# Universal Proxy Re-Encryption

Nico Döttling[1] and Ryo Nishimaki[2]

[1] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
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 a proxy 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)*. The proxy has a re-encryption key generated from the delegator's secret key and the delegatee public key. Thus UPRE generalizes proxy re-encryption by supporting arbitrary PKE schemes and allowing to convert ciphertexts into ones of *possibly different PKE schemes*. In this work, we

- provide syntax and definitions for both UPRE and a variant we call relaxed UPRE. The relaxed variant means that decryption algorithms for re-encrypted ciphertexts are slightly modified but still only use the original delegatee secret keys for decryption.
- construct a UPRE based on probabilistic indistinguishability obfuscation (PIO). It allows us to re-encrypt ciphertexts polynomially many times.
- construct relaxed UPRE from garbled circuits (GCs). We provide two variants of this construction, one which allows us to re-encrypt ciphertexts polynomially many times, and a second one which satisfies a stronger security requirement but only allows us to re-encrypt ciphertexts a constant number of times.

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

## 1 Introduction

### 1.1 Background

Constructing cryptographic systems from scratch is a challenging task. When migrating from a legacy cryptosystem to a new one with better security and functionality, it would be desirable to reuse existing public key infrastructures (PKI) to reduce the cost of migration. In this work, we explore a *universal* methodology to construct a new and easily deployable cryptographic system from existing cryptographic systems and PKI.

As a particular example of cryptographic systems, we consider proxy re-encryption (PRE) [BBS98]. PRE allows to convert a ciphertext under public key $\mathsf{pk}_f$ (we call delegator public key and $f$ denotes "from") into another ciphertext under public key $\mathsf{pk}_t$ (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 $\mathsf{sk}_f$ (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 thus enables delegation of re-encryption and several useful applications. 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 (some additional conversion is needed). Moreover, only delegatees (receivers of converted ciphertexts) can select any PKE scheme and delegators (senders of original ciphertexts) cannot. From a practical point of view, this is unsatisfactory as we need to build a new system using a PRE scheme from scratch if we want 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 the applications of PRE. Ideally, we would like to achieve a re-encryption mechanism that works for any pair of PKE schemes without any modification and setup.

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

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* $\mathsf{sk}_t$. We call this variant *relaxed UPRE*. Here, we emphasize that the delegator uses only $\mathsf{pk}_f$ and $\mathsf{Enc}_f$ to encrypt a message and the delegatee uses only $\mathsf{sk}_t$ 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. In addition, 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 vulnerability is found in an encryption system. That is, we can outsource renewing encrypted storage 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 $\mathsf{pk}_f$ and $\mathsf{pk}_t$ 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. We consider unidirectional single/constant/multi-hop but do not focus on bidirectional since the functionality of a bidirectional re-encryption key is simulated by two unidirectional re-encryption keys.

The main question addressed in this work is how to achieve UPRE. Regarding feasibility, it seems plausible that UPRE can be achieved from indistinguishability obfuscation (IO) [BGI+12,GGH+16] or multilinear maps [GGH13,CLT13,GGH15].[3]. And in fact, we present a construction based on IO as an initial step, though we emphasize that formally proving security is not a trivial task even if we use IO. Consequently, the main focus of this work is concerned with the following question.

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

We give a positive answer to this question.

### 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 present a general construction of multi-hop UPRE for some class of PKE by using probabilistic IO (PIO).
3. 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.
4. 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.

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

The third contribution is notable since we introduce a new design idea and use only weak 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 [Coh19][4]. Thus, we define security against honest re-encryption attacks for UPRE (UPRE-HRA), which implies UPRE-CPA, instead of CPA-security. 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.[5][6] See Section 2 for details.

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. The required properties for PKE schemes depend on the security level of PIO. If we assume additional properties on PKE, then we can achieve UPRE from sub-exponentially secure IO (sub-exp IO) and sub-exponentially secure OWF (sub-exp OWF). Most well-known CPA-secure PKE schemes such as ElGamal, Goldwasser-Micali PKE schemes satisfy the additional properties. However, if we use any PKE, we need PIO with the strongest security for specific circuits (refer to [CLTV15]). If we use the exponential DDH assumption, we can achieve UPRE from any PKE and polynomially secure IO. The advantage of the scheme based on PIO is that it is a multi-hop UPRE scheme and conceptually simple.

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 exist if one-way functions exist (a very weak cryptographic assumption). This relaxed UPRE scheme satisfies HRA-security. However, some meta information (all garbled circuits from the first delegator to the last delegatee)

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

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

[6] Davidson, Deo, Lee, and Martin [DDLM19] independently introduced a stronger notion called strong post-compromised security in the *standard PRE setting*. Note that our work appeared before their publication. Our work appeared on September 7th in 2018 while their work [DDLM19] did on April 5th in 2019. (See the submission dates on Cryptology ePrint Archive.)

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 do not know how to 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.

### 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.

*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 $\mathsf{C}_{\mathsf{re}}$, in which a delegator secret key $\mathsf{sk}_f$ and a delegatee public key $\mathsf{pk}_t$ are hard-wired in and which takes a delegator ciphertext $\mathsf{ct}_f$ as an input. The re-encryption circuit decrypts $\mathsf{ct}_f$ by using $\mathsf{sk}_f$, obtains a plaintext $m$, and generates a ciphertext of $m$ under $\mathsf{pk}_t$. We can hide information about $\mathsf{sk}_f$ by using PIO (note that $\mathsf{C}_{\mathsf{re}}$ 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 $\mathsf{pk}_t$. 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 $\mathsf{sk}_f$, we use a dummy re-encryption circuit that does not run the decryption algorithm of $\Sigma_f$ with $\mathsf{sk}_f$ and just outputs a dummy ciphertext under $\mathsf{pk}_t$ (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* $\mathsf{pk}_t$. 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+18]. 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. Thus, we proceed as follows. First, we change the hard-wired public key $\mathsf{pk}_t$ into a trapdoor key $\mathsf{tk}_t$. Second, we use the security of PIO. The indistinguishability under $\mathsf{tk}_t$ is used to satisfy the condition of PIO.

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 delegator's PKE scheme. Those processes are the notable differences from the security proof of FHE based on PIO by Canetti et al. [CLTV15]. The point is that one vertex can be connected to multiple vertices in the multi -hop (U)PRE setting.

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 [ACH20] 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 the 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 in the following two steps. First, we generate shares $(s_1, s_2)$ of a delegator secret key $\mathsf{sk}_f$ by a secret sharing scheme.

We encrypt share $s_1$ by using $\mathsf{pk}_t$ 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 $\mathsf{sk}_f$ by the privacy property of secret sharing. We define a circuit $\mathsf{C}^{\mathsf{re}}_{\mathsf{de}}$ where $s_2$ and the delegator ciphertext $\mathsf{ct}_f$ are hard-wired. The circuit $\mathsf{C}^{\mathsf{re}}_{\mathsf{de}}$ takes as input $s_1$, reconstructs $\mathsf{sk}_f$ 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 $\{\mathsf{labels}_{i,b}\}_{i\in[|s_1|]b\in\{0,1\}}$. We set a re-encrypted ciphertext to $\mathsf{rct} := (\widetilde{\mathsf{ct}}_t, \widetilde{\mathsf{C}^{\mathsf{re}}_{\mathsf{de}}}, \{\mathsf{labels}_{i,b}\})$ (we omit $\{i \in [|s_1|], b \in \{0,1\}\}$ 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 does not work since sending $\{\mathsf{labels}_{i,b}\}$ breaks the security of GC and $\mathsf{sk}_f$ 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 $2\lambda$ 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 3.1 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 $\mathsf{pk}_t$. That is, we set $\mathsf{rk}_{f\to t} := (\hat{\mathsf{pk}}, s_2, \mathsf{Enc}(\mathsf{pk}_t, \hat{\mathsf{sk}}))$. 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 $\mathsf{rct} := (\widetilde{\mathsf{ct}}_t, \hat{\mathsf{ct}}, \widetilde{\mathsf{C}^{\mathsf{re}}_{\mathsf{de}}})$, where $\widetilde{\mathsf{ct}}_t \leftarrow \mathsf{Enc}(\mathsf{pk}_t, \hat{\mathsf{sk}})$.

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 $\mathsf{sk}_t$, 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 $\mathsf{rk}_{f\to t}$ does not reveal information about $\mathsf{sk}_f$ 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 $\mathsf{sk}_f$ from the other share $s_2$ by the privacy property of the secret sharing scheme. That is, we can erase information about $\mathsf{sk}_f$ and can use the CPA-security of

$\mathsf{pk}_f$. 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 3 for the detail. We note that the secret sharing mechanism was used in previous (non-universal) PRE schemes [CWYD10,HKK+12]. (The technique is called token-controlled technique in some papers.) However, using garbled circuits and batch encryption is new in the PRE setting.

In the construction above, a delegator might obtain information about the plaintext since the re-encrypted ciphertext includes $\mathsf{ct}_f$ in the garbled circuit and the delegator has $\mathsf{sk}_f$. We have no way to prove that the construction above satisfies 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 under the delegatee's public key since we can hide $\mathsf{ct}_f$ by using the security of the delegatee's PKE scheme. 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.

| Instantiation | Type | #Hop | Security | Supported PKE | Assumptions |
|---|---|---|---|---|---|
| Ours from IO + [CLTV15] | UPRE | multi | HRA & CRA | 0-hiding trapdoor | sub-exp IO and OWF |
| Ours from IO + [CLTV15] | UPRE | multi | HRA & CRA | any IND-CPA | di-PIO and OWF |
| Ours from IO + [ACH20] | UPRE | multi | HRA & CRA | any IND-CPA | IO and exp. DDH |
| Ours in Sec. 3 | rUPRE | multi | HRA | any IND-CPA | PKE |
| Ours in Sec. 4 | 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. Hohenberger 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) 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,DWLC08,SC09,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 introduce the syntax and security definitions of UPRE. In Section 3, we present our relaxed UPRE scheme based on GC, and prove its security. In Section 4, we present our CRA-secure relaxed UPRE scheme. We omit many contents (basic preliminaries) due to space limitations. In particular, we omit our UPRE scheme based on IO. See the full version of this paper for omitted contents.

## 2 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 $\Sigma_f$ into ciphertexts of a (possibly) different PKE scheme $\Sigma_t$. 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 $\mathsf{pk}_f$ will be converted into a ciphertext $\mathsf{pk}_t$. We assume that in the description of $\Sigma_{\sigma_i}$, ciphertext space $\mathcal{C}_{\sigma_i}$ and message space $\mathcal{M}_{\sigma_i}$ are also included. When we use $\Sigma_{\sigma_i}$ as an input for algorithms of $\mathsf{UPRE}$, we interpret it as a description of algorithms (rather than Turing machines or circuits). Note that the length of such descriptions is polynomial since algorithms of PKE should be PPT.

### 2.1 Unidirectional UPRE

**Definition 2.1 (Universal Proxy Re-Encryption for PKE: Syntax).** *A universal re-encryption scheme* $\mathsf{UPRE}$ *consists of two PPT 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* $\mathsf{sk}_f$ *of* $\Sigma_{\sigma_f}$*, and a public-key* $\mathsf{pk}_t$ *of* $\Sigma_{\sigma_t}$ *and outputs a re-encryption key* $\mathsf{rk}_{f\to t}$ *for ciphertexts under* $\mathsf{pk}_f$*. 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* $\mathsf{ct}_f$ *under* $\mathsf{pk}_f$ *of* $\Sigma_{\sigma_f}$*, and outputs a re-encrypted ciphertext* $\mathsf{ct}_t$ *under* $\mathsf{pk}_t$*.*

**Definition 2.2 (Relaxed Universal Proxy Re-Encryption for PKE: Syntax).** *A relaxed universal re-encryption scheme* $\mathsf{UPRE}$ *consists of two PPT and one deterministic polynomial-time 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 Def. 2.1.*
- $\mathsf{ReEnc}(\Sigma_{\sigma_f}, \Sigma_{\sigma_t}, \mathsf{rk}_{f\to t}, \mathsf{ct}_f)$ *takes a pair of PKE schemes* $(\Sigma_{\sigma_f}, \Sigma_{\sigma_t})$*, a re-encryption key* $\mathsf{rk}_{f\to t}$*, and a ciphertext* $\mathsf{ct}_f$ *under* $\mathsf{pk}_f$ *of* $\Sigma_{\sigma_f}$*, and outputs a re-encrypted ciphertext* $\mathsf{rct}$*. We implicitly assume that* $\mathsf{rct}$ *includes index* $\ell$ *which indicates how many times* $\mathsf{ReEnc}$ *was applied so far. When we write* $\mathsf{rct}^{(\ell)}$*, it means that* $\mathsf{rct}^{(\ell)}$ *was obtained by applying* $\mathsf{ReEnc}$ $\ell$ *times.*
- $\mathsf{mDec}(\Sigma_{\sigma_t}, \mathsf{sk}_t, \mathsf{rct}^{(\ell)}, \ell)$ *is a deterministic algorithm and takes a PKE scheme* $\Sigma_{\sigma_t}$*, a secret key* $\mathsf{sk}_t$*, a re-encrypted ciphertext* $\mathsf{rct}^{(\ell)}$ *under* $\mathsf{rk}_{f\to t}$*, and index* $\ell$ *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 $\Sigma_{\sigma_t}$ 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 $\mathsf{sk}_t$. 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 $\ell$ 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 $\mathsf{pk}_f$ (resp. $\mathsf{pk}_t$) into ciphertexts that can be decrypted by $\mathsf{sk}_t$ (resp. $\mathsf{sk}_f$). Although unidirectional UPRE can support the functionality of bidirectional UPRE by generating two re-encryption keys $\mathsf{rk}_{f\to t}$ and $\mathsf{rk}_{t\to f}$, it is not clear whether security is preserved. 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. Correctness is easy to understand.

This HRA for UPRE is based on security against HRA of PRE introduced by Cohen [Coh19]. 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[7]. 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).

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 2.3 (UPRE for PKE: Single-Hop Correctness).** *A relaxed UPRE scheme* $\mathsf{UPRE}$ *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* $\Sigma' \coloneqq (\Sigma_{\sigma_f}, \Sigma_{\sigma_t})$. *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 [Coh19] may be able to skip explanations below and jump into the formal definition. Readers who are familiar with PRE-CPA security [ABH09,Coh19] may

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

be able to skip explanations below except "Honest encryption and re-encryption query" part.

**Challenge query:** We consider a natural extension of the CPA security of PKE. The adversary selects a target public-key $\mathsf{pk}_{i^*}$ indexed by $i^*$ and tries to distinguish whether a target ciphertext $\mathsf{ct}_{i^*}$ 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 $\mathsf{pk}_i$ 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 $\mathsf{sk}_j$. 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 [Coh19] 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 [Coh19], 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. $(i, j, \mathsf{ct}_i)$) to the re-encryption key generation (resp. re-encryption) oracle [Coh19]. 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 $\mathsf{Drv}$. 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 2.4 (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 2.5 (UPRE for PKE: Single-Hop selective HRA Security).** *We define the experiment* $\mathsf{Exp}^{\mathsf{upre\text{-}hra}}_{\mathcal{A}}(1^\lambda, b)$ *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* $\#\mathsf{Keys} := 0, \mathsf{HList} := \emptyset, \mathsf{CList} := \emptyset$, $\#\mathsf{CT} := 0$, $\mathsf{KeyCTList} := \emptyset$, $\mathsf{Drv} := \emptyset$. *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* $(i, \sigma_i, \lambda_i)$ *from* $\mathcal{A}$*, if the challenger already received* $(i, *, *)$ *before, it outputs* $\perp$*. Otherwise, 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* $\#\mathsf{Keys} := \#\mathsf{Keys} + 1$*. If* $\lambda_i < \lambda$*, then the challenger ignores the query.*[8]
- *For a corrupted key query* $(i, \sigma_i, \lambda_i)$ *from* $\mathcal{A}$*, if the challenger already received* $(i, *, *)$ *before, it outputs* $\perp$*. Otherwise, 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* $\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_{\#\mathsf{Keys}}}$*. At the end of Phase 1, we assume that the list* $((1, \sigma_1), \ldots, (\#\mathsf{Keys}, \sigma_{\#\mathsf{Keys}}))$ *is broadcasted and all entities know it.*

**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* $i \in \mathsf{HList} \wedge j \in \mathsf{CList}$*. 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* $\mathcal{A}$.

$\mathcal{O}_{\mathsf{reenc}}(i, j, k)$**:** *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* $\mathsf{ct}_i$ *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* $b'$.

[8] If we prefer longer security parameters, then we can change the condition to $\lambda_i < c\lambda$ for some constant $c > 1$.

*We say the* $\mathsf{UPRE}$ *is single-hop UPRE-HRA secure if, for any* $\sigma_i \in [N]$*, for any PPT* $\mathcal{A}$*, it holds that*

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

*Discussion on Def. 2.5.* (1) On security parameter: We can simply set $\forall i\ \lambda_i := \lambda$. Some $\lambda_j$ may be longer than other $\lambda_i$ (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 $\bot$ 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 [FKKP19]. We do not know whether the techniques by Fuchbauer et al. are applicable to the UPRE setting. This is an interesting future work. 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,Coh19].

Knowledgeable readers might think a UPRE definition based on the PRE definition by Chow et al.[CWYD10] is better than the definition above. In the PRE setting, the defition by Chow et al. might be stronger than that by Cohen. However, the relationship between them is not formally studied. Thus, which definition is better or not is out of scope of this paper.

### 2.2 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 $\mathsf{rct}$ 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 2.6 (UPRE for PKE: $L$-hop Correctness).** *A multi-hop UPRE scheme* $\mathsf{mUPRE}$ *for PKE is L-hop correct if for all PKE schemes* $(\Sigma_{\sigma_0}, \Sigma_{\sigma_1}, \ldots, \Sigma_{\sigma_L})$ *that satisfy correctness and* $\sigma_{i-1} \neq \sigma_i$ *for all* $i \in [L]$*,* $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}_{\sigma_i}(1^{\lambda_i})$ *(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 $\mathsf{mDec}$ is indexed by $j$ is that the decryption procedure for $j$-times re-encrypted ciphertexts might be different. See Section 3 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[9].

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 $\mathsf{CList}$. We denote by $i \rightsquigarrow j$ there exists a path from vertex $i$ to vertex $j$ in $G$.

**Definition 2.7 (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 $i^* \in \mathsf{HList}$ (honest user set fixed at the setup phase) to $j^* \in \mathsf{CList}$ such that the path includes edge $(i, j)$ as an intermediate edge (this includes the case $j = j^*$). That is, no $i^* \in \mathsf{HList}$, $j^* \in \mathsf{CList}$ such that a path $i^* \rightsquigarrow j^*$ exists in $G' = (V, E \cup (i, j))$.*

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 graph $G^* = (V^*, E^*)$ at the beginning of an experiment. To formally define this model, we define a *deviating pair with respect to $G^*$ and $G$.*

**Definition 2.8 (deviating pair).** *We say that $(i, j)$ is a deviating pair with respect to $G^* = (V^*, E^*)$ and $G = (V, E)$ in the selective-graph model if $i \in V^* \wedge j \in V$ or $j \in V^* \wedge i \in V$.*

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 and re-encrypted ciphertexts from $\mathcal{O}_{\mathsf{rekey}}$ and $\mathcal{O}_{\mathsf{reenc}}$, respectively, if queried $(i, j)$ is a deviating pair. That is, the structure of DAG that is connected to the target vertex must be determined at the beginning of the game. We focus on security in the selective-graph model in this study since it is what our schemes achieve. For admissible edges in the selective-graph model, we consider $i^* \in V_{\mathsf{h}}^*$ (defined below) instead of $i^* \in \mathsf{HList}$ (i.e., replacing $\mathsf{HList}$ with $V_{\mathsf{h}}^*$ in Def. 2.7).

**Definition 2.9 (UPRE for PKE: Multi-Hop selective-graph HRA Security).** *We define the experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{upre\text{-}msg\text{-}hra}}(1^\lambda, b)$ 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.*

[9] The circular security issue arises in constructions that use general PKE schemes. If there exists a cycle, we have no way to use the CPA-security of a PKE scheme in the cycle since the information of each secret key in the cycle is in a re-encryption key in the cycle. This does not happen in concrete constructions based on some hard problems such as the DDH.

- *The challenger initializes* $\#\mathsf{Keys} := 0, \mathsf{HList} := \emptyset, \mathsf{CList} := \emptyset, \#\mathsf{CT} := 0, \mathsf{KeyCTList} := \emptyset, \mathsf{Drv} := \emptyset, V := \emptyset, E := \emptyset$.
- *At the beginning of this phase,* $\mathcal{A}$ *must commit a graph* $G^* = (V^* = (V_{\mathsf{h}}^*, V_{\mathsf{c}}^*), E^*)$. *We assume that* $V^* = \{1, \ldots, |V^*|\}$ *by using appropriate renaming. If there is an edge* $(i,j) \in E^*$ *such that* $i \in V_{\mathsf{h}}^* \wedge j \in V_{\mathsf{c}}^*$, *then the game aborts. The challenger generates keys* $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}_{\sigma_i}(1^{\lambda_i})$ *for all* $i \in V^*$ *and sends* $\{\mathsf{pk}_i\}_{i \in V_{\mathsf{h}}^*}, \{(\mathsf{pk}_j, \mathsf{sk}_j)\}_{j \in V_{\mathsf{c}}^*}$ *to* $\mathcal{A}$. *We assume that* $\mathcal{A}$ *selects* $(\sigma_i, \lambda_i)$ *for all* $i \in V^*$ *as the key generation queries below (if* $\lambda_i < \lambda$ *for* $i \in V_{\mathsf{h}}^*$, *then the game aborts). The challenger also generates* $\mathsf{rk}_{i \to j} \leftarrow \mathsf{ReKeyGen}(\Sigma_{\sigma_i}, \Sigma_{\sigma_j}, \mathsf{sk}_i, \mathsf{pk}_j)$ *for all* $(i,j) \in E^*$ *and sends them to* $\mathcal{A}$. *The challenger sets* $\mathsf{HList} := \mathsf{HList} \cup V_{\mathsf{h}}^*$, $\mathsf{CList} := \mathsf{CList} \cup V_{\mathsf{c}}^*$, *and* $\#\mathsf{Keys} := \#\mathsf{Keys} + |V^*|$.
- *For the* $i$*-th honest key generation query* $(\sigma_i, \lambda_i)$ *from* $\mathcal{A}$, *if* $\lambda_i < \lambda$, *the challenger outputs* $\bot$. *Otherwise, 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 the* $j$*-th corrupted key generation query* $(j, \sigma_j, \lambda_j)$ *from* $\mathcal{A}$, *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* $i \leq \#\mathsf{Keys}$, *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* $i, j \leq \#\mathsf{Keys}$, *the challenger does the following.*

1. *If* $i \in V^*$ *or* $j \in V^*$ *or* $i = j$, *then output* $\bot$.
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* $E := E \cup (i, j)$ *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* $i, j \leq \#\mathsf{Keys}$ *and* $k \leq \#\mathsf{CT}$, *the challenger does the following.*

1. *If (A)* $(i, j)$ *is a deviating pair with respect to* $G^*$ *and* $G$, *or (B)* $(i, j)$ *is not an admissible edge 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, generates* $\mathsf{rk}_{i \to j} \leftarrow \mathsf{ReKeyGen}(\Sigma_{\sigma_i}, \Sigma_{\sigma_j}, \mathsf{sk}_i, \mathsf{pk}_j)$ *and* $\mathsf{rct}_j \leftarrow \mathsf{ReEnc}(\Sigma_{\sigma_i}, \Sigma_{\sigma_j}, \mathsf{rk}_{i \to j}, \mathsf{rct}_i)$ *from* $\mathsf{rct}_i$ *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* $i^* \in V_{\mathsf{h}}^*$ *and* $m_0, m_1, \in \mathcal{M}_U$ *(same as defined in Def. 2.5), the challenger generates* $\mathsf{ct}^* \leftarrow \mathsf{Enc}_{\sigma^*}(pk_{i^*}, m_b)$ *and gives it to* $\mathcal{A}$. *The challenger also sets* $\#\mathsf{CT} := \#\mathsf{CT} + 1$, $\mathsf{Drv} := \mathsf{Drv} \cup \{\#\mathsf{CT}\}$, $\mathsf{KeyCTList} := \mathsf{KeyCTList} \cup \{(\mathsf{ct}^*, \Sigma_{\sigma_{i^*}}, i^*, \#\mathsf{CT})\}$.

**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 multi-hop selective-graph UPRE-HRA secure if, for any PPT $\mathcal{A}$, it holds that*

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

*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 $\mathsf{Drv}$ nor number $\#\mathsf{CT}$ is used.
3. The condition that $\mathcal{O}_{\mathsf{reenc}}$ outputs $\perp$ for a query $(i, j)$ such that $i \in \mathsf{HList} \wedge j \in \mathsf{CList}$ (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.

### 2.3 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 3. 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.

**Definition 2.10 (Selective-graph UPRE-CRA security).** *The experiment $\mathsf{Exp}^{\mathsf{upre\text{-}msg\text{-}cra}}_{\mathcal{A}}(1^\lambda, b)$ of this security notion is the same as that of multi-hop selective-graph UPRE-HRA security except that the challenge oracle $\mathcal{O}_{\mathsf{cha}}$ is modified as follows.*

$\mathcal{O}_{\mathsf{cha}}(i_{\mathsf{c}}, i^*, m_0, m_1)$**:** *This oracle is invoked only once. For a challenge query $(i_{\mathsf{c}}, i^*, m_0, m_1)$ where $i_{\mathsf{c}} \in V^*_{\mathsf{c}} \ \wedge \ i^* \in V^*_{\mathsf{h}}$ and $m_0, m_1, \in \mathcal{M}_U$ (same as defined in Def. 2.5), 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. *Generates $\mathsf{rk}_{i_{\mathsf{c}} \to i^*} = \mathsf{ReKeyGen}(\Sigma_{\sigma_{i_{\mathsf{c}}}}, \Sigma_{\sigma_{i^*}}, \mathsf{sk}_{i_{\mathsf{c}}}, \mathsf{pk}_{i^*})$.*
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 $\#\mathsf{CT} := \#\mathsf{CT}+1$, $\mathsf{Drv} := \mathsf{Drv} \cup \{\#\mathsf{CT}\}$, $\mathsf{KeyCTList} := \mathsf{KeyCTList} \cup \{(\mathsf{ct}^*, \Sigma_{\sigma_{i^*}}, i^*, \#\mathsf{CT})\}$.*

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

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

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

### 2.4 On Re-Encryption Simulatability

Cohen introduced the notion of re-encryption simulatability for PRE to prove PRE-HRA security in a modular way [Coh19]. He proved that if a PRE scheme is PRE-CPA secure and satisfies re-encryption simulatability[10], then the scheme is PRE-HRA secure.

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 notion. Our relaxed UPRE schemes in Sections 3 and 4 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[11].

### 2.5 UPRE for More Advanced Encryption

We give the basic definitions of UPRE for PKE in Sections 2.1 and 2.2. We can consider more definitions for advanced encryption since UPRE is a general concept.

*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

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

[11] We could define a weaker variant of re-encryption simulatability for UPRE (and PRE) that still implies HRA security. 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.

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.

## 3 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.

### 3.1 Weak Batch Encryption

**Definition 3.1 (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* $\lambda$*-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* $\perp$*.*

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

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

*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*.*

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

1. $\mathcal{A}$ *chooses* $s \in \{0,1\}^\lambda$ *and sends it to the challenger.*
2. *The challenger computes* $(\hat{\mathsf{pk}}, \hat{\mathsf{sk}}) \leftarrow \mathsf{BatchGen}(1^\lambda, s)$ *and sends* $\hat{\mathsf{pk}}$ *to* $\mathcal{A}$*.*

3. $\mathcal{A}$ *sends* $\{(m_{i,0}, m_{i,1})\}_{i\in[\lambda]}$ *to the challenger 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* $\hat{\mathsf{ct}}^* \leftarrow \mathsf{BatchEnc}(\hat{\mathsf{pk}}, \{(m_{i,s[i]}, m_{i,s[i]})\})$.
4. *The challenger sends* $(\hat{\mathsf{sk}}, \hat{\mathsf{ct}}^*)$ *to* $\mathcal{A}$.
5. $\mathcal{A}$ *outputs a guess* $\beta'$ *for* $\beta$*. The experiment outputs* $\beta'$.

*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*

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

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

**IND-CPA Security:** *The experiment* $\mathsf{Exp}^{\mathsf{ind\text{-}cpa}}_{\mathcal{A}}(1^\lambda, \beta)$ *is the same as* $\mathsf{Exp}^{\mathsf{wbe\text{-}cpa}}_{\mathcal{A}}(1^\lambda, \beta)$ *above except that*

1. $\mathcal{A}$ *is not given* $\hat{\mathsf{sk}}$.
2. *If* $\beta = 1$*, then* $\hat{\mathsf{ct}}^* \leftarrow \mathsf{BatchEnc}(\hat{\mathsf{pk}}, \{(\mathbf{0}, \mathbf{0})\})$ *where* $\mathbf{0}$ *is a fixed special message (considered as all zero) that does not depend on* $\beta$.

*If* $\Pr[\mathsf{Exp}^{\mathsf{ind\text{-}cpa}}_{\mathcal{A}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ind\text{-}cpa}}_{\mathcal{A}}(1^\lambda, 1) = 1]$ *is negligible, then the weak batch encryption is IND-CPA secure.*

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 3.1 (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 $\hat{sk} := \{sk_{i,s[i]}\}_{i\in[\lambda]}$.

$\mathsf{BatchEnc}(\hat{\mathsf{pk}}, \{(m_{i,0}, m_{i,1})\}_{i\in[\lambda]})$**:** 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 $\Sigma$ 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. The IND-CPA security trivially holds. ■

### 3.2 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 $(\mathsf{Share}, \mathsf{Reconstruct})$. 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 $\mathsf{labels}$ 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 $\mathsf{labels}_s$ 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 $\mathsf{rk}_{f\to t} := (\hat{\mathsf{pk}}, s_2, \widetilde{\mathsf{ct}}_t)$.
- $\mathsf{ReEnc}(\Sigma_f, \Sigma_t, \mathsf{rk}_{f\to t}, \mathsf{ct}_f)$:
  - Parse $\mathsf{rk}_{f\to t} = (\hat{\mathsf{pk}}, s_2, \widetilde{\mathsf{ct}}_t)$.
  - If $\mathsf{ct}_f$ is in the ciphertext space of $\Sigma_f$ (1st level), set $\mathsf{C} \leftarrow \mathsf{P}[s_2, \mathsf{ct}_f]$; Else if (level $i > 1$), parse $\mathsf{ct}_f = (\hat{\mathsf{ct}}', \widetilde{\mathsf{ct}}_f, \tilde{\mathsf{C}}_{i-1}, \ldots, \tilde{\mathsf{C}}_1)$ and set $\mathsf{C} \leftarrow \mathsf{Q}[s_2, \hat{\mathsf{ct}}', \widetilde{\mathsf{ct}}_f]$
  - 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 $(\hat{\mathsf{ct}}, \widetilde{\mathsf{ct}}_t, \tilde{\mathsf{C}}_i, \ldots, \tilde{\mathsf{C}}_1)$
- $\mathsf{mDec}(\Sigma_t, \mathsf{sk}_t, \mathsf{rct}, i)$: Parse $\mathsf{rct} = (\hat{\mathsf{ct}}, \widetilde{\mathsf{ct}}_t, \tilde{\mathsf{C}}_i, \ldots, \tilde{\mathsf{C}}_1)$.
  - Compute $\hat{\mathsf{sk}}' \leftarrow \mathsf{Dec}(\mathsf{sk}_t, \widetilde{\mathsf{ct}}_t)$.
  - Compute $\widetilde{\mathsf{labels}}_i \leftarrow \mathsf{BatchDec}(\hat{\mathsf{sk}}', \hat{\mathsf{ct}})$
  - 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:** $s_2$, $\mathsf{ct}_f$.
**Input:** A share $s_1$.

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

**Fig. 1.** The description of the first level re-encryption circuit $\mathsf{P}$

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

**Hardwired:** $s_2$, $\hat{\mathsf{ct}}'$, $\widetilde{\mathsf{ct}}_f$.
**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 $\widetilde{\mathsf{labels}} \leftarrow \mathsf{BatchDec}(\hat{\mathsf{sk}}', \hat{\mathsf{ct}}')$.

**Fig. 2.** The description of the higher level re-encryption circuit $\mathsf{Q}$

*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 $\mathsf{rct} = (\hat{\mathsf{ct}}, \widetilde{\mathsf{ct}}_t, \tilde{\mathsf{C}}_1)$ 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 $\widetilde{\mathsf{ct}}_t = \mathsf{Enc}_t(\mathsf{pk}_t, \hat{\mathsf{sk}})$. Consider the computation of $\mathsf{mDec}(\Sigma_t, \mathsf{sk}_t, \mathsf{rct})$. By the correctness of $\Sigma_t$ 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 $(\mathsf{BatchGen}, \mathsf{BatchEnc}, \mathsf{BatchDec})$ 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 $\mathsf{P}$, $\mathsf{P}[s_2, \mathsf{ct}_f](s_1)$ 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 $\Sigma_f$ we get that $m' = m$.

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

### 3.3 Security Proof

**Theorem 3.2 (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 Def. 3.1, and both* $\Sigma_f$ *and* $\Sigma_t$ *are IND-CPA secure PKE, then* $\mathsf{UPRE}_{\mathsf{gc}}$ *is selective-graph 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 $|\Pr[\mathsf{Hyb}^x_{\mathcal{A}}(b) = 1] - \Pr[\mathsf{Hyb}^y_{\mathcal{A}}(b) = 1]| \leq \mathsf{negl}(\lambda)$. We say that a ciphertext $\mathsf{ct}$ is a level $i$ re-encryption, if $\mathsf{ct}$ is of the form $\mathsf{ct} = (\hat{\mathsf{ct}}, \widetilde{\mathsf{ct}}_t, \tilde{\mathsf{C}}_i, \ldots, \tilde{\mathsf{C}}_1)$, i.e. $\mathsf{ct}$ is the result of $i$ re-encryptions.

$\mathsf{Hyb}^0_{\mathcal{A}}(b)$**:** The first experiment is the original security experiment for $b$, $\mathsf{Exp}^{\mathsf{upre\text{-}msg\text{-}hra}}_{\mathcal{A}}(1^\lambda, b)$. That is, it holds that $\mathsf{Hyb}^0_{\mathcal{A}}(b) = \mathsf{Exp}^{\mathsf{upre\text{-}msg\text{-}hra}}_{\mathcal{A}}(1^\lambda, b)$. Note that in the successive experiments, we can easily simulate all keys in $G = (V, E)$ since vertices in $V$ are not connected to the target vertex in $G^*$ 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 $1/|V^*_{\mathsf{h}}|$, so $\Pr[\mathsf{Hyb}^{0'}_{\mathcal{A}}(b) = 1] = \frac{1}{|V^*_{\mathsf{h}}|} \cdot \Pr[\mathsf{Hyb}^0_{\mathcal{A}}(b) = 1]$.

$\mathsf{Hyb}^1_{\mathcal{A}}(b)$**:** 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 $Q_{\mathsf{reenc}}$ 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.[12] 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 $k \notin \mathsf{Drv}$.

[12] 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 $\mathsf{rk}_{\widehat{i}\to j} = (\hat{\mathsf{pk}}, s_2, \widetilde{\mathsf{ct}}_j)$.
- If $\mathsf{ct}_f$ is in the ciphertext space of $\Sigma_f$, set $\mathsf{C} \leftarrow \mathsf{P}[s_2, \mathsf{ct}_{\widehat{i}}]$; Else if, parse $\mathsf{ct}_f = (\hat{\mathsf{ct}}', \widetilde{\mathsf{ct}}_f, \tilde{\mathsf{C}}_{i-1}, \ldots, \tilde{\mathsf{C}}_1)$ and 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 $\mathsf{labels}^* \leftarrow \left\{(\mathsf{labels}_{i,s_1[i]}, \mathsf{labels}_{i,s_1[i]})\right\}_{i\in[\lambda]}$
- Compute $\hat{\mathsf{ct}} \leftarrow \mathsf{BatchEnc}(\hat{\mathsf{pk}}, \mathsf{labels}^*)$.
- Output $(\hat{\mathsf{ct}}, \widetilde{\mathsf{ct}}_j, \tilde{\mathsf{C}}_\iota, \ldots, \tilde{\mathsf{C}}_1)$.

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 $\mathsf{rk}_{\widehat{i}\to j} = (\hat{\mathsf{pk}}, s_2, \widetilde{\mathsf{ct}}_j)$.
- If $\mathsf{ct}_f$ is in the ciphertext space of $\Sigma_f$, set $\mathsf{C} \leftarrow \mathsf{P}[s_2, \mathsf{ct}_{\widehat{i}}]$; Else if, parse $\mathsf{ct}_f = (\hat{\mathsf{ct}}', \widetilde{\mathsf{ct}}_f, \tilde{\mathsf{C}}_{i-1}, \ldots, \tilde{\mathsf{C}}_1)$ and set $\mathsf{C} \leftarrow \mathsf{Q}[s_2, \hat{\mathsf{ct}}', \widetilde{\mathsf{ct}}_{\widehat{i}}]$.
- Compute $(\tilde{\mathsf{C}}_\iota, \widetilde{\mathsf{labels}}) \leftarrow \mathsf{GCSim}(\mathsf{C}(s_1))$.
- Compute $\mathsf{labels}^* \leftarrow (\widetilde{\mathsf{labels}}, \widetilde{\mathsf{labels}})$.
- Compute $\hat{\mathsf{ct}} \leftarrow \mathsf{BatchEnc}(\hat{\mathsf{pk}}, \mathsf{labels}^*)$.
- Output $(\hat{\mathsf{ct}}, \widetilde{\mathsf{ct}}_j, \tilde{\mathsf{C}}_\iota, \ldots, \tilde{\mathsf{C}}_1)$.

$\mathsf{Hyb}^{1,u,3}_{\mathcal{A}}(b)$**:** 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 $\mathsf{rk}_{\widehat{i}\to j} = (\hat{\mathsf{pk}}, s_2, \widetilde{\mathsf{ct}}_j)$.
- If $\mathsf{ct}_f$ is in the ciphertext space of $\Sigma_f$, set compute $(\tilde{\mathsf{C}}_\iota, \widetilde{\mathsf{labels}}) \leftarrow \mathsf{GCSim}(m)$;
  Else if, parse $\mathsf{ct}_f = (\hat{\mathsf{ct}}', \widetilde{\mathsf{ct}}_f, \tilde{\mathsf{C}}_{i-1}, \ldots, \tilde{\mathsf{C}}_1)$ and compute $(\tilde{\mathsf{C}}_\iota, \widetilde{\mathsf{labels}}) \leftarrow \mathsf{GCSim}(\widetilde{\mathsf{labels}}')$.
- Compute $\mathsf{labels}^* \leftarrow (\widetilde{\mathsf{labels}}, \widetilde{\mathsf{labels}})$.
- Compute $\hat{\mathsf{ct}} \leftarrow \mathsf{BatchEnc}(\hat{\mathsf{pk}}, \mathsf{labels}^*)$.
- Output $(\hat{\mathsf{ct}}, \widetilde{\mathsf{ct}}_j, \tilde{\mathsf{C}}_\iota, \ldots, \tilde{\mathsf{C}}_1)$.

For syntactic convention, we let $\mathsf{Hyb}^{1,0,3}_{\mathcal{A}}(b) \coloneqq \mathsf{Hyb}^{1}_{\mathcal{A}}(b)$. 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 $\mathsf{sk}_{i^*}$ to compute re-encrypted ciphertexts as above. However, in $\hat{\mathsf{pk}}$ and $s_2$, information about $\mathsf{sk}_{i^*}$ still remains. We will handles these issues in the following process.

*Process for removing $\mathsf{sk}_{i^*}$ of the target vertex.* Now, we focus on vertices in $V^*$ connected via admissible edges. To use the security of $\Sigma_{i^*}$, we need remove information about $\mathsf{sk}_{i^*}$ from all re-encryption keys in $G^* = (V^*, E^*)$ possibly connected to $i^*$. Let $Q$ be the total number of admissible edges connected to target vertex $i^*$. We call the following procedure a depth-search from vertex $i$: We seek a vertex that is connected to $i$ and does not have an outgoing edge in a forward direction. If there is a vertex $i'$ (possibly $i' = i$) that has two or more than two edges during the search, then we select a vertex $i'_1$ that is not searched

yet and is numbered by the smallest number, and set a flag such that the vertex is already searched to $i'_1$. We scan $G^* = (V^*, E^*)$ by the depth-search as follows.

First, we do a depth-search from $i^*$ and find a vertex $j$ such that $j$ does not have an outgoing edge.
Repeat the following process.

1. (Backward scan process) Go back to the vertex $i'$ that has two or more than two edges. If there is no such a vertex, then we end. If an edge was scanned by this backward scan, then we set a "scanned" flag to the edge.
2. Do the depth-search from $i'$.

During the backward scan process above, we repeat the hybrid transitions $\mathsf{Hyb}^{2,v,1}_{\mathcal{A}}$, $\mathsf{Hyb}^{2,v,2}_{\mathcal{A}}$, and $\mathsf{Hyb}^{2,v,3}_{\mathcal{A}}$ below whenever we move on a edge where $v = 1, \ldots, Q$. We let $\mathsf{Dlist}$ 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 $\mathsf{Dlist}$ 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 $\mathsf{rk}_{i\to j}$.

$\mathsf{Hyb}^{2,v,1}_{\mathcal{A}}(b)$ : At this point, we are at vertex $i$ and edge $(i, j)$ was just scanned.
- 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 3.1.** *It holds that* $\mathsf{Hyb}^{1,u,2}_{\mathcal{A}}(b) = \mathsf{Hyb}^{1,u,3}_{\mathcal{A}}(b)$.

*Proof.* This immediately holds since $m$ and $\widetilde{\mathsf{labels}}'$ are outputs of $\mathsf{C}(s_1)$ when $\mathsf{C} = \mathsf{P}$ and $\mathsf{C} = \mathsf{Q}$, respectively. ■

Indistinguishability of $\mathsf{Hyb}_{\mathcal{A}}^{1,(u-1),3}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^{1,u,1}(b)$is shown in Lemma 3.2, whereas indistinguishability of $\mathsf{Hyb}_{\mathcal{A}}^{1,u,1}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^{1,u,2}(b)$is shown in Lemma 3.3.

**Lemma 3.2.** *If* $(\mathsf{BatchGen}, \mathsf{BatchEnc}, \mathsf{BatchDec})$ *is WBE-CPA secure, then it holds that* $\mathsf{Hyb}_{\mathcal{A}}^{1,u,1}(b) \approx \mathsf{Hyb}_{\mathcal{A}}^{1,(u-1),3}(b)$.

*Proof of Lem. 3.2.* 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 $\mathsf{Hyb}_{\mathcal{A}}^{1,u,1}(b)$. From the $(u+1)$-th to $Q_{\mathsf{reenc}}$-th re-encryption queries are handled as in $\mathsf{Hyb}_{\mathcal{A}}^{1,(u-1),3}(b)$. $\mathcal{B}$ 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 $\mathsf{labels}$ 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 $\mathsf{Hyb}_{\mathcal{A}}^{1,(u-1),3}(b)$. On the other hand, if $\hat{\mathsf{ct}} = \mathsf{BatchEnc}(\hat{\mathsf{pk}}, \mathsf{labels}^*)$, then the query is handled as in $\mathsf{Hyb}_{\mathcal{A}}^{1,u,1}(b)$. ■

**Lemma 3.3.** *If* $\mathsf{gc} = (\mathsf{Garble}, \mathsf{Eval})$ *is a selectively secure garbling scheme, then it holds that* $\mathsf{Hyb}_{\mathcal{A}}^{1,u,2}(b) \approx \mathsf{Hyb}_{\mathcal{A}}^{1,u,1}(b)$.

*Proof of Lem. 3.3.* We will construct a reduction $\mathcal{B}$ which breaks the security of $(\mathsf{Garble}, \mathsf{Eval})$. As in the proof of Lemma 3.2, from the first to $(u-1)$-th re-encryption queries are handled as in $\mathsf{Hyb}_{\mathcal{A}}^{1,u,2}(b)$ and from the $(u+1)$-th to $Q_{\mathsf{reenc}}$-th re-encryption queries are handled as in $\mathsf{Hyb}_{\mathcal{A}}^{1,u,1}(b)$. $\mathcal{B}$ 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 with respect to $G^*$ and $k \notin \mathsf{Drv}$. That is, $\mathcal{B}$ sends $(\mathsf{C}, s_1)$ 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 $\mathsf{Hyb}_{\mathcal{A}}^{1,u,1}(b)$. On the other hand, if $(\tilde{\mathsf{C}}, \widetilde{\mathsf{labels}}) = \mathsf{GCSim}(\mathsf{C}(s_1))$, then the query is handled as in $\mathsf{Hyb}_{\mathcal{A}}^{1,u,2}(b)$. ■

**Lemma 3.4.** *If* $\Sigma_j$ *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 not-scanned edge. That is, we never use $\mathsf{sk}_j$ for simulation at this point. We can construct an adversary $\mathcal{B}$ that is given $\mathsf{pk}_j$. $\mathcal{B}$ 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 $\Sigma_j$. ■

**Lemma 3.5.** *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 3.6.** *If* $(\mathsf{Share}, \mathsf{Reconstruct})$ *is 2-out-of-2 secrete sharing scheme, then it holds that* $\mathsf{Hyb}^{2,v,2}_{\mathcal{A}}(b) \overset{\mathsf{s}}{\approx} \mathsf{Hyb}^{2,v,3}_{\mathcal{A}}(b)$.

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

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

## 4 Constant-Hop Construction Secure against CRA

In this section, we present constant-hop and CRA-secure UPRE schemes for PKE based on GC. The design is almost the same as that of the scheme in Section 3 except that we encrypt the garbled circuit $\tilde{\mathsf{C}}$ by using the delegatee's public key to hide information about the delegator's ciphertext.

### 4.1 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 3, 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 (1st level), set $\mathsf{C} \leftarrow \mathsf{P}[s_2, \mathsf{ct}_f]$; Else if (level $i > 1$), 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:** $s_2$, $\mathsf{ct}_f$.
**Input:** A share $s_1$.

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

**Fig. 3.** The description of the first level re-encryption circuit $\mathsf{P}$

**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 $\widetilde{\mathsf{labels}} \leftarrow \mathsf{BatchDec}(\hat{\mathsf{sk}}', \hat{\mathsf{ct}}')$.

**Fig. 4.** The description of the higher level re-encryption circuit $\mathsf{Q}$

**Theorem 4.1 (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 Def. 3.1, and both* $\Sigma_f$ *and* $\Sigma_t$ *are IND-CPA secure PKE, then* $\mathsf{UPRE}_{\mathsf{cra}}$ *is a selective-graph UPRE-CRA secure UPRE scheme.*

We omit the proof due to the space limit.

## References

ABH09. G. Ateniese, K. Benson, and S. Hohenberger. Key-Private Proxy Re-encryption. In *CT-RSA 2009*, volume 5473 of *LNCS*, pages 279–294. 2009.

ABPW13. Y. Aono, X. Boyen, L. T. Phong, and L. Wang. Key-Private Proxy Re-encryption under LWE. In *INDOCRYPT 2013*, volume 8250 of *LNCS*, pages 1–18. 2013.

ACH20. T. Agrikola, G. Couteau, and D. Hofheinz. The Usefulness of Sparsifiable Inputs: How to Avoid Subexponential iO. In *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 187–219. 2020.

AFGH05. G. Ateniese, K. Fu, M. Green, and S. Hohenberger. Improved Proxy Re-Encryption Schemes with Applications to Secure Distributed Storage. In *NDSS 2005*. 2005.

BBS98. M. Blaze, G. Bleumer, and M. Strauss. Divertible Protocols and Atomic Proxy Cryptography. In *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 127–144. 1998.

BGI+12. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. *Journal of the ACM*, 59(2):6, 2012.

BGLS03. D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 416–432. 2003.

BLSV18. Z. Brakerski, A. Lombardi, G. Segev, and V. Vaikuntanathan. Anonymous IBE, Leakage Resilience and Circular Security from New Assumptions. In *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 535–564. 2018.

CCL+14. N. Chandran, M. Chase, F.-H. Liu, R. Nishimaki, and K. Xagawa. Re-encryption, Functional Re-encryption, and Multi-hop Re-encryption: A Framework for Achieving Obfuscation-Based Security and Instantiations from Lattices. In *PKC 2014*, volume 8383 of *LNCS*, pages 95–112. 2014.

CCV12. N. Chandran, M. Chase, and V. Vaikuntanathan. Functional Re-encryption and Collusion-Resistant Obfuscation. In *TCC 2012*, volume 7194 of *LNCS*, pages 404–421. 2012.

CH07. R. Canetti and S. Hohenberger. Chosen-ciphertext secure proxy re-encryption. In *ACM CCS 2007*, pages 185–194. 2007.

CHN+18. A. Cohen, J. Holmgren, R. Nishimaki, V. Vaikuntanathan, and D. Wichs. Watermarking Cryptographic Capabilities. *SIAM J. Computing*, 47(6):2157–2202, 2018.

CLT13. J.-S. Coron, T. Lepoint, and M. Tibouchi. Practical Multilinear Maps over the Integers. In *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 476–493. 2013.

CLTV15. R. Canetti, H. Lin, S. Tessaro, and V. Vaikuntanathan. Obfuscation of Probabilistic Circuits and Applications. In *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 468–497. 2015.

Coh19. A. Cohen. What About Bob? The Inadequacy of CPA Security for Proxy Reencryption. In *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 287–316. 2019.

CPP16. G. Couteau, T. Peters, and D. Pointcheval. Encryption Switching Protocols. In *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 308–338. 2016.

CWYD10. S. S. M. Chow, J. Weng, Y. Yang, and R. H. Deng. Efficient Unidirectional Proxy Re-Encryption. In *AFRICACRYPT 10*, volume 6055 of *LNCS*, pages 316–332. 2010.

DDLM19. A. Davidson, A. Deo, E. Lee, and K. Martin. Strong Post-Compromise Secure Proxy Re-Encryption. In *ACISP 19*, volume 11547 of *LNCS*, pages 58–77. 2019.

DKL+18. D. Derler, S. Krenn, T. Lorünser, S. Ramacher, D. Slamanig, and C. Striecks. Revisiting Proxy Re-encryption: Forward Secrecy, Improved Security, and Applications. In *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 219–250. 2018.

DWLC08. R. H. Deng, J. Weng, S. Liu, and K. Chen. Chosen-Ciphertext Secure Proxy Re-encryption without Pairings. In *CANS 08*, volume 5339 of *LNCS*, pages 1–17. 2008.

ElG85. T. ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985.

FKKP19. G. Fuchsbauer, C. Kamath, K. Klein, and K. Pietrzak. Adaptively Secure Proxy Re-encryption. In *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 317–346. 2019.

GGH13. S. Garg, C. Gentry, and S. Halevi. Candidate Multilinear Maps from Ideal Lattices. In *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 1–17. 2013.

GGH15. C. Gentry, S. Gorbunov, and S. Halevi. Graph-Induced Multilinear Maps from Lattices. In *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 498–527. 2015.

GGH+16. S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate Indistinguishability Obfuscation and Functional Encryption for All Circuits. *SIAM J. Comput.*, 45(3):882–929, 2016.

GM84. S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

GSL19. S. Gaddam, R. Sinha, and A. Luykx. Applying Proxy-Re-Encryption to Payments. Real World Crypto 2019, 2019. https://rwc.iacr.org/2019/slides/Applying_PRE_Payments.pdf.

HHR16. J. Hesse, D. Hofheinz, and A. Rupp. Reconfigurable Cryptography: A Flexible Approach to Long-Term Security. In *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 416–445. 2016.

HKK+12. G. Hanaoka, Y. Kawai, N. Kunihiro, T. Matsuda, J. Weng, R. Zhang, and Y. Zhao. Generic Construction of Chosen Ciphertext Secure Proxy Re-Encryption. In *CT-RSA 2012*, volume 7178 of *LNCS*, pages 349–364. 2012.

HKW15. S. Hohenberger, V. Koppula, and B. Waters. Universal Signature Aggregators. In *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 3–34. 2015.

HRsV11. S. Hohenberger, G. N. Rothblum, a. shelat, and V. Vaikuntanathan. Securely Obfuscating Re-Encryption. *Journal of Cryptology*, 24(4):694–719, 2011.

ID03. A. Ivan and Y. Dodis. Proxy Cryptography Revisited. In *NDSS 2003*. 2003.

Jak99. M. Jakobsson. On Quorum Controlled Asymmetric Proxy Re-encryption. In *PKC'99*, volume 1560 of *LNCS*, pages 112–121. 1999.

LV08. B. Libert and D. Vergnaud. Unidirectional Chosen-Ciphertext Secure Proxy Re-encryption. In *PKC 2008*, volume 4939 of *LNCS*, pages 360–379. 2008.

NX15. R. Nishimaki and K. Xagawa. Key-Private Proxy Re-Encryption from Lattices, Revisited. *IEICE Transactions*, 98-A(1):100–116, 2015.

PRSV17. Y. Polyakov, K. Rohloff, G. Sahu, and V. Vaikuntanathan. Fast Proxy Re-Encryption for Publish/Subscribe Systems. *ACM Trans. Priv. Secur.*, 20(4):14:1–14:31, 2017.

SC09. J. Shao and Z. Cao. CCA-Secure Proxy Re-encryption without Pairings. In *PKC 2009*, volume 5443 of *LNCS*, pages 357–376. 2009.