Simulation-based Receiver Selective Opening CCA Secure PKE from Standard Computational Assumptions [⋆]

Keisuke Hara[1,2], Fuyuki Kitagawa[1,2], Takahiro Matsuda[2], Goichiro Hanaoka[2], and Keisuke Tanaka[1]

[1] Tokyo Institute of Technology, Tokyo, Japan
hara.k.am@m.titech.ac.jp, {kitagaw1,keisuke}@is.titech.ac.jp
[2] National Institute of Advanced Industrial Science and Technology (AIST), Tokyo, Japan
{t-matsuda,hanaoka-goichiro}@aist.go.jp

Abstract. In the situation where there are one sender and multiple receivers, a receiver selective opening (RSO) attack for a public key encryption (PKE) scheme considers adversaries that can corrupt some of the receivers and get their secret keys and plaintexts. Security against RSO attacks for a PKE scheme ensures confidentiality of ciphertexts of uncorrupted receivers. Simulation-based RSO security against chosen ciphertext attacks (SIM-RSO-CCA) is the strongest security notion in all RSO attack scenarios. Jia, Lu, and Li (INDOCRYPT 2016) proposed the first SIM-RSO-CCA secure PKE scheme. However, their scheme used indistinguishablility obfuscation, which is not known to be constructed from any standard computational assumption. In this paper, we give two contributions for constructing SIM-RSO-CCA secure PKE from standard computational assumptions. Firstly, we propose a generic construction of SIM-RSO-CCA secure PKE using an IND-CPA secure PKE scheme and a non-interactive zero-knowledge proof system satisfying one-time simulation soundness. Secondly, we propose an efficient and concrete construction of SIM-RSO-CCA secure PKE based on the decisional Diffie-Hellman (DDH) assumption. Moreover, we give a method for efficiently expanding the plaintext space of the DDH-based construction. By applying this method to the construction, we obtain the first DDH-based SIM-RSO-CCA secure PKE scheme supporting a super-polynomially large plaintext space with compact ciphertexts.

Keywords: public key encryption, receiver selective opening security, chosen ciphertext security.

[⋆] A preliminary version of this paper appears in the 11th Conference on Security and Cryptography for Networks (SCN 2018) [16]. This is the full version.

Table of Contents

1 Introduction . . . 3
  1.1 Background and Motivation . . . 3
  1.2 Our Contributions . . . 3
  1.3 Technical Overview . . . 4
  1.4 Related work . . . 7
  1.5 Organization . . . 8
2 Preliminaries . . . 8
  2.1 Notations . . . 8
  2.2 Public Key Encryption . . . 8
  2.3 Non-interactive Zero-knowledge Proof System . . . 9
  2.4 “+1”-Decisional Diffie-Hellman (DDH) Assumption . . . 10
  2.5 Universal Hash Family and Leftover Hash Lemma . . . 11
  2.6 Collision-resistant Hash Function . . . 11
3 CCA Security for Receiver Non-commiting Encryption . . . 11
  3.1 Receiver Non-commiting Encryption . . . 11
  3.2 RNC-CCA Secure RNCE Implies SIM-RSO-CCA Secure PKE . . . 12
4 Generic Construction of RNC-CCA Secure RNCE . . . 15
  4.1 Description . . . 15
  4.2 Security Proof . . . 16
5 DDH-based Construction of RNC-CCA Secure RNCE . . . 19
  5.1 Description of the Basic Construction . . . 20
  5.2 Security Proof for the Basic Construction . . . 21
  5.3 Description of the Main Construction . . . 26
  5.4 Security Proof for the Main Construction . . . 27
A Generic Construction of RNC-CCA Secure RNCE with a Multi-bit Plaintext Space . . . 35
B On the “+1”-DDH Assumption . . . 35

1 Introduction

1.1 Background and Motivation

In the context of public key encryption (PKE), the generally accepted security notions are IND-CPA and IND-CCA security [10, 12]. However, Bellare, Hofheinz, and Yilek [4] pointed out that IND-CPA and IND-CCA security might not be strong enough when considering *Selective Opening (SO)* attacks in a multi-user scenario. Intuitively, SO attacks consider the corruptions of some fraction of users and the extortions of their secret information. Motivated by the above problem, they firstly introduced SO security for PKE. Even if an adversary can mount SO attacks, SO security can guarantee confidentiality of ciphertexts of uncorrupted users. In practice, considering secret communication among many users, we should take account of information leakage from some users. Therefore, SO security is an important security notion for PKE in practice. To date, two settings have been considered for SO security: *Sender Selective Opening (SSO) security* [4,5] and *Receiver Selective Opening (RSO) security* [3,17]. The main focus in this paper is on RSO security. In the situation where one sender and multiple receivers exist, RSO security guarantees confidentiality of uncorrupted ciphertexts even if an adversary can corrupt some fraction of receivers and get their plaintexts and secret keys. SO security is defined in both the chosen plaintext attack (CPA) and the chosen ciphertext attack (CCA) settings. In general, we should take active adversaries into account, and thus the CCA security is more desirable than the CPA security.

Furthermore, there are two types of definitions for SO security: indistinguishability-based SO security and simulation-based SO security. The definition of indistinguishability-based SO security usually has a restriction on a plaintext distribution that an adversary can choose. More specifically, the definition of indistinguishability-based SO security usually requires the plaintext distribution to satisfy a notion called *efficient resamplability* [4]. Intuitively, efficient resamplability requires a plaintext distribution to be such that even if some plaintexts are fixed, the other plaintexts can be efficiently sampled. This requirement is somewhat artificial and limits application scenarios since plaintext distributions appearing in practice do not necessarily satisfy this requirement.

On the other hand, simulation-based SO security does not have such a restriction on the plaintext distribution. This security requires that the output of any adversary that is given the public keys, ciphertexts, and plaintexts and secret information of corrupted users, can be simulated by a simulator which only takes the corrupted plaintexts as its input. The secret information corresponds to randomnesses (used in encryptions) of the senders in the SSO setting and secret keys of the receivers in the RSO setting, respectively. Compared to indistinguishability-based SO security, simulation-based SO security can guarantee security even if an adversary chooses an arbitrary plaintext distribution. Since there is no restriction on the plaintext distributions, we can say that simulation-based SO security is preferable to indistinguishability-based SO security considering the utilization of PKE. Also, the previous works [3, 17] showed that simulation-based SO security is stronger than indistinguishability-based SO security in the CPA setting. It seems that this implication also holds in the CCA setting.

From the above arguments, we aim to achieve simulation-based RSO security against chosen ciphertext attacks which we call SIM-RSO-CCA security for PKE. So far, the only construction of SIM-RSO-CCA secure PKE is of Jia, Lu, and Li [19], but their construction is based on a very strong cryptographic primitive, *indistinguishability obfuscation (iO)* [2, 11].[1] This primitive is not known to be constructed from standard computational assumptions. Hence, in this paper, we tackle the following question: *Is it possible to construct a SIM-RSO-CCA secure PKE scheme from standard computational assumptions?*

1.2 Our Contributions

Based on the above motivation, we give affirmative answers to the question. More specifically, our technical results consist of the following three parts.

[1] Very recently, in a concurrent and independent work, Huang et al. [18] gave several constructions of SIM-RSO-CCA secure PKE. We compare the results between theirs and ours in Section 1.4.

SIM-RSO-CCA security derived from RNC-CCA security. As our first technical result, we introduce a new security notion that we call *RNC-CCA security* for receiver non-committing encryption (RNCE) [6, Section 4], which is a variant of PKE with a special non-committing property. Then, we show that RNC-CCA secure RNCE implies SIM-RSO-CCA secure PKE. When considering SIM-RSO-CCA security for PKE, we must take into account information of multiple users, a simulator, and an adversary. Thus, if we try to prove SIM-RSO-CCA security directly from standard computational assumptions, security proofs could become very complex. On the other hand, the merit of considering RNCE with our new security notion is that the definition of RNC-CCA security involves only a single user, a single adversary, and no simulator. Hence, we can potentially avoid a complex security proof when proving RNC-CCA security from standard computational assumptions. We believe that this result gives us a guideline for constructing a new SIM-RSO-CCA secure PKE scheme, and in fact, our proposed SIM-RSO-CCA secure PKE schemes are obtained via this result, as explained below.

A generic construction of RNC-CCA secure RNCE. As our second technical result, we show a generic construction of RNC-CCA secure RNCE using an IND-CPA secure PKE scheme and a non-interactive zero-knowledge (NIZK) proof system satisfying one-time simulation soundness. (In the following, we call this primitive an OTSS-NIZK for simplicity.) This primitive is slightly stronger than an ordinary NIZK proof system. However, the constructions of this primitive based on various standard assumptions are known [13, 14, 22]. Therefore, our second technical result shows that we can construct RNC-CCA secure RNCE schemes from various standard assumptions through our generic construction.

An efficient and concrete construction of RNC-CCA secure RNCE. Although our generic construction of RNC-CCA secure RNCE can be instantiated based on standard computational assumptions, we require an NIZK proof system as a building block. In general, NIZK proof systems are not very efficient, and thus the above construction does not necessarily lead to an efficient construction. Thus, as our third technical result, we show an efficient and concrete construction of RNC-CCA secure RNCE based on the decisional Diffie-Hellman (DDH) assumption. This scheme is a variant of the Cramer-Shoup encryption scheme [7], and thus we do not need general NIZK proof systems. Moreover, we give a method for efficiently expanding the plaintext space of the above construction. By applying this method to the construction, we obtain the first DDH-based RNC-CCA secure RNCE scheme supporting a super-polynomially large plaintext space with compact ciphertexts.

In summary, combining our first and second technical results, we obtain the first generic construction of SIM-RSO-CCA secure PKE from an IND-CPA secure PKE scheme and an OTSS-NIZK. This result enables us to construct SIM-RSO-CCA secure PKE from various standard computational assumptions. Moreover, combining our first and third technical results, we obtain the first efficient and concrete construction of SIM-RSO-CCA secure PKE from the DDH assumption.

The difference between this paper and the conference version [16]. In the conference version of this paper [16], we gave a SIM-RSO-CCA secure PKE scheme the size of whose plaintext space is polynomial in a security parameter based on the DDH assumption. However, when we expand the size of its plaintext space to super-polynomial, the ciphertext overhead (the difference between the ciphertext length and the plaintext length) of the construction increases linearly as the length of a plaintext increases. In the following, we call this construction our "basic" construction. In this paper, based on our basic construction, we give the first DDH-based concrete construction with compact ciphertexts which supports a super-polynomially large plaintext space. Here, "compact" means that the ciphertext overhead of our construction is independent of the length of a plaintext increases. In the following, we call this construction our "main" construction. In addition to the contribution, this paper gives the full proofs of Theorems 1, 2, and 3 omitted in the conference version [16].

1.3 Technical Overview

As mentioned earlier, Jia et al. [19] proposed the first SIM-RSO-CCA secure PKE scheme using iO. They pointed out that there exist common features between an IND-CCA security proof and a SIM-RSO security proof. To date, there are three major techniques for constructing IND-CCA secure PKE schemes: the double encryption technique [29], the hash proof system (HPS) technique [8], and the all-but-one (ABO) technique [27,28]. Sahai and Waters [30] pointed out that the "punctured programming" paradigm is compatible with iO when constructing various cryptographic primitives, and they in particular constructed an IND-CCA secure PKE scheme based on iO. In fact, Jia et al.'s SIM-RSO-CCA secure PKE scheme is obtained from the Sahai-Waters PKE scheme. Since the ABO technique has some similarity to the punctured programming paradigm, in retrospect, Jia et al.'s PKE scheme can be viewed as one constructed via the ABO technique.

In contrast to the Jia et al.'s strategy using the ABO technique, we take two different path of constructing two different SIM-RSO-CCA secure PKE schemes, that is, the double encryption technique and the HPS technique, respectively. Somewhat surprisingly, our SIM-RSO-CCA secure PKE schemes only require underlying cryptographic primitives that were required to construct IND-CCA secure PKE schemes. In particular, our constructions do not need any other strong cryptographic primitive, such as iO, for achieving SIM-RSO-CCA security.

For making the above two approaches work well, we focus on the work of Hazay, Patra, and Warinschi [17], who pointed out that RNCE [6, Section 4] is an appropriate cryptographic primitive for achieving RSO security. Concretely, they showed that CPA secure RNCE implies SIM-RSO-CPA secure PKE. Inspired by their idea, we formalize a new security notion for RNCE which we call RNC-CCA security, and show that RNC-CCA secure RNCE implies SIM-RSO-CCA secure PKE. Then, we propose a generic construction and an efficient and concrete construction of RNC-CCA secure RNCE based on the double encryption technique and the HPS technique, respectively.

The features of RNCE. Here, we explain the features of RNCE. Informally, RNCE is special PKE having the following two algorithms, Fake and Open.[2] Fake is the fake encryption algorithm that takes a public key and trapdoor information (generated at the key generation) as input, and outputs a *fake ciphertext* which has no information about a plaintext. Open is the opening algorithm that takes a public key, trapdoor information, the fake ciphertext, and a certain plaintext as input, and outputs a *fake secret key* which decrypts the fake ciphertext to the plaintext.

RNCE requires the following two security properties. The first one is that an adversary cannot distinguish a real ciphertext generated by the ordinary encryption algorithm and a fake ciphertext generated by Fake. The second one is that an adversary cannot distinguish a real secret key generated by the ordinary key generation algorithm and a fake secret key generated by Open. Canetti, Halevi, and Katz [6, Section 4.1] firstly introduced RNCE and a security notion for it considering only non-adaptive chosen ciphertext attacks (CCA1). We extend their security notion to RNC-CCA security considering adaptive chosen ciphertext attacks.

Sufficient condition for SIM-RSO-CCA secure PKE. We briefly review the security definition of RNCE. Informally, if considering only CPA, the security of RNCE is defined using an experiment that proceeds as follows.

1. An adversary is given a public key and chooses an arbitrary plaintext from the plaintext space.
2. The adversary is given either a real ciphertext or a fake ciphertext depending on the challenge bit chosen uniformly at random.
3. The adversary is given either a real secret key or a fake secret key depending on the above challenge bit.
4. The adversary guesses whether the given ciphertext and secret key are real or fake.

When defining RNC-CCA security for RNCE, it is natural to consider a definition in which an adversary is allowed to make a decryption query at any time in the above security experiment. If we define such a security experiment, an adversary can make a decryption query after he gets a secret key. If we adopt

[2] In fact, our syntax of RNCE has additional algorithms FKG and FDec. These algorithms are needed for defining RNC-CCA security. See Section 3 for the details.

this security definition, it is clear that the adversary of RNC-CCA security can simulate the decryption oracle for an adversary of SIM-RSO-CCA security.

However, there is one technical problem if we adopt the above definition. The problem is that we cannot obtain an efficient and concrete construction of RNCE from the HPS technique. More specifically, it seems hard to construct an RNCE scheme based on the Cramer-Shoup encryption scheme [7]. The critical problem is that when proving the CCA security of a Cramer-Shoup encryption scheme, we use the fact that the entropy of the secret key is sufficiently large. In the security experiment of RNCE, an adversary gets the secret key used in the experiment, and thus the entropy of the secret key is completely lost and the security proof fails if we adopt the above definition.

In order to circumvent the above problem, we define the security experiment for RNC-CCA security of an RNCE scheme so that an adversary is not allowed to make decryption queries after he gets the secret key. Adopting this security definition, we do not have to simulate the decryption oracle for the adversary after he gets the secret key, and we can complete the security proof of our RNCE scheme. See Section 5 for the details.

Here, one might have the following question: Can we show that RNC-CCA security implies SIM-RSO-CCA security when adopting the above modified definition for RNC-CCA security? We show an affirmative answer to this question. In a nutshell, we do not have to simulate the decryption queries which are relative to the secret keys of corrupted users in the definition of SIM-RSO-CCA security, and thus we can still show that RNC-CCA secure RNCE implies SIM-RSO-CCA secure PKE. See Section 3 for the details.

How to derive RNC-CCA secure RNCE from the double encryption technique. Here, we give an overview of our generic construction of RNC-CCA secure RNCE derived from the classical double encryption technique [26, 29]. One can see that our generic construction is an extension of a CPA secure RNCE scheme observed by Canetti et al. [6, Section 4.1]. Their RNCE scheme is inspired by the double encryption technique without considering CCA security. The trick for the non-committing property of their construction is that the secret key used in the decryption algorithm is chosen at random from the two underlying secret keys, and thus their scheme is very simple. In order to upgrade the CPA security of this RNCE scheme to CCA security, we focus on the work by Lindell [22] who constructed an IND-CCA secure PKE scheme based on an IND-CPA secure PKE scheme and an OTSS-NIZK using the double encryption technique. Applying a similar method to the above RNCE scheme, we obtain our generic construction of RNC-CCA secure RNCE. See Section 4 for the details.

We note that the technique for achieving the non-committing property, i.e., generating multiple secret keys and using only one of them for decryption, has been adopted in a number of works, e.g., in the construction of an adaptively and forward secure key-evolving encryption scheme [6, Section 3], and more recently in the construction of a tightly secure key encapsulation mechanism in the multi-user setting with corruption [1]. Furthermore, our construction shares an idea of binding two ciphertexts with an NIZK proof system with [6, Section 3] to resist against active behaviors of an adversary (e.g., decryption queries). However, one difference is that we require one-time simulation-soundness for the underlying NIZK proof system, while they require unbounded simulation-soundness.

How to derive RNC-CCA secure RNCE from the HPS technique. Here, we explain an overview of our concrete construction of RNC-CCA secure RNCE derived from the HPS technique [7, 8]. Our concrete construction is an extension of the CCA1 secure RNCE scheme proposed by Canetti et al. [6, Section 4.2]. Their RNCE scheme is a variant of the Cramer-Shoup-“lite” encryption scheme [7], which is an IND-CCA1 secure PKE scheme based on the DDH assumption. The only difference is that they encode a plaintext m by the group element g^m, where g is a generator of the underlying group. This encoding is essential for the opening algorithm Open of their proposed scheme, and the plaintext space of their scheme is of polynomial-size since they have to compute the discrete logarithm of g^m in the decryption procedure.

As our basic construction, we extend their scheme to a RNC-CCA secure RNCE scheme based on the “full”-Cramer-Shoup encryption scheme [7]. In our construction, we achieve the non-committing property using a technique similar to Canetti et al., and handle an adversary's decryption queries similarly to the CCA security proof of the "full"-Cramer-Shoup encryption scheme. See Sections 5.1 and 5.2 for the details. We note that the size of the plaintext space of this construction is restricted to a polynomial in a security parameter.

How to expand a polynomial-sized plaintext space and its drawback. In general, it is natural to ask whether we can expand a plaintext space of an encryption scheme. In fact, we can expand the polynomial-sized plaintext space of our basic construction to a super-polynomially large one by using the scheme in a parallel way. More specifically, in our basic construction, a ciphertext (resp., secret key) consists of four elements $(u_1, u_2, e = k^r \cdot g_1^m, v)$ of $\mathbb{G}$ (resp., six elements $(x_1, x_2, y_1, y_2, z_1, z_2)$ of $\mathbb{Z}_p$), where $\mathbb{G}$ is a multiplicative cyclic group of prime order p, k is a group element in a public key, g_1 is a generator of $\mathbb{G}$, m is a plaintext that belongs to a polynomial-sized plaintext space (and hence its length is logarithmic), and r is a randomness used in the encryption procedure. Similarly to the original Cramer-Shoup encryption scheme, a secret key $(x_1, x_2, y_1, y_2, z_1, z_2)$ of our construction is divided into two parts having different roles. The component (x_1, x_2) is used for masking a plaintext and the remaining component (y_1, y_2, z_1, z_2) is used for checking the validity of a ciphertext. Based on the above basic construction, we can expand the plaintext space as follows. Let d be a logarithmic function in a security parameter and ℓ be a polynomial in a security parameter. If we want to encrypt a plaintext $m = m_1 \| \cdots \| m_\ell \in \{0,1\}^{d \cdot \ell}$, we can encrypt this plaintext m by generating ℓ independent components $(x_{i1}, x_{i2})_{i \in [\ell]}$ then hiding each block m_i with each component (x_{i1}, x_{i2}). Note that we do not need to increase the number of the component (y_1, y_2, z_1, z_2) for checking the validity of a ciphertext. However, there is a drawback about the ciphertext overhead of the construction obtained in this way. Specifically, in this extended construction, a ciphertext consists of $\ell + 3$ elements $(u_1, u_2, (e_i)_{i \in [\ell]}, v)$ of $\mathbb{G}$, where $e_i := k_i^r \cdot g_1^{m_i}$ and each k_i is a group element in a public key for each $i \in [\ell]$. If we let the bit length of one group element be q, the ciphertext overhead is $q(\ell + 3) - d\ell$. Thus, as the length of a plaintext increases, the ciphertext overhead of this construction increases linearly.

How to expand a plaintext space more efficiently. It is more desirable to efficiently expand a plaintext space of an encryption scheme. Thus, for reducing the ciphertext overhead, we provide a method for efficiently expanding the polynomial-sized plaintext space of our basic construction to a super-polynomially large one. Our technique for achieving compact ciphertexts is inspired by Hofheinz et al. [15]. Compared to our basic construction, when computing a ciphertext $(u_1, u_2, (e_i)_{i \in [\ell]}, v)$, we change how to compute $(e_i)_{i \in [\ell]}$ which are components of masked plaintexts. More specifically, before masking a plaintext $m = m_1 \| \cdots \| m_\ell \in \{0,1\}^{d \cdot \ell}$, we firstly use a universal hash function $\mathsf{H} : \mathbb{G} \to \{0,1\}^d$ to compress each group element k_i^r used for masking each block m_i to a d-bit string for all $i \in [\ell]$. Then, we compute $e_i := \mathsf{H}(k_i^r) \oplus m_i$ for all $i \in [\ell]$. By taking this approach, the number of group elements in a ciphertext $(u_1, u_2, (e_i)_{i \in [\ell]}, v)$ decreases from $\ell + 3$ to 3. Similarly to the above, if we let the bit length of one group element be q, the ciphertext overhead is $(3q + d\ell) - d\ell = 3q$. That is, even if the length of a plaintext increases, the ciphertext overhead of this modified scheme is always constant $3q$. Remarkably, this overhead is exactly the same as that in the original Cramer-Shoup encryption scheme. See Sections 5.3 and 5.4 for the details.

1.4 Related work

To date, SSO secure PKE schemes have been extensively studied, and several constructions of SIM-SSO-CCA secure PKE have been shown based on various standard computational assumptions [21, 23–25]. On the other hand, RSO secure PKE schemes have been much less studied.

At the moment the conference version of this paper was submitted, the only existing construction of SIM-RSO-CCA secure PKE is the construction using iO proposed by Jia et al. [19]. Jia, Lu, and Li [20] proposed indistinguishability-based RSO-CCA (IND-RSO-CCA) secure PKE schemes based on standard computational assumptions. Concretely, they showed two generic constructions of IND-RSO-CCA secure PKE schemes. First, they gave a generic construction based on an IND-RSO-CPA secure PKE scheme, an IND-CCA secure PKE scheme, an NIZK proof system, and a strong one-time signature scheme. Second, they gave a generic construction based on a universal HPS. It is not obvious whether their schemes (can be extended to) satisfy SIM-RSO-CCA security.

Very recently, as a concurrent and independent work, Huang et al. [18] showed three constructions of SIM-RSO-CCA secure PKE. Their first (resp., second) constructions are based on the DDH (resp., decisional composite residuosity (DCR)) assumption the size of whose plaintext space is polynomial (resp., super-polynomial) in a security parameter. Besides, as their third construction, they introduced the new notion of master-key selective opening security for identity-based encryption (IBE) and then showed that SIM-RSO-CCA secure PKE can be constructed from IBE with this security notion. They also showed a construction of IBE with this security notion in the ideal cipher model.

Compared to their work, our results have an overlap on the DDH-based construction. More precisely, their DDH-based construction is essentially the same as our basic construction described in the conference version [16]. However, their work does not treat an analogue of RNC-CCA secure RNCE proposed in this paper. Furthermore, their work does not include our generic construction based on an IND-CPA secure PKE scheme and an OTSS-NIZK, and the DDH-based construction supporting a super-polynomially large plaintext space with compact ciphertexts.

1.5 Organization

The rest of the paper is organized as follows: In Section 2, we review the notations, assumptions, and definitions of cryptographic primitives. In Section 3, we introduce RNC-CCA security for RNCE and show its implication to SIM-RSO-CCA security for PKE. In Section 4, we show a generic construction of RNC-CCA secure RNCE with a binary plaintext space, which is constructed from an IND-CPA secure PKE scheme and an OTSS-NIZK. (In Appendix A, we give a multi-bit version of the geniric construction.) In Section 5, we show a DDH-based constructions of RNC-CCA secure RNCE.

2 Preliminaries

In this section, we define notations, assumptions, and cryptographic primitives.

2.1 Notations

In this paper, $x \leftarrow X$ denotes sampling an element x from a finite set X uniformly at random. $y \leftarrow \mathcal{A}(x; r)$ denotes that a probabilistic algorithm $\mathcal{A}$ outputs y for an input x using a randomness r, and we simply denote $y \leftarrow \mathcal{A}(x)$ when we need not write an internal randomness explicitly. For strings x and y, $x \| y$ denotes the concatenation of x and y. Also, $x := y$ denotes that x is defined by y. λ denotes a security parameter. A function $f(\lambda)$ is a negligible function in λ, if $f(\lambda)$ tends to 0 faster than $\frac{1}{\lambda^c}$ for every constant $c > 0$. $\mathsf{negl}(\lambda)$ denotes an unspecified negligible function. PPT stands for probabilistic polynomial time. If n is a natural number, $[n]$ denotes the set of integers $\{1, \cdots, n\}$. Also, if a and b are integers such that $a \leq b$, $[a, b]$ denotes the set of integers $\{a, \cdots, b\}$. If $\mathbf{m} = (m_1, \cdots, m_n)$ is an n-dimensional vector, $\mathbf{m}_J$ denotes the subset $\{m_j\}_{j \in J}$ where $J \subseteq [n]$. If $\mathcal{O}$ is a function or an algorithm and $\mathcal{A}$ is an algorithm, $\mathcal{A}^{\mathcal{O}}$ denotes that $\mathcal{A}$ has oracle access to $\mathcal{O}$. If M is a square matrix, det(M) denotes the determinant of M.

2.2 Public Key Encryption

A public key encryption (PKE) scheme with a plaintext space $\mathcal{M}$ consists of a tuple of the following three PPT algorithms $\mathit{\Pi} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$.

KG**:** The key generation algorithm, given a security parameter 1^λ, outputs a public key pk and a secret key sk.

Enc**:** The encryption algorithm, given a public key pk and a plaintext $m \in \mathcal{M}$, outputs a ciphertext c.

Dec: The (deterministic) decryption algorithm, given a public key pk, a secret key sk, and a ciphertext c, outputs a plaintext $m \in \{\bot\} \cup \mathcal{M}$.

As the correctness for Π, we require that $\mathsf{Dec}(pk, sk, \mathsf{Enc}(pk, m)) = m$ holds for all $\lambda \in \mathbb{N}$, $m \in \mathcal{M}$, and $(pk, sk) \leftarrow \mathsf{KG}(1^\lambda)$.

Next, we define IND-CPA security and SIM-RSO-CCA security for a PKE scheme.

Definition 1 (IND-CPA security). *We say that $\Pi = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ is IND-CPA secure if for any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$,*

$$\mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\Pi,\mathcal{A}}(\lambda) := 2 \cdot \Big| \Pr[b \leftarrow \{0,1\}; (pk, sk) \leftarrow \mathsf{KG}(1^\lambda); (m_0^*, m_1^*, \mathsf{st}_1) \leftarrow \mathcal{A}_1(pk); c^* \leftarrow \mathsf{Enc}(pk, m_b^*); b' \leftarrow \mathcal{A}_2(c^*, \mathsf{st}_1) : b = b'] - \frac{1}{2} \Big| = \mathsf{negl}(\lambda),$$

where it is required that $|m_0^*| = |m_1^*|$.

Definition 2 (SIM-RSO-CCA security). *Let n be the number of users. For a PKE scheme $\Pi = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$, an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$, and a simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$, we define the following pair of experiments.*

```
$\mathsf{Exp}^{\mathsf{rso\text{-}cca\text{-}real}}_{n,\Pi,\mathcal{A}}(\lambda)$ :
    $(\mathbf{pk}, \mathbf{sk}) := (pk_j, sk_j)_{j \in [n]} \leftarrow (\mathsf{KG}(1^\lambda))_{j \in [n]}$
    $(\mathsf{Dist}, \mathsf{st}_1) \leftarrow \mathcal{A}_1^{\mathcal{O}_{\mathsf{Dec}}(\cdot,\cdot)}(\mathbf{pk})$
    $\mathbf{m}^* := (m_j^*)_{j \in [n]} \leftarrow \mathsf{Dist}$
    $\mathbf{c}^* := (c_j^*)_{j \in [n]} \leftarrow (\mathsf{Enc}(pk_j, m_j^*))_{j \in [n]}$
    $(J, \mathsf{st}_2) \leftarrow \mathcal{A}_2^{\mathcal{O}_{\mathsf{Dec}}(\cdot,\cdot)}(\mathbf{c}^*, \mathsf{st}_1)$
    $\mathsf{out} \leftarrow \mathcal{A}_3^{\mathcal{O}_{\mathsf{Dec}}(\cdot,\cdot)}(\mathbf{sk}_J, \mathbf{m}^*_J, \mathsf{st}_2)$
    Return $(\mathbf{m}^*, \mathsf{Dist}, J, \mathsf{out})$

$\mathsf{Exp}^{\mathsf{rso\text{-}cca\text{-}sim}}_{n,\Pi,\mathcal{S}}(\lambda)$ :
    $(\mathsf{Dist}, \mathsf{st}_1) \leftarrow \mathcal{S}_1(1^\lambda)$
    $\mathbf{m}^* := (m_j^*)_{j \in [n]} \leftarrow \mathsf{Dist}$
    $(J, \mathsf{st}_2) \leftarrow \mathcal{S}_2(\mathsf{st}_1)$
    $\mathsf{out} \leftarrow \mathcal{S}_3(\mathbf{m}^*_J, \mathsf{st}_2)$
    Return $(\mathbf{m}^*, \mathsf{Dist}, J, \mathsf{out})$
```

In both of the experiments, we require that the distributions Dist *output by $\mathcal{A}$ and $\mathcal{S}$ be efficiently samplable. In $\mathsf{Exp}^{\mathsf{rso\text{-}cca\text{-}real}}_{n,\Pi,\mathcal{A}}(\lambda)$, a decryption query (c, j) is answered by $\mathsf{Dec}(pk_j, sk_j, c)$. $\mathcal{A}_2$ and $\mathcal{A}_3$ are not allowed to make a decryption query (c_j^*, j) for any $j \in [n]$. Furthermore, $\mathcal{A}_3$ is not allowed to make a decryption query (c, j) satisfying $j \in J$. (This is without losing generality, since $\mathcal{A}_3$ can decrypt any ciphertext using the given secret keys.)*

We say that Π is SIM-RSO-CCA secure if for any PPT adversary $\mathcal{A}$ and any positive integer $n = n(\lambda)$, there exists a PPT simulator $\mathcal{S}$ such that for any PPT distinguisher $\mathcal{D}$,

$$\mathsf{Adv}^{\mathsf{rso\text{-}cca}}_{n,\Pi,\mathcal{A},\mathcal{S},\mathcal{D}}(\lambda) := |\Pr[\mathcal{D}(\mathsf{Exp}^{\mathsf{rso\text{-}cca\text{-}real}}_{n,\Pi,\mathcal{A}}(\lambda)) = 1] - \Pr[\mathcal{D}(\mathsf{Exp}^{\mathsf{rso\text{-}cca\text{-}sim}}_{n,\Pi,\mathcal{S}}(\lambda)) = 1]| = \mathsf{negl}(\lambda).$$

Remark 1. For simplicity, we consider non-adaptive opening queries by an adversary in our experiments. That is, an adversary can make an opening query $J \subseteq [n]$ only at once. However, our constructions of SIM-RSO-CCA secure PKE remain secure even if we consider adaptive opening queries by an adversary.

Remark 2. In this paper, as in the previous works [19, 20], we consider only the revelation of secret keys in the definition of SIM-RSO-CCA security. Namely, we assume that an adversary cannot obtain a random coin used for generating a secret key. Hazay, Patra, and Warinschi [17] considered the revelation of both secret keys and random coins used in the key generation algorithm in the definition of RSO-CPA security. If we take into account corruptions of both secret keys and random coins, it seems that we need *key simulatability* [9, 17] for building blocks.

2.3 Non-interactive Zero-knowledge Proof System

Let $\mathcal{R}$ be an efficiently computable binary relation and $\mathcal{L} := \{x | \exists w \text{ s.t. } (x, w) \in \mathcal{R}\}$. A non-interactive proof system for $\mathcal{L}$ consists of a tuple of the following five PPT algorithms $\Phi = (\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{SimCRS}, \mathsf{SimPrv})$.

CRSGen**:** The common reference string (CRS) generation algorithm, given a security parameter 1^λ, outputs a CRS crs.

Prove**:** The proving algorithm, given a CRS crs, a statement $x \in \mathcal{L}$, and a witness w for the fact that $x \in \mathcal{L}$, outputs a proof π.

Verify**:** The verification algorithm, given a CRS crs, a statement x, and a proof π, outputs either 1 (meaning "accept") or 0 (meaning "reject").

SimCRS**:** The simulator's CRS generation algorithm, given a security parameter 1^λ, outputs a simulated CRS crs and a trapdoor key tk.

SimPrv**:** The simulator's proving algorithm, given a trapdoor key tk and a (possibly false) statement x, outputs a simulated proof π.

As the correctness for Φ, we require that $\mathsf{Verify}(crs, x, \mathsf{Prove}(crs, x, w)) = 1$ holds for all $\lambda \in \mathbb{N}$, all $crs \leftarrow \mathsf{CRSGen}(1^\lambda)$, all statements $x \in \mathcal{L}$, and all witnesses w for the fact that $x \in \mathcal{L}$.

Next, we define the security notions for a non-interactive proof system: *One-time simulation soundness* (OT-SS) and *zero-knowledge* (ZK).

Definition 3 (One-time simulation soundness). *We say that a non-interactive proof system* $\Phi = (\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{SimCRS}, \mathsf{SimPrv})$ *satisfies one-time simulation soundness (OT-SS) if for any PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*,*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{ot\text{-}ss}}_{\Phi,\mathcal{A}}(\lambda) := \Pr[&(crs, tk) \leftarrow \mathsf{SimCRS}(1^\lambda); (x^*, \mathsf{st}_1) \leftarrow \mathcal{A}_1(crs); \pi^* \leftarrow \mathsf{SimPrv}(tk, x^*);\\ &(x, \pi) \leftarrow \mathcal{A}_2(\pi^*, \mathsf{st}_1) : (x \notin \mathcal{L}) \wedge (\mathsf{Verify}(crs, x, \pi) = 1) \wedge ((x, \pi) \neq (x^*, \pi^*))] = \mathsf{negl}(\lambda).\end{aligned}$$

Definition 4 (Zero-knowledge). *For a non-interactive proof system* $\Phi = (\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{SimCRS}, \mathsf{SimPrv})$ *and an adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, we consider the following pair of experiments.*

```
$\mathsf{Exp}^{\mathsf{zk\text{-}real}}_{\Phi,\mathcal{A}}(\lambda)$ :          | $\mathsf{Exp}^{\mathsf{zk\text{-}sim}}_{\Phi,\mathcal{A}}(\lambda)$ :
    $crs \leftarrow \mathsf{CRSGen}(1^\lambda)$                    |     $(crs, tk) \leftarrow \mathsf{SimCRS}(1^\lambda)$
    $(x, w, \mathsf{st}_1) \leftarrow \mathcal{A}_1(crs)$          |     $(x, w, \mathsf{st}_1) \leftarrow \mathcal{A}_1(crs)$
    $\pi \leftarrow \mathsf{Prove}(crs, x, w)$                     |     $\pi \leftarrow \mathsf{SimPrv}(tk, x)$
    $b' \leftarrow \mathcal{A}_2(\pi, \mathsf{st}_1)$              |     $b' \leftarrow \mathcal{A}_2(\pi, \mathsf{st}_1)$
    Return $b'$                                                    |     Return $b'$
```

In both of the experiments, it is required that $x \in \mathcal{L}$ *and* w *be a witness for* $x \in \mathcal{L}$*. We say that* Φ *is zero-knowledge (ZK) if for any PPT adversary* $\mathcal{A}$*,*

$$\mathsf{Adv}^{\mathsf{zk}}_{\Phi,\mathcal{A}}(\lambda) := |\Pr[\mathsf{Exp}^{\mathsf{zk\text{-}real}}_{\Phi,\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{zk\text{-}sim}}_{\Phi,\mathcal{A}}(\lambda) = 1]| = \mathsf{negl}(\lambda).$$

In this paper, we call a non-interactive proof system satisfying both OT-SS and ZK properties an *OTSS-NIZK*.

2.4 "+1"-Decisional Diffie-Hellman (DDH) Assumption

In this section, we define the "+1"-DDH assumption. Although it is straightforward to see this assumption is implied by the standard DDH assumption, we provide its formal proof in Appendix B for completeness. This assumption is used to simplify the security proof of our concrete construction in Section 5.

Definition 5 ("+1"-DDH assumption). *Let* $p = \Omega(2^\lambda)$ *be a prime number,* $\mathbb{G}$ *be a multiplicative cyclic group of order* p*, and* $\mathbb{Z}_p$ *be the set of integers modulo* p*. We say that the "+1"-DDH assumption holds in* $\mathbb{G}$ *if for any PPT adversary* $\mathcal{A}$*,*

$$\begin{aligned}\mathsf{Adv}^{+1\text{-}\mathsf{ddh}}_{\mathbb{G},\mathcal{A}}(\lambda) := |\Pr[&g \leftarrow \mathbb{G}; a \leftarrow \mathbb{Z}_p^*; b \leftarrow \mathbb{Z}_p : \mathcal{A}(g, g^a, g^b, g^{ab}) = 1]\\ &- \Pr[g \leftarrow \mathbb{G}; a \leftarrow \mathbb{Z}_p^*; b \leftarrow \mathbb{Z}_p : \mathcal{A}(g, g^a, g^b, g^{ab+1}) = 1]| = \mathsf{negl}(\lambda).\end{aligned}$$

2.5 Universal Hash Family and Leftover Hash Lemma

In this section, we firstly recall the definitions of the statistical distance and a universal hash family.

Definition 6 (Statistical distance). *Let $\mathcal{X}$ and $\mathcal{X}'$ be distributions over the same set X. The statistical distance between $\mathcal{X}$ and $\mathcal{X}'$, denoted by $\mathsf{SD}(\mathcal{X}, \mathcal{X}')$, is defined by*

$$\mathsf{SD}(\mathcal{X}, \mathcal{X}') := \max_{f: X \to \{0,1\}} |\Pr[f(\mathcal{X}) = 1] - \Pr[f(\mathcal{X}') = 1]|.$$

Definition 7 (Universal hash family). *Let X and Y be sets. Let $\mathcal{H} := \{\mathsf{H} : X \to Y\}$ be a family of functions. We say that $\mathcal{H}$ is a universal hash family if for any $x_1, x_2 \in X$ ($x_1 \neq x_2$),*

$$\Pr[\mathsf{H}(x_1) = \mathsf{H}(x_2)] \leq \frac{1}{|Y|}$$

holds, where $\mathsf{H} \leftarrow \mathcal{H}$.

Next, we recall the leftover hash lemma.

Lemma 1 (Leftover hash lemma). *Let $\mathcal{H} := \{\mathsf{H} : X \to Y\}$ be a universal hash family. Then, it holds that*

$$\mathsf{SD}((\mathsf{H}, \mathsf{H}(x)), (\mathsf{H}, y)) \leq \sqrt{\frac{|Y|}{4 \cdot |X|}},$$

where $\mathsf{H} \leftarrow \mathcal{H}$, $x \leftarrow X$, and $y \leftarrow Y$.

2.6 Collision-resistant Hash Function

In this section, we recall the definition of a collision-resistant hash function. A hash function consists of a pair of PPT algorithms $\Lambda = (\mathsf{HKG}, \mathsf{Hash})$. HKG is the hash key generation algorithm that, given a security parameter 1^λ, outputs a hash key hk. Hash is the (deterministic) hashing algorithm that, given a hash key hk and a string $x \in \{0,1\}^*$, outputs a hash value h.

Definition 8 (Collision-resistance). *We say that $\Lambda = (\mathsf{HKG}, \mathsf{Hash})$ is a collision-resistant hash function if for any PPT adversary $\mathcal{A}$,*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{cr}}_{\Lambda,\mathcal{A}}(\lambda) := \Pr[&hk \leftarrow \mathsf{HKG}(1^\lambda); (x, x^*) \leftarrow \mathcal{A}(hk) : \\ &(\mathsf{Hash}(hk, x) = \mathsf{Hash}(hk, x^*)) \wedge (x \neq x^*)] = \mathsf{negl}(\lambda).\end{aligned}$$

3 CCA Security for Receiver Non-commiting Encryption

In this section, we introduce a new security notion that we call RNC-CCA security for receiver non-commiting encryption (RNCE). Next, we show that RNC-CCA secure RNCE implies SIM-RSO-CCA secure PKE.

3.1 Receiver Non-commiting Encryption

Here, we give definitions of RNCE and RNC-CCA security for this primitive. Informally, RNCE is PKE having the property that it can generate a fake ciphertext which can be later opened to any plaintext (by showing an appropriate secret key). Canetti, Halevi, and Katz [6, Section 4.1] gave a definition of RNCE considering security against non-adaptive chosen ciphertext attacks (CCA1). We extend their definition to one considering security against adaptive CCA.

Informally, an RNCE scheme Π consists of the seven PPT algorithms $(\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{FKG}, \mathsf{Fake}, \mathsf{Open}, \mathsf{FDec})$. $(\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ are the same algorithms as those of a PKE scheme. The remaining four algorithms $(\mathsf{FKG}, \mathsf{Fake}, \mathsf{Open}, \mathsf{FDec})$ are used for defining the security notion of this primitive. Therefore, these algorithms are not used when using this scheme in practice. We note that the definition of RNCE in [6, Section 4.1] does not contain FKG and FDec, but they are necessary for our formalization of RNC-CCA security. The formal definition is as follows.

Definition 9 (Receiver non-commiting encryption). *An RNCE scheme Π with a plaintext space $\mathcal{M}$ consists of the following seven PPT algorithms* $(\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{FKG}, \mathsf{Fake}, \mathsf{Open}, \mathsf{FDec})$. $(\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ *are the same algorithms as those of a PKE scheme.* $(\mathsf{FKG}, \mathsf{Fake}, \mathsf{Open}, \mathsf{FDec})$ *are defined as follows.*

FKG**:** *The fake key generation algorithm, given a security parameter 1^λ, outputs a public key pk and a trapdoor td.*

Fake**:** *The fake encryption algorithm, given a public key pk and a trapdoor td, outputs a fake ciphertext $\widetilde{c}$.*

Open**:** *The opening algorithm, given a public key pk, a trapdoor td, a fake ciphertext $\widetilde{c}$, and a plaintext m, outputs a fake secret key $\widetilde{sk}$.*

FDec**:** *The fake decryption algorithm, given a public key pk, a trapdoor td, and a ciphertext c, outputs $m \in \{\bot\} \cup \mathcal{M}$.*

Next, we define RNC-CCA security for RNCE as follows.

Definition 10 (RNC-CCA security). *For an RNCE scheme $\Pi = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{FKG}, \mathsf{Fake}, \mathsf{Open}, \mathsf{FDec})$ and an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$, we consider the following pair of experiments.*

$\mathsf{Exp}^{\mathsf{rnc\text{-}real}}_{\Pi,\mathcal{A}}(\lambda)$:	$\mathsf{Exp}^{\mathsf{rnc\text{-}sim}}_{\Pi,\mathcal{A}}(\lambda)$:
$(pk, sk) \leftarrow \mathsf{KG}(1^\lambda)$	$(pk, td) \leftarrow \mathsf{FKG}(1^\lambda)$
$(m^*, \mathsf{st}_1) \leftarrow \mathcal{A}_1^{\mathcal{O}_{\mathsf{Dec}}(\cdot)}(pk)$	$(m^*, \mathsf{st}_1) \leftarrow \mathcal{A}_1^{\mathcal{O}_{\mathsf{Dec}}(\cdot)}(pk)$
$c^* \leftarrow \mathsf{Enc}(pk, m^*)$	$c^* \leftarrow \mathsf{Fake}(pk, td)$
$\mathsf{st}_2 \leftarrow \mathcal{A}_2^{\mathcal{O}_{\mathsf{Dec}}(\cdot)}(c^*, \mathsf{st}_1)$	$\mathsf{st}_2 \leftarrow \mathcal{A}_2^{\mathcal{O}_{\mathsf{Dec}}(\cdot)}(c^*, \mathsf{st}_1)$
$sk^* := sk$	$sk^* \leftarrow \mathsf{Open}(pk, td, c^*, m^*)$
Return $b' \leftarrow \mathcal{A}_3(sk^*, \mathsf{st}_2)$	Return $b' \leftarrow \mathcal{A}_3(sk^*, \mathsf{st}_2)$

In $\mathsf{Exp}^{\mathsf{rnc\text{-}real}}_{\Pi,\mathcal{A}}(\lambda)$, a decryption query c is answered by $\mathsf{Dec}(pk, sk, c)$. On the other hand, in $\mathsf{Exp}^{\mathsf{rnc\text{-}sim}}_{\Pi,\mathcal{A}}(\lambda)$, a decryption query c is answered by $\mathsf{FDec}(pk, td, c)$. In both of the experiments, $\mathcal{A}_2$ is not allowed to make a decryption query $c = c^*$ and $\mathcal{A}_3$ is not allowed to make any decryption query.

We say that Π is RNC-CCA secure if for any PPT adversary $\mathcal{A}$,

$$\mathsf{Adv}^{\mathsf{rnc\text{-}cca}}_{\Pi,\mathcal{A}}(\lambda) := |\Pr[\mathsf{Exp}^{\mathsf{rnc\text{-}real}}_{\Pi,\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{rnc\text{-}sim}}_{\Pi,\mathcal{A}}(\lambda) = 1]| = \mathsf{negl}(\lambda).$$

3.2 RNC-CCA Secure RNCE Implies SIM-RSO-CCA Secure PKE

In this section, we show that an RNC-CCA secure RNCE scheme implies a SIM-RSO-CCA secure PKE scheme. Specifically, we show the following theorem.

Theorem 1. *If an RNCE scheme $\Pi = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{FKG}, \mathsf{Fake}, \mathsf{Open}, \mathsf{FDec})$ is RNC-CCA secure, then $\Pi_{\mathsf{rso}} := (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ is a SIM-RSO-CCA secure PKE scheme.*

Before providing the formal proof, we describe an intuition of the proof. Let n be the number of key pairs and $\mathcal{A}$ be an adversary against the SIM-RSO-CCA security of Π_{rso}. In the proof, we firstly construct a PPT simulator $\mathcal{S}$ in $\mathsf{Exp}^{\mathsf{rso\text{-}cca\text{-}sim}}_{n,\Pi_{\mathsf{rso}},\mathcal{S}}(\lambda)$. Specifically, $\mathcal{S}$ computes fake ciphertexts $(\widetilde{c}_j)_{j\in[n]}$ using Fake and fake secret keys $(\widetilde{sk}_j)_{j\in J}$ using Open, where J is the set of corrupted indices. Here, $\mathcal{S}$ can simulate the decryption oracle for $\mathcal{A}$ by FDec using the trapdoors $(td_j)_{j\in[n]}$ generated by $\mathcal{S}$.

Next, in order to move from the real experiment $\mathsf{Exp}^{\mathsf{rso\text{-}cca\text{-}real}}_{n,\Pi_{\mathsf{rso}},\mathcal{A}}(\lambda)$ to the simulated experiment $\mathsf{Exp}^{\mathsf{rso\text{-}cca\text{-}sim}}_{n,\Pi_{\mathsf{rso}},\mathcal{S}}(\lambda)$, we change, step by step, n real challenge ciphertexts $(c_j^*)_{j\in[n]}$ to n fake ciphertexts $(\widetilde{c}_j)_{j\in[n]}$ and n real secret keys $(sk_j)_{j\in[n]}$ to n fake secret keys $(\widetilde{sk}_j)_{j\in[n]}$ which are given to $\mathcal{A}$, respectively. We can show this by the RNC-CCA security of Π using a hybrid argument. Here, we have to deal with some technically subtle point when simulating the decryption oracle for $\mathcal{A}$. Namely, we have to program the behavior of an adversary $\mathcal{B}$ against the RNC-CCA security of Π depending on whether the index i is contained in the corrupted set J output by $\mathcal{A}_2$, where i is the position that $\mathcal{B}$ embeds his own challenge instance into the challenge instances of $\mathcal{A}$. The formal proof is as follows.

Proof of Theorem 1. Let $n = n(\lambda) > 0$ be an arbitrary polynomial that denotes the number of key pairs. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ be any PPT adversary that attacks the SIM-RSO-CCA security of Π_{rso}. First, we construct the following PPT simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ for $\mathcal{A}$.

$\mathcal{S}_1(1^\lambda)$ **:** First, $\mathcal{S}_1$ computes $(pk_j, td_j) \leftarrow \mathsf{FKG}(1^\lambda)$ for all $j \in [n]$. Next, it sets $\mathbf{pk} := (pk_j)_{j\in[n]}$ and runs $\mathcal{A}_1(\mathbf{pk})$. When $\mathcal{A}_1$ makes a decryption query (c, j), $\mathcal{S}_1$ returns $m \leftarrow \mathsf{FDec}(pk_j, td_j, c)$ to $\mathcal{A}_1$. After $\mathcal{A}_1$ outputs a pair $(\mathsf{Dist}, \mathsf{st}_1)$ of the plaintext distribution and state information and terminates, $\mathcal{S}_1$ sets st'_1 as all the information known to $\mathcal{S}_1$, returns $(\mathsf{Dist}, \mathsf{st}'_1)$ to $\mathsf{Exp}^{\mathsf{rso\text{-}cca\text{-}sim}}_{n,\Pi_{\mathsf{rso}},\mathcal{S}}(\lambda)$, and terminates.

$\mathcal{S}_2(\mathsf{st}'_1)$ **:** First, $\mathcal{S}_2$ computes $c^*_j \leftarrow \mathsf{Fake}(pk_j, td_j)$ for all $j \in [n]$, sets $\mathbf{c}^* := (c^*_j)_{j\in[n]}$, and runs $\mathcal{A}_2(\mathbf{c}^*, \mathsf{st}_1)$. When $\mathcal{A}_2$ makes a decryption query (c, j) (such that $(c, j) \neq (c^*_j, j)$), $\mathcal{S}_2$ returns $m \leftarrow \mathsf{FDec}(pk_j, td_j, c)$ to $\mathcal{A}_2$. After $\mathcal{A}_2$ outputs a set of indices $J \subseteq [n]$ and state information st_2 and terminates, $\mathcal{S}_2$ sets st'_2 as all the information known to $\mathcal{S}_2$, returns (J, st'_2) to $\mathsf{Exp}^{\mathsf{rso\text{-}cca\text{-}sim}}_{n,\Pi_{\mathsf{rso}},\mathcal{S}}(\lambda)$, and terminates.

$\mathcal{S}_3(\mathbf{m}^*_J, \mathsf{st}'_2)$ **:** First, $\mathcal{S}_3$ computes $sk^*_j \leftarrow \mathsf{Open}(pk_j, td_j, c^*_j, m^*_j)$ for all $j \in J$, sets $\mathbf{sk}^*_J := (sk^*_j)_{j\in J}$, and runs $\mathcal{A}_3(\mathbf{sk}^*_J, \mathbf{m}^*_J, \mathsf{st}_2)$. When $\mathcal{A}_3$ makes a decryption query (c, j) (such that $(c, j) \neq (c^*_j, j)$ and $j \notin J$), $\mathcal{S}_3$ returns $m \leftarrow \mathsf{FDec}(pk_j, td_j, c)$ to $\mathcal{A}_3$. (Note that $j \notin J$ holds now due to the rule of the SIM-RSO-CCA experiment.) After $\mathcal{A}_3$ outputs out and terminates, $\mathcal{S}_3$ returns out to $\mathsf{Exp}^{\mathsf{rso\text{-}cca\text{-}sim}}_{n,\Pi_{\mathsf{rso}},\mathcal{S}}(\lambda)$ and terminates.

We let $\mathcal{D}$ be any PPT distinguisher for the adversary $\mathcal{A}$ and the above simulator $\mathcal{S}$. We let p_{real} be the probability that $\mathcal{D}$ outputs 1 given the output by $\mathsf{Exp}^{\mathsf{rso\text{-}cca\text{-}real}}_{n,\Pi_{\mathsf{rso}},\mathcal{A}}(\lambda)$ and p_{sim} be the probability that $\mathcal{D}$ outputs 1 given the output by $\mathsf{Exp}^{\mathsf{rso\text{-}cca\text{-}sim}}_{n,\Pi_{\mathsf{rso}},\mathcal{S}}(\lambda)$. Note that $\mathsf{Adv}^{\mathsf{rso\text{-}cca}}_{n,\Pi_{\mathsf{rso}},\mathcal{A},\mathcal{S},\mathcal{D}}(\lambda) = |p_{\mathsf{real}} - p_{\mathsf{sim}}|$. In the following, we show that there exists a PPT adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3)$ that attacks the RNC-CCA security of Π so that $|p_{\mathsf{real}} - p_{\mathsf{sim}}| = n \cdot \mathsf{Adv}^{\mathsf{rnc\text{-}cca}}_{\Pi,\mathcal{B}}(\lambda)$.

Lemma 2. *There exists a PPT adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3)$ such that $|p_{\mathsf{real}} - p_{\mathsf{sim}}| = n \cdot \mathsf{Adv}^{\mathsf{rnc\text{-}cca}}_{\Pi,\mathcal{B}}(\lambda)$.*

Proof of Lemma 2. We construct a PPT adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3)$ that attacks the RNC-CCA security of Π so that $|p_{\mathsf{real}} - p_{\mathsf{sim}}| = n \cdot \mathsf{Adv}^{\mathsf{rnc\text{-}cca}}_{\Pi,\mathcal{B}}(\lambda)$, using the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ and the distinguisher $\mathcal{D}$ as follows.

$\mathcal{B}_1(pk)$ **:** First, $\mathcal{B}_1$ chooses i from $[n]$ uniformly at random and sets $pk_i := pk$. Next, it computes $(pk_j, td_j) \leftarrow \mathsf{FKG}(1^\lambda)$ for all $j \in [i-1]$ and $(pk_j, sk_j) \leftarrow \mathsf{KG}(1^\lambda)$ for all $j \in [i+1, n]$. Then, it sets $\mathbf{pk} := (pk_j)_{j\in[n]}$ and runs $\mathcal{A}_1(\mathbf{pk})$. When $\mathcal{A}_1$ makes a decryption query (c, j), $\mathcal{B}_1$ answers as follows.
 - If $j \in [i-1]$, then $\mathcal{B}_1$ computes $m \leftarrow \mathsf{FDec}(pk_j, td_j, c)$ and returns m to $\mathcal{A}_1$
 - If $j = i$, then $\mathcal{B}_1$ makes a decryption query c. When it receives m from its decryption oracle, it returns m to $\mathcal{A}_1$.
 - If $j \in [i+1, n]$, then $\mathcal{B}_1$ computes $m \leftarrow \mathsf{Dec}(pk_j, sk_j, c)$ and returns m to $\mathcal{A}_1$.

 After $\mathcal{A}_1$ outputs a distribution Dist and state information st_1, $\mathcal{B}_1$ samples $\mathbf{m} := (m^*_j)_{j\in[n]} \leftarrow \mathsf{Dist}$. Finally, $\mathcal{B}_1$ sets st'_1 as all the information known to $\mathcal{B}_1$, returns (m^*_i, st'_1) to the experiment, and terminates.

$\mathcal{B}_2(c^*, \mathsf{st}'_1)$ **:** First, $\mathcal{B}_2$ sets $c^*_i := c^*$ and computes the challenge ciphertexts c^*_j for all $j \in [n]\backslash\{i\}$ as follows.
 - If $j \in [i-1]$, then $\mathcal{B}_2$ computes $c^*_j \leftarrow \mathsf{Fake}(pk_j, td_j)$.
 - If $j \in [i+1, n]$, then $\mathcal{B}_2$ computes $c^*_j \leftarrow \mathsf{Enc}(pk_j, m^*_j)$.

 Next, $\mathcal{B}_2$ sets $\mathbf{c}^* := (c^*_j)_{j\in[n]}$ and runs $\mathcal{A}_2(\mathbf{c}^*, \mathsf{st}_1)$. When $\mathcal{A}_2$ makes a decryption query (c, j), $\mathcal{B}_2$ answers in the same way as $\mathcal{B}_1$ does. (Note that since $\mathcal{A}_2$ is not allowed to make a decryption query (c^*_j, j) for all $j \in [n]$, $\mathcal{B}_2$ can perfectly simulate the decryption oracle for $\mathcal{A}_2$.) After $\mathcal{A}_2$ outputs a set of indices J and state information st_2 and terminates, depending on whether $i \in J$ holds, $\mathcal{B}_2$ proceeds as follows.
 - If $i \notin J$ holds, then $\mathcal{B}_2$ computes sk^*_j for all $j \in J$ as follows.

    - If $j \in [i-1]$, then $\mathcal{B}_2$ computes $sk_j^* \leftarrow \mathsf{Open}(pk_j, td_j, c_j^*, m_j^*)$.
    - If $j \in [i+1, n]$, then $\mathcal{B}_2$ computes $sk_j^* := sk_j$.

  Next, $\mathcal{B}_2$ sets $\mathbf{sk}_J^* := (sk_j^*)_{j \in J}$ and runs $\mathcal{A}_3(\mathbf{sk}_J^*, \mathbf{m}_J^*, \mathsf{st}_2)$. When $\mathcal{A}_3$ makes a decryption query (c, j), $\mathcal{B}_2$ answers in the same way as $\mathcal{B}_1$ does. (Note that since $\mathcal{A}_3$ is not allowed to make a decryption query (c, j) (such that $((c, j) = (c_j^*, j)) \vee (j \in J)$), $\mathcal{B}_2$ can perfectly simulate the decryption oracle for $\mathcal{A}_3$.) After $\mathcal{A}_3$ outputs out and terminates, $\mathcal{B}_2$ runs $\mathcal{D}(\mathbf{m}^*, \mathsf{Dist}, J, \mathsf{out})$. Then, $\mathcal{B}_2$ sets d as the output of $\mathcal{D}$ and st_2' as the concatenation of all the information known to $\mathcal{B}_2$ and the bit d, returns st_2' to the experiment, and terminates.
- If $i \in J$ holds, then $\mathcal{B}_2$ just sets st_2' as all the information known to $\mathcal{B}_2$, returns st_2' to the experiment, and terminates.

$\mathcal{B}_3(sk^*, \mathsf{st}_2')$ **:** If $i \notin J$, then st_2' must contain the bit d, and $\mathcal{B}_3$ outputs it to the experiment and terminates.

Otherwise, (that is, if $i \in J$ holds,) $\mathcal{B}_3$ runs as follows. First, $\mathcal{B}_3$ computes sk_j^* for all $j \in [n]$ as follows.

- If $j \in [i-1]$, then $\mathcal{B}_3$ computes $sk_j^* \leftarrow \mathsf{Open}(pk_j, td_j, c_j^*, m_j^*)$.
- If $j = i$, then $\mathcal{B}_3$ sets $sk_i^* := sk^*$.
- If $j \in [i+1, n]$, then $\mathcal{B}_3$ sets $sk_j^* := sk_j$.

Next, $\mathcal{B}_3$ sets $\mathbf{sk}_J^* := (sk_j^*)_{j \in J}$ and runs $\mathcal{A}_3(\mathbf{sk}_J^*, \mathbf{m}_J^*, \mathsf{st}_2)$. When $\mathcal{A}_3$ makes a decryption query (c, j) (such that $(c, j) \neq (c_j^*, j)$ and $j \notin J$), $\mathcal{B}_3$ answers in the same way as $\mathcal{B}_1$ does. (Note that since $i \in J$ holds now due to the rule of the SIM-RSO-CCA experiment, $\mathcal{A}_3$ is not allowed to make a decryption query of the form (c, i). Thus, $\mathcal{B}_3$ can perfectly simulate the decryption oracle for $\mathcal{A}_3$ although $\mathcal{B}_3$ is not allowed to access his own decryption oracle.) After $\mathcal{A}_3$ outputs out and terminates, $\mathcal{B}_3$ runs $\mathcal{D}(\mathbf{m}^*, \mathsf{Dist}, J, \mathsf{out})$. Finally, $\mathcal{B}_3$ returns the output of $\mathcal{D}$ to the experiment and terminates.

Note that $\mathcal{B}$ does not make a forbidden query c^* to its decryption oracle because $\mathcal{A}$ does not make a decryption query (c_j^*, j) for any $j \in [n]$. Note also that $\mathcal{B}_3$ does not make any decryption query. We can estimate $\mathsf{Adv}^{\mathsf{rnc\text{-}cca}}_{\Pi,\mathcal{B}}(\lambda)$ by

$$\begin{aligned}
&\mathsf{Adv}^{\mathsf{rnc\text{-}cca}}_{\Pi,\mathcal{B}}(\lambda) \\
&= |\Pr[\mathsf{Exp}^{\mathsf{rnc\text{-}real}}_{\Pi,\mathcal{B}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{rnc\text{-}sim}}_{\Pi,\mathcal{B}}(\lambda) = 1]| \\
&= \Bigl| \sum_{t \in [n]} \Pr[(\mathsf{Exp}^{\mathsf{rnc\text{-}real}}_{\Pi,\mathcal{B}}(\lambda) = 1) \wedge (i = t)] - \sum_{t \in [n]} \Pr[(\mathsf{Exp}^{\mathsf{rnc\text{-}sim}}_{\Pi,\mathcal{B}}(\lambda) = 1) \wedge (i = t)] \Bigr| \\
&= \Bigl| \sum_{t \in [n]} \Pr[i = t] \cdot \Pr[\mathsf{Exp}^{\mathsf{rnc\text{-}real}}_{\Pi,\mathcal{B}}(\lambda) = 1 | i = t] - \sum_{t \in [n]} \Pr[i = t] \cdot \Pr[\mathsf{Exp}^{\mathsf{rnc\text{-}sim}}_{\Pi,\mathcal{B}}(\lambda) = 1 | i = t] \Bigr|.
\end{aligned}$$

In the following, for each $t \in [n]$, we let $q_t^{\mathsf{real}} := \Pr[\mathsf{Exp}^{\mathsf{rnc\text{-}real}}_{\Pi,\mathcal{B}}(\lambda) = 1 | i = t]$ and $q_t^{\mathsf{sim}} := \Pr[\mathsf{Exp}^{\mathsf{rnc\text{-}sim}}_{\Pi,\mathcal{B}}(\lambda) = 1 | i = t]$.

Since i is chosen from $[n]$ uniformly at random, $\Pr[i = t] = \frac{1}{n}$ holds for all $i \in [n]$. Thus, $\mathsf{Adv}^{\mathsf{rnc\text{-}cca}}_{\Pi,\mathcal{B}}(\lambda) = \frac{1}{n} \cdot |\sum_{t \in [n]} q_t^{\mathsf{real}} - \sum_{t \in [n]} q_t^{\mathsf{sim}}|$ holds. Next, for all $t \in [n-1]$, when $\mathcal{B}$ interacts with $\mathsf{Exp}^{\mathsf{rnc\text{-}real}}_{\Pi,\mathcal{B}}(\lambda)$ and $i = t+1$ holds, the information given to $\mathcal{A}$ and $\mathcal{D}$ is exactly the same as that when $\mathcal{B}$ interacts with $\mathsf{Exp}^{\mathsf{rnc\text{-}sim}}_{\Pi,\mathcal{B}}(\lambda)$ and $i = t$ holds. That is, for all $t \in [n-1]$, we have $q_{t+1}^{\mathsf{real}} = q_t^{\mathsf{sim}}$. Thus, we have $|\sum_{t \in [n]} q_t^{\mathsf{real}} - \sum_{t \in [n]} q_t^{\mathsf{sim}}| = |q_1^{\mathsf{real}} - q_n^{\mathsf{sim}}|$.

Finally, $\mathcal{B}$ perfectly simulates $\mathsf{Exp}^{\mathsf{rso\text{-}cca\text{-}real}}_{n,\Pi_{\mathsf{rso}},\mathcal{A}}(\lambda)$ for $\mathcal{A}$ and $\mathcal{D}$ when $\mathcal{B}$ samples $i = 1$ and interacts with $\mathsf{Exp}^{\mathsf{rnc\text{-}real}}_{\Pi,\mathcal{B}}(\lambda)$. On the other hand, $\mathcal{B}$ perfectly simulates the information for $\mathcal{A}$ and $\mathcal{D}$ that is given by $\mathcal{S}$ in $\mathsf{Exp}^{\mathsf{rso\text{-}cca\text{-}sim}}_{n,\Pi_{\mathsf{rso}},\mathcal{S}}(\lambda)$ when $\mathcal{B}$ samples $i = n$ and interacts with $\mathsf{Exp}^{\mathsf{rnc\text{-}sim}}_{\Pi,\mathcal{B}}(\lambda)$. Therefore, $q_1^{\mathsf{real}} = p_{\mathsf{real}}$ and $q_n^{\mathsf{sim}} = p_{\mathsf{sim}}$ hold, that is, we have $|q_1^{\mathsf{real}} - q_n^{\mathsf{sim}}| = |p_{\mathsf{real}} - p_{\mathsf{sim}}|$.

From the above arguments, it holds that $\mathsf{Adv}^{\mathsf{rnc\text{-}cca}}_{\Pi,\mathcal{B}}(\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{rnc\text{-}real}}_{\Pi,\mathcal{B}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{rnc\text{-}sim}}_{\Pi,\mathcal{B}}(\lambda) = 1]| = \frac{1}{n} \cdot |p_{\mathsf{real}} - p_{\mathsf{sim}}|$. That is, $|p_{\mathsf{real}} - p_{\mathsf{sim}}| = n \cdot \mathsf{Adv}^{\mathsf{rnc\text{-}cca}}_{\Pi,\mathcal{B}}(\lambda)$ holds. $\square$ (**Lemma 2**)

At the beginning of this proof, $\mathcal{A}$, n, and $\mathcal{D}$ are chosen arbitrarily. Hence, for any polynomial n and PPT adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that for any PPT distinguisher $\mathcal{D}$, there exists a PPT

```
$\mathsf{KG}'(1^\lambda)$ :
    $\alpha \leftarrow \{0,1\}$
    $(pk_0, sk_0) \leftarrow \mathsf{KG}(1^\lambda)$
    $(pk_1, sk_1) \leftarrow \mathsf{KG}(1^\lambda)$
    $crs \leftarrow \mathsf{CRSGen}(1^\lambda)$
    $pk := (pk_0, pk_1, crs)$
    $sk := (\alpha, sk_\alpha)$
    Return $(pk, sk)$
```

```
$\mathsf{Enc}'(pk, m)$ :
    $(r_0, r_1) \leftarrow \mathcal{R}_\Pi^2$
    $c_0 \leftarrow \mathsf{Enc}(pk_0, m; r_0)$
    $c_1 \leftarrow \mathsf{Enc}(pk_1, m; r_1)$
    $x := (pk_0, pk_1, c_0, c_1)$
    $w := (m, r_0, r_1)$
    $\pi \leftarrow \mathsf{Prove}(crs, x, w)$
    Return $c := (c_0, c_1, \pi)$
```

```
$\mathsf{Dec}'(pk, sk, c)$ :
    $x := (pk_0, pk_1, c_0, c_1)$
    If $\mathsf{Verify}(crs, x, \pi) = 1$ then
        $m \leftarrow \mathsf{Dec}(pk_\alpha, sk_\alpha, c_\alpha)$
        Return $m$
    else Return $\perp$
```

```
$\mathsf{FKG}'(1^\lambda)$ :
    $\alpha \leftarrow \{0,1\}$
    $(pk_0, sk_0) \leftarrow \mathsf{KG}(1^\lambda)$
    $(pk_1, sk_1) \leftarrow \mathsf{KG}(1^\lambda)$
    $(crs, tk) \leftarrow \mathsf{SimCRS}(1^\lambda)$
    $pk := (pk_0, pk_1, crs)$
    $td := (\alpha, sk_0, sk_1, tk)$
    Return $(pk, td)$
```

```
$\mathsf{Fake}'(pk, td)$ :
    $c_\alpha \leftarrow \mathsf{Enc}(pk_\alpha, 0)$
    $c_{1\oplus\alpha} \leftarrow \mathsf{Enc}(pk_{1\oplus\alpha}, 1)$
    $x := (pk_0, pk_1, c_0, c_1)$
    $\pi \leftarrow \mathsf{SimPrv}(tk, x)$
    Return $\widetilde{c} := (c_0, c_1, \pi)$
```

```
$\mathsf{Open}'(pk, td, \widetilde{c}, m)$ :
    $\widetilde{sk} := (\alpha \oplus m, sk_{\alpha\oplus m})$
    Return $\widetilde{sk}$
```

```
$\mathsf{FDec}'(pk, td, c)$ :
    $x := (pk_0, pk_1, c_0, c_1)$
    If $\mathsf{Verify}(crs, x, \pi) = 1$ then
        $m \leftarrow \mathsf{Dec}(pk_0, sk_0, c_0)$
        Return $m$
    else Return $\perp$
```

Fig. 1. Generic construction of RNC-CCA secure RNCE Π'.

adversary $\mathcal{B}$ such that $\mathsf{Adv}^{\mathsf{rso\text{-}cca}}_{n,\Pi_{\mathsf{rso}},\mathcal{A},\mathcal{S},\mathcal{D}}(\lambda) = |\Pr[\mathcal{D}(\mathsf{Exp}^{\mathsf{rso\text{-}cca\text{-}real}}_{n,\Pi_{\mathsf{rso}},\mathcal{A}}(\lambda)) = 1] - \Pr[\mathcal{D}(\mathsf{Exp}^{\mathsf{rso\text{-}cca\text{-}sim}}_{n,\Pi_{\mathsf{rso}},\mathcal{S}}(\lambda)) = 1]| = |p_{\mathsf{real}} - p_{\mathsf{sim}}| = n \cdot \mathsf{Adv}^{\mathsf{rnc\text{-}cca}}_{\Pi,\mathcal{B}}(\lambda)$ holds. Therefore, we have $\mathsf{Adv}^{\mathsf{rso\text{-}cca}}_{n,\Pi_{\mathsf{rso}},\mathcal{A},\mathcal{S},\mathcal{D}}(\lambda) = \mathsf{negl}(\lambda)$ by the assumption that Π satisfies RNC-CCA security. □ (**Theorem 1**)

4 Generic Construction of RNC-CCA Secure RNCE

In this section, we show our generic construction of RNC-CCA secure RNCE with the plaintext space $\{0,1\}$. First, in Section 4.1, we describe our generic construction. Then, in Section 4.2, we give the proof of RNC-CCA security for our generic construction.

4.1 Description

In this section, we formally describe our generic construction of RNC-CCA secure RNCE with the plaintext space $\{0,1\}$. Let $\Pi = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ be a PKE scheme with the plaintext space $\{0,1\}$ and $\mathcal{R}_\Pi$ be a randomness space for the encryption algorithm Enc. Let $\Phi = (\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{SimCRS}, \mathsf{SimPrv})$ be a non-interactive proof system for L_{eq}, where

$$L_{eq} := \left\{ (pk_0, pk_1, c_0, c_1) \middle| \, \exists (m, r_0, r_1) \text{ s.t. } (c_0 = \mathsf{Enc}(pk_0, m; r_0)) \wedge (c_1 = \mathsf{Enc}(pk_1, m; r_1)) \right\}.$$

Then, we construct an RNCE scheme $\Pi' = (\mathsf{KG}', \mathsf{Enc}', \mathsf{Dec}', \mathsf{FKG}', \mathsf{Fake}', \mathsf{Open}', \mathsf{FDec}')$ with the plaintext space $\{0,1\}$ as described in Fig. 1. We note that, considering a real ciphertext c and a real secret key sk, the correctness of the decryption of Π' is straightforward due to the correctness of Π and Φ.

Correctness of decryption using a fake secret key. Although it is not necessary for a security proof, we believe that it is instructive to check the correctness of the decryption procedure using a fake secret key. Hence, we confirm that a fake secret key can decrypt a fake ciphertext and a real ciphertext by the decryption algorithm Dec'.

Let (pk, td) be a public key/trapdoor pair generated by $\mathsf{FKG}'(1^\lambda)$, $m \in \{0,1\}$ be an arbitrary plaintext, $\widetilde{c} = (c_0, c_1, \pi)$ be a fake ciphertext generated by $\mathsf{Fake}'(pk, td)$, and $\widetilde{sk} = (\alpha \oplus m, sk_{\alpha\oplus m})$ be a fake secret key generated by $\mathsf{Open}'(pk, td, \widetilde{c}, m)$. First, we confirm that the fake secret key $\widetilde{sk}$ can decrypt the fake ciphertext $\widetilde{c}$ to the plaintext m. For example, we consider the case $\alpha = 0$. In this case, c_0 is a ciphertext of 0, c_1 is a ciphertext of 1, π is generated by $\mathsf{SimPrv}(tk, (pk_0, pk_1, c_0, c_1))$, and the fake secret key is (m, sk_m). Clearly, the fake ciphertext passes the verification in Dec'. Then, Dec' computes and outputs $\mathsf{Dec}(pk_m, sk_m, c_m)$, which results in m. Therefore, the fake secret key $\widetilde{sk}$ can correctly decrypt the fake ciphertext $\widetilde{c}$ to the plaintext m. The case of $\alpha = 1$ can be confirmed similarly.

Furthermore, for any plaintext $m' \in \{0,1\}$ (not necessarily m), the fake secret key $(\alpha \oplus m, sk_{\alpha \oplus m})$ can decrypt a real ciphertext $c = (c_0, c_1, \pi)$ generated by $\mathsf{Enc}'(pk, m')$ to the plaintext m' because c_0 and c_1 encrypt the same plaintext m'.

How to expand the plaintext space of our generic construction. In the above, we only give the construction whose plaintext space is $\{0,1\}$. However, we can expand the plaintext space by using our single-bit construction in a parallel way except for the generation of a proof of an OTSS-NIZK. More concretely, if we encrypt an ℓ-bit plaintext $m = m_1 \| \cdots \| m_\ell$, the procedure is as follows. Firstly, we generate a public key $pk = ((pk_j^i)_{i \in [\ell], j \in \{0,1\}}, crs)$ and a secret key $sk = (\alpha_i, sk_{\alpha_i}^i)_{i \in [\ell]}$, where $\alpha_1, \cdots, \alpha_\ell \leftarrow \{0,1\}$, $(pk_j^i, sk_j^i) \leftarrow \mathsf{KG}(1^\lambda)$ for all $(i,j) \in [\ell] \times \{0,1\}$, and crs denotes a CRS of an OTSS-NIZK. Next, we compute a ciphertext $c = ((c_j^i)_{i \in [\ell], j \in \{0,1\}}, \pi)$, where $c_j^i \leftarrow \mathsf{Enc}(pk_j^i, m_i)$ for all $(i,j) \in [\ell] \times \{0,1\}$ and π is a proof proving that, for each $i \in [\ell]$, the ciphertexts c_0^i and c_1^i encrypt the same plaintext $m_i \in \{0,1\}$. Similarly, for the other procedures, we execute the corresponding one-bit version algorithms in a parallel way for all $i \in [\ell]$ except for the procedure of the OTSS-NIZK. See Appendix A for the details.

4.2 Security Proof

In this section, we show the following theorem.

Theorem 2. *If Π is an IND-CPA secure PKE scheme and Φ is an OTSS-NIZK, then Π' is RNC-CCA secure.*

Before describing the formal proof, we highlight the flow of the proof. We change $\mathsf{Exp}_{\Pi',\mathcal{A}}^{\mathsf{rnc\text{-}real}}(\lambda)$ to $\mathsf{Exp}_{\Pi',\mathcal{A}}^{\mathsf{rnc\text{-}sim}}(\lambda)$ step by step, where $\mathcal{A}$ is an adversary that attacks the RNC-CCA security of Π'. Although the main part of our proof is similar to that of the original double encryption paradigm [26, 29], we have the following three remarkable changes.

First, toward transforming the challenge ciphertext to a fake ciphertext, we make the challenge ciphertext component $c_{1 \oplus \alpha}^*$ encrypt $1 \oplus m^*$. Second, in order to eliminate the information of the bit α from the decryption oracle, when answering a decryption query $c = (c_0, c_1, \pi)$ made by $\mathcal{A}$, we use the component (pk_0, sk_0, c_0) corresponding to the position 0 instead of the component $(pk_\alpha, sk_\alpha, c_\alpha)$ corresponding to the position α. Third, we use $\alpha \oplus m^*$ instead of α in order to make the challenge ciphertext c^* be independent of the challenge plaintext m^*. Due to these changes, the challenge ciphertext c^* and the real secret key sk are respectively switched to the fake ciphertext $\widetilde{c}$ and the fake secret key $\widetilde{sk}$. The formal proof is as follows.

Proof of Theorem 2. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ be any PPT adversary that attacks the RNC-CCA security of Π'. We introduce the following experiments $\{\mathsf{Exp}_i\}_{i=0}^5$.

Exp_0 : Exp_0 is exactly the same as $\mathsf{Exp}_{\Pi',\mathcal{A}}^{\mathsf{rnc\text{-}real}}(\lambda)$. The detailed description is as follows.

1. First, Exp_0 samples $\alpha \leftarrow \{0,1\}$ and computes $(pk_0, sk_0) \leftarrow \mathsf{KG}(1^\lambda)$, $(pk_1, sk_1) \leftarrow \mathsf{KG}(1^\lambda)$, and $crs \leftarrow \mathsf{CRSGen}(1^\lambda)$. Next, Exp_0 sets $pk := (pk_0, pk_1, crs)$ and $sk := (\alpha, sk_\alpha)$ and runs $\mathcal{A}_1(pk)$. When $\mathcal{A}_1$ makes a decryption query $c = (c_0, c_1, \pi)$, Exp_0 checks whether $\mathsf{Verify}(crs, (pk_0, pk_1, c_0, c_1), \pi) = 1$ holds. If this holds, Exp_0 computes $m \leftarrow \mathsf{Dec}(pk_\alpha, sk_\alpha, c_\alpha)$, and returns m to $\mathcal{A}_1$. Otherwise, Exp_0 returns $\perp$ to $\mathcal{A}_1$.
2. When $\mathcal{A}_1$ outputs (m^*, st_1) and terminates, Exp_0 computes the challenge ciphertext c^* as follows. First, Exp_0 samples $(r_0^*, r_1^*) \leftarrow \mathcal{R}_\Pi^2$ and computes $c_0^* \leftarrow \mathsf{Enc}(pk_0, m^*; r_0^*)$, $c_1^* \leftarrow \mathsf{Enc}(pk_1, m^*; r_1^*)$, and $\pi^* \leftarrow \mathsf{Prove}(crs, (pk_0, pk_1, c_0^*, c_1^*), (m^*, r_0^*, r_1^*))$. Next, Exp_0 sets $c^* = (c_0^*, c_1^*, \pi^*)$ and runs $\mathcal{A}_2(c^*, \mathsf{st}_1)$. When $\mathcal{A}_2$ makes a decryption query c, Exp_0 answers in the same way as above.
3. When $\mathcal{A}_2$ outputs state information st_2 and terminates, Exp_0 runs $\mathcal{A}_3(sk, \mathsf{st}_2)$. When $\mathcal{A}_3$ outputs a bit b' and terminates, Exp_0 outputs b'.

Exp_1 : Exp_1 is identical to Exp_0 except for the following change. The common reference string crs is generated by executing $(crs, tk) \leftarrow \mathsf{SimCRS}(1^\lambda)$ and Exp_1 generates a simulated proof $\pi^* \leftarrow \mathsf{SimPrv}(tk, (pk_0, pk_1, c_0^*, c_1^*))$ when computing the challenge ciphertext c^*.

Exp_2 : Exp_2 is identical to Exp_1 except that when computing the challenge ciphertext c^*, Exp_2 computes $c_{1\oplus\alpha}^* \leftarrow \mathsf{Enc}(pk_{1\oplus\alpha}, 1 \oplus m^*)$ instead of $c_{1\oplus\alpha}^* \leftarrow \mathsf{Enc}(pk_{1\oplus\alpha}, m^*)$.

Exp_3 : Exp_3 is identical to Exp_2 except that when responding to a decryption query $c = (c_0, c_1, \pi)$, Exp_3 answers $m \leftarrow \mathsf{Dec}(pk_0, sk_0, c_0)$ instead of $m \leftarrow \mathsf{Dec}(pk_\alpha, sk_\alpha, c_\alpha)$ if $\mathsf{Verify}(crs, (pk_0, pk_1, c_0, c_1), \pi) = 1$ holds. Note that the decryption procedure in Exp_3 is exactly the same as FDec'.

Exp_4 : Exp_4 is identical to Exp_3 except that $\alpha \oplus m^*$ is used instead of α. That is, when computing the challenge ciphertext c^*, Exp_4 computes c_0^* and c_1^* by $c_{\alpha\oplus m^*}^* \leftarrow \mathsf{Enc}(pk_{\alpha\oplus m^*}, m^*)$ and $c_{\alpha\oplus(1\oplus m^*)}^* \leftarrow \mathsf{Enc}(pk_{\alpha\oplus(1\oplus m^*)}, 1 \oplus m^*)$. Moreover, Exp_4 gives the secret key $sk = (\alpha \oplus m^*, sk_{\alpha\oplus m^*})$ to $\mathcal{A}_3$ instead of $sk = (\alpha, sk_\alpha)$.

Exp_5 : Exp_5 is exactly the same as $\mathsf{Exp}^{\mathsf{rnc\text{-}sim}}_{\Pi',\mathcal{A}}(\lambda)$.

We let $p_i := \Pr[\mathsf{Exp}_i(\lambda) = 1]$ for all $i \in [0, 5]$. Then, we have

$$\mathsf{Adv}^{\mathsf{rnc\text{-}cca}}_{\Pi',\mathcal{A}}(\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{rnc\text{-}real}}_{\Pi',\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{rnc\text{-}sim}}_{\Pi',\mathcal{A}}(\lambda) = 1]| = |p_0 - p_5| \leq \sum_{i=0}^{4} |p_i - p_{i+1}|.$$

It remains to show how each $|p_i - p_{i+1}|$ is upper-bounded. To this end, in the following, we show that there exist an adversary $\mathcal{E} = (\mathcal{E}_1, \mathcal{E}_2)$ against the ZK property of Φ such that $|p_0 - p_1| = \mathsf{Adv}^{\mathsf{zk}}_{\Phi,\mathcal{E}}(\lambda)$ (Lemma 3), an adversary $\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2)$ against the IND-CPA security of Π such that $|p_1 - p_2| = \mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\Pi,\mathcal{F}}(\lambda)$ (Lemma 4), and an adversary $\mathcal{G} = (\mathcal{G}_1, \mathcal{G}_2)$ against the OT-SS of Φ such that $|p_2 - p_3| \leq \mathsf{Adv}^{\mathsf{ot\text{-}ss}}_{\Phi,\mathcal{G}}(\lambda)$ (Lemma 5). Then, we show that $|p_3 - p_4| = 0$ holds (Lemma 6). The main reason why this is true, is because α is chosen uniformly at random, and thus $\alpha \oplus m^*$ is also distributed uniformly at random. Finally, we show that $|p_4 - p_5| = 0$ holds (Lemma 7) by showing that Exp_4 and Exp_5 are identical.

Lemma 3. *There exists a PPT adversary $\mathcal{E} = (\mathcal{E}_1, \mathcal{E}_2)$ such that $|p_0 - p_1| = \mathsf{Adv}^{\mathsf{zk}}_{\Phi,\mathcal{E}}(\lambda)$.*

Proof of Lemma 3. We construct a PPT adversary $\mathcal{E} = (\mathcal{E}_1, \mathcal{E}_2)$ that attacks the ZK property of Φ so that $|p_0 - p_1| = \mathsf{Adv}^{\mathsf{zk}}_{\Phi,\mathcal{E}}(\lambda)$, using the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ as follows.

$\mathcal{E}_1(crs)$ **:** First, $\mathcal{E}_1$ samples $\alpha \leftarrow \{0, 1\}$ and computes $(pk_0, sk_0) \leftarrow \mathsf{KG}(1^\lambda)$ and $(pk_1, sk_1) \leftarrow \mathsf{KG}(1^\lambda)$. Next, it sets $pk := (pk_0, pk_1, crs)$ and $sk := (\alpha, sk_\alpha)$, and runs $\mathcal{A}_1(pk)$. When $\mathcal{A}_1$ makes a decryption query c, $\mathcal{E}_1$ returns $m \leftarrow \mathsf{Dec}'(pk, sk, c)$ to $\mathcal{A}_1$. When $\mathcal{A}_1$ outputs the challenge plaintext m^* and state information st_1 and then terminates, $\mathcal{E}_1$ samples $(r_0^*, r_1^*) \leftarrow \mathcal{R}_\Pi^2$ and computes $c_0^* \leftarrow \mathsf{Enc}(pk_0, m^*; r_0^*)$ and $c_1^* \leftarrow \mathsf{Enc}(pk_1, m^*; r_1^*)$. Finally, $\mathcal{E}_1$ sets $x^* := (pk_0, pk_1, c_0^*, c_1^*)$, $w^* := (m^*, r_0^*, r_1^*)$, and st_1' as all the information known to $\mathcal{E}_1$, returns $(x^*, w^*, \mathsf{st}_1')$, and terminates.

$\mathcal{E}_2(\pi, \mathsf{st}_1')$ **:** First, $\mathcal{E}_2$ sets $\pi^* := \pi$ and $c^* := (c_0^*, c_1^*, \pi^*)$, and then runs $\mathcal{A}_2(c^*, \mathsf{st}_1)$. When $\mathcal{A}_2$ makes a decryption query c, $\mathcal{E}_2$ returns $m \leftarrow \mathsf{Dec}'(pk, sk, c)$ to $\mathcal{A}_2$. When $\mathcal{A}_2$ outputs state information st_2 and terminates, $\mathcal{E}_2$ sets $sk^* := sk = (\alpha, sk_\alpha)$ and runs $\mathcal{A}_3(sk^*, \mathsf{st}_2)$. When $\mathcal{A}_3$ outputs a bit b', $\mathcal{E}_2$ returns b' to its experiment and terminates.

We can see that $\mathcal{E}$ perfectly simulates Exp_0 for $\mathcal{A}$ if it receives real components from $\mathsf{Exp}^{\mathsf{zk\text{-}real}}_{\Phi,\mathcal{E}}(\lambda)$. This ensures that the probability that $\mathcal{E}$ outputs 1 given real components is exactly the same as the probability that $\mathcal{A}$ outputs 1 in Exp_0. That is, $\Pr[\mathsf{Exp}^{\mathsf{zk\text{-}real}}_{\Phi,\mathcal{E}}(\lambda) = 1] = p_0$ holds. On the other hand, $\mathcal{E}$ perfectly simulates Exp_1 for $\mathcal{A}$ if it receives simulated components from $\mathsf{Exp}^{\mathsf{zk\text{-}sim}}_{\Phi,\mathcal{E}}(\lambda)$. This ensures that the probability that $\mathcal{E}$ outputs 1 given simulated components is exactly the same as the probability that $\mathcal{A}$ outputs 1 in Exp_1. That is, $\Pr[\mathsf{Exp}^{\mathsf{zk\text{-}sim}}_{\Phi,\mathcal{E}}(\lambda) = 1] = p_1$ holds. Therefore, it holds that $\mathsf{Adv}^{\mathsf{zk}}_{\Phi,\mathcal{E}}(\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{zk\text{-}real}}_{\Phi,\mathcal{E}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{zk\text{-}sim}}_{\Phi,\mathcal{E}}(\lambda) = 1]| = |p_0 - p_1|$. $\square$ **(Lemma 3)**

Lemma 4. *There exists a PPT adversary $\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2)$ such that $|p_1 - p_2| = \mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\Pi,\mathcal{F}}(\lambda)$.*

Proof of Lemma 4. We construct a PPT adversary $\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2)$ that attacks the IND-CPA security of Π so that $|p_1 - p_2| = \mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\Pi,\mathcal{F}}(\lambda)$, using the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ as follows.

$\mathcal{F}_1(pk')$ **:** First, $\mathcal{F}_1$ samples $\alpha \leftarrow \{0,1\}$ and computes $(pk'', sk'') \leftarrow \mathsf{KG}(1^\lambda)$ and $(crs, tk) \leftarrow \mathsf{SimCRS}(1^\lambda)$. Next, it sets $(pk_\alpha, pk_{1\oplus\alpha}) := (pk'', pk')$ and $sk_\alpha := sk''$, then sets $pk := (pk_0, pk_1, crs)$ and $sk := (\alpha, sk_\alpha)$ and runs $\mathcal{A}_1(pk)$. When $\mathcal{A}_1$ makes a decryption query c, $\mathcal{F}_1$ computes $m \leftarrow \mathsf{Dec}'(pk, sk, c)$ and returns m to $\mathcal{A}_1$. When $\mathcal{A}_1$ outputs the challenge plaintext m^* and state information st_1 and then terminates, $\mathcal{F}_1$ sets st'_1 as all the information known to $\mathcal{F}_1$, returns $((m^*, 1 \oplus m^*), \mathsf{st}'_1)$, and terminates.

$\mathcal{F}_2(c'^*, \mathsf{st}'_1)$ **:** First, $\mathcal{F}_2$ sets $c^*_{1\oplus\alpha} := c'^*$, then computes $c^*_\alpha \leftarrow \mathsf{Enc}(pk_\alpha, m^*)$ and $\pi^* \leftarrow \mathsf{SimPrv}(tk, (pk_0, pk_1, c^*_0, c^*_1))$. Then, it sets $c^* := (c^*_0, c^*_1, \pi^*)$ and runs $\mathcal{A}_2(c^*, \mathsf{st}_1)$. When $\mathcal{A}_2$ makes a decryption query c, $\mathcal{F}_2$ computes $m \leftarrow \mathsf{Dec}'(pk, sk, c)$ and returns m to $\mathcal{A}_2$. When $\mathcal{A}_2$ outputs state information st_2 and terminates, $\mathcal{F}_2$ sets $sk^* := (\alpha, sk_\alpha)$ and runs $\mathcal{A}_3(sk^*, \mathsf{st}_2)$. When $\mathcal{A}_3$ outputs a bit b', $\mathcal{F}_2$ returns b' to its experiment and terminates.

We let b be the challenge bit for $\mathcal{F}$ in its experiment. We can see that $\mathcal{F}$ perfectly simulates Exp_1 for $\mathcal{A}$ if it receives a ciphertext of m^* from its experiment. This ensures that the probability that $\mathcal{F}$ outputs 1 when its challenge bit is $b = 0$ is exactly the same as the probability that $\mathcal{A}$ outputs 1 in Exp_1. On the other hand, $\mathcal{F}$ perfectly simulates Exp_2 for $\mathcal{A}$ if it receives a ciphertext of $1 \oplus m^*$ from its experiment. This ensures that the probability that $\mathcal{F}$ outputs 1 when its challenge bit is $b = 1$ is exactly the same as the probability that $\mathcal{A}$ outputs 1 in Exp_2. This implies that $\mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\Pi,\mathcal{F}}(\lambda) = |p_1 - p_2|$. $\square$ (**Lemma 4**)

Lemma 5. *There exists a PPT adversary $\mathcal{G} = (\mathcal{G}_1, \mathcal{G}_2)$ such that $|p_2 - p_3| \le \mathsf{Adv}^{\mathsf{ot\text{-}ss}}_{\Phi,\mathcal{G}}(\lambda)$.*

Proof of Lemma 5. For $i \in \{2, 3\}$, we let $\mathbf{Bad}_i$ be the event that $\mathcal{A}_2$ makes a decryption query $c = (c_0, c_1, \pi)$ satisfying $(\mathsf{Dec}(pk_0, sk_0, c_0) \ne \mathsf{Dec}(pk_1, sk_1, c_1)) \wedge (\mathsf{Verify}(crs, (pk_0, pk_1, c_0, c_1), \pi) = 1)$ in Exp_i. (We call such a decryption query a *bad decryption query*.) Exp_2 proceeds identically to Exp_3 unless $\mathbf{Bad}_2$ happens. Therefore, the inequality $|p_2 - p_3| \le \Pr[\mathbf{Bad}_2] = \Pr[\mathbf{Bad}_3]$ holds. In the following, we show that one can construct a PPT adversary $\mathcal{G} = (\mathcal{G}_1, \mathcal{G}_2)$ that attacks the OT-SS of Φ so that $\Pr[\mathbf{Bad}_2] = \mathsf{Adv}^{\mathsf{ot\text{-}ss}}_{\Phi,\mathcal{G}}(\lambda)$, using the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$.

$\mathcal{G}_1(crs)$ **:** First, $\mathcal{G}_1$ samples $\alpha \leftarrow \{0,1\}$ and computes $(pk_0, sk_0) \leftarrow \mathsf{KG}(1^\lambda)$ and $(pk_1, sk_1) \leftarrow \mathsf{KG}(1^\lambda)$. Next, $\mathcal{G}_1$ sets $pk := (pk_0, pk_1, crs)$ and $sk := (\alpha, sk_\alpha)$, then runs $\mathcal{A}_1(pk)$. When $\mathcal{A}_1$ makes a decryption query $c = (c_0, c_1, \pi)$, $\mathcal{G}_1$ computes $m \leftarrow \mathsf{Dec}'(pk, sk, c)$ and returns m to $\mathcal{A}_1$. When $\mathcal{A}_1$ outputs the challenge plaintext m^* and state information st_1 and terminates, $\mathcal{G}_1$ computes $c^*_\alpha \leftarrow \mathsf{Enc}(pk_\alpha, m^*)$ and $c^*_{1\oplus\alpha} \leftarrow \mathsf{Enc}(pk_{1\oplus\alpha}, 1 \oplus m^*)$. Finally, $\mathcal{G}_1$ sets st'_1 as all the information known to $\mathcal{G}_1$, returns $((pk_0, pk_1, c^*_0, c^*_1), \mathsf{st}'_1)$, and terminates.

$\mathcal{G}_2(\pi^*, \mathsf{st}'_1)$ **:** First, $\mathcal{G}_2$ sets $c^* := (c^*_0, c^*_1, \pi^*)$ and runs $\mathcal{A}_2(c^*, \mathsf{st}_1)$. When $\mathcal{A}_2$ makes a decryption query c, $\mathcal{G}_2$ parses $c = (c_0, c_1, \pi)$ and checks whether $(\mathsf{Dec}(pk_0, sk_0, c_0) \ne \mathsf{Dec}(pk_1, sk_1, c_1)) \wedge (\mathsf{Verify}(crs, (pk_0, pk_1, c_0, c_1), \pi) = 1)$ holds. If this holds, $\mathcal{G}_2$ returns $((pk_0, pk_1, c_0, c_1), \pi)$ to its experiment, and terminates. Otherwise, $\mathcal{G}_2$ computes $m \leftarrow \mathsf{Dec}'(pk, sk, c)$ and returns m to $\mathcal{A}_2$. When $\mathcal{A}_2$ outputs state information st_2 and terminates, $\mathcal{G}_2$ gives up and terminates.

From the above construction of $\mathcal{G}$, it is easy to see that $\mathcal{G}$ perfectly simulates the experiment Exp_2 for $\mathcal{A}$. Here, the success condition of $\mathcal{G}$ is to output a pair of a statement and a proof $((pk_0, pk_1, c_0, c_1), \pi)$ satisfying $((x^*, \pi^*) \ne (x, \pi)) \wedge (\mathsf{Verify}(crs, (pk_0, pk_1, c_0, c_1), \pi) = 1) \wedge ((pk_0, pk_1, c_0, c_1) \notin L_{eq})$, where $x^* = (pk_0, pk_1, c^*_0, c^*_1)$ and $x = (pk_0, pk_1, c_0, c_1)$. If $\mathcal{A}_2$ makes a bad decryption query $c = (c_0, c_1, \pi)$, then $\mathsf{Dec}(pk_0, sk_0, c_0) \ne \mathsf{Dec}(pk_1, sk_1, c_1)$ and $\mathsf{Verify}(crs, (pk_0, pk_1, c_0, c_1), \pi) = 1$ hold. Here, the correctness of Π implies that $(pk_0, pk_1, c_0, c_1) \notin L_{eq}$. Moreover, if $\mathcal{A}_2$ makes such a decryption query c, $(c^*_0, c^*_1, \pi^*) = c^* \ne c = (c_0, c_1, \pi)$ holds. Thus, when $\mathcal{A}_2$ makes such a decryption query c, $\mathcal{G}$ achieves its success condition by returning $((pk_0, pk_1, c_0, c_1), \pi)$ to its experiment. We note that $\mathcal{G}$ can detect that the event $\mathbf{Bad}_2$ occurs because $\mathcal{G}$ has both of the secret keys sk_0 and sk_1. From the above arguments, the probability that $\mathcal{A}_2$ makes a bad decryption query is exactly the same as the probability that $\mathcal{G}$ breaks the OT-SS of Φ. Therefore, we have $\Pr[\mathbf{Bad}_2] = \mathsf{Adv}^{\mathsf{ot\text{-}ss}}_{\Phi,\mathcal{G}}(\lambda)$, which in turn implies $|p_2 - p_3| \leq \mathsf{Adv}^{\mathsf{ot\text{-}ss}}_{\Phi,\mathcal{G}}(\lambda)$. □ **(Lemma 5)**

Lemma 6. $|p_3 - p_4| = 0$ *holds.*

Proof of Lemma 6. In the experiment Exp_4, we use $\alpha \oplus m^*$ instead of α. This changes the challenge ciphertext components c_0^* and c_1^* so that they are computed by $c^*_{\alpha\oplus m^*} \leftarrow \mathsf{Enc}(pk_{\alpha\oplus m^*}, m^*)$ and $c^*_{\alpha\oplus(1\oplus m^*)} \leftarrow \mathsf{Enc}(pk_{\alpha\oplus(1\oplus m^*)}, 1\oplus m^*)$, and the secret key (α, sk_α) to $(\alpha\oplus m^*, sk_{\alpha\oplus m^*})$.

In the following, we let α' be $\alpha \oplus m^*$. Since $\alpha \in \{0,1\}$ is chosen uniformly at random, α' is also uniformly distributed. Moreover, $\mathcal{A}$'s view does not change between Exp_3 and Exp_4. Concretely, in Exp_3, the challenge ciphertext components c^*_α and $c^*_{1\oplus\alpha}$ are respectively computed by $c^*_\alpha \leftarrow \mathsf{Enc}(pk_\alpha, m^*)$ and $c^*_{1\oplus\alpha} \leftarrow \mathsf{Enc}(pk_{1\oplus\alpha}, 1\oplus m^*)$ and the secret key sk is (α, sk_α). Similarly, in Exp_4, the challenge ciphertext components $c^*_{\alpha'}$ and $c^*_{1\oplus\alpha'}$ are respectively computed by $c_{\alpha'} \leftarrow \mathsf{Enc}(pk_{\alpha'}, m^*)$ and $c_{1\oplus\alpha'} \leftarrow \mathsf{Enc}(pk_{1\oplus\alpha'}, 1\oplus m^*)$, and the secret key sk is $(\alpha', sk_{\alpha'})$. Thus, $\mathcal{A}$'s view does not change between Exp_3 and Exp_4. Hence, we have $|p_3 - p_4| = 0$. □ **(Lemma 6)**

Lemma 7. $|p_4 - p_5| = 0$ *holds.*

Proof of Lemma 7. We explain that the information that the adversary $\mathcal{A}$ gets in Exp_4 is exactly the same as that in Exp_5 based on the constructions of FKG', Fake', Open', and FDec'.

First, the public key $pk = (pk_0, pk_1, crs)$ given to $\mathcal{A}_1$ in Exp_4 is exactly the same as one which $\mathcal{A}$ gets in Exp_5.

Second, we compute the challenge ciphertext components c_0^* and c_1^* in Exp_4 by $c^*_{\alpha\oplus m^*} \leftarrow \mathsf{Enc}(pk_{\alpha\oplus m^*}, m^*)$ and $c^*_{1\oplus(\alpha\oplus m^*)} \leftarrow \mathsf{Enc}(pk_{1\oplus(\alpha\oplus m^*)}, 1\oplus m^*)$. We can see that the above modified challenge ciphertexts do not depend on the value of $m^* \in \{0,1\}$. That is, these are computed by $c^*_\alpha \leftarrow \mathsf{Enc}(pk_\alpha, 0)$ and $c^*_{1\oplus\alpha} \leftarrow \mathsf{Enc}(pk_{1\oplus\alpha}, 1)$ regardless of the value of $m^* \in \{0,1\}$. This is exactly how they are generated in Exp_5.

Third, we can easily confirm that the decryption oracle in Exp_4 behaves in exactly the same way as in Exp_5.

Finally, the secret key $sk = (\alpha\oplus m^*, sk_{\alpha\oplus m^*})$ which $\mathcal{A}$ gets in Exp_4 is exactly the same as one which $\mathcal{A}$ gets in Exp_5.

From the above arguments, we can conclude $|p_4 - p_5| = 0$. □ **(Lemma 7)**

Putting everything together, we obtain

$$\mathsf{Adv}^{\mathsf{rnc\text{-}cca}}_{\Pi',\mathcal{A}}(\lambda) = |p_0 - p_5| \leq \sum_{i=0}^{4} |p_i - p_{i+1}| \leq \mathsf{Adv}^{\mathsf{zk}}_{\Phi,\mathcal{E}}(\lambda) + \mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\Pi,\mathcal{F}}(\lambda) + \mathsf{Adv}^{\mathsf{ot\text{-}ss}}_{\Phi,\mathcal{G}}(\lambda).$$

Since Π is IND-CPA secure and Φ is an OTSS-NIZK, for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{rnc\text{-}cca}}_{\Pi',\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$ holds. Therefore, Π' satisfies RNC-CCA security. □ **(Theorem 2)**

5 DDH-based Construction of RNC-CCA Secure RNCE

In this section, we show our concrete construction of RNC-CCA secure RNCE based on the DDH assumption.

Firstly, in Section 5.1, we describe our basic construction (proposed in the conference version) supporting only a polynomial-sized plaintext space. Then, in Section 5.2, we give the proof of RNC-CCA security for this scheme.

Secondly, in Section 5.3, we give our main construction with compact ciphertexts supporting a super-polynomially large plaintext space. This construction is obtained by using our method for efficiently expanding a polynomial-sized plaintext space to a super-polynomially large one. Then, in Section 5.4, we give the proof of RNC-CCA security for this scheme.

$\mathsf{KG}(1^\lambda)$: $g_1 \leftarrow \mathbb{G}$ $w \leftarrow \mathbb{Z}_p^*$ $g_2 := g_1^w$ $x_1, x_2, y_1, y_2, z_1, z_2 \leftarrow \mathbb{Z}_p$ $k := g_1^{x_1} g_2^{x_2}$ $s := g_1^{y_1} g_2^{y_2}$ $t := g_1^{z_1} g_2^{z_2}$ $hk \leftarrow \mathsf{HKG}(1^\lambda)$ $pk := (g_1, g_2, k, s, t, hk)$ $sk := (x_1, x_2, y_1, y_2, z_1, z_2)$ Return (pk, sk)	$\mathsf{Enc}(pk, m)$: $r \leftarrow \mathbb{Z}_p$ $u_1 := g_1^r$ $u_2 := g_2^r$ $e := k^r \cdot g_1^m$ $\mu \leftarrow \mathsf{Hash}(hk, u_1 \| u_2 \| e)$ $v := s^r t^{r\mu}$ Return $c := (u_1, u_2, e, v)$	$\mathsf{Dec}(pk, sk, c)$: Parse $c = (u_1, u_2, e, v)$ $\mu \leftarrow \mathsf{Hash}(hk, u_1 \| u_2 \| e)$ If $u_1^{y_1+z_1\mu} u_2^{y_2+z_2\mu} = v$ then $m := \log_{g_1}\left(\frac{e}{(u_1^{x_1} u_2^{x_2})}\right)$ Return m else Return $\perp$
$\mathsf{FKG}(1^\lambda)$: $g_1 \leftarrow \mathbb{G}$ $w \leftarrow \mathbb{Z}_p^*$ $g_2 := g_1^w$ $x_1, x_2, y_1, y_2, z_1, z_2 \leftarrow \mathbb{Z}_p$ $k := g_1^{x_1} g_2^{x_2}$ $s := g_1^{y_1} g_2^{y_2}$ $t := g_1^{z_1} g_2^{z_2}$ $hk \leftarrow \mathsf{HKG}(1^\lambda)$ $pk := (g_1, g_2, k, s, t, hk)$ $\widetilde{r} \leftarrow \mathbb{Z}_p$ $td := (\widetilde{r}, w, x_1, x_2,$ $y_1, y_2, z_1, z_2)$ Return (pk, td)	$\mathsf{Fake}(pk, td)$: $\widetilde{u_1} := g_1^{\widetilde{r}}$ $\widetilde{u_2} := g_1 g_2^{\widetilde{r}}$ $\widetilde{e} := g_1^{x_2} k^{\widetilde{r}}$ $\widetilde{\mu} \leftarrow \mathsf{Hash}(hk, \widetilde{u_1} \| \widetilde{u_2} \| \widetilde{e})$ $\widetilde{v} := g_1^{y_2+z_2\widetilde{\mu}} s^{\widetilde{r}} t^{\widetilde{r}\widetilde{\mu}}$ Return $\widetilde{c} := (\widetilde{u_1}, \widetilde{u_2}, \widetilde{e}, \widetilde{v})$ $\mathsf{Open}(pk, td, \widetilde{c}, m)$: $x_1' := x_1 + mw \pmod p$ $x_2' := x_2 - m \pmod p$ $\widetilde{sk} := (x_1', x_2', y_1, y_2, z_1, z_2)$ Return $\widetilde{sk}$	$\mathsf{FDec}(pk, td, c)$: Parse $c = (u_1, u_2, e, v)$ $\mu \leftarrow \mathsf{Hash}(hk, u_1 \| u_2 \| e)$ $\widetilde{u_1} := g_1^{\widetilde{r}}$ $\widetilde{u_2} := g_1 g_2^{\widetilde{r}}$ $\widetilde{e} := g_1^{x_2} k^{\widetilde{r}}$ $\widetilde{\mu} \leftarrow \mathsf{Hash}(hk, \widetilde{u_1} \| \widetilde{u_2} \| \widetilde{e})$ If $(\mu \neq \widetilde{\mu}) \wedge (u_1^w = u_2)$ $\wedge (u_1^{y_1+z_1\mu} u_2^{y_2+z_2\mu} = v)$ then $m := \log_{g_1}\left(\frac{e}{(u_1^{x_1} u_2^{x_2})}\right)$ Return m else Return $\perp$

Fig. 2. Basic construction of RNC-CCA secure RNCE Π_{ddh} based on the DDH assumption.

5.1 Description of the Basic Construction

Here, we give the formal description of our basic construction of RNC-CCA secure RNCE with a polynomial-sized plaintext space. One can see that our basic scheme is a variant of the Cramer-Shoup encryption scheme [7]. The only difference is that we encode a plaintext m by the group element g_1^m, where g_1 is a generator of the underlying group. This encoding is essential for the opening algorithm Open of our proposed scheme. The plaintext space of our basic scheme needs to be of polynomial-size since we need to compute the discrete logarithm of g_1^m for the decryption procedure.

Formally, we let $\Lambda = (\mathsf{HKG}, \mathsf{Hash})$ be a hash function. Let $\mathbb{G}$ be a multiplicative cyclic group of prime order p, where $p = \Omega(2^\lambda)$. We naturally encode an element in $\{0,1\}^\lambda$ as one in $\mathbb{Z}_p$. Then, we construct our RNCE scheme $\Pi_{\mathsf{ddh}} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{FKG}, \mathsf{Fake}, \mathsf{Open}, \mathsf{FDec})$ as described in Fig. 2. We note that the correctness of the decryption of Π_{ddh} is straightforward due to the correctness of the original Cramer-Shoup encryption scheme.

Correctness of decryption using a fake secret key. Similarly to Section 4.1, we confirm that a fake secret key can decrypt a fake ciphertext and a real ciphertext by the decryption algorithm Dec.

Let (pk, td) be a public key/trapdoor pair generated by $\mathsf{FKG}(1^\lambda)$, $m \in \mathbb{Z}_p$ be an arbitrary plaintext, $\widetilde{c} = (\widetilde{u_1}, \widetilde{u_2}, \widetilde{e}, \widetilde{v})$ be a fake ciphertext generated by $\mathsf{Fake}(pk, td)$, and $\widetilde{sk} = (x_1', x_2', y_1, y_2, z_1, z_2)$ be a fake secret key generated by $\mathsf{Open}(pk, td, \widetilde{c}, m)$. First, we confirm that a fake secret key $\widetilde{sk}$ can decrypt a fake ciphertext $\widetilde{c} = (\widetilde{u_1}, \widetilde{u_2}, \widetilde{e}, \widetilde{v})$ to the plaintext m as follows. We can check that the fake ciphertext $\widetilde{c}$ satisfies the following validity condition (checked in Dec)

$$\widetilde{u_1}^{y_1+z_1\widetilde{\mu}} \widetilde{u_2}^{y_2+z_2\widetilde{\mu}} = {g_1}^{\widetilde{r}(y_1+z_1\widetilde{\mu})} (g_1 {g_2}^{\widetilde{r}})^{y_2+z_2\widetilde{\mu}} = {g_1}^{y_2+z_2\widetilde{\mu}} ({g_1}^{y_1} {g_2}^{y_2})^{\widetilde{r}} ({g_1}^{z_1} {g_2}^{z_2})^{\widetilde{r}\widetilde{\mu}} = {g_1}^{y_2+z_2\widetilde{\mu}} s^{\widetilde{r}} t^{\widetilde{r}\widetilde{\mu}} = \widetilde{v},$$

where $\widetilde{\mu} = \mathsf{Hash}(hk, \widetilde{u_1} \| \widetilde{u_2} \| \widetilde{e})$. Next, we can check that the fake ciphertext $\widetilde{c}$ is decrypted to m using the fake secret key $\widetilde{sk}$ as

$$\frac{\widetilde{e}}{\widetilde{u_1}^{x_1'} \widetilde{u_2}^{x_2'}} = \frac{g_1^{x_2} k^{\widetilde{r}}}{g_1^{\widetilde{r}(x_1+mw)} g_1^{x_2-m} g_2^{\widetilde{r}(x_2-m)}} = \frac{g_1^{\widetilde{r}x_1+x_2} g_2^{\widetilde{r}x_2}}{g_1^{\widetilde{r}x_1+\widetilde{r}mw+x_2-m} g_2^{\widetilde{r}x_2-\widetilde{r}m}} = g_1^m.$$

Furthermore, let $c = (u_1, u_2, e, v)$ be a real ciphertext output by $\mathsf{Enc}(pk, m')$ for any plaintext $m' \in \mathbb{Z}_p$ (which is not necessarily m). Then, the fake secret key $\widetilde{sk}$ can decrypt the real ciphertext c to m' as

$$\frac{e}{u_1{}^{x'_1} u_2{}^{x'_2}} = \frac{k^r \cdot g_1^{m'}}{g_1^{r(x_1+mw)} g_2^{r(x_2-m)}} = \frac{g_1^{rx_1} g_2^{rx_2} \cdot g_1^{m'}}{g_1^{rx_1+rmw} g_2^{rx_2-rm}} = g_1^{m'}.$$

5.2 Security Proof for the Basic Construction

In this section, we show the following theorem.

Theorem 3. *If the "+1"-DDH assumption holds in* $\mathbb{G}$*, and* $\Lambda = (\mathsf{HKG}, \mathsf{Hash})$ *is a collision-resistant hash function, then* Π_{ddh} *is RNC-CCA secure.*

Since the "+1"-DDH assumption is implied by the ordinary DDH assumption, Theorem 3 implies that our construction Π_{ddh} is indeed RNC-CCA secure under the ordinary DDH assumption.

Before describing the formal proof, we highlight the flow of the proof. We change $\mathsf{Exp}^{\mathsf{rnc\text{-}real}}_{\Pi_{\mathsf{ddh}},\mathcal{A}}(\lambda)$ to $\mathsf{Exp}^{\mathsf{rnc\text{-}sim}}_{\Pi_{\mathsf{ddh}},\mathcal{A}}(\lambda)$ step by step, where $\mathcal{A}$ is an adversary that attacks the RNC-CCA security of Π_{ddh}. Although the main part of our proof is similar to that of the Cramer-Shoup encryption scheme [7, 8], we have the following two remarkable changes in order to change the challenge ciphertext c^* (resp., the real secret key sk) to a fake ciphertext $\widetilde{c}$ (resp., the fake secret key $\widetilde{sk}$).

First, toward transforming the challenge ciphertext to a fake ciphertext, we change the challenge ciphertext component $u_2^* = g_2^{r^*}$ to $u_2^* = g_1 g_2^{r^*}$. Second, we change the real secret key component (x_1, x_2) to the fake secret key component (x'_1, x'_2) computed by Open described in Fig. 2. Due to these changes, the challenge ciphertext component e^* is changed to the fake ciphertext component $\widetilde{e}$ and the real secret key sk is changed to the fake secret key $\widetilde{sk}$. The proof is as follows.

Proof of Theorem 3. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ be any PPT adversary that attacks the RNC-CCA security of Π_{ddh} and makes $Q_{dec} > 0$ decryption queries. We introduce the following experiments $\{\mathsf{Exp}_i\}_{i=0}^{7}$.

Exp_0 : Exp_0 is exactly the same as $\mathsf{Exp}^{\mathsf{rnc\text{-}real}}_{\Pi_{\mathsf{ddh}},\mathcal{A}}(\lambda)$. The detailed description is as follows.

1. First, Exp_0 samples $g_1 \leftarrow \mathbb{G}$ and $w \leftarrow \mathbb{Z}_p^*$ and sets $g_2 := g_1^w$. Next, Exp_0 samples $x_1, x_2, y_1, y_2, z_1, z_2 \leftarrow \mathbb{Z}_p$ and sets $k := g_1^{x_1} g_2^{x_2}$, $s := g_1^{y_1} g_2^{y_2}$, and $t := g_1^{z_1} g_2^{z_2}$. Then, it samples $hk \leftarrow \mathsf{HKG}(1^\lambda)$, sets $pk := (g_1, g_2, k, s, t, hk)$ and $sk := (x_1, x_2, y_1, y_2, z_1, z_2)$, and runs $\mathcal{A}_1(pk)$. When $\mathcal{A}_1$ makes a decryption query $c = (u_1, u_2, e, v)$, Exp_0 computes $\mu \leftarrow \mathsf{Hash}(hk, u_1 \| u_2 \| e)$ and checks whether $u_1^{y_1+z_1\mu} u_2^{y_2+z_2\mu} = v$ holds. If this holds, Exp_0 returns $m = \log_{g_1}(e \cdot (u_1^{x_1} u_2^{x_2})^{-1})$ to $\mathcal{A}_1$. Otherwise, Exp_0 returns $\perp$ to $\mathcal{A}_1$.
2. When $\mathcal{A}_1$ outputs (m^*, st_1) and terminates, Exp_0 computes the challenge ciphertext c^* as follows. First, Exp_0 samples $r^* \leftarrow \mathbb{Z}_p$ and sets $u_1^* := g_1^{r^*}$, $u_2^* := g_2^{r^*}$, and $e^* := k^{r^*} \cdot g_1^{m^*}$. Next, Exp_0 computes $\mu^* \leftarrow \mathsf{Hash}(hk, u_1^* \| u_2^* \| e^*)$, sets $v^* := s^{r^*} t^{r^* \mu^*}$ and $c^* := (u_1^*, u_2^*, e^*, v^*)$, and runs $\mathcal{A}_2(c^*, \mathsf{st}_1)$. When $\mathcal{A}_2$ makes a decryption query $c = (u_1, u_2, e, v)$, Exp_0 answers the query in the same way as above.
3. When $\mathcal{A}_2$ outputs state information st_2 and terminates, Exp_0 runs $\mathcal{A}_3(sk, \mathsf{st}_2)$. When $\mathcal{A}_3$ outputs b' and terminates, Exp_0 outputs b'.

Exp_1 : Exp_1 is identical to Exp_0 except for the following change. When computing the challenge ciphertext $c^* = (u_1^*, u_2^*, e^*, v^*)$, Exp_1 computes e^* and v^* by $e^* := (u_1^*)^{x_1} (u_2^*)^{x_2} \cdot g_1^{m^*}$ and $v^* := (u_1^*)^{y_1} (u_2^*)^{y_2} ((u_1^*)^{z_1} (u_2^*)^{z_2})^{\mu^*}$, respectively.

Exp_2 : Exp_2 is identical to Exp_1 except that when computing the challenge ciphertext $c^* = (u_1^*, u_2^*, e^*, v^*)$, Exp_2 computes u_2^* by $u_2^* := g_1^{wr^*+1}$.

Exp_3 : Exp_3 is identical to Exp_2 except that when responding to a decryption query $c = (u_1, u_2, e, v)$ made by $\mathcal{A}_2$, Exp_3 answers $\perp$ if $\mathsf{Hash}(hk, u_1^* \| u_2^* \| e^*) = \mathsf{Hash}(hk, u_1 \| u_2 \| e)$ holds.

Exp_4 : Exp_4 is identical to Exp_3 except that when responding to a decryption query $c = (u_1, u_2, e, v)$ made by $\mathcal{A}_1$ or $\mathcal{A}_2$, Exp_4 answers $\perp$ if $u_1^w \neq u_2$ holds.

Exp_5 : Exp_5 is identical to Exp_4 except that $x_1' := x_1 + wm^*$ (mod p) and $x_2' := x_2 - m^*$ (mod p) are used instead of x_1 and x_2, respectively. That is, when computing the challenge ciphertext $c^* := (u_1^*, u_2^*, e^*, v^*)$, Exp_5 computes e^* by $e^* := (u_1^*)^{x_1'}(u_2^*)^{x_2'} \cdot g_1^{m^*}$ instead of $(u_1^*)^{x_1}(u_2^*)^{x_2} \cdot g_1^{m^*}$. Note that $e^* = (u_1^*)^{x_1'}(u_2^*)^{x_2'} \cdot g_1^{m^*} = g_1^{r^*(x_1+wm^*)} g_1^{(wr^*+1)(x_2-m^*)} \cdot g_1^{m^*} = g_1^{x_2}(g_1^{x_1}g_2^{x_2})^{r^*}$ holds, and thus e^* is independent of m^*. Furthermore, Exp_5 gives the secret key $sk' := (x_1', x_2', y_1, y_2, z_1, z_2)$ to $\mathcal{A}_3$ instead of $sk := (x_1, x_2, y_1, y_2, z_1, z_2)$.

Since u_1^*, u_2^*, and e^* in Exp_5 are independent of the challenge plaintext m^*, without loss of generality we generate them before $\mathcal{A}_1$ is run.

Exp_6 : Exp_6 is identical to Exp_5 except that when responding to a decryption query $c = (u_1, u_2, e, v)$ made by $\mathcal{A}_1$, Exp_6 answers $\perp$ if $\mathsf{Hash}(hk, u_1^* \| u_2^* \| e^*) = \mathsf{Hash}(hk, u_1 \| u_2 \| e)$ holds. Note that the procedure of the decryption oracle in Exp_6 is exactly the same as that of $\mathsf{FDec}(pk, td, c)$.

Exp_7 : Exp_7 is exactly the same as $\mathsf{Exp}^{\mathsf{rnc\text{-}sim}}_{\Pi_{\mathsf{ddh}},\mathcal{A}}(\lambda)$.

We let $p_i := \Pr[\mathsf{Exp}_i(\lambda) = 1]$ for all $i \in [0, 7]$. Then, we have

$$\mathsf{Adv}^{\mathsf{rnc\text{-}cca}}_{\Pi_{\mathsf{ddh}},\mathcal{A}}(\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{rnc\text{-}real}}_{\Pi_{\mathsf{ddh}},\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{rnc\text{-}sim}}_{\Pi_{\mathsf{ddh}},\mathcal{A}}(\lambda) = 1]| = |p_0 - p_7| \leq \sum_{i=0}^{6} |p_i - p_{i+1}|.$$

It remains to show how each $|p_i - p_{i+1}|$ is upper-bounded. In the following, we will show that $|p_0 - p_1| = 0$ holds (Lemma 8) since the difference between Exp_0 and Exp_1 is only conceptual. Then, we will show that there exist a PPT adversary $\mathcal{E}$ against the "+1"-DDH assumption in $\mathbb{G}$ such that $|p_1 - p_2| = \mathsf{Adv}^{+1\text{-}\mathsf{ddh}}_{\mathbb{G},\mathcal{E}}(\lambda)$ (Lemma 9), and a PPT adversary $\mathcal{F}$ against the collision-resistance of Λ such that $|p_2 - p_3| \leq \mathsf{Adv}^{\mathsf{cr}}_{\Lambda,\mathcal{F}}(\lambda)$ (Lemma 10). Next, we will show that $|p_3 - p_4| \leq \frac{Q_{dec}}{p}$ holds (Lemma 11) by showing that the probability that each of $\mathcal{A}$'s valid queries is rejected in Exp_5 but not in Exp_4, is at most $\frac{1}{p}$. Then, we will show that $|p_4 - p_5| = 0$ holds (Lemma 12) since (x_1, x_2) and (x_1', x_2') are information-theoretically indistinguishable from $\mathcal{A}$. Next, we will show that there exists a PPT adversary $\mathcal{G}$ against the collision-resistance of Λ such that $|p_5 - p_6| \leq \mathsf{Adv}^{\mathsf{cr}}_{\Lambda,\mathcal{G}}(\lambda) + \frac{Q_{dec}}{p}$ (Lemma 13). Finally, we will show that $|p_6 - p_7| = 0$ holds (Lemma 14) by showing that Exp_6 and Exp_7 are identical.

Lemma 8. *$|p_0 - p_1| = 0$ holds.*

Proof of Lemma 8. In Exp_1, we change $e^* = k^{r^*} \cdot g_1^{m^*}$ to $e^* := (u_1^*)^{x_1}(u_2^*)^{x_2} \cdot g_1^{m^*}$, and $v^* = s^{r^*}t^{r^*\mu^*}$ to $v^* := (u_1^*)^{y_1}(u_2^*)^{y_2}((u_1^*)^{z_1}(u_2^*)^{z_2})^{\mu^*}$ in the challenge ciphertext c^*. This is only a conceptual change, and thus Exp_0 is exactly the same as Exp_1 from the viewpoint of the adversary $\mathcal{A}$. Hence, we have $|p_0 - p_1| = 0$. □ (**Lemma 8**)

Lemma 9. *There exists a PPT adversary $\mathcal{E}$ such that $|p_1 - p_2| = \mathsf{Adv}^{+1\text{-}\mathsf{ddh}}_{\mathbb{G},\mathcal{E}}(\lambda)$.*

Proof of Lemma 9. We construct a PPT adversary $\mathcal{E}$ that attacks the "+1"-DDH assumption in $\mathbb{G}$ so that $|p_1 - p_2| = \mathsf{Adv}^{+1\text{-}\mathsf{ddh}}_{\mathbb{G},\mathcal{E}}(\lambda)$, using the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ as follows.

$\mathcal{E}(g_1, g_2, g_3, g_4)$ **:** First, $\mathcal{E}$ samples $x_1, x_2, y_1, y_2, z_1, z_2 \leftarrow \mathbb{Z}_p$ and sets $k := g_1^{x_1}g_2^{x_2}$, $s := g_1^{y_1}g_2^{y_2}$, and $t := g_1^{z_1}g_2^{z_2}$. Next, it samples $hk \leftarrow \mathsf{HKG}(1^\lambda)$, sets $pk := (g_1, g_2, k, s, t, hk)$ and $sk := (x_1, x_2, y_1, y_2, z_1, z_2)$, and runs $\mathcal{A}_1(pk)$. When $\mathcal{A}_1$ makes a decryption query $c = (u_1, u_2, e, v)$, $\mathcal{E}$ returns $\mathsf{Dec}(pk, sk, c)$ to $\mathcal{A}_1$. When $\mathcal{A}_1$ outputs the tuple of the challenge plaintext and state information (m^*, st_1) and terminates, $\mathcal{E}$ sets $u_1^* := g_3$, $u_2^* := g_4$, and $e^* := ((u_1^*)^{x_1}(u_2^*)^{x_2}) \cdot g_1^{m^*}$. Then, $\mathcal{E}$ computes $\mu^* \leftarrow \mathsf{Hash}(hk, u_1^* \| u_2^* \| e^*)$, sets $v^* := (u_1^*)^{y_1}(u_2^*)^{y_2}((u_1^*)^{z_1}(u_2^*)^{z_2})^{\mu^*}$ and $c^* := (u_1^*, u_2^*, e^*, v^*)$, and runs $\mathcal{A}_2(c^*, \mathsf{st}_1)$. When $\mathcal{A}_2$ makes a decryption query c, $\mathcal{E}$ returns $\mathsf{Dec}(pk, sk, c)$ to $\mathcal{A}_2$. When $\mathcal{A}_2$ outputs state information st_2 and terminates, $\mathcal{E}$ runs $\mathcal{A}_3(sk, \mathsf{st}_2)$. When $\mathcal{A}_3$ outputs a bit b' and terminates, $\mathcal{E}$ outputs b' to its experiment.

In the following, we call (g_1, g_2, g_3, g_4) a DH tuple if $(g_1, g_2, g_3, g_4) = (g, g^a, g^b, g^{ab})$ and a DH+1 tuple if $(g_1, g_2, g_3, g_4) = (g, g^a, g^b, g^{ab+1})$. Furthermore, we will use the following notation

$$p_{\mathsf{dh}} := \Pr[\mathcal{E}(g_1, g_2, g_3, g_4) = 1 | g \leftarrow \mathbb{G}; a \leftarrow \mathbb{Z}_p^*; b \leftarrow \mathbb{Z}_p; (g_1, g_2, g_3, g_4) := (g, g^a, g^b, g^{ab})],$$

$$p_{\mathsf{dh}+1} := \Pr[\mathcal{E}(g_1, g_2, g_3, g_4) = 1 | g \leftarrow \mathbb{G}; a \leftarrow \mathbb{Z}_p^*; b \leftarrow \mathbb{Z}_p; (g_1, g_2, g_3, g_4) := (g, g^a, g^b, g^{ab+1})].$$

Here, a and b in the "+1"-DDH assumption correspond to w and r^*, respectively. Thus, $g_2 = g_1^w$, $g_3 = g_1^{r^*}$, and $g_4 = g_1^{wr^*}$ hold if $\mathcal{E}$ receives a DH tuple. Therefore, we can see that $\mathcal{E}$ perfectly simulates Exp_1 for $\mathcal{A}$ if it receives a DH tuple. This ensures that the probability that $\mathcal{E}$ outputs 1 given a DH tuple is exactly the same as the probability that Exp_1 outputs 1. That is, we have $p_{\mathsf{dh}} = p_1$. On the other hand, $g_2 = g_1^w$, $g_3 = g_1^{r^*}$, and $g_4 = g_1^{wr^*+1} = g_1 g_2^{r^*}$ hold if $\mathcal{E}$ receives a DH+1 tuple. Therefore, $\mathcal{E}$ perfectly simulates Exp_2 for $\mathcal{A}$ if it receives a DH+1 tuple. This ensures that the probability that $\mathcal{E}$ outputs 1 given a DH+1 tuple is exactly the same as the probability that Exp_2 outputs 1. That is, we have $p_{\mathsf{dh}+1} = p_2$. This implies that $\mathsf{Adv}^{+1\text{-}\mathsf{ddh}}_{\mathbb{G},\mathcal{E}}(\lambda) = |p_{\mathsf{dh}} - p_{\mathsf{dh}+1}| = |p_1 - p_2|$. □ (**Lemma 9**)

Lemma 10. *There exists a PPT adversary $\mathcal{F}$ such that $|p_2 - p_3| \leq \mathsf{Adv}^{\mathsf{cr}}_{\Lambda,\mathcal{F}}(\lambda)$.*

Proof of Lemma 10. For $i \in \{2, 3\}$, we let $\mathbf{Bad}_i^{\mathsf{cr}}$ be the event that $\mathcal{A}_2$ makes a decryption query $c = (u_1, u_2, e, v)$ satisfying $(\mathsf{Hash}(hk, u_1^* \| u_2^* \| e^*) = \mathsf{Hash}(hk, u_1 \| u_2 \| e)) \wedge (u_1^{y_1+z_1\mu} u_2^{y_2+z_2\mu} = v)$ in Exp_i. (We call such a decryption query a *bad decryption query*.) Exp_2 proceeds identically to Exp_3 unless $\mathbf{Bad}_2^{\mathsf{cr}}$ happens. Therefore, the inequality $|p_2 - p_3| \leq \Pr[\mathbf{Bad}_2^{\mathsf{cr}}] = \Pr[\mathbf{Bad}_3^{\mathsf{cr}}]$ holds. In the following, we show that one can construct a PPT adversary $\mathcal{F}$ that attacks the collision-resistance of Λ so that $\Pr[\mathbf{Bad}_2^{\mathsf{cr}}] = \mathsf{Adv}^{\mathsf{cr}}_{\Lambda,\mathcal{F}}(\lambda)$, using the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$.

$\mathcal{F}(hk)$ **:** First, $\mathcal{F}$ samples $g_1 \leftarrow \mathbb{G}$, and $w \leftarrow \mathbb{Z}_p^*$ and sets $g_2 := g_1^w$. Next, it samples $x_1, x_2, y_1, y_2, z_1, z_2 \leftarrow \mathbb{Z}_p$ and sets $k := g_1^{x_1} g_2^{x_2}$, $s := g_1^{y_1} g_2^{y_2}$, and $t := g_1^{z_1} g_2^{z_2}$. Then, it sets $pk := (g_1, g_2, k, s, t, hk)$ and $sk := (x_1, x_2, y_1, y_2, z_1, z_2)$, and runs $\mathcal{A}_1(pk)$. When $\mathcal{A}_1$ makes a decryption query c, $\mathcal{F}$ returns $\mathsf{Dec}(pk, sk, c)$ to $\mathcal{A}_1$. When $\mathcal{A}_1$ outputs the challenge plaintext and state information (m^*, st_1) and terminates, $\mathcal{F}$ samples $r^* \leftarrow \mathbb{Z}_p$ and sets $u_1^* := g_1^{r^*}$, $u_2^* := g_1^{wr^*+1}$, and $e^* := (u_1^*)^{x_1} (u_2^*)^{x_2} \cdot g_1^{m^*}$. Next, it computes $\mu^* \leftarrow \mathsf{Hash}(hk, u_1^* \| u_2^* \| e^*)$, sets $v^* := (u_1^*)^{y_1} (u_2^*)^{y_2} ((u_1^*)^{z_1} (u_2^*)^{z_2})^{\mu^*}$ and $c^* := (u_1^*, u_2^*, e^*, v^*)$, and runs $\mathcal{A}_2(c^*, \mathsf{st}_1)$. When $\mathcal{A}_2$ makes a decryption query $c = (u_1, u_2, e, v)$, $\mathcal{F}$ sets $\mu := \mathsf{Hash}(hk, u_1 \| u_2 \| e)$ and checks whether $(\mu = \mu^*) \wedge (u_1^{y_1+z_1\mu} u_2^{y_2+z_2\mu} = v)$ holds. If this holds, $\mathcal{F}$ returns $(u_1 \| u_2 \| e, u_1^* \| u_2^* \| e^*)$ to its experiment and terminates. Otherwise, $\mathcal{F}$ returns $\mathsf{Dec}(pk, sk, c)$ to $\mathcal{A}_2$. When $\mathcal{A}_2$ outputs state information st_2 and terminates, $\mathcal{F}$ gives up and terminates.

From the above construction of $\mathcal{F}$, it is easy to see that $\mathcal{F}$ perfectly simulates Exp_2 for $\mathcal{A}$. Here, the success condition of $\mathcal{F}$ is to output a pair $(u_1 \| u_2 \| e, u_1^* \| u_2^* \| e^*)$ satisfying $(\mathsf{Hash}(hk, u_1 \| u_2 \| e) = \mathsf{Hash}(hk, u_1^* \| u_2^* \| e^*)) \wedge (u_1 \| u_2 \| e \neq u_1^* \| u_2^* \| e^*)$. If $\mathcal{A}_2$ makes a bad decryption query $c = (u_1, u_2, e, v)$, $\mathsf{Hash}(hk, u_1 \| u_2 \| e) = \mathsf{Hash}(hk, u_1^* \| u_2^* \| e^*)$ and $u_1^{y_1+z_1\mu} u_2^{y_2+z_2\mu} = v$ hold. In the following, we see that $u_1 \| u_2 \| e \neq u_1^* \| u_2^* \| e^*$ holds if $u_1^{y_1+z_1\mu} u_2^{y_2+z_2\mu} = v$ holds. Assume towards a contradiction, $u_1 \| u_2 \| e = u_1^* \| u_2^* \| e^*$ holds. Then, $v \neq v^*$ holds because the decryption query c must satisfy $c \neq c^*$. Also, $u_1^{y_1+z_1\mu} u_2^{y_2+z_2\mu} = (u_1^*)^{y_1+z_1\mu^*} (u_2^*)^{y_2+z_2\mu^*}$ holds due to $u_1 \| u_2 \| e = u_1^* \| u_2^* \| e^*$. Here, since $v^* = (u_1^*)^{y_1+z_1\mu^*} (u_2^*)^{y_2+z_2\mu^*}$ holds, $v \neq u_1^{y_1+z_1\mu} u_2^{y_2+z_2\mu}$ must also hold. However this contradicts to $u_1^{y_1+z_1\mu} u_2^{y_2+z_2\mu} = v$, and thus $u_1 \| u_2 \| e \neq u_1^* \| u_2^* \| e^*$ holds. Hence, when $\mathcal{A}_2$ makes a bad decryption query (u_1, u_2, e, v), $\mathcal{F}$ achieves its success condition by returning $(u_1 \| u_2 \| e, u_1^* \| u_2^* \| e^*)$ to its experiment. From the above arguments, the probability that $\mathcal{F}$ breaks the collision-resistance of Λ is exactly the same as the probability that $\mathcal{A}_2$ makes a bad decryption query in Exp_2. Therefore, we have $\Pr[\mathbf{Bad}_2^{\mathsf{cr}}] = \mathsf{Adv}^{\mathsf{cr}}_{\Lambda,\mathcal{F}}(\lambda)$. Thus, $|p_2 - p_3| \leq \mathsf{Adv}^{\mathsf{cr}}_{\Lambda,\mathcal{F}}(\lambda)$ holds. □ (**Lemma 10**)

Lemma 11. $|p_3 - p_4| \leq \frac{Q_{dec}}{p}$ *holds.*

Proof of Lemma 11. For $j \in \{3, 4\}$, we let $\mathbf{Bad}_j^i$ be the event that $\mathcal{A}$'s ith decryption query $c = (u_1, u_2, e, v)$ satisfies $(u_1^w \neq u_2) \wedge (u_1^{y_1+z_1\mu} u_2^{y_2+z_2\mu} = v) \wedge (\mu \neq \mu^*)$ in Exp_j. (We call such a decryption query a *bad decryption query*.) Furthermore, we let $\mathbf{Bad}_j$ be the event that $\mathcal{A}$ makes at least one decryption query $c = (u_1, u_2, e, v)$ satisfying $(u_1^w \neq u_2) \wedge (u_1^{y_1+z_1\mu} u_2^{y_2+z_2\mu} = v) \wedge (\mu \neq \mu^*)$ in Exp_j. Clearly, $\Pr[\mathbf{Bad}_j] \leq \sum_{i \in [Q_{dec}]} \Pr[\mathbf{Bad}_j^i]$ holds for $j \in \{3, 4\}$. Exp_3 proceeds identically to Exp_4

unless the event $\mathbf{Bad}_3$ happens. Therefore, the inequality $|p_3 - p_4| \le \Pr[\mathbf{Bad}_3] = \Pr[\mathbf{Bad}_4]$ holds. In the following, we show that $\Pr[\mathbf{Bad}_4] \le \frac{Q_{dec}}{p}$ holds. To show it, we introduce $Q_{dec,1} > 0$ as the number of decryption queries made by $\mathcal{A}_1$.

To this end, we firstly estimate the probability that $\mathbf{Bad}_4^i$ (for $i \in [Q_{dec,1}]$) happens based on the information that $\mathcal{A}_1$ gets about y_1, y_2, z_1, z_2. Before $\mathcal{A}_1$ makes a decryption query, the information that $\mathcal{A}_1$ gets about y_1, y_2, z_1, z_2 consists of $s = g_1^{y_1} g_2^{y_2}$ and $t = g_1^{z_1} g_2^{z_2}$ in the public key pk. When $\mathcal{A}_1$ makes a decryption query $c = (u_1, u_2, e, v)$ satisfying $u_1^w \ne u_2$, we let $v' := u_1^{y_1 + z_1\mu} u_2^{y_2 + z_2\mu} = (g_1^{r_1 y_1} g_1^{wr_2 y_2})(g_1^{r_1 z_1} g_1^{wr_2 z_2})^{\mu}$, where $r_1 = \log_{g_1} u_1 \in \mathbb{Z}_p$, $r_2 = \log_{g_2} u_2 \in \mathbb{Z}_p \backslash \{r_1\}$, and $\mu = \mathsf{Hash}(hk, u_1 \| u_2 \| e)$, and consider the discrete logarithms with the base g_1 for s, t, and v' as

$$\begin{pmatrix} \log_{g_1} s \\ \log_{g_1} t \\ \log_{g_1} v' \end{pmatrix} = M_1 \begin{pmatrix} y_1 \\ y_2 \\ z_1 \\ z_2 \end{pmatrix}, \text{where } M_1 = \begin{pmatrix} 1 & w & 0 & 0 \\ 0 & 0 & 1 & w \\ r_1 & wr_2 & r_1\mu & wr_2\mu \end{pmatrix}.$$

Since $r_1 \ne r_2$ and $w \ne 0$ hold, the rank of M_1 is 3. Thus, the above equations are linearly independent and there are exactly p equally-likely possibilities for (y_1, y_2, z_1, z_2), which in turn implies that the probability that the decryption query c satisfies the condition $v' = u_1^{y_1+z_1\mu} u_2^{y_2+z_2\mu} = v$ is exactly $\frac{1}{p}$. Here, if $u_1^w \ne u_2$ holds, $\mathcal{A}_1$ always gets $\perp$ regardless of the condition $u_1^{y_1+z_1\mu} u_2^{y_2+z_2\mu} = v$ in Exp_4.Therefore, the probability that $\mathcal{A}_1$'s ith query is a bad decryption query is exactly $\frac{1}{p}$ for all $i \in [Q_{dec,1}]$, that is, $\Pr[\mathbf{Bad}_4^i] = \frac{1}{p}$ holds for all $i \in [Q_{dec,1}]$.

Second, we estimate the probability that $\mathbf{Bad}_4^i$ (for $i \in [Q_{dec,1} + 1, Q_{dec}]$) happens based on the information that $\mathcal{A}_2$ gets about y_1, y_2, z_1, z_2. Before $\mathcal{A}_2$ makes a decryption query, the information that $\mathcal{A}_2$ gets about y_1, y_2, z_1, z_2 consists of $s = g_1^{y_1} g_2^{y_2}$ and $t = g_1^{z_1} g_2^{z_2}$ in the public key pk and $v^* = (u_1^*)^{y_1}(u_2^*)^{y_2}((u_1^*)^{z_1}(u_2^*)^{z_2})^{\mu^*} = (g_1^{r^* y_1} g_1^{(1+wr^*)y_2})(g_1^{r^* z_1} g_1^{(1+wr^*)z_2})^{\mu^*}$ in the challenge ciphertext c^*. When $\mathcal{A}_2$ makes a decryption query $c = (u_1, u_2, e, v)$ satisfying $u_1^w \ne u_2$, we let $v' := u_1^{y_1+z_1\mu} u_2^{y_2+z_2\mu} = (g_1^{r_1 y_1} g_1^{wr_2 y_2})(g_1^{r_1 z_1} g_1^{wr_2 z_2})^{\mu}$, where $r_1 = \log_{g_1} u_1 \in \mathbb{Z}_p$, $r_2 = \log_{g_2} u_2 \in \mathbb{Z}_p \backslash \{r_1\}$, and $\mu = \mathsf{Hash}(hk, u_1 \| u_2 \| e)$, and consider the discrete logarithms with the base g_1 for s, t, v^*, and v' as

$$\begin{pmatrix} \log_{g_1} s \\ \log_{g_1} t \\ \log_{g_1} v^* \\ \log_{g_1} v' \end{pmatrix} = M_2 \begin{pmatrix} y_1 \\ y_2 \\ z_1 \\ z_2 \end{pmatrix}, \text{where } M_2 = \begin{pmatrix} 1 & w & 0 & 0 \\ 0 & 0 & 1 & w \\ r^* & wr^*+1 & r^*\mu^* & \mu^*(wr^*+1) \\ r_1 & wr_2 & r_1\mu & wr_2\mu \end{pmatrix}.$$

Since $r_1 \ne r_2$, $\mu^* \ne \mu$, and $w \ne 0$ hold, we have $\det(M_2) = w(r_1 - r_2)(\mu^* - \mu) \ne 0$. Thus, the above equations are linearly independent and there are exactly p equally-likely possibilities for (y_1, y_2, z_1, z_2), which in turn implies that the probability that the decryption query c satisfies the condition $v' = u_1^{y_1+z_1\mu} u_2^{y_2+z_2\mu} = v$ is exactly $\frac{1}{p}$. Here, if $u_1^w \ne u_2$ holds, $\mathcal{A}_2$ always gets $\perp$ regardless of the condition $v' = u_1^{y_1+z_1\mu} u_2^{y_2+z_2\mu} = v$ in Exp_4. Therefore, the probability that $\mathcal{A}_2$'s ith query is a bad decryption query is exactly $\frac{1}{p}$ for all $i \in [Q_{dec,1} + 1, Q_{dec}]$, that is, $\Pr[\mathbf{Bad}_4^i] = \frac{1}{p}$ holds for all $i \in [Q_{dec,1} + 1, Q_{dec}]$.

From the above arguments, we have $|p_3 - p_4| \le \Pr[\mathbf{Bad}_4] \le \frac{Q_{dec}}{p}$. □ (**Lemma 11**)

Lemma 12. $|p_4 - p_5| = 0$ *holds.*

Proof of Lemma 12. In the experiment Exp_5, we use $x_1' := x_1 + wm^* \pmod p$ and $x_2' := x_2 - m^* \pmod p$ instead of x_1 and x_2, respectively.

Here, x_1 and x_2 are chosen uniformly at random from $\mathbb{Z}_p$ and the only information that $\mathcal{A}$ gets about (x_1, x_2), before $\mathcal{A}$ gets the challenge ciphertext, consists of $k = g_1^{x_1} g_2^{x_2}$. Thus, there are p equally-likely possibilities for (x_1, x_2). Therefore, $x_1' := x_1 + wm^* \pmod p$ and $x_2' := x_2 - m^* \pmod p$ are also uniformly distributed in $\mathbb{Z}_p$ conditioned on that $\log_{g_1} k = x_1 + wx_2$ holds. Hence, the view of $\mathcal{A}$ in Exp_4 is exactly the same as that in Exp_5, which in turn implies that $|p_4 - p_5| = 0$ holds. □ (**Lemma 12**)

Lemma 13. *There exists a PPT adversary* $\mathcal{G}$ *such that* $|p_5 - p_6| \le \mathsf{Adv}^{\mathsf{cr}}_{\Lambda,\mathcal{G}}(\lambda) + \frac{Q_{dec}}{p}$.

Proof of Lemma 13. For $i \in \{5,6\}$, we let $\mathbf{Bad}_i^{\mathsf{cr}}$ be the event that $\mathcal{A}_1$ makes a decryption query $c = (u_1, u_2, e, v)$ satisfying $(\mathsf{Hash}(hk, u_1^* \| u_2^* \| e^*) = \mathsf{Hash}(hk, u_1 \| u_2 \| e)) \wedge (u_1^{y_1+z_1\mu} u_2^{y_2+z_2\mu} = v)$ in Exp_i. (We call such a decryption query a *bad decryption query*.) Exp_5 proceeds identically to Exp_6 unless $\mathbf{Bad}_5^{\mathsf{cr}}$ happens. Therefore, the inequality $|p_5 - p_6| \le \Pr[\mathbf{Bad}_5^{\mathsf{cr}}] = \Pr[\mathbf{Bad}_6^{\mathsf{cr}}]$ holds. Here, we let $\mathbf{Bad}_5^{c^*}$ be the event that $\mathcal{A}_1$ makes a decryption query $c = c^*$ in Exp_5. Then, we can estimate $\Pr[\mathbf{Bad}_5^{\mathsf{cr}}]$ by $\Pr[\mathbf{Bad}_5^{\mathsf{cr}}] = \Pr[\mathbf{Bad}_5^{\mathsf{cr}} \wedge \mathbf{Bad}_5^{c^*}] + \Pr[\mathbf{Bad}_5^{\mathsf{cr}} \wedge \neg \mathbf{Bad}_5^{c^*}]$.

We can see that the probability $\Pr[\mathbf{Bad}_5^{\mathsf{cr}} \wedge \mathbf{Bad}_5^{c^*}]$ is at most $\frac{Q_{dec}}{p}$. This is because $\mathcal{A}_1$ cannot obtain the information about r^* from its decryption queries.

In the following, similarly to Lemma 10, we show that one can construct a PPT adversary $\mathcal{G}$ that attacks the collision-resistance of Λ so that $\Pr[\mathbf{Bad}_5^{\mathsf{cr}} \wedge \neg \mathbf{Bad}_5^{c^*}] = \mathsf{Adv}_{\Lambda,\mathcal{G}}^{\mathsf{cr}}(\lambda)$, using the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$.

$\mathcal{G}(hk)$ **:** First, $\mathcal{G}$ samples $g_1 \leftarrow \mathbb{G}$ and $w \leftarrow \mathbb{Z}_p^*$ and sets $g_2 := g_1^w$. Next, it samples $x_1, x_2, y_1, y_2, z_1, z_2 \leftarrow \mathbb{Z}_p$ and sets $k := g_1^{x_1} g_2^{x_2}$, $s := g_1^{y_1} g_2^{y_2}$, and $t := g_1^{z_1} g_2^{z_2}$. Then, it sets $pk := (g_1, g_2, k, s, t, hk)$ and $sk := (x_1, x_2, y_1, y_2, z_1, z_2)$. Next, it samples $r^* \leftarrow \mathbb{Z}_p$, sets $u_1^* := g_1^{r^*}$, $u_2^* := g_1^{wr^*+1}$, and $e^* := g_1^{x_2}(g_1^{x_1} g_2^{x_2})^{r^*}$, computes $\mu^* \leftarrow \mathsf{Hash}(hk, u_1^* \| u_2^* \| e^*)$, and runs $\mathcal{A}_1(pk)$. When $\mathcal{A}_1$ makes a decryption query $c = (u_1, u_2, e, v)$, $\mathcal{G}$ sets $\mu := \mathsf{Hash}(hk, u_1 \| u_2 \| e)$ and checks whether $(\mu = \mu^*) \wedge (u_1^{y_1+z_1\mu} u_2^{y_2+z_2\mu} = v) \wedge (c \neq c^*)$ holds. If this holds, $\mathcal{G}$ returns $(u_1 \| u_2 \| e, u_1^* \| u_2^* \| e^*)$ to its experiment and terminates. Otherwise, $\mathcal{G}$ answers the query in the same way as the decryption oracle in Exp_5 does. When $\mathcal{A}_1$ outputs the challenge plaintext and state information (m^*, st_1) and terminates, $\mathcal{G}$ gives up and terminates.

From the above construction of $\mathcal{G}$, it is easy to see that $\mathcal{G}$ perfectly simulates the experiment Exp_5 for $\mathcal{A}_1$. Moreover, with essentially the same argument as in the proof of Lemma 10, the probability that $\mathcal{G}$ succeeds in breaking the collision-resistance of Λ is exactly the same as the probability that $\mathcal{A}_1$ makes a bad decryption query with the condition $(c \neq c^*)$ in Exp_5, that is, $\Pr[\mathbf{Bad}_5^{\mathsf{cr}} \wedge \neg \mathbf{Bad}_5^{c^*}] = \mathsf{Adv}_{\Lambda,\mathcal{G}}^{\mathsf{cr}}(\lambda)$ holds.

From the above arguments, we have $\Pr[\mathbf{Bad}_5^{\mathsf{cr}}] = \Pr[\mathbf{Bad}_5^{\mathsf{cr}} \wedge \mathbf{Bad}_5^{c^*}] + \Pr[\mathbf{Bad}_5^{\mathsf{cr}} \wedge \neg \mathbf{Bad}_5^{c^*}] \le \mathsf{Adv}_{\Lambda,\mathcal{G}}^{\mathsf{cr}}(\lambda) + \frac{Q_{dec}}{p}$. Thus, $|p_5 - p_6| \le \mathsf{Adv}_{\Lambda,\mathcal{G}}^{\mathsf{cr}}(\lambda) + \frac{Q_{dec}}{p}$ holds. □ (**Lemma 13**)

Lemma 14. *$|p_6 - p_7| = 0$ holds.*

Proof of Lemma 14. From the constructions of FKG, Fake, Open, and FDec, the information that the adversary $\mathcal{A}$ gets in Exp_6 is exactly the same as that in Exp_7. Specifically, we can confirm it as follows.

First, it is easy to see that the public key $pk = (g_1, g_2, k, s, t, hk)$ given to $\mathcal{A}_1$ in Exp_6 is exactly the same as one given to $\mathcal{A}_1$ in Exp_7.

Second, we recall that the challenge ciphertext $c^* = (u_1^*, u_2^*, e^*, v^*)$ in Exp_6 is generated as

$$(u_1^*, u_2^*, e^*, v^*) = \left(g_1^{r^*},\ g_1 g_2^{r^*},\ g_1^{x_2}(g_1^{x_1} g_2^{x_2})^{r^*},\ (g_1^{r^* y_1} g_1^{(1+wr^*)y_2})(g_1^{r^* z_1} g_1^{(1+wr^*)z_2})^{\mu^*} \right),$$

where $\mu^* := \mathsf{Hash}(hk, u_1^* \| u_2^* \| e^*)$. We can confirm that $e^* = g_1^{x_2}(g_1^{x_1} g_2^{x_2})^{r^*} = g_1^{x_2} k^{r^*}$ and $v^* = (g_1^{r^* y_1} g_1^{(1+wr^*)y_2})(g_1^{r^* z_1} g_1^{(1+wr^*)z_2})^{\mu^*} = g_1^{y_2+z_2\mu^*} s^{r^*} t^{r^*\mu^*}$. These e^* and v^* are of the same form as those generated by Fake. Thus, the challenge ciphertext (u_1^*, u_2^*, e^*, v^*) is exactly the same as that in Exp_7.

Third, the behavior of the decryption oracle in Exp_6 is exactly the same as that in Exp_7. Concretely, when $\mathcal{A}$ makes a decryption query $c = (u_1, u_2, e, v)$ in Exp_6, Exp_6 computes $\mu = \mathsf{Hash}(hk, u_1 \| u_2 \| e)$ and checks whether $(\mu = \mu^*) \vee (u_1^w \neq u_2) \vee (u_1^{y_1+z_1\mu} u_2^{y_2+z_2\mu} \neq v)$ holds. If this condition holds, Exp_6 returns $\bot$ to $\mathcal{A}$. Otherwise, Exp_6 returns $\log_{g_1}(e \cdot (u_1^{x_1} u_2^{x_2})^{-1})$ to $\mathcal{A}$. This behavior is exactly the same as that of FDec, and thus this decryption oracle's procedure in Exp_6 is exactly the same as that in Exp_7.

Finally, it is easy to see that the secret key $sk' = (x_1', x_2', y_1, y_2, z_1, z_2)$ which $\mathcal{A}_3$ gets in Exp_6 is exactly the same as one which $\mathcal{A}_3$ gets in Exp_7.

From the above arguments, we can conclude $|p_6 - p_7| = 0$. □ (**Lemma 14**)

Putting everything together, we obtain

$$\mathsf{Adv}^{\mathsf{rnc\text{-}cca}}_{\Pi_{\mathsf{ddh}},\mathcal{A}}(\lambda) = |p_0 - p_7| \le \sum_{i=0}^{6} |p_i - p_{i+1}| \le \mathsf{Adv}^{+1\text{-}\mathsf{ddh}}_{\mathbb{G},\mathcal{E}}(\lambda) + \mathsf{Adv}^{\mathsf{cr}}_{\Lambda,\mathcal{F}}(\lambda) + \mathsf{Adv}^{\mathsf{cr}}_{\Lambda,\mathcal{G}}(\lambda) + \frac{2Q_{dec}}{p}.$$

Since the "+1"-DDH assumption holds in $\mathbb{G}$, Λ is a collision-resistant hash function, Q_{dec} is a polynomial of λ, and $p = \Omega(2^\lambda)$, for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{rnc\text{-}cca}}_{\Pi_{\mathsf{ddh}},\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$ holds. Therefore, Π_{ddh} satisfies RNC-CCA security. □ (**Theorem 3**)

5.3 Description of the Main Construction

As mentioned in Section 1.3, the disadvantage of our basic construction is that when we expand the size of its plaintext space to super-polynomial, the ciphertext overhead of the construction increases linearly as the length of a plaintext increases.

More precisely, based on the basic construction, we can expand the plaintext space as follows. Let $d = d(\lambda)$ be a logarithmic function in λ and $\ell = \ell(\lambda)$ be a polynomial in λ. If we want to encrypt a plaintext $m = m_1 \| \cdots \| m_\ell \in \{0,1\}^{d\cdot\ell}$ (that belongs to the super-polynomially large plaintext space), we can encrypt this plaintext m by generating ℓ independent components $(x_{i1}, x_{i2})_{i\in[\ell]}$ then hiding each block m_i with each component (x_{1i}, x_{i2}). Note that we do not need to increase the number of the component (y_1, y_2, z_1, z_2) for checking the validity of a ciphertext. Here, there is a drawback about the size of a ciphertext of the construction obtained in this way. Specifically, in this basic construction, a ciphertext consists of $\ell + 3$ elements $(u_1, u_2, (e_i)_{i\in[\ell]}, v)$ of $\mathbb{G}$. If we let the bit length of one group element be q, the ciphertext overhead is $q(\ell+3) - d\ell = \ell(q-d) + 3q$. Thus, as the length of a plaintext increases, the ciphertext overhead of this construction increases linearly.

Hence, in the following, we propose our main construction with compact ciphertexts supporting the super-polynomially large plaintext space $\{0,1\}^{d\cdot\ell}$ rather than a polynomial-sized one. The main difference from the above basic construction is that when encrypting a plaintext $m = m_1 \| \cdots \| m_\ell \in \{0,1\}^{d\cdot\ell}$, we compute $e_i = \mathsf{H}(k_i^r) \oplus m_i$ instead of $e_i = k_i^r \cdot g_1^{m_i}$ for each $i \in [\ell]$, where H is a universal hash function, k_i are group elements in a public key, $r \leftarrow \mathbb{Z}_p$, and g_1 is a generator of the group $\mathbb{G}$.

Formally, we let $\mathbb{G}$ be a multiplicative cyclic group of prime order p, where $p = \Omega(2^\lambda)$. Let $\mathcal{H} := \{\mathsf{H} : \mathbb{G} \to \{0,1\}^d\}$ be a family of functions and $\Lambda = (\mathsf{HKG}, \mathsf{Hash})$ be a hash function. Let $d = d(\lambda) = O(\log\lambda)$, $\ell = \ell(\lambda)$ be a polynomial in λ, and $Q_h = Q_h(\lambda)$ be a polynomial in λ satisfying $Q_h = \omega(2^d \cdot \log\lambda)$. Then, we construct our RNCE scheme $\Pi'_{\mathsf{ddh}} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{FKG}, \mathsf{Fake}, \mathsf{Open}, \mathsf{FDec})$ with the plaintext space $\mathcal{M} := \{0,1\}^{d\cdot\ell}$ as described in Fig. 3. We note that the correctness of the decryption of Π'_{ddh} is straightforward due to the correctness of the original Cramer-Shoup encryption scheme.

Correctness of decryption using a fake secret key. Similarly to Section 4.1, we confirm that a fake secret key can decrypt a fake ciphertext and a real ciphertext by the decryption algorithm Dec. (In the following, we consider the case that a fake secret key $\widetilde{sk}$ generated by Open is not $\perp$.)

Let (pk, td) be a public key/trapdoor pair generated by $\mathsf{FKG}(1^\lambda)$, $m = m_1 \| \cdots \| m_\ell \in \{0,1\}^{d\cdot\ell}$ be an arbitrary plaintext, $\widetilde{c} = (\widetilde{u_1}, \widetilde{u_2}, (\widetilde{e_i})_{i\in[\ell]}, \widetilde{v})$ be a fake ciphertext generated by $\mathsf{Fake}(pk, td)$, and $\widetilde{sk} = ((x_{ij})_{i\in[\ell], j\in[2]}, y_1, y_2, z_1, z_2)$ be a fake secret key generated by $\mathsf{Open}(pk, td, \widetilde{c}, m)$. First, we confirm that the fake secret key $\widetilde{sk}$ can decrypt the fake ciphertext $\widetilde{c}$ to the plaintext m as follows. We can check the fake ciphertext $\widetilde{c}$ satisfies the following validity condition (checked in Dec)

$$\widetilde{u_1}^{y_1+z_1\widetilde{\mu}}\widetilde{u_2}^{y_2+z_2\widetilde{\mu}} = g_1^{\widetilde{r}(y_1+z_1\widetilde{\mu})}(g_1g_2^{\widetilde{r}})^{y_2+z_2\widetilde{\mu}} = g_1^{y_2+z_2\widetilde{\mu}}(g_1^{y_1}g_2^{y_2})^{\widetilde{r}}(g_1^{z_1}g_2^{z_2})^{\widetilde{r}\widetilde{\mu}} = g_1^{y_2+z_2\widetilde{\mu}}s^{\widetilde{r}}t^{\widetilde{r}\widetilde{\mu}} = \widetilde{v},$$

where $\widetilde{\mu} = \mathsf{Hash}(hk, \widetilde{u_1}\|\widetilde{u_2}\|\widetilde{e})$. Next, we can check that the fake ciphertext $\widetilde{c} = (\widetilde{u_1}, \widetilde{u_2}, (\widetilde{e_i})_{i\in[\ell]}, \widetilde{v})$ is decrypted to m_i using the fake secret key components $(x'_{ij})_{i\in[\ell], j\in[2]}$ as follows. For each $i \in [\ell]$, we have

$$\widetilde{e_i} \oplus \mathsf{H}(\widetilde{u_1}^{x'_{i1}}\widetilde{u_2}^{x'_{i2}}) = \widetilde{e_i} \oplus \mathsf{H}(g_1^{\widetilde{r}(\alpha_i - w\gamma_i + w\widetilde{r}\alpha_i)}(g_1g_2^{\widetilde{r}})^{\gamma_i - \widetilde{r}\alpha_i}) = \widetilde{e_i} \oplus \mathsf{H}(g_1^{\gamma_i}) = \widetilde{e_i} \oplus \widetilde{e_i} \oplus m_i = m_i.$$

```
$\mathsf{KG}(1^\lambda)$ :
  $g_1 \leftarrow \mathbb{G}$
  $w \leftarrow \mathbb{Z}_p^*$
  $g_2 := g_1^w$
  $x_{11}, \cdots, x_{\ell 1}, x_{12}, \cdots, x_{\ell 2} \leftarrow \mathbb{Z}_p$
  $y_1, y_2, z_1, z_2 \leftarrow \mathbb{Z}_p$
  $(k_i)_{i \in [\ell]} := (g_1^{x_{i1}} g_2^{x_{i2}})_{i \in [\ell]}$
  $s := g_1^{y_1} g_2^{y_2}$
  $t := g_1^{z_1} g_2^{z_2}$
  $\mathsf{H} \leftarrow \mathcal{H}$
  $hk \leftarrow \mathsf{HKG}(1^\lambda)$
  $pk := (g_1, g_2, (k_i)_{i \in [\ell]}, s, t, \mathsf{H}, hk)$
  $sk := ((x_{ij})_{i \in [\ell], j \in [2]}, y_1, y_2, z_1, z_2)$
  Return $(pk, sk)$
```

```
$\mathsf{Enc}(pk, m)$ :
  Parse $m = m_1 \| \cdots \| m_\ell \in (\{0,1\}^d)^\ell$
  $r \leftarrow \mathbb{Z}_p$
  $u_1 := g_1^r$
  $u_2 := g_2^r$
  $(e_i)_{i \in [\ell]} := (\mathsf{H}(k_i^r) \oplus m_i)_{i \in [\ell]}$
  $\mu \leftarrow \mathsf{Hash}(hk, u_1 \| u_2 \| (e_i)_{i \in [\ell]})$
  $v := s^r t^{r\mu}$
  Return $c := (u_1, u_2, (e_i)_{i \in [\ell]}, v)$
```

```
$\mathsf{Dec}(pk, sk, c)$ :
  Parse $c = (u_1, u_2, (e_i)_{i \in [\ell]}, v)$
  $\mu \leftarrow \mathsf{Hash}(hk, u_1 \| u_2 \| (e_i)_{i \in [\ell]})$
  If $u_1^{y_1 + z_1 \mu} u_2^{y_2 + z_2 \mu} = v$ then
    $(m_i)_{i \in [\ell]}$
      $:= (e_i \oplus \mathsf{H}(u_1^{x_{i1}} u_2^{x_{i2}}))_{i \in [\ell]}$,
  else Return $\perp$
  Return $m := m_1 \| \cdots \| m_\ell$
```

```
$\mathsf{FKG}(1^\lambda)$ :
  $g_1 \leftarrow \mathbb{G}$
  $w \leftarrow \mathbb{Z}_p^*$
  $g_2 := g_1^w$
  $\alpha_1, \cdots, \alpha_\ell \leftarrow \mathbb{Z}_p$
  $y_1, y_2, z_1, z_2 \leftarrow \mathbb{Z}_p$
  $(k_i)_{i \in [\ell]} := (g_1^{\alpha_i})_{i \in [\ell]}$
  $s := g_1^{y_1} g_2^{y_2}$
  $t := g_1^{z_1} g_2^{z_2}$
  $\mathsf{H} \leftarrow \mathcal{H}$
  $hk \leftarrow \mathsf{HKG}(1^\lambda)$
  $pk := (g_1, g_2, (k_i)_{i \in [\ell]}, s, t, \mathsf{H}, hk)$
  $\widetilde{r} \leftarrow \mathbb{Z}_p$
  $td := (\widetilde{r}, w, (\alpha_i)_{i \in [\ell]}, y_1, y_2, z_1, z_2)$
  Return $(pk, td)$
```

```
$\mathsf{Fake}(pk, td)$ :
  $\widetilde{u_1} := g_1^{\widetilde{r}}$
  $\widetilde{u_2} := g_1 g_2^{\widetilde{r}}$
  $\widetilde{e_1}, \cdots, \widetilde{e_\ell} \leftarrow \{0,1\}^d$
  $\widetilde{\mu} \leftarrow \mathsf{Hash}(hk, \widetilde{u_1} \| \widetilde{u_2} \| (\widetilde{e_i})_{i \in [\ell]})$
  $\widetilde{v} := g_1^{y_2 + z_2 \widetilde{\mu}} s^{\widetilde{r}} t^{\widetilde{r}\widetilde{\mu}}$
  Return $\widetilde{c} := (\widetilde{u_1}, \widetilde{u_2}, (\widetilde{e_i})_{i \in [\ell]}, \widetilde{v})$
```

```
$\mathsf{Open}(pk, td, \widetilde{c}, m)$ :
  Parse $m = m_1 \| \cdots \| m_\ell \in (\{0,1\}^d)^\ell$
  Parse $\widetilde{c} := (\widetilde{u_1}, \widetilde{u_2}, (\widetilde{e_i})_{i \in [\ell]}, \widetilde{v})$
  For $i := 1, \cdots, \ell$
    $(x'_{ij})_{j \in [2]} := \perp$
    For $v := 1, \cdots, Q_h$
      $\gamma_i \leftarrow \mathbb{Z}_p$
      If $\mathsf{H}(g_1^{\gamma_i}) = \widetilde{e_i} \oplus m_i$ then
        $x'_{i1} := \alpha_i - w\gamma_i + w\widetilde{r}\alpha_i \pmod p$
        $x'_{i2} := \gamma_i - \widetilde{r}\alpha_i \pmod p$
        Break
  If $\exists i$ s.t. $x'_{i1} = x'_{i2} = \perp$ then
    $\widetilde{sk} := \perp$
  else $\widetilde{sk} := ((x'_{ij})_{i \in [\ell], j \in [2]}, y_1, y_2, z_1, z_2)$
  Return $\widetilde{sk}$
```

```
$\mathsf{FDec}(pk, td, c)$ :
  Parse $c = (u_1, u_2, (e_i)_{i \in [\ell]}, v)$
  $\mu \leftarrow \mathsf{Hash}(hk, u_1 \| u_2 \| (e_i)_{i \in [\ell]})$
  $\widetilde{u_1} := g_1^{\widetilde{r}}$
  $\widetilde{u_2} := g_1 g_2^{\widetilde{r}}$
  $\widetilde{e_1}, \cdots, \widetilde{e_\ell} \leftarrow \{0,1\}^d$
  $\widetilde{\mu} \leftarrow \mathsf{Hash}(hk, \widetilde{u_1} \| \widetilde{u_2} \| (\widetilde{e_i})_{i \in [\ell]})$
  If $(\mu \neq \widetilde{\mu}) \wedge$
  $(u_1^w = u_2) \wedge$
  $(u_1^{y_1 + z_1 \mu} u_2^{y_2 + z_2 \mu} = v)$ then
    $(m_i)_{i \in [\ell]} := (e_i \oplus \mathsf{H}(u_1^{\alpha_i}))_{i \in [\ell]}$,
  else Return $\perp$
  Return $m := m_1 \| \cdots \| m_\ell$
```

Fig. 3. Main construction of RNC-CCA secure RNCE Π'_{ddh} based on the DDH assumption.

Furthermore, let $c = (u_1, u_2, (e_i)_{i \in [\ell]}, v)$ be a real ciphertext output by $\mathsf{Enc}(pk, m')$ for any plaintext $m' = m'_1 \| \cdots \| m'_\ell \in \{0,1\}^{d \cdot \ell}$ (which is not necessarily m). Then, we can see that the fake secret key $\widetilde{sk}$ can decrypt the real ciphertext c to m' as follows. For each $i \in [\ell]$, we have

$$e_i \oplus \mathsf{H}(u_1^{x'_{i1}} u_2^{x'_{i2}}) = \mathsf{H}(k_i^r) \oplus m'_i \oplus \mathsf{H}(u_1^{\alpha_i - w\gamma_i + w\widetilde{r}\alpha_i} u_2^{\gamma_i - \widetilde{r}\alpha_i})$$
$$= \mathsf{H}(k_i^r) \oplus m'_i \oplus \mathsf{H}(u_1^{\alpha_i}) = \mathsf{H}(k_i^r) \oplus m'_i \oplus \mathsf{H}(k_i^r) = m'_i.$$

5.4 Security Proof for the Main Construction

In this section, we show the following theorem.

Theorem 4. *If the "+1"-DDH assumption holds in $\mathbb{G}$, $\mathcal{H}$ is a universal hash family, and $\Lambda = (\mathsf{HKG}, \mathsf{Hash})$ is a collision-resistant hash function, then Π'_{ddh} is RNC-CCA secure.*

Since the "+1"-DDH assumption is implied by the ordinary DDH assumption, Theorem 4 implies that our construction Π'_{ddh} is indeed RNC-CCA secure under the ordinary DDH assumption.

Here, for simplicity, we firstly give the proof for our main construction Π'_{ddh} conditioned on $\ell = 1$. Then, at the end of the proof, we explain the proof for the general case $\ell > 1$. Also, in the proof, we eliminate an index $i \in [\ell]$ from the description of our main construction Π'_{ddh} for simplicity.

Before describing the formal proof, we explain an important difference from the proof of Theorem 3. Although the proof below is almost the same as the proof of Theorem 3, we need the following additional argument when changing from a real secret key to a fake secret key. Unlike our basic construction in Section 5.1, when computing a fake secret key $x'_1, x'_2 \in \mathbb{Z}_p$ for opening a fake ciphertext component $\widetilde{e}$ to a plaintext $m \in \{0,1\}^d$, we need to sample a value γ from $\mathbb{Z}_p$ satisfying the condition $\mathsf{H}(g_1^{\gamma}) = \widetilde{e} \oplus m$ in our main construction, where $e \leftarrow \{0,1\}^d$ and g_1 is a generator of $\mathbb{G}$.

Here, we have to consider the event that we cannot sample γ from $\mathbb{Z}_p$ satisfying such a condition. In our proof, we show that the probability that this event happens is negligible. This is because the plaintext space $\mathcal{M}$ is $\{0,1\}^d$ and the hash function $\mathsf{H} : \mathbb{G} \to \{0,1\}^d$ is universal, and thus when we sample γ from $\mathbb{Z}_p$ Q_h times, where Q_h is a polynomial in λ satisfying $Q_h = \omega(2^d \cdot \log \lambda)$, we can make this probability negligible by using information-theoretic arguments and the leftover hash lemma. Therefore, we show that we can compute a fake secret key x'_1, x'_2 for opening the fake ciphertext component $\widetilde{e}$ to the plaintext m with an overwhelming probability. The formal proof is as follows.

Proof of Theorem 4. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ be any PPT adversary that attacks the RNC-CCA security of Π'_{ddh} and makes $Q_{dec} > 0$ decryption queries. Let Q_h be a polynomial in λ satisfying $Q_h = \omega(2^d \cdot \log \lambda)$. We introduce the following experiments $\{\mathsf{Exp}_i\}_{i=0}^{4}$.

Exp_0 : Exp_0 is exactly the same as $\mathsf{Exp}^{\mathsf{rnc\text{-}real}}_{\Pi'_{\mathsf{ddh}},\mathcal{A}}(\lambda)$. The detailed description is as follows.

1. First, Exp_0 samples $g_1 \leftarrow \mathbb{G}$ and $w \leftarrow \mathbb{Z}_p^*$ and sets $g_2 := g_1^w$. Next, Exp_0 samples $x_1, x_2, y_1, y_2, z_1, z_2 \leftarrow \mathbb{Z}_p$ and sets $k := g_1^{x_1} g_2^{x_2}$, $s := g_1^{y_1} g_2^{y_2}$, and $t := g_1^{z_1} g_2^{z_2}$. Then, it samples $\mathsf{H} \leftarrow \mathcal{H}$ and $hk \leftarrow \mathsf{HKG}(1^\lambda)$, sets $pk := (g_1, g_2, k, s, t, \mathsf{H}, hk)$ and $sk := (x_1, x_2, y_1, y_2, z_1, z_2)$, and runs $\mathcal{A}_1(pk)$. When $\mathcal{A}_1$ makes a decryption query $c = (u_1, u_2, e, v)$, Exp_0 computes $\mu \leftarrow \mathsf{Hash}(hk, u_1 \| u_2 \| e)$ and checks whether $u_1^{y_1 + z_1 \mu} u_2^{y_2 + z_2 \mu} = v$ holds. If this holds, Exp_0 returns $m = e \oplus \mathsf{H}(u_1^{x_1} u_2^{x_2})$ to $\mathcal{A}_1$. Otherwise, Exp_0 returns $\perp$ to $\mathcal{A}_1$.
2. When $\mathcal{A}_1$ outputs (m^*, st_1) and terminates, Exp_0 computes the challenge ciphertext c^* as follows. First, Exp_0 samples $r^* \leftarrow \mathbb{Z}_p$ and sets $u_1^* := g_1^{r^*}$, $u_2^* := g_2^{r^*}$, and $e^* := \mathsf{H}(k^{r^*}) \oplus m^*$. Next, Exp_0 computes $\mu^* \leftarrow \mathsf{Hash}(hk, u_1^* \| u_2^* \| e^*)$, sets $v^* := s^{r^*} t^{r^* \mu^*}$ and $c^* := (u_1^*, u_2^*, e^*, v^*)$, and runs $\mathcal{A}_2(c^*, \mathsf{st}_1)$. When $\mathcal{A}_2$ makes a decryption query $c = (u_1, u_2, e, v)$, Exp_0 answers the query for $\mathcal{A}_2$ in the same way as above.
3. When $\mathcal{A}_2$ outputs state information st_2 and terminates, Exp_0 runs $\mathcal{A}_3(sk, \mathsf{st}_2)$. When $\mathcal{A}_3$ outputs b' and terminates, Exp_0 outputs b'.

Exp_1 : Exp_1 is identical to Exp_0 except for the following changes.

- When computing the challenge ciphertext $c^* = (u_1^*, u_2^*, e^*, v^*)$, Exp_1 computes u_2^*, e^*, and v^* by $u_2^* := g_1^{wr^*+1}$, $e^* := \mathsf{H}((u_1^*)^{x_1}(u_2^*)^{x_2}) \oplus m^*$, and $v^* := (u_1^*)^{y_1}(u_2^*)^{y_2}((u_1^*)^{z_1}(u_2^*)^{z_2})^{\mu^*}$, respectively.
- When responding to a decryption query $c = (u_1, u_2, e, v)$ made by $\mathcal{A}_2$, Exp_1 answers $\perp$ if $\mathsf{Hash}(hk, u_1^* \| u_2^* \| e^*) = \mathsf{Hash}(hk, u_1 \| u_2 \| e)$ holds.
- When responding to a decryption query $c = (u_1, u_2, e, v)$ made by $\mathcal{A}_1$ or $\mathcal{A}_2$, Exp_1 answers $\perp$ if $u_1^w \neq u_2$ holds.

Exp_2 : Exp_2 is identical to Exp_1 except for the following changes.

- When computing the challenge ciphertext $c^* = (u_1^*, u_2^*, e^*, v^*)$, Exp_2 samples $\widetilde{e} \leftarrow \{0,1\}^d$ and computes $\mathsf{H}(k^{r^*}) := \widetilde{e} \oplus m^*$.
- $x'_1 := \alpha - w\gamma + wr^*\alpha \pmod p$ and $x'_2 := \gamma - r^*\alpha \pmod p$ are used instead of x_1 and x_2, respectively. Here, γ is chosen exactly the same as the opening algorithm Open and α is the discrete logarithm of k with the base g_1. Hence, when computing the challenge ciphertext $c^* := (u_1^*, u_2^*, e^*, v^*)$, Exp_2 computes e^* by $e^* := \mathsf{H}((u_1^*)^{x'_1}(u_2^*)^{x'_2}) \oplus m^*$ instead of $\mathsf{H}((u_1^*)^{x_1}(u_2^*)^{x_2}) \oplus m^*$. Note that $e^* = \mathsf{H}((u_1^*)^{x'_1}(u_2^*)^{x'_2}) \oplus m^* = \mathsf{H}(g_1^{r^*(\alpha - w\gamma + wr^*\alpha)} g_1^{(wr^*+1)(\gamma - r^*\alpha)}) \oplus m^* = \mathsf{H}(g_1^{\gamma}) \oplus m^* = \widetilde{e}$, and thus e^* is independent of the challenge plaintext m^*.
- When responding to a decryption query $c = (u_1, u_2, e, v)$ made by $\mathcal{A}_1$ and $\mathcal{A}_2$, Exp_2 computes $\mu \leftarrow \mathsf{Hash}(hk, u_1 \| u_2 \| e)$ and executes as follows.

    - For $\mathcal{A}_1$'s query, Exp_2 checks whether $(u_1^w = u_2) \wedge (u_1^{y_1+z_1\mu} u_2^{y_2+z_2\mu} = v)$ holds.
    - For $\mathcal{A}_2$'s query, Exp_2 checks whether $(u_1^w = u_2) \wedge (u_1^{y_1+z_1\mu} u_2^{y_2+z_2\mu} = v) \wedge (\mu \neq \mu^*)$ holds.

    If the first (resp., second) check is valid, Exp_2 returns $e \oplus \mathsf{H}(u_1^\alpha)$ instead of $e \oplus \mathsf{H}(u_1^{x_1} u_2^{x_2})$ to $\mathcal{A}_1$ (resp., $\mathcal{A}_2$). Note that the above conditions for the decryption oracle is exactly the same as one in Exp_1 and the only difference here is the value which is returned to $\mathcal{A}_1$ and $\mathcal{A}_2$.
  – Exp_2 gives the secret key $sk' := (x_1', x_2', y_1, y_2, z_1, z_2)$ to $\mathcal{A}_3$ instead of $sk := (x_1, x_2, y_1, y_2, z_1, z_2)$.

  Since u_1^*, u_2^*, and e^* in Exp_2 are independent of the challenge plaintext m^*, without loss of generality we generate them before $\mathcal{A}_1$ is run.

Exp_3 : Exp_3 is identical to Exp_2 except that when responding to a decryption query $c = (u_1, u_2, e, v)$ made by $\mathcal{A}_1$, Exp_3 answers $\perp$ if $\mathsf{Hash}(hk, u_1^* \| u_2^* \| e^*) = \mathsf{Hash}(hk, u_1 \| u_2 \| e)$ holds. Note that the procedure of the decryption oracle in Exp_3 is exactly the same as that of $\mathsf{FDec}(pk, td, c)$.

Exp_4 : Exp_4 is exactly the same as $\mathsf{Exp}^{\mathsf{rnc\text{-}sim}}_{\Pi'_{\mathsf{ddh}},\mathcal{A}}(\lambda)$.

We let $p_i := \Pr[\mathsf{Exp}_i(\lambda) = 1]$ for all $i \in [0, 4]$. Then, we have

$$\mathsf{Adv}^{\mathsf{rnc\text{-}cca}}_{\Pi'_{\mathsf{ddh}},\mathcal{A}}(\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{rnc\text{-}real}}_{\Pi'_{\mathsf{ddh}},\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{rnc\text{-}sim}}_{\Pi'_{\mathsf{ddh}},\mathcal{A}}(\lambda) = 1]| = |p_0 - p_4| \le \sum_{i=0}^{3} |p_i - p_{i+1}|.$$

It remains to show how each $|p_i - p_{i+1}|$ is upper-bounded. We show that $|p_0 - p_1| \le \mathsf{Adv}^{+1\text{-}\mathsf{ddh}}_{\mathbb{G},\mathcal{E}}(\lambda) + \mathsf{Adv}^{\mathsf{cr}}_{\Lambda,\mathcal{F}}(\lambda) + \frac{Q_{dec}}{p}$ holds (Lemma 15), where $\mathcal{E}$ (resp., $\mathcal{F}$) is a PPT adversary against the "+1"-DDH assumption in $\mathbb{G}$ (resp., the collision-resistance of Λ). Next, we show that $|p_1 - p_2| \le \sqrt{\frac{2^{d-2}}{p}} + 2 \cdot \left(\left(\frac{1}{2}\right)^{\frac{Q_h}{2^d}} + Q_h \sqrt{\frac{2^{d-2}}{p}} \right)$ holds (Lemma 16) by using information-theoretic arguments and the leftover hash lemma. Then, we show that there exists a PPT adversary $\mathcal{G}$ against the collision-resistance of Λ such that $|p_2 - p_3| \le \mathsf{Adv}^{\mathsf{cr}}_{\Lambda,\mathcal{G}}(\lambda) + \frac{Q_{dec}}{p}$ (Lemma 17). Finally, we show that $|p_3 - p_4| = 0$ holds (Lemma 18) by showing that Exp_3 and Exp_4 are identical.

Lemma 15. *There exist PPT adversaries $\mathcal{E}$ and $\mathcal{F}$ such that* $|p_0 - p_1| \le \mathsf{Adv}^{+1\text{-}\mathsf{ddh}}_{\mathbb{G},\mathcal{E}}(\lambda) + \mathsf{Adv}^{\mathsf{cr}}_{\Lambda,\mathcal{F}}(\lambda) + \frac{Q_{dec}}{p}$.

Proof of Lemma 15. Firstly, we define the following sub-experiments between Exp_0 and Exp_1.

$\mathsf{Exp}_{0.0}$: $\mathsf{Exp}_{0.0}$ is identical to Exp_0.

$\mathsf{Exp}_{0.1}$: $\mathsf{Exp}_{0.1}$ is identical to $\mathsf{Exp}_{0.0}$ except that when computing the challenge ciphertext $c^* = (u_1^*, u_2^*, e^*, v^*)$, $\mathsf{Exp}_{0.1}$ computes $e^* := \mathsf{H}((u_1^*)^{x_1}(u_2^*)^{x_2}) \oplus m^*$ and $v^* := (u_1^*)^{y_1}(u_2^*)^{y_2}((u_1^*)^{z_1}(u_2^*)^{z_2})^{\mu^*}$.

$\mathsf{Exp}_{0.2}$: $\mathsf{Exp}_{0.2}$ is identical to $\mathsf{Exp}_{0.1}$ except that when computing the challenge ciphertext $c^* = (u_1^*, u_2^*, e^*, v^*)$, $\mathsf{Exp}_{0.2}$ computes u_2^* by $u_2^* := g_1^{wr^*+1}$.

$\mathsf{Exp}_{0.3}$: $\mathsf{Exp}_{0.3}$ is identical to $\mathsf{Exp}_{0.2}$ except that when responding to a decryption query $c = (u_1, u_2, e, v)$ made by $\mathcal{A}_2$, $\mathsf{Exp}_{0.3}$ answers $\perp$ if $\mathsf{Hash}(hk, u_1^* \| u_2^* \| e^*) = \mathsf{Hash}(hk, u_1 \| u_2 \| e)$ holds.

$\mathsf{Exp}_{0.4}$: $\mathsf{Exp}_{0.4}$ is identical to $\mathsf{Exp}_{0.3}$ except that when responding to a decryption query $c = (u_1, u_2, e, v)$ made by $\mathcal{A}_1$ or $\mathcal{A}_2$, $\mathsf{Exp}_{0.4}$ answers $\perp$ if $u_1^w \neq u_2$ holds. (This experiment is exactly the same as Exp_1.)

We let $p_{0.j} := \Pr[\mathsf{Exp}_{0.j}(\lambda) = 1]$ for all $j \in [0, 4]$. Then, we can show that $|p_{0.0} - p_{0.1}| = 0$ holds, there exists a PPT adversary $\mathcal{E}$ (resp., $\mathcal{F}$) such that $|p_{0.1} - p_{0.2}| = \mathsf{Adv}^{+1\text{-}\mathsf{ddh}}_{\mathbb{G},\mathcal{E}}(\lambda)$ (resp., $|p_{0.2} - p_{0.3}| \le \mathsf{Adv}^{\mathsf{cr}}_{\Lambda,\mathcal{F}}(\lambda)$), and $|p_{0.3} - p_{0.4}| \le \frac{Q_{dec}}{p}$ holds. These proofs are essentially the same as those of Lemma 8, Lemma 9, Lemma 10, and Lemma 11 in the proof of Theorem 3, respectively, and thus omitted. Putting everything together, we obtain

$$|p_0 - p_1| \le \sum_{i=0}^{3} |p_{0.i} - p_{0.(i+1)}| \le \mathsf{Adv}^{+1\text{-}\mathsf{ddh}}_{\mathbb{G},\mathcal{E}}(\lambda) + \mathsf{Adv}^{\mathsf{cr}}_{\Lambda,\mathcal{F}}(\lambda) + \frac{Q_{dec}}{p}.$$

□ (**Lemma 15**)

Lemma 16. *$|p_1 - p_2| \leq \sqrt{\frac{2^{d-2}}{p}} + 2 \cdot \left(\left(\frac{1}{2}\right)^{\frac{Q_h}{2^d}} + Q_h\sqrt{\frac{2^{d-2}}{p}} \right)$ holds.*

Proof of Lemma 16. Firstly, we define the following sub-experiments between Exp_1 and Exp_2.

$\mathsf{Exp}_{1.0}$: $\mathsf{Exp}_{1.0}$ is identical to Exp_1. We let $K^* := k^{r^*}$ and $h^* := \mathsf{H}(k^{r^*})$.

$\mathsf{Exp}_{1.1}$: $\mathsf{Exp}_{1.1}$ is identical to $\mathsf{Exp}_{1.0}$ except that $\mathsf{Exp}_{1.1}$ firstly samples $k, K^* \leftarrow \mathbb{G}$ then computes (x_1, x_2) satisfying $(g_1^{x_1} g_2^{x_2} = k) \wedge ((u_1^*)^{x_1}(u_2^*)^{x_2} = K^*)$ instead of firstly sampling $x_1, x_2 \leftarrow \mathbb{Z}_p$ then computing $k = g_1^{x_1} g_2^{x_2}$ and $K^* = (u_1^*)^{x_1}(u_2^*)^{x_2}$. Note that $x_1 = \alpha - w\gamma + wr^*\alpha \pmod p$ and $x_2 = \gamma - r^*\alpha \pmod p$ holds, where $\alpha := \log_{g_1} k$ and $\gamma := \log_{g_1} K^*$.

$\mathsf{Exp}_{1.2}$: $\mathsf{Exp}_{1.2}$ is identical to $\mathsf{Exp}_{1.1}$ except that $\mathsf{Exp}_{1.2}$ computes $K' \leftarrow \mathbb{G}, h^* := \mathsf{H}(K')$, and $K^* \leftarrow \mathsf{HashInv}(h^*)$, where $\mathsf{HashInv}$ is an algorithm explained below, instead of sampling $K^* \leftarrow \mathbb{G}$ and then computing $h^* := \mathsf{H}(K^*)$. The algorithm $\mathsf{HashInv}$ takes an input $h^* \in \{0,1\}^d$ and if the set of preimages $\mathsf{H}^{-1}(h^*) := \{x \in \mathbb{G} | \mathsf{H}(x) = h^*\}$ is not empty, it outputs x chosen uniformly at random from $\mathsf{H}^{-1}(h^*)$. Otherwise, it outputs $\perp$.

$\mathsf{Exp}_{1.3}$: $\mathsf{Exp}_{1.3}$ is identical to $\mathsf{Exp}_{1.2}$ except that $\mathsf{Exp}_{1.3}$ samples $h^* \leftarrow \{0,1\}^d$ instead of sampling $K' \leftarrow \mathbb{G}$ and computing $h^* := \mathsf{H}(K')$.

$\mathsf{Exp}_{1.4}$: $\mathsf{Exp}_{1.4}$ is identical to $\mathsf{Exp}_{1.3}$ except that $\mathsf{Exp}_{1.4}$ samples $\widetilde{e} \leftarrow \{0,1\}^d$ and computes $h^* := \widetilde{e} \oplus m^*$ instead of sampling $h^* \leftarrow \{0,1\}^d$.

$\mathsf{Exp}_{1.5}$: $\mathsf{Exp}_{1.5}$ is identical to $\mathsf{Exp}_{1.4}$ except for the following two changes. Firstly, γ is chosen exactly the same as the opening algorithm Open instead of sampling $K^* (= g_1^\gamma) \leftarrow \mathsf{HashInv}(h^*)$. That is, if we cannot find $\gamma \in \mathbb{Z}_p$ that satisfies $\mathsf{H}(g_1^\gamma) = \widetilde{e} \oplus m^*$ within Q_h trials, $\mathsf{Exp}_{1.5}$ stops and gives $sk := \perp$ to $\mathcal{A}_3$. Otherwise, $\mathsf{Exp}_{1.5}$ gives the secret key $sk' := (x_1', x_2', y_1, y_2, z_1, z_2)$ to $\mathcal{A}_3$.
Secondly, when responding to a decryption query $c = (u_1, u_2, e, v)$ made by $\mathcal{A}_1$ and $\mathcal{A}_2$, Exp_2 computes $\mu \leftarrow \mathsf{Hash}(hk, u_1 \| u_2 \| e)$ and executes as follows.
- For $\mathcal{A}_1$'s query, Exp_2 checks whether $(u_1^w = u_2) \wedge (u_1^{y_1 + z_1\mu} u_2^{y_2 + z_2\mu} = v)$ holds.
- For $\mathcal{A}_2$'s query, Exp_2 checks whether $(u_1^w = u_2) \wedge (u_1^{y_1 + z_1\mu} u_2^{y_2 + z_2\mu} = v) \wedge (\mu \neq \mu^*)$ holds.

If the first (resp., second) check is valid, Exp_2 returns $e \oplus \mathsf{H}(u_1^\alpha)$ instead of $e \oplus \mathsf{H}(u_1^{x_1} u_2^{x_2})$ to $\mathcal{A}_1$ (resp., $\mathcal{A}_2$). Note that the above conditions for the decryption oracle is exactly the same as one in Exp_1 and the only difference here is the value which is returned to $\mathcal{A}_1$ and $\mathcal{A}_2$. We can see that this experiment is exactly the same as Exp_2 by renaming x_1 (resp., x_2) to x_1' (resp., x_2').

We note that the experiments $\mathsf{Exp}_{1.j}$ for all $j \in [4]$ cannot always be executed in polynomial time, but the experiment $\mathsf{Exp}_{1.5}$ can always be executed in polynomial time. Here, we let $p_{1.j} := \Pr[\mathsf{Exp}_{1.j}(\lambda) = 1]$ for all $j \in [0,5]$. In the following, we show that $|p_{1.0} - p_{1.1}| = 0$, $|p_{1.1} - p_{1.2}| = 0$, $|p_{1.2} - p_{1.3}| \leq \sqrt{\frac{2^{d-2}}{p}}$, $|p_{1.3} - p_{1.4}| = 0$, and $|p_{1.4} - p_{1.5}| \leq 2 \cdot \left(\left(\frac{1}{2}\right)^{\frac{Q_h}{2^d}} + Q_h\sqrt{\frac{2^{d-2}}{p}} \right)$ hold.

Firstly, we see that $|p_{1.0} - p_{1.1}| = 0$ holds. In $\mathsf{Exp}_{1.1}$, we firstly sample $k, K^* \leftarrow \mathbb{G}$ then computes (x_1, x_2) satisfying $(g_1^{x_1} g_2^{x_2} = k) \wedge ((u_1^*)^{x_1}(u_2^*)^{x_2} = K^*)$. In fact, we can find such (x_1, x_2) by considering the discrete logarithms with the base g_1 for k and K^* as

$$\begin{pmatrix} \log_{g_1} k \\ \log_{g_1} K^* \end{pmatrix} = M \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \text{where } M = \begin{pmatrix} 1 & w \\ r^* & 1 + wr^* \end{pmatrix}.$$

Since det$(M) = 1 \ (\neq 0)$ holds, we can have $x_1 = \log_{g_1} k - w \log_{g_1} K^* + wr^* \log_{g_1} k \pmod p$ and $x_2 = \log_{g_1} K^* - r^* \log_{g_1} k \pmod p$. Here, the distribution of (x_1, x_2) is exactly the same as one in Exp_1. Thus, we have $|p_{1.0} - p_{1.1}| = 0$.

Next, we see that $|p_{1.1} - p_{1.2}| = 0$ holds. In $\mathsf{Exp}_{1.2}$, we compute $K' \leftarrow \mathbb{G}, h^* := \mathsf{H}(K')$, and $K^* \leftarrow \mathsf{H}^{-1}(h^*)$ instead of $K^* \leftarrow \mathbb{G}$ and $h^* := \mathsf{H}(K^*)$. Here, the distribution $\{K^* \leftarrow \mathbb{G}; h^* \leftarrow \mathsf{H}(K^*) : (K^*, h^*)\}$ and $\{K' \leftarrow \mathbb{G}; h^* \leftarrow \mathsf{H}(K'); K^* \leftarrow \mathsf{H}^{-1}(h^*) : (K^*, h^*)\}$ are exactly the same. Hence, the view of $\mathcal{A}$ in $\mathsf{Exp}_{1.2}$ is exactly the same as that in $\mathsf{Exp}_{1.1}$. Thus, we have $|p_{1.1} - p_{1.2}| = 0$.

Then, we see that $|p_{1.2} - p_{1.3}| \leq \sqrt{\frac{2^{d-2}}{p}}$ holds. The only difference between $\mathsf{Exp}_{1.2}$ and $\mathsf{Exp}_{1.3}$ is that we sample $h^* \leftarrow \{0,1\}^d$ in $\mathsf{Exp}_{1.3}$ instead of sampling $K' \leftarrow \mathbb{G}$ and computing $h^* := \mathsf{H}(K')$.

Thus, we have $|p_{1.2} - p_{1.3}| \leq \mathsf{SD}((\mathsf{H}, \mathsf{H}(K')), (\mathsf{H}, h^*))$. Here, applying the leftover hash lemma for $\mathcal{H} = \{\mathsf{H} : \mathbb{G} \to \{0,1\}^d\}$, we have $\mathsf{SD}((\mathsf{H}, \mathsf{H}(K')), (\mathsf{H}, h^*)) \leq \sqrt{\frac{2^{d-2}}{p}}$ due to $|\mathbb{G}| = p$ and $|\{0,1\}^d| = 2^d$, where $\mathsf{H} \leftarrow \mathcal{H}$, $K' \leftarrow \mathbb{G}$, and $h^* \leftarrow \{0,1\}^d$. Therefore, we can conclude that $|p_{1.2} - p_{1.3}| \leq \mathsf{SD}((\mathsf{H}, \mathsf{H}(K')), (\mathsf{H}, h^*)) \leq \sqrt{\frac{2^{d-2}}{p}}$ holds.

Next, we can see that $|p_{1.3} - p_{1.4}| = 0$ holds since in $\mathsf{Exp}_{1.4}$, $h^* := \widetilde{e} \oplus m^*$ is also uniformly distributed in $\{0,1\}^d$ due to $\widetilde{e} \leftarrow \{0,1\}^d$, and thus the view of $\mathcal{A}$ is exactly the same as that in $\mathsf{Exp}_{1.3}$.

Finally, we show that $|p_{1.4} - p_{1.5}| \leq 2 \cdot \left(\left(\frac{1}{2}\right)^{\frac{Q_h}{2^d}} + Q_h \sqrt{\frac{2^{d-2}}{p}} \right)$ holds. In $\mathsf{Exp}_{1.5}$, γ is chosen exactly the same as the opening algorithm Open instead of sampling $K^* (= g_1^\gamma) \leftarrow \mathsf{HashInv}(h^*)$. In Open, we firstly sample $\gamma \leftarrow \mathbb{Z}_p$ then check whether $\mathsf{H}(g_1^\gamma) = \widetilde{e} \oplus m^*$ holds. In the following, we let $\mathbf{Bad}$ be the event that we cannot find $\gamma \in \mathbb{Z}_p$ that satisfies $\mathsf{H}(g_1^\gamma) = \widetilde{e} \oplus m^*$ within Q_h trials in Open in $\mathsf{Exp}_{1.5}$. That is, $\mathbf{Bad}$ is the event that when sampling $\gamma \in \mathbb{Z}_p$ uniformly at random Q_h times, there exists no γ satisfying $\mathsf{H}(g_1^\gamma) = \widetilde{e} \oplus m^*$ in $\mathsf{Exp}_{1.5}$. Thus, the probability that the event $\mathbf{Bad}$ happens is estimated as

$$\Pr[\mathbf{Bad}] \leq \Pr_{\substack{\mathsf{H} \leftarrow \mathcal{H} \\ \gamma_1, \cdots, \gamma_{Q_h} \leftarrow \mathbb{Z}_p}} \left[\bigwedge_{i=1}^{Q_h} (\mathsf{H}(g_1^{\gamma_i}) \neq h^*) \right].$$

If $\mathbf{Bad}$ does not happen, we can see that the distribution of γ sampled in $\mathsf{HashInv}$ is exactly the same as one sampled in Open conditioned on that γ satisfying $\mathsf{H}(g_1^\gamma) = \widetilde{e} \oplus m^*$ is found. Therefore, in this case, the distribution of γ is not changed between $\mathsf{Exp}_{1.4}$ and $\mathsf{Exp}_{1.5}$. As for responses to decryption queries $c = (u_1, u_2, e, v)$ made by $\mathcal{A}$, recall that the rejection rules are exactly the same in $\mathsf{Exp}_{1.4}$ and $\mathsf{Exp}_{1.5}$. Also, if c is not rejected,

$$\mathsf{H}(u_1^\alpha) = \mathsf{H}((g_1^\alpha)^r) = \mathsf{H}(k^r) = \mathsf{H}((g_1^{x_1} g_2^{x_2})^r) = \mathsf{H}(u_1^{x_1} u_2^{x_2})$$

holds, where $r = \log_{g_1} u_1$. Therefore, the answer $e \oplus \mathsf{H}(u_1^\alpha)$ in $\mathsf{Exp}_{1.5}$ is exactly the same as the answer $e \oplus \mathsf{H}(u_1^{x_1} u_2^{x_2})$ in $\mathsf{Exp}_{1.4}$. Hence, the view of $\mathcal{A}$ in $\mathsf{Exp}_{1.5}$ is exactly the same as that in $\mathsf{Exp}_{1.4}$ if we can find $\gamma \in \mathbb{Z}_p$ satisfying the condition within Q_h trials. Thus, $p_{1.4} = \Pr[\mathsf{Exp}_{1.5}(\lambda) = 1 | \neg\mathbf{Bad}]$ holds. Then, we have the following inequality

$$\begin{aligned}
|p_{1.4} - p_{1.5}| &= |p_{1.4} - (\Pr[(\mathsf{Exp}_{1.5}(\lambda) = 1) \wedge \neg\mathbf{Bad}] + \Pr[(\mathsf{Exp}_{1.5}(\lambda) = 1) \wedge \mathbf{Bad}])| \\
&\leq |p_{1.4} - \Pr[(\mathsf{Exp}_{1.5}(\lambda) = 1) \wedge \neg\mathbf{Bad}]| + \Pr[(\mathsf{Exp}_{1.5}(\lambda) = 1) \wedge \mathbf{Bad}] \\
&\leq |p_{1.4} - \Pr[(\mathsf{Exp}_{1.5}(\lambda) = 1) | \neg\mathbf{Bad}] \cdot \Pr[\neg\mathbf{Bad}]| + \Pr[\mathbf{Bad}] \\
&= |p_{1.4} - (1 - \Pr[\mathbf{Bad}]) \cdot \Pr[(\mathsf{Exp}_{1.5}(\lambda) = 1) | \neg\mathbf{Bad}]| + \Pr[\mathbf{Bad}] \\
&= \Pr[\mathbf{Bad}] \cdot \Pr[(\mathsf{Exp}_{1.5}(\lambda) = 1) | \neg\mathbf{Bad}] + \Pr[\mathbf{Bad}] \\
&\leq 2 \cdot \Pr[\mathbf{Bad}] \\
&\leq 2 \cdot \left(\Pr_{\substack{\mathsf{H} \leftarrow \mathcal{H} \\ \gamma_1, \cdots, \gamma_{Q_h} \leftarrow \mathbb{Z}_p}} \left[\bigwedge_{i=1}^{Q_h} (\mathsf{H}(g_1^{\gamma_i}) \neq h^*) \right] \right) \\
&\leq 2 \cdot \left| \Pr_{\substack{\mathsf{H} \leftarrow \mathcal{H} \\ \gamma_1, \cdots, \gamma_{Q_h} \leftarrow \mathbb{Z}_p}} \left[\bigwedge_{i=1}^{Q_h} (\mathsf{H}(g_1^{\gamma_i}) \neq h^*) \right] - \Pr_{h_1, \cdots, h_{Q_h} \leftarrow \{0,1\}^d} \left[\bigwedge_{i=1}^{Q_h} (h_i \neq h^*) \right] \right| \\
&\quad + 2 \cdot \left(\Pr_{h_1, \cdots, h_{Q_h} \leftarrow \{0,1\}^d} \left[\bigwedge_{i=1}^{Q_h} (h_i \neq h^*) \right] \right). \qquad (1)
\end{aligned}$$

Here, since $h_1, \cdots, h_{Q_h}$ are chosen uniformly at random from $\{0,1\}^d$, we have

$$\Pr_{h_1, \cdots, h_{Q_h} \leftarrow \{0,1\}^d} \left[\bigwedge_{i=1}^{Q_h} (h_i \neq h^*) \right] = \left(1 - \frac{1}{2^d}\right)^{Q_h} = \left(1 - \frac{1}{2^d}\right)^{2^d \cdot \frac{Q_h}{2^d}} \overset{(*)}{<} \left(\frac{1}{2}\right)^{\frac{Q_h}{2^d}},$$

where in the inequality $(*)$ we used the fact that $\left(1 - \frac{1}{x}\right)^x < \frac{1}{2}$ holds for all $x > 0$.

Also, due to the definition of the statistical distance, we can bound the first term of Equation (1)

$$\begin{aligned}&\left| \Pr_{\substack{\mathsf{H} \leftarrow \mathcal{H} \\ \gamma_1, \cdots, \gamma_{Q_h} \leftarrow \mathbb{Z}_p}} \left[\bigwedge_{i=1}^{Q_h} (\mathsf{H}(g_1^{\gamma_i}) \neq h^*) \right] - \Pr_{h_1, \cdots, h_{Q_h} \leftarrow \{0,1\}^d} \left[\bigwedge_{i=1}^{Q_h} (h_i \neq h^*) \right] \right| \\ &\leq \mathsf{SD}((\mathsf{H}, \mathsf{H}(g_1^{\gamma_1}), \cdots, \mathsf{H}(g_1^{\gamma_{Q_h}})), (\mathsf{H}, h_1, \cdots, h_{Q_h})),\end{aligned}$$

where $\mathsf{H} \leftarrow \mathcal{H}, h_1, \cdots, h_{Q_h} \leftarrow \{0,1\}^d$.

By using a hybrid argument, we can see that $\mathsf{SD}((\mathsf{H}, \mathsf{H}(g_1^{\gamma_1}), \cdots, \mathsf{H}(g_1^{\gamma_{Q_h}})), (\mathsf{H}, h_1, \cdots, h_{Q_h})) \leq Q_h \cdot \mathsf{SD}((\mathsf{H}, \mathsf{H}(g_1^{\gamma_1})), (\mathsf{H}, h_1))$ holds. Furthermore, by applying the leftover hash lemma for $\mathcal{H} = \{\mathsf{H} : \mathbb{G} \to \{0,1\}^d\}$, we can see that $\mathsf{SD}((\mathsf{H}, \mathsf{H}(g_1^{\gamma_1})), (\mathsf{H}, h_1)) \leq \sqrt{\frac{2^{d-2}}{p}}$ holds due to $|\mathbb{G}| = p$ and $|\{0,1\}^d| = 2^d$. Thus, we have

$$\left| \Pr_{\substack{\mathsf{H} \leftarrow \mathcal{H} \\ \gamma_1, \cdots, \gamma_{Q_h} \leftarrow \mathbb{Z}_p}} \left[\bigwedge_{i=1}^{Q_h} (\mathsf{H}(g_1^{\gamma_i}) \neq h^*) \right] - \Pr_{h_1, \cdots, h_{Q_h} \leftarrow \{0,1\}^d} \left[\bigwedge_{i=1}^{Q_h} (h_i \neq h^*) \right] \right| \leq Q_h \sqrt{\frac{2^{d-2}}{p}}.$$

Putting everything together, we obtain

$$\begin{aligned}|p_{1.4} - p_{1.5}| \leq 2 \cdot &\left| \Pr_{\substack{\mathsf{H} \leftarrow \mathcal{H} \\ \gamma_1, \cdots, \gamma_{Q_h} \leftarrow \mathbb{Z}_p}} \left[\bigwedge_{i=1}^{Q_h} (\mathsf{H}(g_1^{\gamma_i}) \neq h^*) \right] - \Pr_{h_1, \cdots, h_{Q_h} \leftarrow \{0,1\}^d} \left[\bigwedge_{i=1}^{Q_h} (h_i \neq h^*) \right] \right| \\ &+ 2 \cdot \left(\Pr_{h_1, \cdots, h_{Q_h} \leftarrow \{0,1\}^d} \left[\bigwedge_{i=1}^{Q_h} (h_i \neq h^*) \right] \right) \\ &< 2 \cdot \left(\left(\frac{1}{2}\right)^{\frac{Q_h}{2^d}} + Q_h \sqrt{\frac{2^{d-2}}{p}} \right).\end{aligned}$$

From the above arguments, we conclude

$$|p_1 - p_2| \leq \sum_{i=0}^{4} |p_{1.i} - p_{1.(i+1)}| \leq \sqrt{\frac{2^{d-2}}{p}} + 2 \cdot \left(\left(\frac{1}{2}\right)^{\frac{Q_h}{2^d}} + Q_h \sqrt{\frac{2^{d-2}}{p}} \right).$$

□ **(Lemma 16)**

Lemma 17. *There exists a PPT adversary $\mathcal{G}$ such that* $|p_2 - p_3| \leq \mathsf{Adv}^{\mathsf{cr}}_{\Lambda,\mathcal{G}}(\lambda) + \frac{Q_{dec}}{p}$.

The proof of this lemma is essentially the same as that of Lemma 13. Thus, we omit it here.

Lemma 18. $|p_3 - p_4| = 0$ *holds.*

Proof of Lemma 18. From the constructions of FKG, Fake, FDec, and Open, the information that the adversary $\mathcal{A}$ gets in Exp_3 is exactly the same as that in Exp_4. Specifically, we can confirm it as follows.

First, it is easy to see that the public key $pk = (g_1, g_2, k, s, t, \mathsf{H}, hk)$ given to $\mathcal{A}_1$ in Exp_3 is exactly the same as the one given to $\mathcal{A}_1$ in Exp_4.

Second, we recall that the challenge ciphertext $c^* = (u_1^*, u_2^*, e^*, v^*)$ is generated in Exp_3 as

$$(u_1^*, u_2^*, e^*, v^*) = (g_1^{r^*},\ \ g_1 g_2^{r^*},\ \ \widetilde{e},\ \ (g_1^{r^* y_1} g_1^{(1+wr^*)y_2})(g_1^{r^* z_1} g_1^{(1+wr^*)z_2})^{\mu^*}).$$

We can confirm that $v^* = (g_1^{r^* y_1} g_1^{(1+wr^*)y_2})(g_1^{r^* z_1} g_1^{(1+wr^*)z_2})^{\mu^*} = g_1^{y_2 + z_2 \mu^*} s^{r^*} t^{r^* \mu^*}$. Therefore, the above (u_1^*, u_2^*, e^*, v^*) are of the same form as those generated by Fake. Thus, the challenge ciphertext (u_1^*, u_2^*, e^*, v^*) is exactly the same as that in Exp_4.

Third, the behavior of the decryption oracle in Exp_3 is exactly the same as that in Exp_4. Concretely, when $\mathcal{A}$ makes a decryption query $c = (u_1, u_2, e, v)$ in Exp_3, Exp_3 computes $\mu = \mathsf{Hash}(hk, u_1 \| u_2 \| e)$ and checks whether $(\mu = \mu^*) \vee (u_1^w \neq u_2) \vee (u_1^{y_1+z_1\mu} u_2^{y_2+z_2\mu} \neq v)$ holds. If this condition holds, Exp_3 returns $\perp$ to $\mathcal{A}$. Otherwise, Exp_3 returns $e \oplus \mathsf{H}(u_1^{\alpha})$ to $\mathcal{A}$. This behavior is exactly the same as that of FDec, and thus this decryption oracle's procedure in Exp_3 is exactly the same as that in Exp_4.

Finally, it is easy to see that the secret key $sk' = (x_1', x_2', y_1, y_2, z_1, z_2)$ which $\mathcal{A}_3$ gets in Exp_3 is exactly the same as the one which $\mathcal{A}_3$ gets in Exp_4.

From the above arguments, we can conclude $|p_3 - p_4| = 0$. □ **(Lemma 18)**

Putting everything together, we obtain

$$\begin{aligned}\mathsf{Adv}^{\mathsf{rnc\text{-}cca}}_{\Pi'_{\mathsf{ddh}},\mathcal{A}}(\lambda) &= |p_0 - p_4| \\ &\leq \sum_{i=0}^{3} |p_i - p_{i+1}| \\ &\leq \mathsf{Adv}^{+1\text{-}\mathsf{ddh}}_{\mathbb{G},\mathcal{E}}(\lambda) + \mathsf{Adv}^{\mathsf{cr}}_{\Lambda,\mathcal{F}}(\lambda) + \mathsf{Adv}^{\mathsf{cr}}_{\Lambda,\mathcal{G}}(\lambda) + \frac{2Q_{dec}}{p} + \left(\frac{1}{2}\right)^{\frac{Q_h}{2^d}-1} + (2Q_h+1)\sqrt{\frac{2^{d-2}}{p}}.\end{aligned}$$

Since the "+1"-DDH assumption holds in $\mathbb{G}$, $\mathcal{H}$ is a universal hash family, Λ is a collision-resistant hash function, Q_{dec} is a polynomial of λ, $d = O(\log \lambda)$, $Q_h = \omega(2^d \cdot \log \lambda)$, and $p = \Omega(2^{\lambda})$, for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{rnc\text{-}cca}}_{\Pi'_{\mathsf{ddh}},\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$ holds. Therefore, Π'_{ddh} (in the case $\ell = 1$) satisfies RNC-CCA security.

Finally, we explain the case that the plaintext space consists of multiple blocks ($\ell > 1$), where ℓ is a polynomial in λ. In fact, a proof for this case is almost the same as the above proof. The only difference from the above proof is that we have to change all real components $(x_{ij})_{i\in[\ell],j\in[2]}$ to fake ones $(x'_{ij})_{i\in[\ell],j\in[2]}$. To this end, we apply the same argument as in the proof of Lemma 16 for each component (x_{ij}) for all $i \in [\ell]$ and $j \in [2]$. By using a hybrid argument, we can show that the difference between the probability p_1 and p_2 is bounded by $\ell \cdot |p_1 - p_2| = \ell \cdot \left(\sqrt{\frac{2^{d-2}}{p}} + 2 \cdot \left(\left(\frac{1}{2}\right)^{\frac{Q_h}{2^d}} + Q_h\sqrt{\frac{2^{d-2}}{p}}\right)\right)$. Thus, due to the same arguments as in the above proof, we can actually show that Π'_{ddh} for any polynomial $\ell > 1$ satisfies RNC-CCA security. □ (**Theorem 4**)

Acknowledgement

The authors would like to thank the members of the study group "Shin-Akarui-Angou-BenkyouKai", and the anonymous reviewers of SCN 2018 for their helpful comments and suggestions. This work is partially supported by Input Output Hong Kong, Nomura Research Institute, NTT Secure Platform Laboratories, Mitsubishi Electric, I-System, JST CREST JPMJCR14D6, JPMJCR1688 JST OPERA and JSPS KAKENHI 16H01705, 16J10322, 17H01695.

References

1. C. Bader, D. Hofheinz, T. Jager, E. Kiltz, and Y. Li. Tightly-secure authenticated key exchange. TCC 2015, LNCS 9014, pp. 629-658.
2. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. CRYPTO 2001, LNCS 2139, pp. 1-18.
3. M. Bellare, R. Dowsley, B. Waters, and S. Yilek. Standard security does not imply security against selective-opening. EUROCRYPT 2012, LNCS 7237, pp. 645-662.
4. M. Bellare, D. Hofheinz, and S. Yilek. Possibility and impossibility results for encryption and commitment secure under selective opening. EUROCRYPT 2009, LNCS 5479, pp. 1-35.
5. M. Bellare and S. Yilek. Encryption schemes secure under selective opening attack. *IACR Cryptology ePrint Archive*, 2009:101.
6. R. Canetti, S. Halevi, and J. Katz. Adaptively-secure, non-interactive public-key encryption. TCC 2005, LNCS 3378, pp. 150-168.

7. R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. CRYPTO 1998, LNCS 1462, pp. 13-25.
8. R. Cramer and V. Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. EUROCRYPT 2002, LNCS 2332, pp. 45-64.
9. I. Damgård and J.B. Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. CRYPTO 2000, LNCS 1880, pp. 432-450.
10. D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography (extended abstract). STOC 1991, pp. 542-552.
11. S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *IACR Cryptology ePrint Archive*, 2013:451.
12. S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270-299, 1984.
13. J. Groth, R. Ostrovsky, and A. Sahai. Perfect non-interactive zero knowledge for NP. EUROCRYPT 2006, LNCS 4004, pp. 339-358.
14. J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. EUROCRYPT 2008, LNCS 4965, pp. 415-432.
15. D. Hofheinz, T. Jager, and A. Rupp. Public-key encryption with simulation-based selective-opening security and compact ciphertexts. TCC 2016, LNCS 9986, pp. 146-168.
16. Keisuke Hara, Fuyuki Kitagawa, Takahiro Matsuda, Goichiro Hanaoka, Keisuke Tanaka. Simulation-based receiver selective opening CCA secure PKE from standard computational assumptions. SCN 2018, LNCS 11035, pp. 140-159.
17. C. Hazay, A. Patra, and B. Warinschi. Selective opening security for receivers. ASIACRYPT 2015 Part I, LNCS 9452, pp. 443-469.
18. Z. Huang, J. Lai, W. Chen, M. H. Au, Z. Peng, and J. Li. Simulation-based selective opening security for receivers under chosen-ciphertext attacks. *IACR Cryptology ePrint Archive*, 2018:755. To appear in Designs, Codes and Cryptography.
19. D. Jia, X. Lu, and B. Li. Receiver selective opening security from indistinguishability obfuscation. INDOCRYPT 2016, LNCS 10095, pp. 393-410.
20. D. Jia, X. Lu, and B. Li. Constructions secure against receiver selective opening and chosen ciphertext attacks. CT-RSA 2017, LNCS 10159, pp. 417-431.
21. B. Libert, A. Sakzad, D. Stehlé, and R. Steinfeld. All-but-many lossy trapdoor functions and selective opening chosen-ciphertext security from LWE. CRYPTO 2017 Part III, LNCS 10403, pp. 332-364.
22. Y. Lindell. A simpler construction of CCA2-secure public-key encryption under general assumptions. EUROCRYPT 2003, LNCS 2656, pp. 241-254.
23. S. Liu and Kenneth G. Paterson. Simulation-based selective opening CCA security for PKE from key encapsulation mechanisms. PKC 2015, LNCS 9020, pp. 3-26.
24. S. Liu, F. Zhang, and K. Chen. Selective opening chosen ciphertext security directly from the DDH assumption. NSS 2012, LNCS 7645, pp.100-112.
25. L. Lyu, S. Liu, S. Han, and D. Gu. Tightly sim-so-cca secure public key encryption from standard assumptions. PKC 2018, LNCS 10769, pp. 62-92.
26. M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. STOC 1990, pp. 427-437.
27. C. Peikert and B. Waters. Lossy trapdoor functions and their applications. STOC 2008, pp. 187-196.
28. A. Rosen and G. Segev. Chosen-ciphertext security via correlated products. TCC 2009, LNCS 5444, pp. 419-436.
29. A. Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. FOCS 1999, pp. 543-553.
30. A. Sahai and B. Waters. How to use indistinguishability obfuscation: deniable encryption, and more. STOC 2014, pp. 475-484.

```
$\mathsf{KG}''(1^\lambda)$ :
    $\alpha_1, \cdots, \alpha_\ell \leftarrow \{0,1\}$
    $(pk_0^i, sk_0^i)_{i\in[\ell]} \leftarrow (\mathsf{KG}(1^\lambda))_{i\in[\ell]}$
    $(pk_1^i, sk_1^i)_{i\in[\ell]} \leftarrow (\mathsf{KG}(1^\lambda))_{i\in[\ell]}$
    $crs \leftarrow \mathsf{CRSGen}(1^\lambda)$
    $pk := ((pk_j^i)_{i\in[\ell],j\in\{0,1\}}, crs)$
    $sk := (\alpha_i, sk_{\alpha_i}^i)_{i\in[\ell]}$
    Return $(pk, sk)$
```

```
$\mathsf{Enc}''(pk, m)$ :
    Parse $m = m_1 \| \cdots \| m_\ell$
    $(r_j^1)_{j\in\{0,1\}}, \cdots, (r_j^\ell)_{j\in\{0,1\}} \leftarrow \mathcal{R}_\Pi^2$
    $(c_0^i)_{i\in[\ell]} \leftarrow (\mathsf{Enc}(pk_0^i, m_i; r_0^i))_{i\in[\ell]}$
    $(c_1^i)_{i\in[\ell]} \leftarrow (\mathsf{Enc}(pk_1^i, m_i; r_1^i))_{i\in[\ell]}$
    $x := (pk_j^i, c_j^i)_{i\in[\ell],j\in\{0,1\}}$
    $w := (m_i, r_j^i)_{i\in[\ell],j\in\{0,1\}}$
    $\pi \leftarrow \mathsf{Prove}(crs, x, w)$
    Return $c := ((c_j^i)_{i\in[\ell],j\in\{0,1\}}, \pi)$
```

```
$\mathsf{Dec}''(pk, sk, c)$ :
    $x := (pk_j^i, c_j^i)_{i\in[\ell],j\in\{0,1\}}$
    If $\mathsf{Verify}(crs, x, \pi) = 1$ then
        $(m_i)_{i\in[\ell]}$
            $\leftarrow (\mathsf{Dec}(pk_{\alpha_i}^i, sk_{\alpha_i}^i, c_{\alpha_i}^i))_{i\in[\ell]}$,
    else Return $\perp$
    If $\exists i$ s.t. $m_i = \perp$ then
        Return $\perp$,
    else Return $m := m_1 \| \cdots \| m_\ell$
```

```
$\mathsf{FKG}''(1^\lambda)$ :
    $\alpha_1, \cdots, \alpha_\ell \leftarrow \{0,1\}$
    $(pk_0^i, sk_0^i)_{i\in[\ell]} \leftarrow (\mathsf{KG}(1^\lambda))_{i\in[\ell]}$
    $(pk_1^i, sk_1^i)_{i\in[\ell]} \leftarrow (\mathsf{KG}(1^\lambda))_{i\in[\ell]}$
    $(crs, tk) \leftarrow \mathsf{SimCRS}(1^\lambda)$
    $pk := ((pk_j^i)_{i\in[\ell],j\in\{0,1\}}, crs)$
    $td := ((\alpha_i, sk_{\alpha_i}^i)_{i\in[\ell]}, tk)$
    Return $(pk, td)$
```

```
$\mathsf{Fake}''(pk, td)$ :
    $(c_{\alpha_i}^i)_{i\in[\ell]} \leftarrow (\mathsf{Enc}(pk_{\alpha_i}^i, 0))_{i\in[\ell]}$
    $(c_{1\oplus\alpha_i}^i)_{i\in[\ell]} \leftarrow (\mathsf{Enc}(pk_{1\oplus\alpha_i}^i, 1))_{i\in[\ell]}$
    $\pi \leftarrow \mathsf{SimPrv}(tk, (pk_j^i, c_j^i)_{i\in[\ell],j\in\{0,1\}})$
    Return $\widetilde{c} := ((c_j^i)_{i\in[\ell],j\in\{0,1\}}, \pi)$
```

```
$\mathsf{Open}''(pk, td, \widetilde{c}, m)$ :
    Return $\widetilde{sk} := (\alpha_i \oplus m_i, sk_{\alpha_i\oplus m_i})_{i\in[\ell]}$
```

```
$\mathsf{FDec}''(pk, td, c)$ :
    $x := (pk_j^i, c_j^i)_{i\in[\ell],j\in\{0,1\}}$
    If $\mathsf{Verify}(crs, x, \pi) = 1$ then
        $(m_i)_{i\in[\ell]}$
            $\leftarrow (\mathsf{Dec}(pk_0^i, sk_0^i, c_0^i))_{i\in[\ell]}$,
    else Return $\perp$
    If $\exists i$ s.t. $m_i = \perp$ then
        Return $\perp$,
    else Return $m := m_1 \| \cdots \| m_\ell$
```

Fig. 4. Generic construction of RNC-CCA secure RNCE Π'' with the plaintext space $\{0,1\}^\ell$.

A Generic Construction of RNC-CCA Secure RNCE with a Multi-bit Plaintext Space

Here, using an IND-CPA secure PKE scheme and an OTSS-NIZK, we show our generic construction of an RNC-CCA secure RNCE scheme with the plaintext space $\{0,1\}^\ell$, where $\ell = \ell(\lambda) > 0$ is a polynomial. Let $\Pi = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ be a PKE scheme with the plaintext space $\{0,1\}$ and $\mathcal{R}_\Pi$ be the randomness space for the encryption algorithm Enc. Let $\Phi = (\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{SimCRS}, \mathsf{SimPrv})$ be a non-interactive proof system for L''_{eq}, where

$$L''_{eq} := \Big\{ (pk_j^i, c_j^i)_{i\in[\ell],j\in\{0,1\}} \Big| \; \exists (m_i, r_j^i)_{i\in[\ell],j\in\{0,1\}} \text{ s.t.} \\ (c_0^i = \mathsf{Enc}(pk_0^i, m_i; r_0^i)) \wedge (c_1^i = \mathsf{Enc}(pk_1^i, m_i; r_1^i)) \text{ for all } i \in [\ell] \Big\}.$$

Then, we construct our scheme $\Pi'' = (\mathsf{KG}'', \mathsf{Enc}'', \mathsf{Dec}'', \mathsf{FKG}'', \mathsf{Fake}'', \mathsf{FDec}'', \mathsf{Open}'')$ as described in Fig. 4. The following theorem holds.

Theorem 5. *If Π is an IND-CPA secure PKE scheme and Φ is an OTSS-NIZK, then Π'' is RNC-CCA secure.*

The proof of Theorem 5 is very similar to the proof of Theorem 2, and thus we omit it here.

B On the "+1"-DDH Assumption

In this section, we show that the DDH assumption implies the "+1"-DDH assumption. In the following, we let $p = \Omega(2^\lambda)$ be a prime number, $\mathbb{G}$ be a multiplicative cyclic group of order p, and $\mathbb{Z}_p$ be the set of integers modulo p. First of all, we recall the definition of the standard DDH assumption.

Definition 11 (DDH assumption). *We say that the DDH assumption holds in $\mathbb{G}$ if for any PPT adversary $\mathcal{A}$,*

$$\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G},\mathcal{A}}(\lambda) := |\Pr[g \leftarrow \mathbb{G}; a \leftarrow \mathbb{Z}_p^*; b \leftarrow \mathbb{Z}_p : \mathcal{A}(g, g^a, g^b, g^{ab}) = 1] \\ - \Pr[g \leftarrow \mathbb{G}; a \leftarrow \mathbb{Z}_p^*; b, c \leftarrow \mathbb{Z}_p : \mathcal{A}(g, g^a, g^b, g^c) = 1]| = \mathsf{negl}(\lambda).$$

We show the following proposition.

Proposition 1. *If the DDH assumption holds in $\mathbb{G}$, then the "+1"-DDH assumption holds in $\mathbb{G}$.*

Proof of Proposition 1. Let $\mathcal{A}$ be any PPT adversary of the "+1"-DDH assumption in $\mathbb{G}$. We let $g \leftarrow \mathbb{G}$, $a \leftarrow \mathbb{Z}_p^*$, and $b, c \leftarrow \mathbb{Z}_p$. We introduce the following experiments $\{\mathsf{Exp}_i\}_{i=0}^2$.

Exp_0 : Exp_0 is an experiment that $\mathcal{A}$ gets a DH tuple (g, g^a, g^b, g^{ab}).
Exp_1 : Exp_1 is an experiment that $\mathcal{A}$ gets a random tuple (g, g^a, g^b, g^c).
Exp_2 : Exp_2 is an experiment that $\mathcal{A}$ gets a DH+1 tuple (g, g^a, g^b, g^{ab+1}).

We let p_i be the probability that $\mathcal{A}$ outputs 1 in Exp_i for all $i \in [0, 2]$. Then, we have $\mathsf{Adv}^{\text{+1-ddh}}_{\mathbb{G},\mathcal{A}}(\lambda) = |p_0 - p_2| \leq |p_0 - p_1| + |p_1 - p_2|$. Note that $|p_0 - p_1| = \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G},\mathcal{A}}(\lambda)$ holds due to the definition of the DDH assumption.

It remains to see that there exists a PPT adversary $\mathcal{B}$ such that $|p_1 - p_2| = \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G},\mathcal{B}}(\lambda)$, using the adversary $\mathcal{A}$. Concretely, when $\mathcal{B}$ receives a tuple (g_1, g_2, g_3, g_4), $\mathcal{B}$ computes $(g_1, g_2, g_3, g_1 \cdot g_4)$ and gives it to $\mathcal{A}$. When $\mathcal{A}$ outputs a bit and terminates, $\mathcal{B}$ returns it to the experiment and terminates. Here, if $\mathcal{B}$ receives a random tuple (g, g^a, g^b, g^c) from its experiment, we can see that $\mathcal{B}$ computes a random tuple (g, g^a, g^b, g^{c+1}) and perfectly simulates Exp_1 for $\mathcal{A}$. (Note that $c + 1$ is also distributed uniformly at random in $\mathbb{Z}_p$.) On the other hand, if $\mathcal{B}$ receives a DH tuple (g, g^a, g^b, g^{ab}) from its experiment, we can see that $\mathcal{B}$ computes a tuple (g, g^a, g^b, g^{ab+1}) and perfectly simulates Exp_2 for $\mathcal{A}$. Therefore, it holds that $|p_1 - p_2| = \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G},\mathcal{B}}(\lambda)$. $\square$ **(Proposition 1)**