsf{BatchEnc}(\hat{\mathsf{pk}}, \mathsf{labels})$.

$\mathsf{Hyb}^{1,u,2}_{\mathcal{A}}(b)$**:** This is the same as $\mathsf{Hyb}^{1,u,1}_{\mathcal{A}}$ except that: Retrieve s_1 of $\widehat{i}$,

- Parse $\mathsf{rk}_{\widehat{i}\to j} = (\hat{\mathsf{pk}}, s_2, \widetilde{\mathsf{ct}}_j)$ and $\mathsf{ct}_{\widehat{i}} = (\hat{\mathsf{ct}}', \widetilde{\mathsf{ct}}_{\widehat{i}}, \tilde{\mathsf{C}}_{\iota-1}, \dots, \tilde{\mathsf{C}}_1)$.
- If $\iota = 1$ set $\mathsf{C} \leftarrow \mathsf{P}[s_2, \mathsf{ct}_{\widehat{i}}]$; Else if $\iota > 1$ set $\mathsf{C} \leftarrow \mathsf{Q}[s_2, \hat{\mathsf{ct}}', \widetilde{\mathsf{ct}}_{\widehat{i}}]$
- Compute $\underline{(\tilde{\mathsf{C}}_\iota, \widetilde{\mathsf{labels}}) \leftarrow \mathsf{GCSim}(\mathsf{C}(s_1))}$
- Compute $\underline{\mathsf{labels}^* \leftarrow \left\{(\widetilde{\mathsf{labels}}_i, \widetilde{\mathsf{labels}}_i)\right\}_{i\in[\lambda]}}$
- Compute $\hat{\mathsf{ct}} \leftarrow \mathsf{BatchEnc}(\hat{\mathsf{pk}}, \mathsf{labels}^*)$.
- Output $(\hat{\mathsf{ct}}, \widetilde{\mathsf{ct}}_j, \tilde{\mathsf{C}}_\iota, \dots, \tilde{\mathsf{C}}_1)$

$\mathsf{Hyb}^{1,u,3}_{\mathcal{A}}(b)$**:** This is the same as $\mathsf{Hyb}^{1,u,2}_{\mathcal{A}}(b)$ except that: Retrieve m and labels $\widetilde{\mathsf{labels}}'$ (corresponding to $\hat{\mathsf{ct}}'$),

- Parse $\mathsf{rk}_{\widehat{i}\to j} = (\hat{\mathsf{pk}}, s_2, \widetilde{\mathsf{ct}}_j)$ and $\mathsf{ct}_{\widehat{i}} = (\hat{\mathsf{ct}}', \widetilde{\mathsf{ct}}_{\widehat{i}}, \tilde{\mathsf{C}}_{\iota-1}, \dots, \tilde{\mathsf{C}}_1)$.
- If $\iota = 1$ compute $\underline{(\tilde{\mathsf{C}}_\iota, \widetilde{\mathsf{labels}}) \leftarrow \mathsf{GCSim}(m)}$; Else if $\iota > 1$ compute $\underline{(\tilde{\mathsf{C}}_\iota, \widetilde{\mathsf{labels}}) \leftarrow \mathsf{GCSim}(\widetilde{\mathsf{labels}}')}$
- Compute $\mathsf{labels}^* \leftarrow \left\{(\widetilde{\mathsf{labels}}_i, \widetilde{\mathsf{labels}}_i)\right\}_{i\in[\lambda]}$
- Compute $\hat{\mathsf{ct}} \leftarrow \mathsf{BatchEnc}(\hat{\mathsf{pk}}, \mathsf{labels}^*)$.
- Output $(\hat{\mathsf{ct}}, \widetilde{\mathsf{ct}}_j, \tilde{\mathsf{C}}_\iota, \dots, \tilde{\mathsf{C}}_1)$

For syntactic convention, we let $\mathsf{Hyb}^{1,0,3}_{\mathcal{A}}(b) := \mathsf{Hyb}^{1}_{\mathcal{A}}(b)$. Moreover, notice that at hybrid $\mathsf{Hyb}^{1,Q_{\mathsf{reenc}},3}_{\mathcal{A}}(b)$ all re-encryption queries are simulated with garbled circuits and their labels that do not depends on secret keys (or more specifically, without values that depend on secret keys). That is, we do not explicitly use sk_{i^*} to compute re-encrypted ciphertexts as above. However, in $\hat{\mathsf{pk}}$ and s_2, information about sk_{i^*} still remains. We will handles these issues in the following process.

Process for removing sk_{i^*} of the target vertex. Now, we focus on vertices in V^* connected via admissible edges. To use the security of Σ_{i^*}, we need remove information about sk_{i^*} from all re-encryption keys in $G^* = (V^*, E^*)$ possibly connected to i^*. For all (honest) vertex $j \in V^*$ that have incoming edge (i, j) such that $i \in V^*$ and do not have outgoing edge (j, j') for some j', we repeat the processes below for $v = 1, \dots, Q$ where Q is the total number of admissible edges connected to target vertex i^*. We let Dlist be the list of vertices whose re-encryption key consists of a simulated and dummy values. That is, if $j \in \mathsf{Dlist}$, then $\mathsf{rk}_{i\to j} = (\hat{\mathsf{pk}}, s_2, \mathsf{Enc}(\mathsf{pk}_j, 0^n))$ where $(\hat{\mathsf{pk}}, \hat{\mathsf{sk}}) \leftarrow \mathsf{BatchGen}(0^n)$ and $(s_1, s_2) \leftarrow \mathsf{Share}(0^n)$ for any i. We initialize $\mathsf{Dlist} := \emptyset$ and maintain Dlist during the repeated processes below.

In the following hybrids we modify the key-generation for honest vertices. That is, all changes in the experiments are in the computation of $\mathsf{rk}_{i\to j}$.

$\mathsf{Hyb}^{2,v,1}_{\mathcal{A}}(b)$:

- Compute $(s_1, s_2) \leftarrow \mathsf{Share}(\mathsf{sk}_i)$
- Compute $(\hat{\mathsf{pk}}, \hat{\mathsf{sk}}) \leftarrow \mathsf{BatchGen}(s_1)$.
- Compute $\widetilde{\mathsf{ct}}_j \leftarrow \underline{\mathsf{Enc}_j(\mathsf{pk}_j, 0^n)}$
- Output $\mathsf{rk}_{i \to j} := (\hat{\mathsf{pk}}, s_2, \widetilde{\mathsf{ct}}_j)$.

That is, we compute $\widetilde{\mathsf{ct}}_j \leftarrow \mathsf{Enc}_j(\mathsf{pk}_j, 0^n)$ instead of $\widetilde{\mathsf{ct}}_j \leftarrow \mathsf{Enc}_j(\mathsf{pk}_j, (s_1, \hat{\mathsf{sk}}))$.

$\mathsf{Hyb}^{2,v,2}_{\mathcal{A}}(b)$:

- Compute $(s_1, s_2) \leftarrow \mathsf{Share}(\mathsf{sk}_i)$
- Compute $(\hat{\mathsf{pk}}, \hat{\mathsf{sk}}) \leftarrow \underline{\mathsf{BatchGen}(0^n)}$.
- Compute $\widetilde{\mathsf{ct}}_j \leftarrow \mathsf{Enc}_j(\mathsf{pk}_j, 0^n)$
- Output $\mathsf{rk}_{i \to j} := (\hat{\mathsf{pk}}, s_2, \widetilde{\mathsf{ct}}_j)$.

$\mathsf{Hyb}^{2,v,3}_{\mathcal{A}}(b)$:

- Compute $(s_1, s_2) \leftarrow \underline{\mathsf{Share}(0^n)}$
- Compute $(\hat{\mathsf{pk}}, \hat{\mathsf{sk}}) \leftarrow \mathsf{BatchGen}(0^n)$.
- Compute $\widetilde{\mathsf{ct}}_j \leftarrow \mathsf{Enc}_j(\mathsf{pk}_j, 0^n)$
- Output $\mathsf{rk}_{i \to j} := (\hat{\mathsf{pk}}, s_2, \widetilde{\mathsf{ct}}_j)$ and renew $\mathsf{Dlist} := \mathsf{Dlist} \cup \{j\}$.

For syntactic convention, we let $\mathsf{Hyb}^{2,0,3}_{\mathcal{A}}(b) := \mathsf{Hyb}^{1,Q_{\mathsf{reenc}},3}_{\mathcal{A}}(b)$.

Now, we prove indistinguishability of hybrid games. First notice that by correctness of $(\mathsf{Share}, \mathsf{Reconstruct})$ and $(\mathsf{BatchGen}, \mathsf{BatchEnc}, \mathsf{BatchDec})$ the modification between $\mathsf{Hyb}^{1,u,2}_{\mathcal{A}}(b)$ and $\mathsf{Hyb}^{1,u,3}_{\mathcal{A}}(b)$ is merely syntactic and the following lemma holds.

Lemma 6.4. *It holds that* $\mathsf{Hyb}^{1,u,2}_{\mathcal{A}}(b) = \mathsf{Hyb}^{1,u,3}_{\mathcal{A}}(b)$.

Indistinguishability of $\mathsf{Hyb}^{1,(u-1),3}_{\mathcal{A}}(b)$ and $\mathsf{Hyb}^{1,u,1}_{\mathcal{A}}(b)$is shown in Lemma 6.5, whereas indistinguishability of $\mathsf{Hyb}^{1,u,1}_{\mathcal{A}}(b)$ and $\mathsf{Hyb}^{1,u,2}_{\mathcal{A}}(b)$is shown in Lemma 6.6.

Lemma 6.5. *If* $(\mathsf{BatchGen}, \mathsf{BatchEnc}, \mathsf{BatchDec})$ *is WBE-CPA secure, then it holds that* $\mathsf{Hyb}^{1,u,1}_{\mathcal{A}}(b) \approx \mathsf{Hyb}^{1,(u-1),3}_{\mathcal{A}}(b)$.

Proof of Lemma 6.5. We will construct a reduction $\mathcal{B}$ which breaks the sender privacy of $(\mathsf{BatchGen}, \mathsf{BatchEnc}, \mathsf{BatchDec})$. The reduction $\mathcal{B}$ answers re-encryption queries as follows. From the first to $(u-1)$-th re-encryption queries are handled as in $\mathsf{Hyb}^{1,u,1}_{\mathcal{A}}(b)$. From the $(u+1)$-th to Q_{reenc}-th re-encryption queries are handled as in $\mathsf{Hyb}^{1,(u-1),3}_{\mathcal{A}}(b)$. $\mathcal{B}$ can simulate all oracles since $\mathcal{B}$ can generate secret keys by itself. For the u-th query $(\widehat{i}, j, k)$ such that $(\widehat{i}, j)$ is not an admissible edge with respect to G and $k \notin \mathsf{Drv}$, $\mathcal{B}$ embeds its own challenge. That is, $\mathcal{B}$ sends s_1 and labels to the experiment and obtains $(\hat{\mathsf{pk}}, \hat{\mathsf{sk}}, \hat{\mathsf{ct}})$. It then uses these values in its own simulation. Clearly, if $\hat{\mathsf{ct}} = \mathsf{BatchEnc}(\hat{\mathsf{pk}}, \mathsf{labels})$, then this query is handled as in $\mathsf{Hyb}^{1,(u-1),3}_{\mathcal{A}}(b)$. On the other hand, if $\hat{\mathsf{ct}} = \mathsf{BatchEnc}(\hat{\mathsf{pk}}, \mathsf{labels}^*)$, then the query is handled as in $\mathsf{Hyb}^{1,u,1}_{\mathcal{A}}(b)$. ■

Lemma 6.6. *If* $\mathsf{gc} = (\mathsf{Garble}, \mathsf{Eval})$ *is a selectively secure garbling scheme, then it holds that* $\mathsf{Hyb}^{1,u,2}_{\mathcal{A}}(b) \approx \mathsf{Hyb}^{1,u,1}_{\mathcal{A}}(b)$.

Proof of Lemma 6.6. We will construct a reduction $\mathcal{B}$ which breaks the security of $(\mathsf{Garble}, \mathsf{Eval})$. As in the proof of Lemma 6.5, from the first to $(u-1)$-th re-encryption queries are handled as in $\mathsf{Hyb}^{1,u,2}_{\mathcal{A}}(b)$ and from the $(u+1)$-th to Q_{reenc}-th re-encryption queries are handled as in $\mathsf{Hyb}^{1,u,1}_{\mathcal{A}}(b)$. $\mathcal{B}$ can simulate all oracles since $\mathcal{B}$ can generate secret keys by itself. $\mathcal{B}$ will embed its challenge in the u-th re-encryption query $(\widehat{i}, j, k)$ such that $(\widehat{i}, j)$ is not an admissible edge and $k \notin \mathsf{Drv}$. That is, $\mathcal{B}$ sends (C, s_1) to the experiment and obtains $(\tilde{\mathsf{C}}, \widetilde{\mathsf{labels}})$. It then uses these values in its own simulation. Clearly, if $(\tilde{\mathsf{C}}, \mathsf{labels}) = \mathsf{Grbl}(\mathsf{C})$ and $\widetilde{\mathsf{labels}} = \mathsf{labels}_{s_1}$, then this query is handled as in $\mathsf{Hyb}^{1,u,1}_{\mathcal{A}}(b)$. On the other hand, if $(\tilde{\mathsf{C}}, \widetilde{\mathsf{labels}}) = \mathsf{GCSim}(\mathsf{C}(s_1))$, then the query is handled as in $\mathsf{Hyb}^{1,u,2}_{\mathcal{A}}(b)$. ■

Lemma 6.7. *If Σ_j is CPA-secure, then it holds that* $\mathsf{Hyb}_{\mathcal{A}}^{2,(v-1),3} \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_{\mathcal{A}}^{2,v,1}$.

Proof. First, at this point, honest vertex j does not have any outgoing edge. That is, we never use sk_j for simulation at this point. We can construct an adversary $\mathcal{B}$ that is given pk_j. $\mathcal{B}$ sends $(\hat{\mathsf{sk}}, 0^n)$ as a challenge message pair and receive a target ciphertext $\widetilde{\mathsf{ct}}_j^*$. $\mathcal{B}$ uses $\widetilde{\mathsf{ct}}_j^*$ as a part of $\mathsf{rk}_{i \to j}$. Thus, the lemma immediately follows from the CPA-security of Σ_j. ■

Lemma 6.8. *It holds that* $\mathsf{Hyb}_{\mathcal{A}}^{2,v,2}(b) \equiv \mathsf{Hyb}_{\mathcal{A}}^{2,v,1}(b)$

Proof. This follows from the fact that the distribution of $\hat{\mathsf{pk}}$ is independent of s_1 ■

Lemma 6.9. *If* $(\mathsf{Share}, \mathsf{Reconstruct})$ *is 2-out-of-2 secrete sharing scheme, then it holds that* $\mathsf{Hyb}_{\mathcal{A}}^{2,v,2}(b) \overset{\mathsf{s}}{\approx} \mathsf{Hyb}_{\mathcal{A}}^{2,v,3}(b)$.

Proof. This immediately follows from the security of $(\mathsf{Share}, \mathsf{Reconstruct})$ since s_1 is not used anywhere at this point. ■

In $\mathsf{Hyb}_{\mathcal{A}}^{2,Q,3}(b)$, sk_{i^*} is neither written in any re-encryption key nor used to generate a re-encrypted ciphertext. Thus, we can use the security of Σ_{i^*}. As in Lemma 5.9, we can prove that $\mathsf{Hyb}_{\mathcal{A}}^{2,Q,3}(0) \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_{\mathcal{A}}^{2,Q,3}(1)$ holds due to the CPA-security of Σ_{i^*}. Therefore, it holds that $\mathsf{Hyb}_{\mathcal{A}}^{0}(0) \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_{\mathcal{A}}^{0}(1)$ since Q_{reenc}, Q and $|V^*|$ are polynomials. ■

7 Constant-Hop Construction Secure against CRA

In this section, we present constant-hop and CRA-secure UPRE schemes for PKE based on GC.

7.1 Our Constant-Hop Scheme from GC

Our scheme $\mathsf{UPRE}_{\mathsf{cra}}$ is based on a on a garbling scheme $(\mathsf{Garble}, \mathsf{Eval})$, a weak batch encryption scheme $(\mathsf{BatchGen}, \mathsf{BatchEnc}, \mathsf{BatchDec})$ and a 2-player secret-sharing scheme $(\mathsf{Share}, \mathsf{Reconstruct})$. As in Section 5, we overload the notation $\Sigma_{\sigma_i} = (\mathsf{Gen}_{\sigma_i}, \mathsf{Enc}_{\sigma_i}, \mathsf{Dec}_{\sigma_i})$ by $\Sigma_i = (\mathsf{Gen}_i, \mathsf{Enc}_i, \mathsf{Dec}_i)$ for ease of notation.

- $\mathsf{ReKeyGen}(1^\lambda, \Sigma_f, \Sigma_t, \mathsf{sk}_f, \mathsf{pk}_t)$:
  - Compute $(s_1, s_2) \leftarrow \mathsf{Share}(\mathsf{sk}_f)$
  - $(\hat{\mathsf{pk}}, \hat{\mathsf{sk}}) \leftarrow \mathsf{BatchGen}(1^\lambda, s_1)$
  - Compute $\widetilde{\mathsf{ct}}_t \leftarrow \mathsf{Enc}_t(\mathsf{pk}_t, \hat{\mathsf{sk}})$
  - Output $\mathsf{rk}_{f \to t} := (\hat{\mathsf{pk}}, s_2, \widetilde{\mathsf{ct}}_t)$.
- $\mathsf{ReEnc}(\Sigma_f, \Sigma_t, \mathsf{rk}_{f \to t}, \mathsf{ct}_f)$:
  - Parse $\mathsf{rk}_{f \to t} = (\hat{\mathsf{pk}}, s_2, \widetilde{\mathsf{ct}}_t)$.
  - Parse $\mathsf{ct}_f = (\hat{\mathsf{ct}}', \widetilde{\mathsf{ct}}_f, \widetilde{\mathsf{ct}}_f')$.
  - If this is the first re-encryption set $\mathsf{C} \leftarrow \mathsf{P}[s_2, \mathsf{ct}_f]$; Else if set $\mathsf{C} \leftarrow \mathsf{Q}[s_2, \hat{\mathsf{ct}}', \widetilde{\mathsf{ct}}_f, \widetilde{\mathsf{ct}}_f']$
  - Compute $(\tilde{\mathsf{C}}, \mathsf{labels}) \leftarrow \mathsf{Garble}(\mathsf{C})$.
  - Compute $\hat{\mathsf{ct}} \leftarrow \mathsf{BatchEnc}(\hat{\mathsf{pk}}, \mathsf{labels})$.
  - Compute $\widetilde{\mathsf{ct}}_t' \leftarrow \mathsf{Enc}_t(\mathsf{pk}_t, \tilde{\mathsf{C}})$.
  - Output $(\hat{\mathsf{ct}}, \widetilde{\mathsf{ct}}_t, \widetilde{\mathsf{ct}}_t')$.
- $\mathsf{mDec}(\Sigma_t, \mathsf{sk}_t, \mathsf{rct}, i)$: Parse $\mathsf{rct} = (\hat{\mathsf{ct}}, \widetilde{\mathsf{ct}}_t, \widetilde{\mathsf{ct}}_t')$.
  - Compute $\hat{\mathsf{sk}}' \leftarrow \mathsf{Dec}(\mathsf{sk}_t, \widetilde{\mathsf{ct}}_t)$.

- Compute $\widetilde{\mathsf{labels}}_i \leftarrow \mathsf{BatchDec}(\hat{\mathsf{sk}}', \hat{\mathsf{ct}})$.
- Compute $\tilde{\mathsf{C}}_i := \tilde{\mathsf{C}} \leftarrow \mathsf{Dec}_t(\mathsf{sk}_t, \widetilde{\mathsf{ct}}'_t)$.
- For $j = i, \ldots, 2$ do: Compute $(\tilde{\mathsf{C}}_{j-1}, \widetilde{\mathsf{labels}}_{j-1}) \leftarrow \mathsf{Eval}(\tilde{\mathsf{C}}_j, \widetilde{\mathsf{labels}}_j)$.
- Compute and output $m' \leftarrow \mathsf{Eval}(\tilde{\mathsf{C}}_1, \widetilde{\mathsf{labels}}_1)$.

First Level Re-Encryption Circuit $\mathsf{P}[s_2, \mathsf{ct}_f](s_1)$

Hardwired: s_2, ct_f.
Input: A share s_1.

- Compute $\mathsf{sk}'_f \leftarrow \mathsf{Reconstruct}(s_1, s_2)$.
- Compute and output $m' \leftarrow \mathsf{Dec}_f(\mathsf{sk}'_f, \mathsf{ct}_f)$.

Figure 8: The description of the first level re-encryption circuit P

Higher Level Re-Encryption Circuit $\mathsf{Q}[s_2, \hat{\mathsf{ct}}', \widetilde{\mathsf{ct}}_f, \widetilde{\mathsf{ct}}'_f](s_1)$

Hardwired: s_2, $\hat{\mathsf{ct}}'$, $\widetilde{\mathsf{ct}}_f$, $\widetilde{\mathsf{ct}}'_f$.
Input: A share s_1.

- Compute $\mathsf{sk}'_f \leftarrow \mathsf{Reconstruct}(s_1, s_2)$.
- Compute $\hat{\mathsf{sk}}' \leftarrow \mathsf{Dec}(\mathsf{sk}'_f, \widetilde{\mathsf{ct}}_f)$.
- Compute and output $\tilde{\mathsf{C}} \leftarrow \mathsf{Dec}_f(\mathsf{sk}'_f, \widetilde{\mathsf{ct}}'_f)$ and $\widetilde{\mathsf{labels}} \leftarrow \mathsf{BatchDec}(\hat{\mathsf{sk}}', \hat{\mathsf{ct}}')$.

Figure 9: The description of the higher level re-encryption circuit Q

Efficiency. Encrypted garbled circuit $\widetilde{\mathsf{ct}}'_f \leftarrow \mathsf{Enc}(\mathsf{pk}_f, \tilde{\mathsf{C}})$ is hard-wired in $\mathsf{Q}[s_2, \hat{\mathsf{ct}}', \widetilde{\mathsf{ct}}_f, \widetilde{\mathsf{ct}}'_f]$ and Q is garbled. That is, garbled circuits are nested. This incurs polynomial blow-up. Thus, we can apply the re-encryption algorithm only constant time.

Correctness. The correcntess follows by a similar argument to that of $\mathsf{UPRE}_{\mathsf{gc}}$ in Section 6.2.

We will first show correctness for level 1 ciphertexts. Let thus $\mathsf{rct} = (\hat{\mathsf{ct}}, \widetilde{\mathsf{ct}}_t, \widetilde{\mathsf{ct}}'_t)$ be a level 1 ciphertext, where $\widetilde{\mathsf{ct}}'_t \leftarrow \mathsf{Enc}_t(\mathsf{pk}_t, \tilde{\mathsf{C}}_1)$ and $(\tilde{\mathsf{C}}_1, \mathsf{labels}) \leftarrow \mathsf{Garble}(\mathsf{P}[s_2, \mathsf{ct}_f])$, $\hat{\mathsf{ct}} = \mathsf{BatchEnc}(\hat{\mathsf{pk}}, \mathsf{labels})$ and $\widetilde{\mathsf{ct}}_t = \mathsf{Enc}_t(\mathsf{pk}_t, \hat{\mathsf{sk}})$. Consider the computation of $\mathsf{mDec}(\Sigma_t, \mathsf{sk}_t, \mathsf{rct})$. By the correctness of Σ_t it holds that $\hat{\mathsf{sk}}' = \mathsf{Dec}(\mathsf{sk}_t, \widetilde{\mathsf{ct}}_t) = \hat{\mathsf{sk}}$ and $\tilde{\mathsf{C}}_1 = \mathsf{Dec}(\mathsf{sk}_t, \widetilde{\mathsf{ct}}'_t)$. Next, by the correctness of the batch public key encryption $(\mathsf{BatchGen}, \mathsf{BatchEnc}, \mathsf{BatchDec})$ it holds that that $\widetilde{\mathsf{labels}} = \mathsf{BatchDec}(\hat{\mathsf{sk}}, \hat{\mathsf{ct}}) = \mathsf{labels}_{s_1}$. Thus, by the correctness of the garbling scheme $(\mathsf{Garble}, \mathsf{Eval})$ it holds that $\mathsf{Eval}(\tilde{\mathsf{C}}_1, \widetilde{\mathsf{labels}}) = \mathsf{Eval}(\tilde{\mathsf{C}}_1, \mathsf{labels}_{s_1}) = \mathsf{P}[s_2, \mathsf{ct}_f](s_1)$. By the definition of P, $\mathsf{P}[s_2, \mathsf{ct}_f](s_1)$ computes $\mathsf{sk}_f \leftarrow \mathsf{Reconstruct}(s_1, s_2)$ and outputs $m' \leftarrow \mathsf{Dec}_f(\mathsf{sk}'_f, \mathsf{ct}_f)$. Thus, by the correctness of $(\mathsf{Share}, \mathsf{Reconstruct})$ it holds that $\mathsf{sk}'_f = \mathsf{sk}_f$ and finally by the correctness of Σ_f we get that $m' = m$.

Now we consider a ciphertext $\mathsf{rct} = (\hat{\mathsf{ct}}, \widetilde{\mathsf{ct}}_t, \widetilde{\mathsf{ct}}'_t)$ at level $i > 1$. As before, it holds that $\widetilde{\mathsf{ct}}'_t \leftarrow \mathsf{Enc}(\mathsf{pk}_t, \tilde{\mathsf{C}}_i)$, $(\tilde{\mathsf{C}}_i, \mathsf{labels}) \leftarrow \mathsf{Garble}(\mathsf{Q}[s_2, \hat{\mathsf{ct}}', \widetilde{\mathsf{ct}}_f, \widetilde{\mathsf{ct}}'_f])$, $\hat{\mathsf{ct}} = \mathsf{BatchEnc}(\hat{\mathsf{pk}}, \mathsf{labels})$ and $\widetilde{\mathsf{ct}}_t = \mathsf{Enc}_t(\mathsf{pk}_t, \hat{\mathsf{sk}})$. Again consider the computation of $\mathsf{mDec}(\Sigma_t, \mathsf{sk}_t, \mathsf{rct})$. By the correctness of Σ_t it holds that $\hat{\mathsf{sk}}' = \mathsf{Dec}(\mathsf{sk}_t, \widetilde{\mathsf{ct}}_t) = \hat{\mathsf{sk}}$ and $\tilde{\mathsf{C}}_i = \mathsf{Dec}(\mathsf{sk}_t, \widetilde{\mathsf{ct}}'_t)$. Next, by the correctness of the batch public key encryption scheme $(\mathsf{BatchGen}, \mathsf{BatchEnc}, \mathsf{BatchDec})$ it holds that that $\widetilde{\mathsf{labels}} = \mathsf{BatchDec}(\hat{\mathsf{sk}}, \hat{\mathsf{ct}}) = \mathsf{labels}_{s_1}$. Thus, by the correctness of the garbling scheme $(\mathsf{Garble}, \mathsf{Eval})$ it holds that $\mathsf{Eval}(\tilde{\mathsf{C}}_i, \widetilde{\mathsf{labels}}_i) = \mathsf{Eval}(\tilde{\mathsf{C}}_i, \widetilde{\mathsf{labels}}_{s_1}) = \mathsf{Q}[s_2, \hat{\mathsf{ct}}', \widetilde{\mathsf{ct}}_f, \widetilde{\mathsf{ct}}'_f](s_1)$.

Notice now that we can substitute $\mathsf{Q}[s_2, \hat{\mathsf{ct}}', \widetilde{\mathsf{ct}}_f, \widetilde{\mathsf{ct}}'_f](s_1)$ by

- Compute $\mathsf{sk}'_f \leftarrow \mathsf{Reconstruct}(s_1, s_2)$.
- Compute $\hat{\mathsf{sk}} \leftarrow \mathsf{Dec}(\mathsf{sk}_f, \widetilde{\mathsf{ct}}_f)$.

- Compute $\tilde{\mathsf{C}}_{i-1} \leftarrow \mathsf{Dec}(\mathsf{sk}_f, \widetilde{\mathsf{ct}}'_f)$ and $\widetilde{\mathsf{labels}} \leftarrow \mathsf{BatchDec}(\hat{\mathsf{sk}}, \hat{\mathsf{ct}}')$.

By the correctness of $(\mathsf{Share}, \mathsf{Reconstruct})$ it holds that $\mathsf{sk}'_f = \mathsf{Reconstruct}(s_1, s_2) = \mathsf{sk}_f$. By inspection we see that the remaining steps of the computation are evaluating garbled circuits again and again until $i = 2$. By combining with the level 1 case, the correctness follows.

7.2 Security Proof

Theorem 7.1 (UPRE-CRA security). *Assume that* $\mathsf{gc} = (\mathsf{Garble}, \mathsf{Eval})$ *is a selectively secure garbling scheme,* $(\mathsf{Share}, \mathsf{Reconstruct})$ *is a 2-out-of-2 secret sharing scheme and* $(\mathsf{BatchGen}, \mathsf{BatchEnc}, \mathsf{BatchDec})$ *is a weak batch encryption scheme in the sense of Definition 6.1, and both* Σ_f *and* Σ_t *are IND-CPA secure PKE, then* $\mathsf{UPRE}_{\mathsf{cra}}$ *is selectively UPRE-CRA secure.*

Proof. The proof is basically the same as that of Theorem 6.3 in Section 6 except that we need two more hybrids in addition to the hybrids in Theorem 6.3. Thus, we write only the new hybrids.

$\mathsf{Hyb}_{\mathcal{A}}^{3,1}(b)$**:** This experiment is the same as $\mathsf{Hyb}_{\mathcal{A}}^{2,Q,3}(b)$ except that for all query (i_{-1}, i^*) to $\mathcal{O}_{\mathsf{rekey}}$ such that $i_{-1} \in \mathsf{CList}$, ciphertext $\widetilde{\mathsf{ct}}_{i^*}$ in the re-encryption key $\mathsf{rk}_{i_{-1} \to i^*}$ is generated by $\mathsf{Enc}_{i^*}(\mathsf{pk}_{i^*}, \underline{0^{\ell_{i^*}}})$ instead of $\mathsf{Enc}_{i^*}(\mathsf{pk}_{i^*}, \underline{\hat{\mathsf{sk}}})$. We can prove that $\mathsf{Hyb}_{\mathcal{A}}^{3,1}(b) \stackrel{\mathsf{c}}{\approx} \mathsf{Hyb}_{\mathcal{A}}^{2,Q,3}(b)$ hold due to the CPA-security of Σ_{i^*} in a similar way to Lemma 6.7.

$\mathsf{Hyb}_{\mathcal{A}}^{3,2}(b)$**:** This experiment is the same as $\mathsf{Hyb}_{\mathcal{A}}^{3,1}(b)$ except that for the challenge query $(i_{\mathsf{c}}, i^*, m_0, m_1)$ to $\mathcal{O}_{\mathsf{cha}}$, $\widehat{\mathsf{ct}}_{i^*}$ in the target ciphertext rct_{i^*} is generated by $\mathsf{Enc}_{i^*}(\mathsf{pk}_{i^*}, \underline{0^{\ell_j}})$ instead of $\mathsf{Enc}_{i^*}(\mathsf{pk}_{i^*}, \underline{\tilde{\mathsf{C}}})$ where $\tilde{\mathsf{C}} \leftarrow \mathsf{Garble}(\mathsf{P}[s_2, \mathsf{ct}_{\mathsf{c}}])$. Note that we can easily simulate $\mathsf{rk}_{i_{-1} \to i^*}$ for $i_{-1} \in \mathsf{CList}$ since sk_{-1} is revealed. We prove that $\mathsf{Hyb}_{\mathcal{A}}^{3,2}(b) \stackrel{\mathsf{c}}{\approx} \mathsf{Hyb}_{\mathcal{A}}^{3,1}(b)$ hold due to the CPA-security of Σ_{i^*} in Lemma 7.3.

In $\mathsf{Hyb}_{\mathcal{A}}^{3,2}(b)$, $\mathsf{rct}_{i^*} = (\hat{\mathsf{pk}}, \mathsf{Enc}_{i^*}(\mathsf{pk}_{i^*}, 0^{\ell_j}), \mathsf{Enc}_{i^*}(\mathsf{pk}_{i^*}, 0^{\ell_j}))$. Thus, it is easy to see that the advantage of $\mathcal{A}$ is just $\frac{1}{2}$ since there is no information about b in these hybrids. Thus, $\mathsf{Hyb}_{\mathcal{A}}^{3,2}(0) = \mathsf{Hyb}_{\mathcal{A}}^{3,2}(1) = \frac{1}{2}$ and the theorem follows. ■

Lemma 7.2. *If* Σ_{i^*} *is IND-CPA secure PKE, then it holds that* $\mathsf{Hyb}_{\mathcal{A}}^{3,1}(b) \stackrel{\mathsf{c}}{\approx} \mathsf{Hyb}_{\mathcal{A}}^{2,Q,3}(b)$.

Proof. We can prove in a similar way to the proof of Lemma 6.7, so we omit this. ■

Lemma 7.3. *If* Σ_{i^*} *is IND-CPA secure PKE, then it holds that* $\mathsf{Hyb}_{\mathcal{A}}^{3,2}(b) \stackrel{\mathsf{c}}{\approx} \mathsf{Hyb}_{\mathcal{A}}^{3,1}(b)$.

Proof. We construct an adversary $\mathcal{B}$ of Σ_{i^*}, which is given pk_{i^*} as a target public key. To use $\mathcal{A}$ of UPRE, $\mathcal{B}$ generates key pairs $(\mathsf{pk}_{i'}, \mathsf{sk}_{i'})$ for all $i' \in \mathsf{HList} \setminus \{i^*\}$. $\mathcal{B}$ sets pk_{i^*} as a public-key of user i^*. In experiments $\mathsf{Hyb}_{\mathcal{A}}^{2,Q,3}(b)$, sk_{i^*} is not used anywhere by the definition of the experiments. When (i^*, j) is queried to $\mathcal{O}_{\mathsf{rekey}}$ such that $(i^*, j) \in E^*$, $\mathcal{B}$ can generate a re-encryption key without sk_{i^*}. When (i^*, j, k) is queried to $\mathcal{O}_{\mathsf{reenc}}$ such that $j \in \mathsf{CList} \wedge k \notin \mathsf{Drv}$ and $(\mathsf{ct}_i, \Sigma_i, i, \#\mathsf{CT}, m) \in \mathsf{KeyCTList}$, $\mathcal{B}$ can generate a re-encrypted ciphertext without sk_{i^*} as we see above. When $(i_{\mathsf{c}}, i^*, m_0, m_1)$ is queried to the challenge oracle $\mathcal{O}_{\mathsf{cha}}$, then $\mathcal{B}$ passes $(\tilde{\mathsf{C}}, 0^{\ell_{i^*}})$ where $\tilde{\mathsf{C}} \leftarrow \mathsf{Garble}(\mathsf{P}[s_1, \mathsf{ct}_{\mathsf{c}}])$ and $\mathsf{ct}_{\mathsf{c}} \leftarrow \mathsf{Enc}_{\mathsf{c}}(\mathsf{pk}_{\mathsf{c}}, m_b)$ to the challenger of IND-CPA game of Σ_{i^*} and receives a target ciphertext $\widehat{\mathsf{ct}}_{i^*}^*$. $\mathcal{B}$ returns $(\hat{\mathsf{pk}}, \mathsf{Enc}_{i^*}(\mathsf{pk}_{i^*}, 0^{\ell_{i^*}}), \widehat{\mathsf{ct}}_{i^*}^*)$ to $\mathcal{A}$. If $\mathcal{A}$ can distinguish two experiments, then $\mathcal{B}$ can break the security of Σ_{i^*} since the case $\widehat{\mathsf{ct}}_{i^*}^* = \mathsf{Enc}_{i^*}(\mathsf{pk}_{i^*}, \tilde{\mathsf{C}})$ and the case $\widehat{\mathsf{ct}}_{i^*}^* = \mathsf{Enc}_{i^*}(\mathsf{pk}_{i^*}, 0^{\ell_{i^*}})$ perfectly simulate $\mathsf{Hyb}_{\mathcal{A}}^{3,1}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^{3,2}(b)$, respectively. ■

References

[ABG$^+$13] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. Cryptology ePrint Archive, Report 2013/689, 2013. http://eprint.iacr.org/2013/689. (Cited on page 10.)

[ABH09] Giuseppe Ateniese, Karyn Benson, and Susan Hohenberger. Key-private proxy re-encryption. In Marc Fischlin, editor, *CT-RSA 2009*, volume 5473 of *LNCS*, pages 279–294. Springer, Heidelberg, April 2009. (Cited on page 6, 11, 12, 15, 38.)

[ABPW13] Yoshinori Aono, Xavier Boyen, Le Trieu Phong, and Lihua Wang. Key-private proxy re-encryption under LWE. In Goutam Paul and Serge Vaudenay, editors, *INDOCRYPT 2013*, volume 8250 of *LNCS*, pages 1–18. Springer, Heidelberg, December 2013. (Cited on page 6.)

[ACH18] Thomas Agrikola, Geoffroy Couteau, and Dennis Hofheinz. The usefulness of sparsifiable inputs: How to avoid subexponential iO. Cryptology ePrint Archive, Report 2018/470, 2018. https://eprint.iacr.org/2018/470. (Cited on page 4, 5, 26.)

[AFGH05] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. In *NDSS 2005*. The Internet Society, February 2005. (Cited on page 1, 6, 10, 12.)

[BBS98] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 127–144. Springer, Heidelberg, May / June 1998. (Cited on page 1, 6, 10.)

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 52–73. Springer, Heidelberg, February 2014. (Cited on page 10.)

[BGI$^+$12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *Journal of the ACM*, 59(2):6, 2012. (Cited on page 2.)

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 501–519. Springer, Heidelberg, March 2014. (Cited on page 7.)

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 416–432. Springer, Heidelberg, May 2003. (Cited on page 6.)

[BLSV18] Zvika Brakerski, Alex Lombardi, Gil Segev, and Vinod Vaikuntanathan. Anonymous IBE, leakage resilience and circular security from new assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 535–564. Springer, Heidelberg, April / May 2018. (Cited on page 5, 27.)

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 280–300. Springer, Heidelberg, December 2013. (Cited on page 7.)

[CCL$^+$14] Nishanth Chandran, Melissa Chase, Feng-Hao Liu, Ryo Nishimaki, and Keita Xagawa. Re-encryption, functional re-encryption, and multi-hop re-encryption: A framework for achieving obfuscation-based security and instantiations from lattices. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 95–112. Springer, Heidelberg, March 2014. (Cited on page 6.)

[CCV12] Nishanth Chandran, Melissa Chase, and Vinod Vaikuntanathan. Functional re-encryption and collusion-resistant obfuscation. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 404–421. Springer, Heidelberg, March 2012. (Cited on page 6.)

[CH07] Ran Canetti and Susan Hohenberger. Chosen-ciphertext secure proxy re-encryption. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 07*, pages 185–194. ACM Press, October 2007. (Cited on page 6, 10, 12, 16.)

[CHN$^+$16] Aloni Cohen, Justin Holmgren, Ryo Nishimaki, Vinod Vaikuntanathan, and Daniel Wichs. Watermarking cryptographic capabilities. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 1115–1127. ACM Press, June 2016. (Cited on page 4.)

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 476–493. Springer, Heidelberg, August 2013. (Cited on page 2.)

[CLTV15] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 468–497. Springer, Heidelberg, March 2015. (Cited on page 3, 4, 5, 9, 10, 21, 22, 25.)

[Coh17] Aloni Cohen. What about bob? The inadequacy of CPA security for proxy reencryption. Cryptology ePrint Archive, Report 2017/785, 2017. http://eprint.iacr.org/2017/785. (Cited on page 2, 11, 12, 15, 38.)

[CPP16] Geoffroy Couteau, Thomas Peters, and David Pointcheval. Encryption switching protocols. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 308–338. Springer, Heidelberg, August 2016. (Cited on page 5.)

[DHRW16] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 93–122. Springer, Heidelberg, August 2016. (Cited on page 9.)

[DJ01] Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In Kwangjo Kim, editor, *PKC 2001*, volume 1992 of *LNCS*, pages 119–136. Springer, Heidelberg, February 2001. (Cited on page 21, 22.)

[DKL$^+$18] David Derler, Stephan Krenn, Thomas Lorünser, Sebastian Ramacher, Daniel Slamanig, and Christoph Striecks. Revisiting proxy re-encryption: Forward secrecy, improved security, and applications. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 219–250. Springer, Heidelberg, March 2018. (Cited on page 2.)

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985. (Cited on page 1, 21, 22.)

[FKKP18] Georg Fuchsbauer, Chethan Kamath, Karen Klein, and Krzysztof Pietrzak. Adaptively secure proxy re-encryption. Cryptology ePrint Archive, Report 2018/426, 2018. https://eprint.iacr.org/2018/426. (Cited on page 12.)

[GGH13] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 1–17. Springer, Heidelberg, May 2013. (Cited on page 2.)

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 498–527. Springer, Heidelberg, March 2015. (Cited on page 2.)

[GGH$^+$16] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM J. Comput.*, 45(3):882–929, 2016. (Cited on page 2.)

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986. (Cited on page 7.)

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984. (Cited on page 1, 21, 22.)

[GSL19] Sivanarayana Gaddam, Rohit Sinha, and Atul Luykx. Applying proxy-re-encryption to payments. Real World Crypto 2019, 2019. https://rwc.iacr.org/2019/slides/Applying_PRE_Payments.pdf. (Cited on page 1.)

[HHR16] Julia Hesse, Dennis Hofheinz, and Andy Rupp. Reconfigurable cryptography: A flexible approach to long-term security. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 416–445. Springer, Heidelberg, January 2016. (Cited on page 6.)

[HJK$^+$16] Dennis Hofheinz, Tibor Jager, Dakshita Khurana, Amit Sahai, Brent Waters, and Mark Zhandry. How to generate and use universal samplers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 715–744. Springer, Heidelberg, December 2016. (Cited on page 6.)

[HKK$^+$12] Goichiro Hanaoka, Yutaka Kawai, Noboru Kunihiro, Takahiro Matsuda, Jian Weng, Rui Zhang, and Yunlei Zhao. Generic construction of chosen ciphertext secure proxy re-encryption. In Orr Dunkelman, editor, *CT-RSA 2012*, volume 7178 of *LNCS*, pages 349–364. Springer, Heidelberg, February / March 2012. (Cited on page 1, 6, 16.)

[HKW15] Susan Hohenberger, Venkata Koppula, and Brent Waters. Universal signature aggregators. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 3–34. Springer, Heidelberg, April 2015. (Cited on page 6.)

[HMLS10] Dennis Hofheinz, John Malone-Lee, and Martijn Stam. Obfuscation for cryptographic purposes. *Journal of Cryptology*, 23(1):121–168, January 2010. (Cited on page 2.)

[HRsV11] Susan Hohenberger, Guy N. Rothblum, abhi shelat, and Vinod Vaikuntanathan. Securely obfuscating re-encryption. *Journal of Cryptology*, 24(4):694–719, October 2011. (Cited on page 2, 6, 16.)

[ID03] Anca Ivan and Yevgeniy Dodis. Proxy cryptography revisited. In *NDSS 2003*. The Internet Society, February 2003. (Cited on page 1, 6.)

[Jak99] Markus Jakobsson. On quorum controlled asymmetric proxy re-encryption. In Hideki Imai and Yuliang Zheng, editors, *PKC'99*, volume 1560 of *LNCS*, pages 112–121. Springer, Heidelberg, March 1999. (Cited on page 1.)

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pages 669–684. ACM Press, November 2013. (Cited on page 7.)

[LV08] Benoît Libert and Damien Vergnaud. Unidirectional chosen-ciphertext secure proxy re-encryption. In Ronald Cramer, editor, *PKC 2008*, volume 4939 of *LNCS*, pages 360–379. Springer, Heidelberg, March 2008. (Cited on page 6, 12, 16.)

[NX15] Ryo Nishimaki and Keita Xagawa. Key-private proxy re-encryption from lattices, revisited. *IEICE Transactions*, 98-A(1):100–116, 2015. (Cited on page 6.)

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Heidelberg, May 1999. (Cited on page 21, 22.)

[PRSV17] Yuriy Polyakov, Kurt Rohloff, Gyana Sahu, and Vinod Vaikuntanathan. Fast proxy re-encryption for publish/subscribe systems. *ACM Trans. Priv. Secur.*, 20(4):14:1–14:31, 2017. (Cited on page 1.)

A Re-Encryption Simulatability

We review the notion of re-encryption simulatability of PRE by Cohen. For the syntax and standard CPA-security of PRE, see previous works [Coh17, ABH09]. Roughly speaking, re-encryption simulatability means that a re-encrypted ciphertext generated from $\mathsf{rk}_{i \to j}$ and ct_i under pk_i can be simulated without $\mathsf{rk}_{i \to j}$ if pk_i, pk_j, ct_i, and m are given where ct_i is an encryption of m under pk_i. Moreover, the simulated re-encrypted ciphertext is *statistically* indistinguishable from the honestly generated re-encrypted ciphertext even if $\mathsf{sk}_i, \mathsf{sk}_j, \mathsf{rk}_{i \to j}$ are given as auxiliary information. The definition below is found in the paper by Choen [Coh17, Definition 7].

Definition A.1 (Re-Encryption Simulatability [Coh17]). *A proxy re-encryption scheme is re-encryption simulatable if there exists a PPT algorithm* $\mathsf{ReEncSim}$ *such that for all* $m \in \mathcal{M}$

$$(\mathsf{ReEncSim}(\mathsf{z}), \mathsf{z}) \overset{\mathsf{s}}{\approx} (\mathsf{ReEnc}(\mathsf{rk}_{i \to j}, \mathsf{ct}_i), \mathsf{z}),$$

where $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp}), (\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \mathsf{KeyGen}(\mathsf{pp}), \mathsf{rk}_{i \to j} \leftarrow \mathsf{ReKeyGen}(\mathsf{sk}_j, \mathsf{pk}_i), \mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{pk}_i, m), \mathsf{z} := (\mathsf{pp}, \mathsf{pk}_i, \mathsf{pk}_j, \mathsf{sk}_j, \mathsf{ct}_i, m)$.

Theorem A.2 ([Coh17]). *If a PRE scheme is PRE-CPA secure and re-encryption simulatable, then it is PRE-HRA secure.*

We can consider re-encryption simulatability for UPRE as Definition A.3.

Definition A.3 (Re-encryption simulatability for UPRE). *A UPRE scheme is re-encryption simulatable if there exists a PPT algorithm* $\mathsf{ReEncSim}$ *such that for all* $m \in \mathcal{M}$

$$(\mathsf{ReEncSim}(\mathsf{z}), \mathsf{z}) \overset{\mathsf{s}}{\approx} (\mathsf{ReEnc}(\mathsf{rk}_{f \to t}, \mathsf{ct}_f), \mathsf{z}),$$

where $(\mathsf{pk}_f, \mathsf{sk}_f) \leftarrow \mathsf{Gen}_{\sigma_f}(1^{\lambda_f}), (\mathsf{pk}_t, \mathsf{sk}_t) \leftarrow \mathsf{Gen}_{\sigma_t}(1^{\lambda_t}), \mathsf{rk}_{f \to t} \leftarrow \mathsf{ReKeyGen}(\mathsf{sk}_f, \mathsf{pk}_t), \mathsf{ct}_f \leftarrow \mathsf{Enc}(\mathsf{pk}_f, m), \mathsf{z} := (\mathsf{pk}_f, \mathsf{pk}_t, \mathsf{sk}_t, \mathsf{ct}_f, m)$.

Remark A.4. Cohen updated his paper and revised the re-encryption simulatability [Coh17]. In the latest definition, $\mathsf{ReEncSim}$ takes sk_j as an input too, and $(\mathsf{sk}_i, \mathsf{rk}_{i \to j})$ are not given as auxiliary input as above.

A.1 Our Relaxed UPRE schemes are not Re-Encryption Simulatable

A re-encrypted ciphertext of $\mathsf{UPRE}_{\mathsf{gc}}$ is $\mathsf{rct} = (\widehat{\mathsf{ct}}_t, \widetilde{\mathsf{ct}}_t, \widetilde{\mathsf{C}}_i, \ldots, \widetilde{\mathsf{C}}_1)$ where $\widehat{\mathsf{ct}}_t \leftarrow \mathsf{BatchEnc}(\hat{\mathsf{pk}}, \mathsf{labels})$, $\widetilde{\mathsf{ct}}_t \leftarrow \mathsf{Enc}_t(\mathsf{pk}_t, \hat{\mathsf{sk}})$, $(\widetilde{\mathsf{C}}_i, \mathsf{labels}) \leftarrow \mathsf{Garble}(\mathsf{C}_i)$, and $\mathsf{C}_1 := \mathsf{P}[s_2, \mathsf{ct}_f]$ and $\mathsf{C}_i \leftarrow \mathsf{Q}[s_2, \widehat{\mathsf{ct}}', \widetilde{\mathsf{ct}}_f]$ for $i > 1$. We can simulate ct_f since we have m. However, we do not know how to simulate $\widetilde{\mathsf{C}}_i$ in a *statistically* indistinguishable way because a simulator $\mathsf{ReEncSim}$ does not have sk_f in the re-encryption simulatability setting. Due to a similar reason, $\mathsf{UPRE}_{\mathsf{cra}}$ does not satisfy Definition A.3.

However, as we see in the proof of Theorem 6.3 (in particular, Lemmata 6.5 and 6.6), we can simulate $\tilde{C}_i$ in a *computationally* indistinguishable way by using the security of GC and weak batch encryption. The point is that sk_f (delegator's key) is not revealed to adversaries though sk_t (delegatee's key) is revealed when we consider honest re-encryption attacks. Moreover, in that case (delegatee's key is corrupted), the re-encryption key $\mathsf{rk}_{f \to t} = (\hat{\mathsf{pk}}, s_2, \widetilde{\mathsf{ct}}_t)$ is not given to adversaries. That is, we do not need s_2 when delegator/delegatee are honest/corrupted, respectively. Therefore, we can simulate the honest encryption oracle in a indistinguishable way in the proof of Theorem 6.3 without re-encryption simulatability. We can observe a similar fact in the proof of Theorem 7.1.

Based on the observation above, it seems that giving sk_f and $\mathsf{rk}_{f \to t}$ to adversaries makes the re-encryption simulatability stronger. Moreover, there is a possibility to weaken re-encryption simulatability, yet the weaker simulatability still implies the HRA security. We introduce such a weaker simulatability in the next section.

A.2 Weak Re-Encryption Simulatability

We can consider a weak re-encryption simulatability for UPRE (and PRE).

Definition A.5 (Weak Re-encryption simulatability for UPRE). *Let* $\mathsf{ReEncSim}$ *be a PPT simulator. We define the following experiments* $\mathsf{Exp}_{\mathcal{D}}^{\mathsf{w\text{-}re\text{-}sim}}(1^\lambda, b)$ *between a challenger and a distinguisher* $\mathcal{D}$ *as follows.*

1. *The challenger generates* $(\mathsf{pk}_f, \mathsf{sk}_f) \leftarrow \mathsf{Gen}_f(1^{\lambda_f})$, $(\mathsf{pk}_t, \mathsf{sk}_t) \leftarrow \mathsf{Gen}_t(1^{\lambda_t})$, *and sends* $(1^{\lambda_f}, \mathsf{pk}_f, 1^{\lambda_t}, \mathsf{pk}_t, \mathsf{sk}_t)$ *to* $\mathcal{D}$.

2. *The challenger and* $\mathcal{D}$ *do the setup phase as in Definition 3.9 and set* $\mathsf{HList} := \mathsf{HList} \cup \{f\}$ *and* $\mathsf{CList} := \mathsf{CList} \cup \{t\}$.

3. $\mathcal{D}$ *has the re-encryption key oracle* $\mathcal{O}_{\mathsf{rekey}}$ *as in Definition 3.9.*

4. $\mathcal{D}$ *chooses a message* $m \in \mathcal{M}_f$, *generates a ciphertext* $\mathsf{ct}_f \leftarrow \mathsf{Enc}_f(\mathsf{pk}_f, m)$ *and sends* (m, ct_f) *to the challenger.*

5. *If* $b = 0$, *the challenger computes* $\mathsf{rk}_{f \to t} \leftarrow \mathsf{ReKeyGen}(\mathsf{sk}_f, \mathsf{pk}_t)$ *and* $\mathsf{ct}^* \leftarrow \mathsf{ReEnc}(\mathsf{rk}_{f \to t}, \mathsf{ct}_f)$ *and returns* ct^* *to* $\mathcal{D}$. *Otherwise, the challenger returns* $\mathsf{ct}^* \leftarrow \mathsf{ReEncSim}(\mathsf{pk}_f, \mathsf{pk}_t, \mathsf{ct}_f, m)$.

6. $\mathcal{D}$ *outputs* $b' \in \{0,1\}$. *The experiment outputs* b'.

We say that UPRE *is weakly re-encryption simulatable if there exists a simulator* ReEncSim, *for any PPT* $\mathcal{D}$, *it holds that*

$$|\Pr[\mathsf{Exp}_{\mathcal{D}}^{\mathsf{w\text{-}re\text{-}sim}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Exp}_{\mathcal{D}}^{\mathsf{w\text{-}re\text{-}sim}}(1^\lambda, 1) = 1]| \leq \mathsf{negl}(\lambda).$$

The difference between the re-encryption simulatability and a weak one is that the indistinguishability is only computational. Moreover, the distinguisher is given oracle access to the re-encryption key oracle $\mathcal{O}_{\mathsf{rekey}}$. Note that $\mathcal{O}_{\mathsf{rekey}}$ does not give $\mathsf{rk}_{f \to t}$ since $f \in \mathsf{HList} \wedge t \in \mathsf{CList}$. This weak variant is sufficient to prove UPRE-HRA security. That is, we can prove that if a UPRE scheme is UPRE-CPA secure and weakly re-encryption simulatable, then it is UPRE-HRA secure.

Theorem A.6. *If a UPRE scheme* UPRE *is multi-hop selectively UPRE-CPA secure and satisfies weak re-encryption simulatability, then* UPRE *is multi-hop selectively UPRE-HRA secure.*

Proof. We define hybrid games.

$\mathsf{Hyb}_{\mathcal{A}}^0(b)$: The first experiment is the original security experiment for b, $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{ms\text{-}upre\text{-}hra}}(1^\lambda, b)$. That is, it holds that $\mathsf{Hyb}_{\mathcal{A}}^0(b) = \mathsf{Exp}_{\mathcal{A}}^{\mathsf{ms\text{-}upre\text{-}hra}}(1^\lambda, b)$. Note that in the successive experiments, we can easily simulate all keys outside of G^* since vertices in $V \setminus V^*$ are not connected to the target vertex and simulators can generate keys for them by itself.

$\mathsf{Hyb}_{\mathcal{A}}^1(b)$: This experiment is the same as $\mathsf{Hyb}_{\mathcal{A}}^0(b)$ except that

1. we record not only $(\mathsf{ct}_i, \Sigma_i, i, \#\mathsf{CT})$ but also m in KeyCTList for honest encryption query (i, m) and
2. for re-encryption query (i, j', k) such that $j' \in \mathsf{CList} \wedge k \notin \mathsf{Drv}$, the re-encrypted ciphertext is differently generated as follows. First, we retrieve $(\mathsf{ct}_i, \Sigma_i, i, \#\mathsf{CT} = k, m)$ from KeyCTList (if there is no such an entry, just outputs $\bot$). Then, we compute the following value instead of computing $\mathsf{rk}_{i \to j'}$.
   (a) $\mathsf{rct} \leftarrow \mathsf{ReEncSim}(\mathsf{pk}_i, \mathsf{pk}_{j'}, \mathsf{ct}_i, m)$.

   Finally, we set rct as a re-encrypted ciphertext for user j' and send it to $\mathcal{A}$.

Note that for (i, j') such that $i \in \mathsf{HList} \wedge j' \in \mathsf{CList}$, we do not need sk_i and $\mathsf{rk}_{i \to j'}$ since we just output $\bot$ for such re-encryption key query (i, j'). The change above is for ciphertexts that $\mathcal{A}$ can decrypt. Here, $\mathcal{A}$ can obtain $\mathsf{sk}_{j'}$ since user j' is corrupted. However, it is not an issue since a distinguisher is given $\mathsf{sk}_{j'}$ as auxiliary input in the weak re-encryption simulatability game. In Lemma A.7, we prove that $\mathsf{Hyb}_{\mathcal{A}}^1(b) \overset{\mathsf{s}}{\approx} \mathsf{Hyb}_{\mathcal{A}}^0(b)$ holds due to the weak re-encryption simulatability.

In Lemma A.8, we prove that $\mathsf{Hyb}_{\mathcal{A}}^1(0) \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_{\mathcal{A}}^1(1)$ holds due to the UPRE-CPA security of Σ_{i^*}. Therefore, it holds that $\mathsf{Hyb}_{\mathcal{A}}^0(0) \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_{\mathcal{A}}^0(1)$ by Lemmata A.7 and A.8 ■

Lemma A.7. *If* UPRE *is weakly re-encryption simulatable, then it holds* $\mathsf{Hyb}_{\mathcal{A}}^0(b) \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_{\mathcal{A}}^1(b)$.

Proof. In fact, we use q' intermediate hybrids to prove this where q' is the number of uncorrupted key pk_i such that re-encryption query (i, j', k) is sent and $i \in \mathsf{HList} \wedge j \in \mathsf{CList} \wedge k \notin \mathsf{Drv}$. For each hybrid, we use the weak re-encryption simulatability. Below, we write only the case for one pk_i for simplicity.

We construct a distinguisher $\mathcal{D}$ of the weak re-encryption simulatability. To use $\mathcal{A}$ of UPRE, $\mathcal{D}$ generates key pairs $(\mathsf{pk}_{i'}, \mathsf{sk}_{i'})$ for all $i' \in \mathsf{HList} \setminus \{i\}$ and $i' \in \mathsf{CList} \setminus \{j'\}$. For $\mathsf{pk}_i, \mathsf{pk}_{j'}, \mathsf{sk}_{j'}$, we use keys $(1^{\lambda_i}, \mathsf{pk}_i, 1^{\lambda_{j'}} \mathsf{pk}_{j'}, \mathsf{sk}_{j'})$ from the challenger, which is given to $\mathcal{D}$. The only issue is that we do not have sk_i and $\mathsf{rk}_{i \to j'}$. First, we do not need $\mathsf{rk}_{i \to j'}$ since $i \in \mathsf{HList} \wedge j' \in \mathsf{CList}$. Second, for re-encryption keys $(i, \hat{j})$ such that $\hat{j} \in \mathsf{HList}$, $\mathcal{D}$ passes the query $(i, \hat{j})$ to the re-encryption key oracle $\mathcal{O}_{\mathsf{rekey}}$ in the weak re-encryption simulatability game, receives $\mathsf{rk}_{i \to \hat{j}}$, and returns it to $\mathcal{A}$. This is possible since $i, \hat{j} \in \mathsf{HList}$. Therefore, $\mathcal{B}$ can simulate all oracles.

However, to use $\mathcal{A}$, $\mathcal{D}$ simulates $\mathcal{O}_{\mathsf{reenc}}$ in a slightly different way. As we define $\mathsf{Hyb}^1_{\mathcal{A}}(b)$, the simulation for query (i, j', k) to $\mathcal{O}_{\mathsf{reenc}}$ such that $j' \in \mathsf{CList} \wedge k \notin \mathsf{Drv}$ and $(\mathsf{ct}_i, \Sigma_i, i, \#\mathsf{CT}, m) \in \mathsf{KeyCTList}$ is different. When $\mathcal{D}$ receives a re-encryption query for such (i, j', k), $\mathcal{D}$ generates $\mathsf{ct}_i \leftarrow \mathsf{Enc}_i(\mathsf{pk}_i, m)$ and sends it to the challenger of the weak re-encryption simulatability game. If $\mathcal{D}$ is given ct^*, then $\mathcal{D}$ returns ct^* as a re-encrypted ciphertext for (i, j', k) Note that $\mathcal{B}$ does not need sk_i for this query. This completes the simulation. If $\mathsf{ct}^* = \mathsf{ReEnc}(\mathsf{rk}_{i \to j'}, \mathsf{ct}_i)$ where $\mathsf{rk}_{i \to j'} \leftarrow \mathsf{ReKeyGen}(\mathsf{sk}_i, \mathsf{pk}_{j'})$, then the view is totally the same as $\mathsf{Hyb}^0_{\mathcal{A}}(b)$. If $\mathsf{ct}^* \leftarrow \mathsf{ReEncSim}(\mathsf{pk}_i, \mathsf{pk}_{j'}, \mathsf{ct}_i, m)$, then the view is totally the same as $\mathsf{Hyb}^1_{\mathcal{A}}(b)$. Therefore, if $\mathcal{A}$ can distinguishes two experiment, $\mathcal{D}$ can break the weak re-encryption simulatability. ■

Lemma A.8. *If* UPRE *is UPRE-CPA secure, then it holds* $\mathsf{Hyb}^1_{\mathcal{A}}(0) \overset{\mathsf{c}}{\approx} \mathsf{Hyb}^1_{\mathcal{A}}(1)$.

Proof. We construct an adversary $\mathcal{B}$ of UPRE-CPA, which is given oracle access to $\mathcal{O}_{\mathsf{rekey}}, \mathcal{O}_{\mathsf{reenc}}, \mathcal{O}_{\mathsf{cha}}$ and can send hones/corrupted key queries. To use a distinguisher $\mathcal{A}$ of these two hybrids, $\mathcal{B}$ must simulate oracles of the HRA security. Basically, $\mathcal{B}$ can easily simulate them by using its oracles except $\mathcal{O}_{\mathsf{enc}}$ and $\mathcal{O}_{\mathsf{reenc}}$ (note that re-encryption key oracles in the CPA/HRA-security are the same). Moreover, it is easy to simulate $\mathcal{O}_{\mathsf{enc}}$ since all encryption keys are public. The only issue is the simulation of $\mathcal{O}_{\mathsf{reenc}}$ in the case that re-encryption queries (i, j', k) such that $j' \in \mathsf{CList} \wedge k \notin \mathsf{Drv}$ are sent. This is already solved since we use $\mathsf{ReEncSim}$ in these hybrids. Thus, $\mathcal{B}$ can simulate all oracles by using its oracles and $\mathsf{ReEncSim}$.

When (i^*, m_0, m_1) is queried to the challenge oracle $\mathcal{O}_{\mathsf{cha}}$, then $\mathcal{B}$ passes (i^*, m_0, m_1) to the challenger of the CPA game and receives a target ciphertext $\mathsf{ct}^*_{i^*}$. $\mathcal{B}$ returns $\mathsf{ct}^*_{i^*}$ to $\mathcal{A}$. If $\mathcal{A}$ can distinguish two experiments, then $\mathcal{B}$ can break the CPA security since $\mathsf{ct}^*_{i^*} = \mathsf{Enc}_{i^*}(\mathsf{pk}_{i^*}, m_0)$ and $\mathsf{ct}^*_{i^*} = \mathsf{Enc}_{i^*}(\mathsf{pk}_{i^*}, m_1)$ perfectly simulate $\mathsf{Hyb}^1_{\mathcal{A}}(0)$ and $\mathsf{Hyb}^1_{\mathcal{A}}(1)$, respectively. ■