박 사 학 위 논 문
Ph.D. Dissertation

양자 내성을 가지는 래티스 문제를 이용한 키 재사용이 가능한 동적 그룹 키 교환 방식의 설계 및 분석

Key-reusable Dynamic Group Key Exchange from Lattice with Quantum Resistance

2020

최 락 용 (崔 洛 龍 Choi, Rakyong)

한 국 과 학 기 술 원

Korea Advanced Institute of Science and Technology

박 사 학 위 논 문

양자 내성을 가지는 래티스 문제를 이용한 키 재사용이 가능한 동적 그룹 키 교환 방식의 설계 및 분석

2020

최 락 용

한 국 과 학 기 술 원

전산학부

양자 내성을 가지는 래티스 문제를 이용한 키 재사용이 가능한 동적 그룹 키 교환 방식의 설계 및 분석

최 락 용

위 논문은 한국과학기술원 박사학위논문으로
학위논문 심사위원회의 심사를 통과하였음

2020년 06월 12일

심사위원장	김 광 조	(인)
심 사 위 원	한 상 근	(인)
심 사 위 원	이 주 영	(인)
심 사 위 원	권 대 성	(인)
심 사 위 원	Jintai Ding	(인)

Key-reusable Dynamic Group Key Exchange from Lattice with Quantum Resistance

Rakyong Choi

Advisor: Kwangjo Kim

A dissertation submitted to the faculty of
Korea Advanced Institute of Science and Technology in
partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer Science

Daejeon, Korea
June 12, 2020

Approved by

Kwangjo Kim
Professor of School of Computing

The study was conducted in accordance with Code of Research Ethics[1].

[1] Declaration of Ethical Conduct in Research: I, as a graduate student of Korea Advanced Institute of Science and Technology, hereby declare that I have not committed any act that may damage the credibility of my research. This includes, but is not limited to, falsification, thesis written by someone else, distortion of research findings, and plagiarism. I confirm that my thesis contains honest conclusions based on my own careful research under the guidance of my advisor.

DCS 최락용. 양자 내성을 가지는 래티스 문제를 이용한 키 재사용이 가능한 동적 그룹 키 교환 방식의 설계 및 분석. 전산학부 . 2020년. 76+v 쪽. 지도교수: 김광조. (영문 논문)
Rakyong Choi. Key-reusable Dynamic Group Key Exchange from Lattice with Quantum Resistance. School of Computing . 2020. 76+v pages. Advisor: Kwangjo Kim. (Text in English)

초 록

최근 키 재사용 공격에 의해 Ding Key Exchange, NewHope 뿐 아니라 Peikert의 키 조정 메커니즘을 이용한 래티스 기반 키 교환 방식에 대한 다양한 공격 논문이 제안되었다. 그러나 기존의 키 재사용 공격의 경우 양자간 키 교환 방식에 대한 공격만을 고려하여 그룹 키 교환 프로토콜에 대해서는 같은 가정으로 적용하기가 어려운 단점이 있다. 한편, 기존의 래티스 기반 그룹 키 교환 방식은 정적 환경에서 제안되었으며 동적 환경을 고려하지 않았다.

본 논문에서는 기존 래티스 기반 그룹 키 교환 방식에 대한 새로운 키 재사용 공격 방식인 {one, two}-neighbour 공격에 대해서 소개하고 2012년 Ding 등이 제안한 그룹 키 교환 방식, 2019년 Apon 등이 제안한 그룹 키 교환 방식이 이러한 키 재사용 공격으로 공격 가능하다는 것을 제시하였다. 이 때 공격자는 파티 P_{N-1}의 비밀키를 얻기 위해 파티 P_{N-2}과 P_{N-3}을 조작 가능하다고 가정한다.

또한 본 논문에서는 래티스 연산에서도 동적 환경에서 그룹 간의 키 교환이 가능한 다양한 그룹 키 교환 방식 설계 모델을 제시 및 구현하였으며, 키 재사용 공격에 대한 대응책으로 Pasteurization 기법을 이용하여 각 참여자의 공개키를 다시 랜덤화 시켜서 키 재사용이 가능한 동적 그룹 키 교환 방식의 설계를 Dutta-Barua 프로토콜을 이용하여 동적 환경에서 RLWE 문제를 기반으로 설계 및 랜덤 오라클 모델 상에서 증명하였다.

핵 심 낱 말 키 교환 프로토콜, 격자 기반 암호, 양자 내성 암호, 키 재사용 공격

Abstract

Recently, several key reuse attacks against Ding Key Exchange, NewHope and other lattice-based key exchange schemes using Peikert's key reconciliation mechanism were suggested. But all known key reuse attacks are designed for two-party setting instead of group key exchange. On the other hand, all previous known lattice-based group key exchanges are designed for static setting.

This paper is organized in three folds. We present the first key reuse attack called {one, two}-neighbour attack against lattice-based group key exchanges, namely Ding *et al.*'s group key exchange scheme in 2012 and Apon *et al.*'s group key exchange scheme from PQCrypto 2019. We consider that the adversary manipulates one or two neighbour parties P_{N-2} (and P_{N-3} for two-neighbour attack) of the last party P_{N-1} to recover the secret key of the last party P_{N-1} among N parties.

We also suggest several constructions of dynamic group key exchange protocols that could be instantiated by lattice. As a countermeasure of our attack, we design the first key-reusable group key exchange based on lattice. where GKE protocol Π_{GKE} is key-reusable if the protocol participants of Π_{GKE} can re-use their public key. By adopting existing pasteurization technique for two-party key exchange from lattice, our protocol becomes resistant to known key reuse attacks. We give a rigorous proof of our protocol in the random oracle model. Our underlying dynamic group key exchange protocol is the modification of Dutta-Barua protocol in RLWE setting.

Keywords Key exchange protocol, dynamic group key exchange, lattice-based cryptography, post-quantum cryptography, key-reuse attack

Contents

Contents i
List of Tables iv
List of Figures v

Chapter 1. Introduction **1**
1.1 Our Contributions 2
1.2 Outline of the Paper 4

Chapter 2. Preliminaries **5**
2.1 Notation 5
2.2 Discrete Gaussian Distribution 5
2.3 Ring Learning with Errors 5
2.4 Rényi Divergence 6
2.5 Generic Key Reconciliation Algorithm 7
2.6 Pasteurization Technique 8

Chapter 3. Related Work **10**
3.1 Constant-round Group Key Exchange 10
3.2 Security Model of Group Key Exchange 10
3.3 Lattice-based Key Exchange 11
3.4 Key Reuse Attacks 11

Chapter 4. {One, two}-neighbour Attack against known Lattice-based GKE **13**
4.1 Ding *et al.*'s Group Key Exchange 13
4.1.1 One-neighbour Attack 13
4.2 Apon *et al.*'s Group Key Exchange 14
4.2.1 Two-neighbour Attack 16

Chapter 5. Novel Construction of GKE Protocols **18**
5.1 Choi *et al.*'s Tree-based GKE 18
5.1.1 Basic Construction 18
5.1.2 Dynamic TGKE 20
5.1.3 Security Analysis 22
5.2 Homomorphic Encryption based GKE 23

5.2.1 Homomorphic Encryption 23
5.2.2 Basic Operations from Graph Theory 24
5.2.3 Construction in Static Setting 24
5.2.4 Parcel-S Protocol 26
5.2.5 Parcel-D Protocol 27

Chapter 6. Key-reusable GKE in Static Setting 32
6.1 Construction of Key-reusable GKE 32
6.2 Security Analysis 32
6.2.1 Correctness Proof 32
6.2.2 Security Proof 35

Chapter 7. Dynamic GKE from RLWE 43
7.1 Security Model 43
7.2 Unauthenticated Group Key Exchange 45
7.3 Authenticated Group Key Exchange 45
7.4 Dynamic Group Key Exchange 45
7.4.1 Join Algorithm 45
7.4.2 Leave Algorithm 49
7.5 Security Analysis 49
7.5.1 Correctness Proof 49
7.5.2 Security Proof 50

Chapter 8. Implementation 60
8.1 Instantiation 60
8.1.1 Restrictions on the Parameters 60
8.1.2 Parameter Selection 60
8.2 Network Topology 61
8.3 Our Tests 61
8.3.1 Test 0: Verifying the Success of Key Exchange 62
8.3.2 Test 1: Performance Check on Components 62
8.3.3 Test 2: Performance Check on Dynamic Operations . . 64

Chapter 9. Key-resuable Dynamic GKE from RLWE 65
9.1 Tweaked Two-neighbour Attack 65
9.2 Security-enhanced Dynamic GKE from RLWE 66

Chapter 10. Comparison with Other Protocols 68

Chapter 11. Concluding Remarks 70

Bibliography 71

List of Tables

8.1 Parameter choice 61

8.2 Average runtime and cycles of each operation 63

8.3 Average runtime and cycles of each function 63

8.4 Performance evaluation of U.Join and U.Leave 64

10.1 Comparison with other lattice-based (authenticated) GKE protocols 68

List of Figures

5.1 TGKE protocol in static setting 19

5.2 Join example 20

5.3 Remove examples for Leave and TreeRefresh 21

5.4 HE-KE protocol 24

5.5 Parcel-S protocol 26

5.6 Join algorithm in Parcel-D protocol 29

5.7 Leave algorithm in Parcel-D protocol 29

5.8 Merge algorithm in Parcel-D protocol 30

8.1 Group secret key of STUG (N=4) 62

8.2 U.Join procedure in DRUG (N=4 $\rightarrow$ N=5) 62

8.3 U.Leave procedure in DRUG (N=5 $\rightarrow$ N=4) 62

Chapter 1. Introduction

These days, network topology is becoming more and more complicated such as group chatting in instant messaging applications, file sharing between multiple parties, *etc.* Hence, secure communication between multiple parties is required to keep the confidentiality of their messages.

Key establishment is a pre-determined protocol where two or more parties make a shared secret for subsequent cryptographic use [48]. This can be subdivided into key transport and key agreement protocols. Key transport protocol enables one party to create a secret value and securely transfer it to other parties but key agreement protocol derives the shared secret of two or more parties as a function of information contributed by each party, so that no party can estimate the result.

A group key exchange (GKE) protocol provides a set of specific cryptographic procedures that establishes a group secret key that is derived from group members. Compared to secret sharing scheme, GKE protocol allows distinct keys for distinct groups while secret sharing scheme starts with a secret and divides it into pieces called shares. Information exchanged between parties in GKE protocol is non-secret and transferred over open channels, while shares are distributed secretly. Each party individually computes the session key in GKE protocol but pooling shares can be reconstructed among K participants of all N parties where $K \leq N$ for secret sharing scheme [48]. In general, GKE protocol consists of three phases: key generation phase, intermediate value broadcasting phase, and key computation phase.

Authenticated key exchange protocol authenticates the identity of parties in the protocol to prevent any attacks like the man-in-the-middle attack even in the presence of active adversaries who controls the underlying communication by eavesdropping and modifying transmitted messages during communication over a network.

To construct (authenticated) GKE protocol, we need to define the computational power of a group member. If all group members are assumed to have equal power, we need to handle some disputes that happen by the malicious actions of group members. To overcome this issue, a trusted authority (TA) must be provided as a communication infrastructure, but it is quite costly.

As the membership status of a party changes or remains the same, we say GKE protocol is in either a static or a dynamic setting. The static setting keeps membership status for a long time while the dynamic setting provides frequent membership status changes in a short time, *i.e.*, any member can join or leave the protocol at any time in a dynamic setting. It is suitable to deploy a TA in a protocol in a static setting, but hard to deploy a TA in a dynamic setting such as resource-constrained environments like IoT. There have been numerous publications on GKE protocols [4,9,13–17,28,38,40,41,45,49,51,52,58–60,64].

It is well-known that quantum algorithms like Shor's algorithm [57] can solve number-theoretic problems like integer factorization and discrete logarithm problems including their elliptic curve versions in polynomial time so that quantum adversaries can break the cryptographic protocol like RSA, Diffie-Hellman key exchange, or ECDSA. As quantum computer becomes realistic in the near future in a best scenario, the National Institute of Standards and Technology (NIST) has been selecting standard post-quantum cryptographic algorithms like key exchange, encryption, and signature schemes which are the most critical functions that require public key cryptosystem. It would be ideal to have drop-in replacements for quantum-vulnerable algorithms like RSA, ElGamal, and Diffie-Helman key exchange [6]. There were a lot of submissions about lattice-based key exchanges / key encapsulation mechanisms including Ding Key Exchange, NewHope, Frodo, Kyber and Round5 [3, 5, 10, 11, 26]. Among them, lattice-based key exchanges face an issue on key reuse.

The history of key reuse attack against lattice-based key exchange protocol started along with the development of lattice-based key exchanges. Since Fluhrer [30] presented an attack to leak the information of the secret key of one participant from the key reconciliation function, there have been several key reuse attacks against Ding Key Exchange, NewHope, and other lattice-based key exchange protocols using Peikert's key reconciliation mechanism [7,23,25,46,53]. But to the best of our knowledge, all the previous results are limited on two-party key exchange protocols.

There are a few publications on post-quantum GKE protocols. Ding *et al.* [26] constructed the first lattice-based GKE protocol and Yang *et al.* [64] and Apon *et al.* [4] independently suggested *constant-round* lattice-based GKE protocols, where *constant-round* means that the number of phases for each party does not change regardless of the number of group members.

While Ding *et al.* and Apon *et al.*'s protocols do not rely on a TA to agree on a group secret key among the group members, Yang *et al.*'s protocol requires the role of a TA that calculates the group secret key by getting the ephemeral key of each party and sends it to each party.

However, to the best of our knowledge, there exists no post-quantum dynamic GKE in the open literature regardless of the existence of TA and there is no lattice-based GKE protocol resistant to key reuse attack, either. Our goal is to design a novel post-quantum constant-round dynamic GKE protocol from Ring Learning with Errors (RLWE) resistant to key reuse attacks, without a TA.

1.1 Our Contributions

In this paper, we give three main contributions.

First, we give a modified key reuse attack called as {*one, two*}*-neighbour attack* to check the vulnerability of Ding *et al.*'s GKE protocol and Apon *et al.*'s protocol (ADGK19) [4] when a specific key reconciliation mechanism is embedded. We describe briefly how to recover the secret key of the last party P_{N-1} among N parties in polynomial time.

We assume that the adversary is impersonating as one or two previous neighbouring parties P_{N-3} and P_{N-2} of the last party P_{N-1} while the secret keys of other parties in the group are fixed. This can be interpreted as a kind of insider's attack. Then, with similar approach of Ding *et al.*'s key reuse attack [23], we can obtain the secret key of the last party P_{N-1} by checking the number of signal changes of the output of key reconciliation function.

Secondly, as a counter-example of our attack, we design the first key-reusable group key exchange based on lattice where GKE protocol Π_{GKE} is *key-reusable* if the protocol participants of Π_{GKE} can re-use their public key (as well as their secret key). We adopt the existing pasteurization technique from Ding *et al.* [24] into ADGK19 so that our new scheme becomes resistant to our two-neighbour attack. We focus on constructing an unauthenticated scheme like ADGK19 since known techniques such as Katz-Yung compiler [38] can be applied to obtain an authenticated one.

Finally, we give a constant-round key-reusable dynamic GKE protocol based on the hardness of RLWE assumption [47] where a party can join or leave the group. We extend two-round Dutta-Barua protocol [28] into RLWE setting.

Given a group $\mathbb{G}$ of prime order q and a generator $g \in \mathbb{G}$, we briefly describe Burmester-Desmedt and Dutta-Barua protocols as below:

1. **(Round 1)** Each party P_i chooses "uniform" value $r_i \in \mathbb{Z}_q$ and broadcasts $z_i = g^{r_i}$ to all other parties.

2. **(Round 2)** Each party P_i broadcasts $X_i = (z_{i+1}/z_{i-1})^{r_i}$ to all other parties.

3. **(Key Computation)**

   - Burmester-Desmedt protocol: $b_i = {z_{i-1}}^{Nr_i} \cdot {X_i}^{N-1} \cdot {X_{i+1}}^{N-2} \cdots X_{i+N-2}$.
   - Dutta-Barua protocol: Each party P_i calculate $Y_{i+1} = X_{i+1}{z_{i+1}}^{r_i}$ and $Y_{i+j} = X_{i+j}Y_{i+(j-1)}$ for $j = 2$ to $N-1$, then $b_i = \prod_{j=0}^{N-1} Y_{i+j}$.

Since Dutta-Barua protocol is a modification from Burmester-Desmedt protocol [16, 17, 38] used in Apon *et al.*'s recent work [4], our unauthenticated GKE protocol in static setting is somewhat similar to Apon *et al.*'s protocol.

We apply this relationship into Apon *et al.*'s protocol. Given a ring R_q and a ring element $a \leftarrow R_q$, we sketch our unauthenticated GKE protocol compared to Apon *et al.*'s as below:

1. **(Round 1)** Each party P_i chooses 'small' secret value $s_i \in R_q$ and 'small' noise $e_i \in R_q$ and broadcasts $z_i = as_i + e_i$ to all other parties.

2. **(Round 2)** Each party P_i chooses another 'small' noise $e'_i \in R_q$ and broadcasts $X_i = (z_{i+1} - z_{i-1})\, s_i + e'_i$ to all other parties.

3. **(Key Computation)**

   - Apon *et al.*'s protocol: $b_i = Nz_{i-1}s_i + (N-1)X_i + (N-2)X_{i+1} + \cdots + X_{i+N-2}$.
   - Our protocol: Each party P_i calculate $Y_i = X_i + z_{i-1}s_i$ and $Y_{i+j} = X_{i+j} + Y_{i+(j-1)}$ for $j = 1$ to $N-1$, then $b_i = \sum_{j=0}^{N-1} Y_{i+j}$.

Hence, we follow security analysis of Apon *et al.*'s protocol with slight modification in the presence of the passive adversary. We adopt "unpredictability-based" security analysis (*i.e.*, given the transcript, it is infeasible to determine the real session key) instead of "indistinguishability-based" one (*i.e.*, given the transcript, the real session key should be indistinguishable from random) to apply the characteristic of bounded Rényi divergence.

But instead of applying Katz-Yung compiler [38] for authenticated GKE with active adversary, we adopt the security model of Bresson *et al.* [13] to give a full security analysis of the dynamic case. Hence, our authenticated GKE protocol also achieves forward secrecy, almost fully symmetric and being constant-round but we do not require one more round to achieve AKE security, compared to Apon *et al.*'s protocol.

1.2 Outline of the Paper

The rest of this paper is organized as follows. We define the notation and preliminaries in Chapter 2. We review the related work on lattice-based (group) key exchanges and key reuse attacks in Chapter 3. In Chapter 4, our novel {one, two}-neighbour attack on previous lattice-based group key exchanges is described. Then, we introduce a novel construction of GKE protocols with trusted authority in Chapter 5. We give key-reusable group key exchange from lattice in Chapter 6 with a security analysis. In Chapter 7, dynamic group key exchange protocol is suggested and the implementation is given in Chapter 8. In Chapter 9, we further claim that our dynamic GKE is not secure against the type of key reuse attacks and give a key-reusable dynamic GKE protocol from RLWE with constant-round property. Finally, we give a comparison table and concluding remarks in Chapters 10 and 11, respectively.

Chapter 2. Preliminaries

2.1 Notation

Let $\mathbb{Z}$ be the set of integers and $[N] = \{0, 1, 2, \cdots N-1\}$. $\lceil x \rceil$ is a ceiling function that maps x to the least integer greater than or equal to x. For a set A, $x \leftarrow A$ denotes a uniformly random sampling of $x \in A$.

Let $\chi(E)$ stand for a probability of a set E of events occurs under a distribution χ. We set $\text{Supp}(\chi) = \{\epsilon : \chi(\epsilon) \neq 0\}$ and let $\bar{E}$ be the complement of an event set E. χ_α denotes the distribution with standard deviation α.

A function μ is negligible if and only if for all $c \in \mathbb{N}$, there exists $n_c \in \mathbb{N}$ such that $\mu(n) < n^{-c}$ for all $n \geq n_c$. Given a ring element p, $p[i]$ denotes the i-th coefficient of p. We use $\log(x)$ and $\exp(x)$ to denote $\log_2(x)$ and e^x, respectively. We denote P_i and $P[0, 1, \cdots, k] = \{P_0, P_1, \cdots, P_k\}$ for i-th party and an array of parties of a protocol, respectively.

2.2 Discrete Gaussian Distribution

For $\sigma \in \mathbb{R}$, denote $\rho_\sigma(\mathbf{x}) = exp\left(-\frac{\pi \cdot \|\mathbf{x}\|^2}{\sigma^2}\right)$ as the Gaussian function scaled by σ where $\mathbf{x} \in \mathbb{R}^m$ and let $\rho_\sigma(\mathbb{Z}^m) = \sum_{\mathbf{x} \in \mathbb{Z}^m} \rho_\sigma(\mathbf{x})$. Define the *$m$-dimensional discrete Gaussian distribution* $D_{\mathbb{Z}^m,\sigma}(\mathbf{x}) = \frac{\rho_\sigma(\mathbf{x})}{\rho_\sigma(\mathbb{Z}^m)}$ for $\mathbf{x} \in \mathbb{Z}^m$. To sample a polynomial of degree n, (1) take $m = 1$ and sample each coefficient n times from $D_{\mathbb{Z},\sigma}(x)$ or (2) take $m = n$ and sample coefficient vectors from $D_{\mathbb{Z}^n,}(\mathbf{x})$. We write $D^m_{\mathbb{Z},\sigma}$ to denote sampling m times from $D_{\mathbb{Z},\sigma}$.s We can restrict the sample domain to $\mathbb{Z}_q$, in other words, i-th coordinate x_i of the input $\mathbf{x}$ is in the range $-\frac{q}{2} < x_i \leq \frac{q}{2}$. Figure **??** shows a 2-dimensional discrete Gaussian distribution with standard deviation $\sigma = 2\sqrt{8}$

Remark that the ring-Learning with Errors (ring-LWE) problem is still hard if the secret $s \in R_q$ is sampled from an error distribution instead of sampled from a uniform distribution of R_q. [**?**, **?**, 47]

But the discrete Gaussian distribution has some issues on its implementation and efficiency.

2.3 Ring Learning with Errors

RLWE problem [47] states that it is hard to find a secret value $s \in R_q$ from l independent samples $(a_i, a_i s + e_i)$ in $R_q \times R_q$ where R_q is a ring.

But, decisional version of RLWE problem is commonly used in cryptographic primitives as a building block. Decisional version of RLWE states that it is hard to distinguish whether a sample $(a, b) \in R_q \times R_q$

is from the RLWE distribution of the form $(a, as+e) \in R_q \times R_q$ for uniform $a \leftarrow R_q$, secret key $s \leftarrow \chi_s$ and error $e \leftarrow \chi_e$ or it is sampled from uniform distribution of $R_q \times R_q$.

We let $\mathsf{Adv}^{RLWE}_{n,q,\chi_s,\chi_e,l}(\mathcal{B})$ denote the advantage of algorithm $\mathcal{B}$ in distinguishing RLWE distribution and uniform distribution of $R_q \times R_q$. In addition, let $\mathsf{Adv}^{RLWE}_{n,q,\chi_s,\chi_e,l}(t)$ be the maximum advantage of any algorithm running in time t. If $\chi = \chi_s = \chi_e$, we write $\mathsf{Adv}^{RLWE}_{n,q,\chi,l}$ for simplicity. For the remaining paper, we set R, q, χ and l as follows:

1. $R = \mathbb{Z}[x]/(f(x))$ is a polynomial ring with an irreducible polynomial $f(x) = x^n + 1$ where x is the indeterminate and n is a power of 2.

2. q is a positive integer modulus defining a quotient ring $R_q = R/qR = \mathbb{Z}_q[x]/(f(x))$. We consider the case where q is prime with $q \equiv 1 \mod 2n$.

3. $\chi = (\chi_s, \chi_e)$ is a pair of noise distributions over R_q where χ_s is a secret-key distribution and χ_e is an error distribution that are concentrated on small elements. We choose Gaussian distribution $D_{\mathbb{Z}_q,\sigma}$ for both χ_s and χ_e.

4. l is the number of samples given to the adversary.

2.4 Rényi Divergence

For two discrete probability distributions P and Q with $\mathrm{Supp}(P) \subseteq \mathrm{Supp}(Q)$, their Rényi divergence is defined as

$$\mathrm{RD}_2(P||Q) = \sum_{x \in \mathrm{Supp}(P)} \frac{P(x)^2}{Q(x)}.$$

Rényi divergence measures closeness of two probability distributions and it is widely used in cryptographic designs [8, 43, 47, 62]. We introduce some important results related to Rényi divergence that can be used in our protocol.

Proposition 1. For discrete distributions P and Q with $\mathrm{Supp}(P) \subseteq \mathrm{Supp}(Q)$, let $E \subseteq \mathrm{Supp}(Q)$ be an arbitrary event. We have

$$Q(E) \geq P(E)^2 / RD_2(P\|Q)$$

Roughly, the proposition says that if $\mathrm{RD}_2(P\|Q)$ is bounded by some polynomial, then any event set E that occurs with negligible probability $Q(E)$ under Q also occurs with negligible probability $P(E)$ under P. With this proposition, **Theorem 1** claims that Rényi divergence between the 1-dimensional discrete Gaussian distribution $D_{\mathbb{Z}_q,\sigma}$ (which centered at the origin) and other distribution $e + D_{\mathbb{Z}_q,\sigma}$ (which centered at a point near the origin, *i.e.*, e is a small value) is bounded.

Theorem 1 ([5]). Let $m, q, \lambda \in \mathbb{Z}$ and fix a bound $\beta_{\textsf{Rényi}}$ and σ with $\beta_{\textsf{Rényi}} < \sigma < q$. Let $e \in \mathbb{Z}$ satisfying $|e| \leq \beta_{\textsf{Rényi}}$. Then

$$\mathrm{RD}_2((e + D_{\mathbb{Z}_q,\sigma})^m || D^m_{\mathbb{Z}_q,\sigma}) \leq \exp(2\pi m(\beta_{\textsf{Rényi}}/\sigma)^2)$$

where χ^m means that we sample m times independently from the distribution χ. Moreover, if we take $\sigma = \Omega(\beta_{\textsf{Rényi}}\sqrt{m/\log\lambda})$ with security parameter λ, we deduce that $\mathrm{RD}_2((e+D_{\mathbb{Z}_q,\sigma})^m||D^m_{\mathbb{Z}_q,\sigma}) \leq \mathrm{poly}(\lambda)$.

2.5 Generic Key Reconciliation Algorithm

The concept of key reconciliation was first introduced by Ding *et al.* [26] to treat errors between two approximately agreed ring elements in R_q. Then, it has been used in several works on lattice-based two-party key exchange protocols [3, 10, 12, 50, 65]. From [4], we describe a generic key reconciliation algorithm which is performed between two-party in one-round.

A key reconciliation $\textsf{KeyRec} = (\textsf{recMsg}, \textsf{recKey})$ allows two parties to derive the same key from close ring elements. One of two participants runs the first algorithm $\textsf{recMsg}$ taking the security parameter λ and a ring element $b \in R_q$ and outputs $\textsf{rec}$ and a key $k \in \{0,1\}^\lambda$. The other participant runs $\textsf{recKey}$ taking $\textsf{rec}$ and a ring element $b' \in R_q$ and outputs a key value $k' \in \{0,1\}^\lambda$.

We claim that a key exchange protocol works correctly when two participants have the same key (i.e. $k = k'$). To hold this equality, b and b' have to be sufficiently close. Especially, if $b - b'$ are bounded by some value $\beta_{\textsf{Rec}}$ and two participants run $\textsf{KeyRec}$ algorithm, then they share the same key except with negligible probability.

Security is defined by the indistinguishability between a key k, result of key exchange and uniformly random value. Formally, an adversary $\mathcal{A}$ should be computationally infeasible to distinguish two distribution,

$$\{(\textsf{rec}, k) : b \leftarrow R_q; (\textsf{rec}, k) \leftarrow \textsf{recMsg}(1^\lambda, b)\}_{\lambda\in\mathbb{N}} \text{ and}$$

$$\{(\textsf{rec}, k') : b \leftarrow R_q; (\textsf{rec}, k) \leftarrow \textsf{recMsg}(1^\lambda, b); k' \leftarrow \{0,1\}^\lambda\}_{\lambda\in\mathbb{N}}$$

For a fixed value of λ, we denote the advantage of adversary $\mathcal{A}$ in distinguishing these two distributions by $\textsf{Adv}_{\textsf{KeyRec}}(\mathcal{A})$, and the maximum advantage of any such adversary running in time t by $\textsf{Adv}_{\textsf{KeyRec}}(t)$.

Ding's Reconciliation Mechanism

Ding *et al.* [26] proposed the first two-party key exchange protocol based on lattice as **Protocol 1**.

To make their protocol to get the correctness, they consider three types of functions for reconciliation mechanism.

For a ring element $\boldsymbol{k}$, they define *hint functions* $\textsf{sgn}_0(x)$ and $\textsf{sgn}_1(x)$ from $\mathbb{Z}_q^n$ to $\{0,1\}^n$ as $\textsf{sgn}_b(\boldsymbol{k}) = (\textsf{sgn}_b(\boldsymbol{k}[0]), \textsf{sgn}_b(\boldsymbol{k}[1]), \cdots, \textsf{sgn}_b(\boldsymbol{k}[n-1]))$ for $b \in \{0,1\}$ where

Protocol 1: Ding Key Exchange

Alice		**Bob**
$\boldsymbol{s}_A, \boldsymbol{e}_A \leftarrow R_q$		
$\boldsymbol{p}_A = \boldsymbol{a}\boldsymbol{s}_A + 2\boldsymbol{e}_A$	$\xrightarrow{\boldsymbol{p}_A}$	$\boldsymbol{s}_B, \boldsymbol{e}_B \leftarrow R_q$
		$\boldsymbol{k}_B = \boldsymbol{p}_A\boldsymbol{s}_B + 2\boldsymbol{e}'_B$
		$\boldsymbol{p}_B = \boldsymbol{a} \cdot \boldsymbol{s}_B + 2\boldsymbol{e}_B$
	$\xleftarrow{(\boldsymbol{p}_B,\ \mathsf{rec})}$	$\mathsf{rec} \leftarrow S(\boldsymbol{k}_B)$
$\boldsymbol{e}'_A \leftarrow R_q$		
$\boldsymbol{k}_A = \boldsymbol{s}_A\boldsymbol{p}_B + 2\boldsymbol{e}'_A$		
$\mathsf{sk}_A \longleftarrow E(\boldsymbol{k}_A, \mathsf{rec})$		$\mathsf{sk}_B \leftarrow E(\boldsymbol{k}_B, \mathsf{rec})$

$$\mathsf{sgn}_0(\boldsymbol{k}[i]) = \begin{cases} 0, \boldsymbol{k}[i] \in \left[-\lfloor \frac{q}{4} \rfloor, \lfloor \frac{q}{4} \rfloor\right] \\ 1,\ \text{otherwise} \end{cases}$$

and

$$\mathsf{sgn}_1(\boldsymbol{k}[i]) = \begin{cases} 0, \boldsymbol{k}[i] \in \left[-\lfloor \frac{q}{4} \rfloor + 1, \lfloor \frac{q}{4} \rfloor + 1\right] \\ 1,\ \text{otherwise.} \end{cases}$$

A *signal function* $S(\boldsymbol{k})$ outputs $\mathsf{rec} = \mathsf{sgn}_b(\boldsymbol{k})$ where b is randomly chosen. For a ring element $\boldsymbol{k}$ and a signal value rec, a *reconciliation function* $E(x, \sigma)$ is a deterministic function with error tolerance δ where $E(\boldsymbol{k}, \mathsf{rec}) = (\boldsymbol{k} + \frac{q-1}{2}\mathsf{rec} \mod q)$.

2.6 Pasteurization Technique

Key reuse attack leaks the information of the secret key of party P_j by the misbehaviour of the other party P_i. More precisely, party P_i sends the public value z_i, which is not a RLWE instance, for key reuse attack. If P_i's value z_j is a RLWE instance, then the value b_j computed by P_j is indistinguishable from a uniformly chosen value and thus, the key reconciliation value of b_j is also indistinguishable from a uniformly chosen value.

In [24], Ding *et al.* suggested an idea called *pasteurization technique* to force the parties involved in key exchange protocol to behave honestly. The idea of this technique is as follows: After receiving z_i from P_i, the party P_j pasteurizes z_i, *i.e.*, P_i computes $z'_i = z_i + a\mathcal{H}(z_i) + 2f_j$, where $\mathcal{H}$ is a random oracle whose output is sampled from χ_α and f_j is sampled from χ_α. If z_i is indeed a RLWE instance, then the pasteurization z'_i is also a RLWE instance, for which P_i knows the secret. However, when z_i is not a RLWE instance, then z'_i looks random to P_i. Thus, the signal of k_j is also random and P_i cannot extract information about P_j's secret key from it. We conclude that party P_i gains nothing if he/she does not follow the protocol.

The pasteurization technique can be seen as the analogue of checking if the exchanged messages are in a group G, in the Diffie-Hellman key exchange, since the technique also enforces honest behaviour of the involved parties.

Chapter 3. Related Work

3.1 Constant-round Group Key Exchange

Burmester and Desmedt [16] proposed the first constant-round GKE protocol (hereinafter, BD94). In BD94, all participants in a protocol are assumed to form a ring topology to generate a group secret key and every group member participates in key generation with equal privilege during the protocol execution. This property is called *contributory*. Just and Vaudenay [36] proposed an authenticated GKE protocol by combining the idea from BD94 and a public key signature scheme. This protocol is more efficient than BD94 from the view of communication bandwidth but requires four-round to generate the group secret key.

A compiler proposed by Katz and Yung (hereinafter, Katz-Yung compiler) [38] can convert any unauthenticated GKE protocol into an authenticated one. They also suggested an authenticated GKE protocol by applying Katz-Yung compiler to BD94.

For dynamic GKE, Kim *et al.* [39] suggested a two-round authenticated GKE protocol for an ad-hoc network, in which no TA is involved. In their protocol, XOR operation is introduced into the generation of the group secret key to reduce the computational cost of each group member. For a dynamic setting, the computation and communication overheads of each group member rely on the number of joining/leaving members rather than relying on the number of previous group members.

Dutta and Barua [28, 29] proposed a two-round authenticated GKE protocol (hereinafter, DB05), which is constructed by combining a variant of BD94 and a signature scheme modified from [38]. For a dynamic setting, the membership addition procedure generates a new group secret key by making a new ring topology with the joining members with the support of the previously agreed group members. For the membership deletion procedure, a new ring topology with the remaining members is formed to run the protocol.

3.2 Security Model of Group Key Exchange

Bresson *et al.* [15] suggested the first formal security model called BCPQ model for authenticated GKE protocols in a static setting. In their paper, they defined AKE security and mutual authentication (MA) security. AKE security guarantees that an active adversary who does not participate in the session cannot distinguish a group secret key from a random number. MA security ensures that only legitimate participants can compute an identical session group secret key.

After that, Katz and Yung [38] revised this model to compile unauthenticated GKE protocol into authenticated GKE protocol. They proved the security of BD94 [16] in the presence of a passive adversary who can only eavesdrop on messages and make a compiler from unauthenticated GKE to authenticated GKE with an active adversary. After that, Katz and Shin [37] proposed another compiler that can transform an implicitly secure authenticated GKE into a secure authenticated GKE resistant to insider attacks, with the universally-composable (UC) model.

For a dynamic setting, Bresson *et al.* [13,14] suggested two formal security models for authenticated GKE protocols depending on the power of corruption and the presence of MA security.

A strong corruption model enables an adversary $\mathcal{A}$ to reveal the long-term secret key as well as the ephemeral keys or internal states of the protocol but a weak corruption model only leaks the long-term secret key of the party while the ephemeral keys or internal states of protocol participants are not corrupted.

3.3 Lattice-based Key Exchange

Ding *et al.* [26] suggested the first lattice-based key exchange protocol in 2012 by modifying Diffie-Hellman key exchange protocol [21] into RLWE setting. Following this research, numerous publications [2–5, 10–12, 18, 19, 22, 31, 33, 44, 50, 55, 56, 61, 64, 65] have looked at the construction and implementation of key exchange protocols based on lattices, but most of them are only designed for two-party key exchange.

For lattice-based GKE protocol, Ding *et al.* [21] suggested the natural extension to GKE protocol (hereinafter, DXL12.G) based on their key exchange protocol using the GKE compiler by Bresson *et al.* [13]. After that, Yang *et al.* [64] proposed the first provably-secure (authenticated) GKE protocol (hereinafter, YMZ15) based on the hardness of LWE/RLWE assumption and security property of secure sketch in the random oracle model. For the secure sketch, TA is necessary and YMZ15 is said to be not contributory.

Recently, Apon *et al.* [4] proposed the first constant-round authenticated GKE protocol (hereinafter, ADGK19) based on the hardness of RLWE assumption, without TA. ADGK19 uses Katz-Yung compiler for authentication and is contributory since they adopt the protocol in [16].

3.4 Key Reuse Attacks

Recent results [7, 23, 25, 30, 46, 53] have noticed that partial/total information of a secret key of the participants can be leaked by the output of the key reconciliation mechanism, in RLWE-based key exchange protocols. The first attack on RLWE-based key exchange with reused keys was described by Fluhrer [30].

Ding *et al.* [23] presented a detailed description on how such an attack is given by analyzing the number of signal changes of each coefficient on Ding Key Exchange as shown in **Algorithm 1**. In [23], when the public value $\boldsymbol{z}_i$ sent by party P_i is not computed honestly, then party P_i can recover information about party P_j's secret key from the behaviour of the signal sent by P_j. Hence, party P_j cannot reuse the same key in several executions of the protocol, which leads to the exposure of its secret key to the other communicating party P_i.

Liu *et al.* [46] proposed a new key reuse attack against NewHope protocol whose key reconciliation function is much more complex and the signal function doesn't change regularly as Ding Key Exchange.

Since then, there are few discussion [24, 32, 63] on how to make a key-reusable RLWE-based key exchange protocol but none of previous researches consider the vulnerability of lattice-based GKE against key reuse attacks.

Algorithm 1: Key-reuse attack [23, 30]

```
1. For $k = 0, 1, \cdots, q-1$, $\mathcal{A}$ does followings:
    (i) take $s_i = 0, e_i = 1$ and sets $pk_{\mathcal{A},k} = k$
    (ii) invoke the oracle $\mathcal{S}$ with $pk_{\mathcal{A},k}$ and obtains output from $\mathcal{S}$;
    (iii) analyze the number of changing signal values
    (iv) guess an absolute value of $i$-th coefficient of Bob's secret $s'_B[i]$
2. For $k = 0, 1, \cdots, q-1$, $\mathcal{A}$ does followings:
    (i) set $pk_{\mathcal{A},k} = (1 + x) \cdot k$,
    (ii) invoke the oracle $\mathcal{S}$ with $pk_{\mathcal{A},k}$ and obtains output from $\mathcal{S}$;
    (iii) analyze relations between two adjacent coefficients $s_B[i]$ and $s_B[i+1]$
    (iv) guess a sign of $s'_B[i]$ and analyzes a distribution of $p_B - a \cdot s'_B$
If $p_B - a \cdot s'_B$ follows the discrete Gaussian distribution, then $s'_B = s_B$, and if $p_B - a \cdot s'_B$ looks
random, then $s'_B = -s_B$.
```

Chapter 4. {One, two}-neighbour Attack against known Lattice-based GKE

4.1 Ding *et al.*'s Group Key Exchange

Ding *et al.* [26] also extended their two-party key exchange protocol into group setting described in **Protocol 2**. But Ding *et al.* did not describe the exact security proof of their GKE protocol.

Protocol 2: Ding Group Key Exchange $(P[0,1,\cdots,N-1], q, n, \chi, \boldsymbol{a})$

1. For a party P_i,
   (i) $\boldsymbol{s}_i, \boldsymbol{e}_{i_0} \longleftarrow \chi$;
   (ii) Compute $\boldsymbol{p}_{i_0} = \boldsymbol{a}\boldsymbol{s}_i + 2\boldsymbol{e}_{i_0} \in R_q$;
   (iii) Send $\boldsymbol{p}_{i_0}$ to P_{i+1};
2. For P_{i+j}, $(1 \leq j \leq N-2)$,
   (i) Computes $\boldsymbol{p}_{i_j} = \boldsymbol{p}_{i_{j-1}}\boldsymbol{s}_{i+j} + 2\boldsymbol{e}_{i_j} \mod q$ where $\boldsymbol{e}_{i_j} \longleftarrow \chi$;
   (ii) Sends $\boldsymbol{p}_{i_j}$ to a party P_{i+j+1};
3. For the party P_{N-1},
   (i) $\boldsymbol{e}'_{N-1} \longleftarrow \chi$ and computes $K_{N-1} = \boldsymbol{p}_{0_{N-2}}\boldsymbol{s}_{N-1} + 2\boldsymbol{e}'_{N-1}$;
   (ii) Computes and broadcast a signal $\mathsf{rec} \leftarrow S(K_{N-1})$;
   (iii) $\mathsf{sk}_{N-1} \leftarrow E(K_{N-1}, \mathsf{rec})$;
4. For a party P_j except P_{N-1},
   (i) $\boldsymbol{e}'_j \longleftarrow \chi$ and computes $K_i = \boldsymbol{p}_{(i+1)_{N-2}}\boldsymbol{s}_i + 2\boldsymbol{e}'_j$;
   (ii) $\mathsf{sk}_i \longleftarrow E(K_i, \mathsf{rec})$;

4.1.1 One-neighbour Attack

Suppose that party P_{N-1} reuses its public key $\boldsymbol{p}_{N-1}$ and $\mathcal{A}$ is an active adversary who behaves like P_{N-2} with the knowledge of P_{N-1} and with the ability to initiate multiple key exchange sessions to query party P_{N-1}. We present an attack in the one pass case of key exchange protocol, in which the adversary can initiate multiple key exchange sessions with party P_{N-1} and use key mismatch in each session to retrieve the secret $\boldsymbol{s}_{N-1}$.

An oracle $\mathcal{S}$ performs the action of party P_{N-1} and the adversary $\mathcal{A}$ has access to this oracle to make multiple queries. The adversary behaves like party P_{N-2} and sends $\boldsymbol{p}_{0_{\mathcal{A},k}} = k\boldsymbol{e}_{\mathcal{A}}$ instead of $\boldsymbol{p}_{0_{N-2}}$, where $\boldsymbol{e}_{\mathcal{A}}$ is the identity element in the ring. We first consider the simpler case when party P_{N-1} does not add the error term to its key computation K_{N-1} in **Algorithm 2**, to explain the attack strategy.

Algorithm 2: Simplified one-neighbour attack

```
1. For $k = 0, 1, \cdots, q-1$, $\mathcal{A}$ does the following:
    (i) Set $\boldsymbol{p}_{0_{\mathcal{A},k}} = k\boldsymbol{e}_{\mathcal{A}}$;
    (ii) Invoke the oracle $\mathcal{S}$ and obtains the signal value rec from $K_{N-1} = k\boldsymbol{s}_{N-1}$;
    (iii) Analyze the number of changes in each coefficient of rec;
    (iv) Guess an absolute value of $i$-th coefficient of $\boldsymbol{s}_{N-1}$;
2. For $k = 0, 1, \cdots, q-1$, $\mathcal{A}$ does the following:
    (i) Set $\boldsymbol{p}_{0_{\mathcal{A},k}} = (1+x)k\boldsymbol{e}_{\mathcal{A}}$;
    (ii) Invoke the oracle $\mathcal{S}$ and obtains the signal value rec from $K_{N-1} = k(1+x)\boldsymbol{s}_{N-1}$ from $\mathcal{S}$;
    (iii) Analyze relations between two adjacent coefficients $\boldsymbol{s}_{N-1}[i]$ and $\boldsymbol{s}_{N-1}[i+1]$;
    (iv) Let $\boldsymbol{s}'_{N-1}$ be the guess of $\mathcal{A}$ about $\boldsymbol{s}_{N-1}$. Then, from the relationship between two adjacent
         coefficients and their absolute values, $\mathcal{A}$ gets either $\boldsymbol{s}'_{N-1} = \boldsymbol{s}_{N-1}$ or $\boldsymbol{s}'_{N-1} = -\boldsymbol{s}_{N-1}$;
    (v) Analyze a distribution of $\boldsymbol{p}_{N-1} - \boldsymbol{a}\boldsymbol{s}'_{N-1}$;
If $\boldsymbol{p}_{N-1} - \boldsymbol{a}\boldsymbol{s}'_{N-1}$ follows the discrete Gaussian distribution, then $\boldsymbol{s}'_{N-1} = \boldsymbol{s}_{N-1}$;
If $\boldsymbol{p}_{N-1} - \boldsymbol{a}\boldsymbol{s}'_{N-1}$ looks random, then $\boldsymbol{s}'_{N-1} = -\boldsymbol{s}_{N-1}$;
```

For the original case with the noise, the number of queries required to recover $\boldsymbol{s}_{N-1}$ increases compared to the steps above, due to the complexity involved in eliminating the effect of the noise $2\boldsymbol{e}'_{N-1}$. By repeating the procedures in **Algorithm 1**, we check the pattern of the noise $2\boldsymbol{e}'_{N-1}$ which is from Gaussian distribution and get the secret key P_{N-1}. Likewise, we can get all secret keys for each party but this becomes unapplicable when all parties check whether they get the same common secret key.

4.2 Apon *et al.*'s Group Key Exchange

Given a ring R_q, a ring element $\boldsymbol{a}$, two standard deviations of Gaussian distributions σ_1 and σ_2 and a hash function $\mathcal{H}(\cdot)$ that samples from χ_{σ_1}, we describe ADGK19 and show how to apply our modified key reuse attack. Apon *et al.* proposed the first constant-round GKE protocol from lattice desribed in **Protocol 3** [4]. We consider two security parameters λ and ρ for computational and statistical security parameters, respectively. The parameters $N, n, \sigma_1, \sigma_2, \lambda$ and ρ of the protocol are required to satisfy some constraints described as follows:

$$(N^2 + 2N) \cdot \sqrt{n} \cdot \rho^{3/2} \cdot \sigma_1^2 + (\frac{N^2}{2} + 1) \cdot \sigma_1 + (N-2) \cdot \sigma_2 \leq \beta_{Rec}$$

$$2N\sqrt{n}\lambda^{3/2}\sigma_1^2 + (N-1)\sigma_1 \leq \beta_{\text{Rényi}}$$

$$\sigma_2 = \Omega(\beta_{\text{Rényi}}\sqrt{n/log(\lambda)}$$

Apon *et al.* did not specify the key reconciliation mechanism, executed as subroutine in **Key Computation** step and seems not to consider attacks on reconciliation mechanisms like key reuse attacks.

Protocol 3:
$\mathsf{ADGK19}(P\,[0,1,\cdots,N-1]\,, \boldsymbol{a}, \mathcal{H}, \sigma_1, \sigma_2)$

(Round 1) For each party P_i for $i = 0$ to $N-1$, do the following in parallel.

1. Computes $\boldsymbol{z}_i = \boldsymbol{a}\boldsymbol{s}_i + \boldsymbol{e}_i$ where $\boldsymbol{s}_i, \boldsymbol{e}_i \leftarrow \chi_{\sigma_1}$;
2. Broadcasts $\boldsymbol{z}_i$;

(Round 2) For $i = 0$ to $N-1$, do the following in parallel.

1. If $i = 0$, party P_0 samples $\boldsymbol{e}'_0 \leftarrow \chi_{\sigma_2}$ and otherwise, party P_i samples $\boldsymbol{e}'_i \leftarrow \chi_{\sigma_1}$;
2. Each party P_i broadcasts $X_i = (\boldsymbol{z}_{i+1} - \boldsymbol{z}_{i-1})\,\boldsymbol{s}_i + \boldsymbol{e}'_i$;

(Round 3) For a party P_{N-1}, do the following.

1. Samples $\boldsymbol{e}''_{N-1} \leftarrow \chi_{\sigma_1}$ and calculates $\boldsymbol{b}_{N-1} = \boldsymbol{z}_{N-2}N\boldsymbol{s}_{N-1} + (N-1)X_{N-1} +$ $(N-2)X_0 + \cdots + X_{N-3} + \boldsymbol{e}''_{N-1}$;
2. Runs $\mathsf{recMsg}(\cdot)$ to output $(\mathsf{rec}, \boldsymbol{k}_{N-1}) = \mathsf{recMsg}(\boldsymbol{b}_{N-1})$;
3. Broadcasts rec and gets session key as $\mathsf{sk}_{N-1} = \mathcal{H}(\boldsymbol{k}_{N-1})$;

(Key Computation) For each party P_i $(i \neq N-1)$, do the following:

1. Computes $\boldsymbol{b}_i = \boldsymbol{z}_{i-1}N\boldsymbol{s}_i +\, (N-1)X_i + (N-2)X_{i+1} + \cdots + X_{i+N-2}$;
2. Runs $\mathsf{recKey}()$ to output $\boldsymbol{k}_i = \mathsf{recKey}\,(\boldsymbol{b}_i, \mathsf{rec})$ and
   gets session key as $\mathsf{sk}_i = \mathcal{H}(\boldsymbol{k}_i)$;

As we discussed earlier, lattice-based two-party key exchange protocols such as Ding Key Exchange and NewHope protocols have been broken by the weakness of their key reconciliation mechanisms and to the best of our knowledge, all previously known key reconciliation mechanisms are vulnerable to key reuse attack.

We will check the vulnerability of $\mathsf{ADGK19}$ by applying two well-known key reconciliation mechanisms; Ding's reconciliation mechanism and NewHope reconciliation mechanism.

4.2.1 Two-neighbour Attack

If we apply Ding's reconciliation mechanism to ADGK19, $\mathsf{recMsg}(\boldsymbol{b}_{N-1})$ outputs $\mathsf{rec} \leftarrow S(\boldsymbol{b}_{N-1})$ and $\boldsymbol{k}_{N-1} \leftarrow E(\boldsymbol{b}_{N-1}, \mathsf{rec})$ and $\mathsf{recKey}(\boldsymbol{b}_i, \mathsf{rec})$ outputs $\boldsymbol{k}_i \leftarrow E(\boldsymbol{b}_i, \mathsf{rec})$.

We assume that a public value $\boldsymbol{a} \in R_q$ and public keys $\boldsymbol{z}_i$ for each party P_i are fixed except P_{N-2}. $\mathcal{A}$ can initiate many sessions with all parties in the group and can access to the oracle $\mathcal{S}$. In performing the attack, an adversary $\mathcal{A}$ plays the role of two previous neighbouring parties P_{N-2} and P_{N-3} of the last party P_{N-1}. $\mathcal{A}$ creates key pairs $(\boldsymbol{s}_{N-2}, \boldsymbol{z}_{N-2})$ of P_{N-2} by deviating from the protocol. We denote the public key $\boldsymbol{z}_{N-2}$ deviated by $\mathcal{A}$ as $\boldsymbol{z}_{\mathcal{A}}$ and the corresponding secret key $\boldsymbol{s}_{N-2}$ and error $\boldsymbol{e}_{N-2}$ deviated by $\mathcal{A}$ as $\boldsymbol{s}_{\mathcal{A}}$ and $\boldsymbol{e}_{\mathcal{A}}$, respectively.

We describe an attack on simplified ADGK19 where the error term $\boldsymbol{e}''_{N-1}$ is not added to the key computation of $\boldsymbol{b}_{N-1}$. We set an oracle $\mathcal{S}$ that simulates party P_{N-1}'s action from a given input public key. On receiving $\boldsymbol{z}_{\mathcal{A}}$ from $\mathcal{A}$, S computes $\boldsymbol{b}_{N-1}$ and outputs $(\mathsf{rec}, \boldsymbol{k}_{N-1})$ from $\boldsymbol{b}_{N-1}$ according to the protocol.

For the term $\boldsymbol{b}_{N-1}$, we only consider the term $\boldsymbol{z}_{N-2} N \boldsymbol{s}_{N-1} + (N-1)X_{N-1}$ since all X_i's except X_{N-1} and X_{N-3} are fixed and X_{N-3} is controlled by $\mathcal{A}$ since P_{N-3} is corrupted. Then, $\boldsymbol{z}_{\mathcal{A}} N \boldsymbol{s}_{N-1} + (N-1)X_{N-1} = \boldsymbol{z}_{\mathcal{A}} \boldsymbol{s}_{N-1} + (N-1)\boldsymbol{z}_0 \boldsymbol{s}_{N-1}$ and by the assumption, $(N-1)\boldsymbol{z}_0 \boldsymbol{s}_{N-1}$ is also fixed. Hence, $\boldsymbol{z}_{N-2} N \boldsymbol{s}_{N-1}$ is the only non-constant term of $\boldsymbol{b}_{N-1}$.

Hence, a coefficient of rec sent by the party P_{N-1} represents the key reconciliation output of a matching coefficient of $\boldsymbol{z}_{\mathcal{A}} \boldsymbol{s}_{N-1}$ by shifting a constant term and we describe the attack on rec as follows:

Step 1. $\mathcal{A}$ invokes the oracle $\mathcal{S}$ with input $\boldsymbol{z}_{\mathcal{A}} = k\boldsymbol{e}_{\mathcal{A}}$ $(k = 0, 1, \cdots, q-1)$ where $\boldsymbol{s}_{\mathcal{A}}$ is 0 and $\boldsymbol{e}_{\mathcal{A}}$ is the identity element 1 in R_q so that $\boldsymbol{b}_{N-1}$ becomes $kN\boldsymbol{s}_{N-1} + (N-1)X_{N-1} + (N-2)X_0 + \cdots + X_{N-3}$. $\mathcal{A}$ can make a correct guess of the value of $\boldsymbol{s}_{N-1}[i]$ based on the number of times the signal of rec for $\{0, \boldsymbol{s}_{N-1}, \cdots, (q-1)\boldsymbol{s}_{N-1}\}$ changes for each coefficient $\boldsymbol{s}_{N-1}[i]$.

Step 2. $\mathcal{A}$ invokes $\mathcal{S}$ with input $(1+x)\boldsymbol{z}_{\mathcal{A}} = (1+x)k\boldsymbol{e}_{\mathcal{A}}$ $(k = 0, 1, \cdots, q-1)$. $\mathcal{A}$ is able to see the key reconciliation value of $(1+x)k\boldsymbol{s}_{N-1}$ that is the output by $\mathcal{S}$. Thus, again by checking the number of signal changes, $\mathcal{A}$ finds values of the coefficients of $(1+x)\boldsymbol{s}_{N-1}$, which are $\boldsymbol{s}_{N-1}[0] - \boldsymbol{s}_{N-1}[n-1], \boldsymbol{s}_{N-1}[1] + \boldsymbol{s}_{N-1}[0], ..., \boldsymbol{s}_{N-1}[n-1] + \boldsymbol{s}_{N-1}[n-2]$ up to sign.

Step 3. From **Steps 1** and **2**, we can determine if each pair of coefficients $\boldsymbol{s}_{N-1}[i]$, $\boldsymbol{s}_{N-1}[i+1]$ have equal or opposite signs, hence narrowing down to only two possibilities such that the guess $\boldsymbol{s}'_{N-1} = \boldsymbol{s}_{N-1}$ or $-\boldsymbol{s}_{N-1}$.

Step 4. Since $\boldsymbol{a}$ and $\boldsymbol{z}_{N-1}$ are public, $\mathcal{A}$ computes $\boldsymbol{z}_{N-1} - \boldsymbol{a}\boldsymbol{s}'_{N-1}$ and verifies the distribution of the result. If $\mathcal{A}$ correctly guesses the sign of $\boldsymbol{s}_{N-1}[0]$ and so does all the coefficients of $\boldsymbol{s}_{N-1}$, the resulting distribution of $\boldsymbol{z}_{N-1} - \boldsymbol{a}\boldsymbol{s}'_{N-1}$ is same as the distribution of $\boldsymbol{z}_{N-1} - \boldsymbol{a}\boldsymbol{s}_{N-1} = \boldsymbol{e}_{N-1}$, which is the Gaussian distribution. Otherwise, the output becomes random and $\mathcal{A}$ obtain the correct $\boldsymbol{s}_{N-1}$ value by flipping the sign.

Thus, $\mathcal{A}$ is able to determine the exact value of $\boldsymbol{s}_{N-1}$ without any ambiguity at the end of the execution when P_{N-1} reuses the same key for several executions. The success of the attack also shows the significance of the role of the key reconciliation function in the group key exchange protocol.

When the error term is added to $\boldsymbol{b}_{N-1}$, there are some frequent changes in the signal value at the boundary values. But, similar to one-neighbour attack in Ding's GKE protocol, we check the pattern of the noise $\boldsymbol{e}''_{N-1}$, by repeating all steps in two-neighbour attack, which is from Gaussian distribution and get the secret key P_{N-1}.

Chapter 5. Novel Construction of GKE Protocols

5.1 Choi *et al.*'s Tree-based GKE

5.1.1 Basic Construction

In the following construction based on RLWE, we assume that all parties are trustful so the protocol doesn't get influenced by which party is chosen as the root node of the tree. Also, no party reveals the other's ephemeral key.

Network topology can be interpreted as a graph where the connection becomes an edge and each party becomes a node of the graph. From network topology of a given group G, we can find a tree structure efficiently if the graph is the connected graph. For the sake of simplicity, we assume that the balanced binary tree is chosen from the network topology and call it as a keygen tree $\tilde{T}$ of our protocol.

Finding the keygen tree $\tilde{T}$ is almost the same as finding the spanning tree $\tilde{T}_{\tilde{G}}$ of the given graph $\tilde{G}$. This can be done efficiently by using path-finding algorithms in graph theory, like well-known Dijkstra's algorithm.

If there are N parties who participate in the communication, we set a keygen tree $\tilde{T}$ with a depth $d = \lceil \log N \rceil$ and define a level of a party $P_{\tilde{u}}$ as $l_{\tilde{u}} = d - l_{\mathsf{root},\tilde{u}}$ where $l_{\mathsf{root},\tilde{u}}$ is the length of a path from the node $\tilde{u}$ to the root node.

We assume that the number of parties are at least three so that the root node always has two nodes as a child. Then, we construct Tree-based Group Key Exchange (TGKE) in static version as follows:

Step S1. Setup.

Find a keygen tree $\tilde{T} = (\tilde{V}, \tilde{E}_T)$ with the depth $d = \lceil \log N \rceil$ from the network topology.

Step S2. Key Construction.

1. Set the leaf nodes as level 0 party, their parent nodes as level 1 party, till the root node as level d party.
2. Between a parent node $\tilde{v}_p$ and its child node $\tilde{v}_c$, we run a two-party KE $\mathcal{TKE}$ to find the two-party common secret key $epk_{p,c}$ between two parties P_p and P_c as an ephemeral key.
3. We run $\mathcal{TKE}$ between level 0 parry and level 1 node. Then, each level 1 party does XOR operation to get an initial value for the next level. *e.g.*, in Figure 5.1(a), since party P_2 is the parent node of P_4 and P_5, P_2 has two ephemeral keys $epk_{2,4}$ and $epk_{2,5}$ and compute the XORed values $epk_2 = epk_{2,4} \oplus epk_{2,5}$.

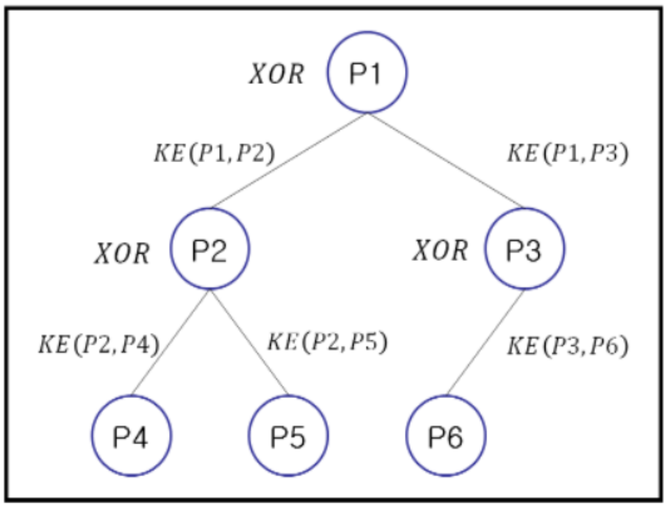
4. Similarly, from level 1 party to level d party, we run $\mathcal{TKE}$.

5. Once root node gets the ephemeral keys with his/her child node, it computes the common group secret key sk_{root} by XOR operation of two-party common secret key.

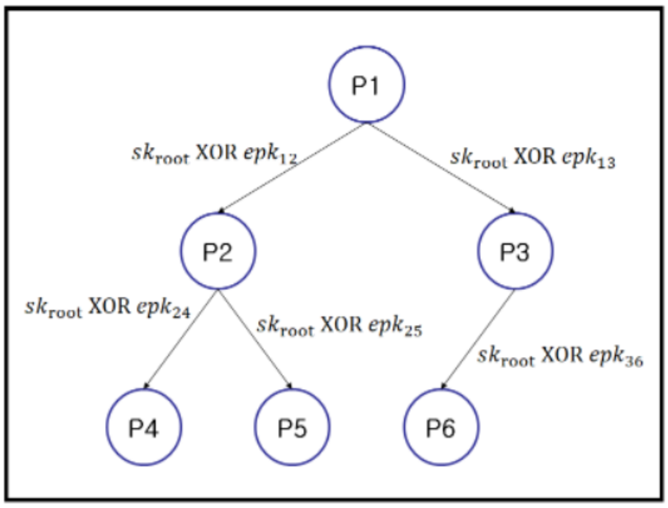
Step S3. Key Sharing.

1. The root node sends the encrypted common secret key k_c to its child nodes $\tilde{v}_c$ by computing $k_c = sk_{\text{root}} \oplus epk_{p,c}$

2. All parties get the same value sk_g after d rounds of sending encrypted common secret key.



(a) common key construction example


P1
sk_{root} XOR epk_{12}
sk_{root} XOR epk_{13}
P2
P3
sk_{root} XOR epk_{24}
sk_{root} XOR epk_{25}
sk_{root} XOR epk_{36}
P4
P5
P6

(b) common key distribution example

Figure 5.1: TGKE protocol in static setting

5.1.2 Dynamic TGKE

To extend TGKE into dynamic setting by describing the procedure when a new party is joining or some party is leaving the group communication.

Join describes the process when a party is joining the group communication. Figure 5.2 shows how a new party is joined to the network.

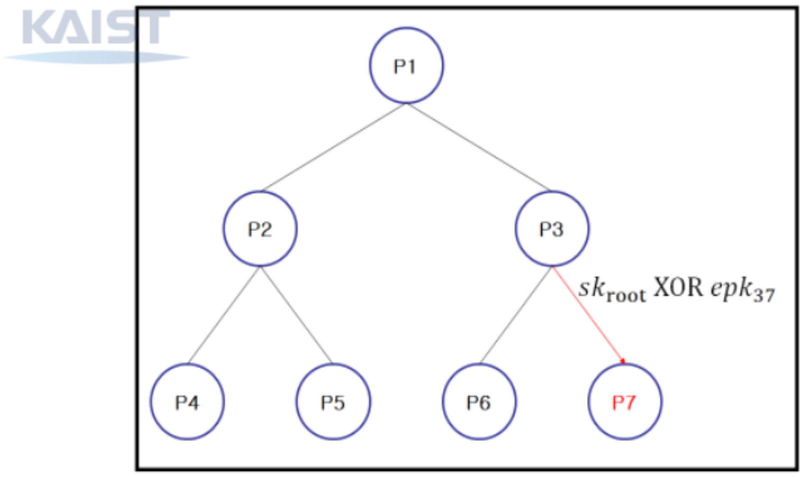


Figure 5.2: Join example

Step D1. Join.

1. Add a node $\tilde{v}_N$ to the original keygen tree $\tilde{T} = \{\tilde{V}, \tilde{E}_T\}$. Then, add an edge between $\tilde{v}_N$ and some node $\tilde{v}_i \in \tilde{V}$ with level 0 or 1, which becomes the balanced binary tree after edge addition. We get a new keygen tree $\tilde{T}' = \{\tilde{V} \bigcup \{\tilde{v}_N\}, \tilde{E}_T \bigcup \{(\tilde{v}_i, \tilde{v}_N)\}\}$.
2. Between parent node $\tilde{v}_i$ and child node $\tilde{v}_N$, we run two-party key exchange protocol $\mathcal{TKE}$ to find the ephemeral key $epk_{i,N}$ between two parties $\tilde{v}_i$ and $\tilde{v}_N$.
3. $\tilde{v}_i$ sends the encrypted common secret key $k_c = sk_{\text{root}} \oplus epk_{i,N}$ to $\tilde{v}_N$ and $\tilde{v}_N$ gets the common secret key sk_{root} by XOR operation.

Leave and TreeRefresh describe the member revocation mechanism since there are two examples when the revoked member is the leaf node (deleting the node P_4 in Figure 5.3(a)) or non-leaf node (deleting the node P_2 in Figure 5.3(b)).

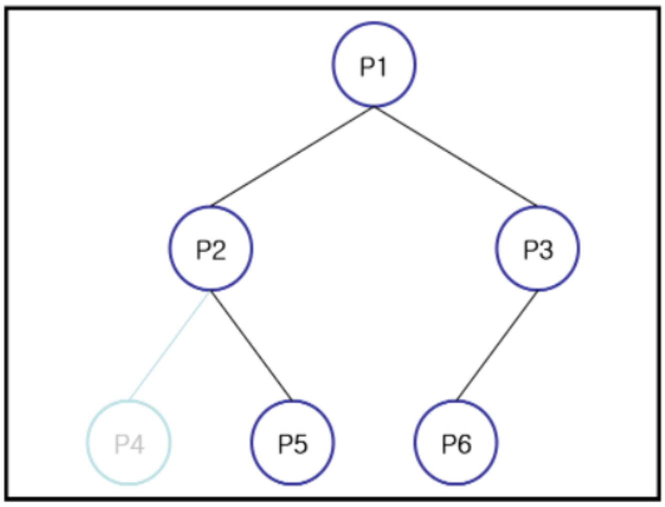
Step D2-1. Leave (leaf case).

1. Delete a node $\tilde{v}_i$ from the original keygen tree $\tilde{T} = \{\tilde{V}, \tilde{E}_T\}$. Then, we get a new keygen tree $\tilde{T}' = \{\tilde{V} \setminus \{\tilde{v}_i\}, \tilde{E}_T \setminus \{1 \le j \le g | (\tilde{v}_i, \tilde{v}_j)\}\}$.

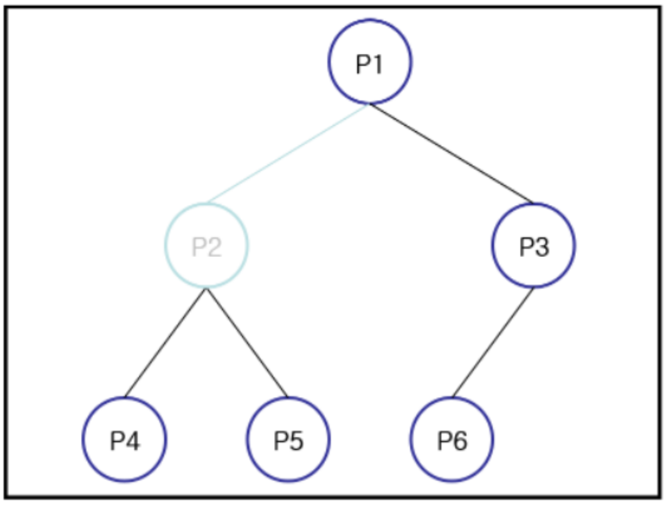
2. From this new tree, run key construction phase and key sharing phase to get a new common secret key sk'.

Step D2-2. TreeRefresh (non-leaf case).

1. Delete a node $\tilde{v}_i$ from the original keygen tree $\tilde{T} = \{\tilde{V}, \tilde{E}_T\}$. Then, run the setup phase to construct a new keygen tree.
2. From this new tree, run key construction phase and key sharing phase to get a new common secret key sk'.



(a) membership revocation (leaf case)



(b) membership revocation (non-leaf case)

Figure 5.3: Remove examples for Leave and TreeRefresh

Note that if we have the connected network topology, we can easily generate the tree by contracting an edge between $\tilde{v}_i$ and its child node. In this case, both leave and tree contraction algorithms make the tree $\tilde{T}'$ which is a minor of the tree $\tilde{T}$.

5.1.3 Security Analysis

In this section, we give a security analysis of static TGKE. Any probabilistic polynomial-time (PPT) adversary should not distinguish a real common group secret key to a random one even if he/she gets the transcripts of the protocol. We assume that every party is trustful and no party does insider attacks.

We derive the correctness proof and the security proof in **Theorems 4** and **3**, respectively.

Theorem 2. Our protocol has the same common secret key between all parties in the group.

Proof. Since we focus on designing conceptually-simpler model for fast and efficient implementation result, we send the common secret key by XORing two-party common secret key. By repeating the same process between parent node and his/her child node from level 0 to level d, we always get the correct common secret key for the group.

□

Before proving **Theorem 3**, we remark that the root node has the level d and a leaf node has the level 0 where $d = \log N$ is a depth of a keygen tree from the network of the group.

Theorem 3. If underlying two-party key exchange protocol $\mathcal{TKE}$ is secure against the passive adversary, TGKE protocol is also secure.

Proof. We prove a security by a hybrid game between the real shared secret key and the random one for each level of the tree.

Then, $\mathbf{Game}_0$ is the real game which the adversary gets the real common group secret key sk and $\mathbf{Game}_d$ is the game which the adversary gets the random value t_d. We show that the views of $\mathbf{Game}_0$ and $\mathbf{Game}_d$ are computationally indistinguishable for any PPT adversaries.

$\mathbf{Game}_0$. This game is the real game between protocol challenger and the passive adversary $\mathcal{A}$, adversary obtains the common secret key sk from our protocol described in Section 6.

$\mathbf{Game}_1$. This game is identical to $\mathbf{Game}_0$ except level 1 nodes changes his initial key as a random value instead of two-party common secret key with level 0 nodes.

Since the underlying two-party key exchange protocol is secure against passive adversary $\mathcal{A}$. Random values are indistinguishable from the key generated by two-party KE for any PPT adversaries. Thus, the adversary cannot distinguish $\mathbf{Game}_0$ and $\mathbf{Game}_1$.

$\mathbf{Game}_i$ $(2 \leq i \leq d-1)$. This game is identical to $\mathbf{Game}_{i-1}$ except level i nodes changes his initial key as a random value r_i instead of two-party common secret key with level $i-1$ nodes. Since the underlying two-party key exchange protocol is secure against passive adversary $\mathcal{A}$. Random values are indistinguishable from keys generated by two-party KE for any PPT adversaries. Thus, the adversary cannot distinguish $\mathbf{Game}_{i-1}$ and $\mathbf{Game}_i$.

Game$_d$. This game is identical to **Game$_{d-1}$** except that encrypted values from parties with level $d-1$ to the root node are replaced by random value t_d during the protocol. Then, similar to the game between **Game$_{i-1}$** and **Game$_i$**, the views between **Game$_{d-1}$** and **Game$_d$** are computationally indistinguishable for any PPT adversaries.

Now, the claim follows since the real game **Game$_0$** and the random game **Game$_d$** are computationally indistinguishable assuming the security of underlying two-party key exchange protocol.

□

5.2 Homomorphic Encryption based GKE

5.2.1 Homomorphic Encryption

In 1978, Rivest *et al.* [54] suggested the concept of homomorphic encryption that supports some computation on encrypted data without the knowledge of the secret key. A homomorphic encryption scheme is defined as follows:

Definition 1. (homomorphic encryption) A homomorphic encryption scheme $\mathcal{HE}$ is a tuple of PPT algorithms $\mathcal{HE}$ = (HE.Gen, HE.Enc, HE.Eval, HE.Dec) with the following functionality:

HE.Gen$(n,\ \alpha)$:

Given the security parameter n and an auxiliary input α, this algorithm outputs a key triple (pk, sk, evk), where pk is the key used for encryption, sk is the key used for decryption and evk is the key used for evaluation.

HE.Enc(pk, m) :

Given a public key pk and a message m, this algorithm outputs a ciphertext c of the message m.

HE.Eval$(evk, C, c_1, \cdots, c_n)$:

Given an evaluation key evk, a Boolean circuit C, and pairs $\{c_i\}_{i=1}^{n}$ where c_i is either a ciphertext or previous evaluation results, this algorithm produces an evaluation output.

HE.Dec(sk, c) :

Given a secret key sk and a ciphertext or an evaluation output c, this algorithm outputs a message m.

Depending on what kind of Boolean circuit the scheme supports, we call an encryption scheme as multiplicative (*resp.* additive) HE if it only supports multiplication (*resp.* addition).

Early work on HE was not practical but there are many cryptographic algorithm tools that support HE efficiently such as HElib, FHEW [27, 34].

5.2.2 Basic Operations from Graph Theory

We define a graph $\tilde{G}$ as a pair of the set of vertices $\tilde{V} = \{\tilde{v}_1, \tilde{v}_2, \cdots, \tilde{v}_n\}$ and the set of edges $\tilde{E}$ between two vertices. Among many terminologies in graph theory, we use vertex/edge addition, vertex/edge deletion, and edge contraction. Since the process of vertex/edge addition and vertex/edge deletion operations is clear, we give a definition of edge contraction and graph minor.

Definition 2. (edge contraction) Let $\tilde{G} = \left(\tilde{V}, \tilde{E}\right)$ be a graph containing an edge $\tilde{e} = (\tilde{u}, \tilde{v})$ with $\tilde{u} \neq \tilde{v}$. Let f be a function which maps every vertex in $\tilde{V} \setminus \{\tilde{u}, \tilde{v}\}$ to itself, and maps $\tilde{u}$ and $\tilde{v}$ to a new vertex $\tilde{w}$. The edge contraction of e makes a new graph $G' = \left(\tilde{V}', \tilde{E}'\right)$, where $\tilde{V}' = \left(\tilde{V} \setminus \{\tilde{u}, \tilde{v}\}\right) \cup \tilde{w}$, $\tilde{E}' = \tilde{E} \cup \tilde{e}$, and for every $\tilde{v} \in \tilde{V}, \tilde{x}' = f(\tilde{x}) \in \tilde{V}2$ is incident to an edge $\tilde{e}' \in \tilde{E}2$ if and only if, the corresponding edge, $\tilde{e} \in \tilde{E}$ is incident to $\tilde{x}$ in $\tilde{G}$.

Definition 3. (graph minor) Let $\tilde{G} = \left(\tilde{V}_G, \tilde{E}_G\right)$ and $\tilde{H} = \left(\tilde{V}_H, \tilde{E}_H\right)$ be a graph. If $\tilde{H}$ can be formed from $\tilde{G}$ by deleting vertices/edges and contracting edges, we say that $\tilde{H}$ is called a minor of $\tilde{G}$.

We use these terms for membership addition and revocation in dynamic setting.

5.2.3 Construction in Static Setting

In Figure 5.4, we state a methodology to build a (non-interactive) multi-party key exchange protocol fro HE by Choi and Kim [20] (HE-KE protocol). All parties pre-share the master secret key sk from HE.Gen algorithm of HE scheme $\mathcal{HE}$. A circuit C can be public in this protocol.

Assuming that the server is honest but curious, HE-KE protocol runs as follows.

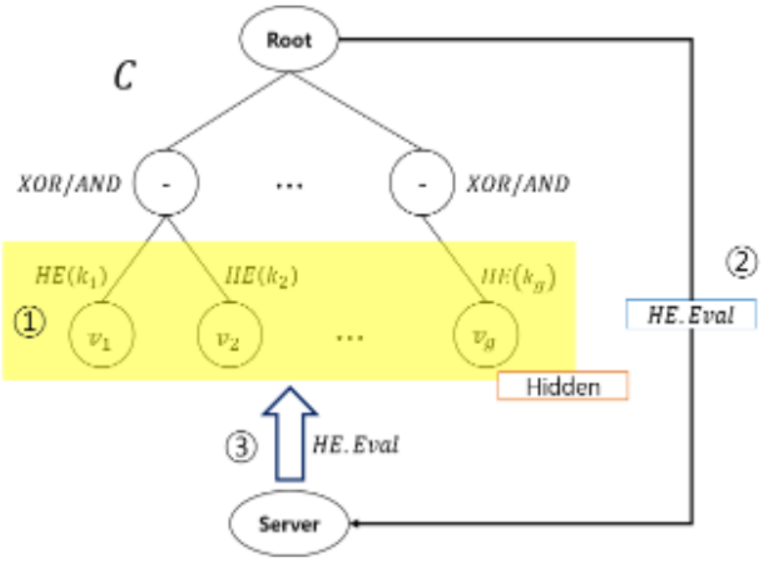


Figure 5.4: HE-KE protocol

Step 1. Each party pre-shares the master group key $sk \leftarrow \mathsf{HE.Gen}(n, \alpha)$ with each other.

Step 2. Make the Boolean circuit C with g leaf nodes.

Step 3. Each party makes ephemeral session key k_i and encrypts it with its public key pk_i, $c_i = \mathsf{HE.Enc}(pk_i, k_i)$. (① from Figure 5.4.)

Step 4. The server computes $c = \mathsf{HE.Eval}(evk, C, c_1, \cdots, c_g)$ with the Boolean circuit C and broadcasts c. (② and ③ from Figure 5.4, respectively.)

Step 5. Each party decrypts the evaluated value c and get the session group key $k = \mathsf{HE.Dec}(sk, c)$.

Lemma 1. [20] If underlying homomorphic encryption $\mathcal{HE}$ is secure, HE-KE protocol is also secure, *i.e.*, it satisfies session key security, known key security, and key privacy.

System Model and Security Requirement

HE-KE protocol [20] suggests non-interactive property by pre-sharing the master group key between each party. But, in this protocol, the way each party pre-shares the master secret key is unclear. Even more, since there is no key refresh algorithm, we should assume that the former group member is trustful so that the adversary $\mathcal{A}$ doesn't get the master secret key.

To resolve this, we assume that the server is fully trustful so that the server distributes the ephemeral key as well as the evaluated value of session group key to each party for each membership event (*e.g., join, leave, merge, or partition*). Compared to HE-KE protocol [20], the former group member cannot compute the session group key in our protocol.

We assume that the adversary $\mathcal{A}$ can (i) send messages to some party, (ii) run the protocol to get the appropriate session group key, and (iii) get some information from a previous group member like the former session group keys.

To check the security of our dynamic key exchange protocol, one of the most important security requirements is key freshness. A key is called *fresh* if the generated key is guaranteed to be new to prevent an old key being reused by an adversary. To guarantee key freshness, we have to prove the following security requirements:

1. **Group Key Secrecy**

   If all group members in the protocol are not corrupted, it is computationally infeasible for a passive adversary to discover any session group key.

2. **Forward Secrecy**

   Even after a passive adversary $\mathcal{A}$ has acquired some session group keys, new session group keys must remain out of reach of the adversary and former group members.

3. **Backward Secrecy**

   Even after a passive adversary $\mathcal{A}$ has acquired some session group key, previously used session group keys must not be discovered by the adversary and new group members.

4. **Key Independence**

   A passive adversary who knows a proper subset of session group keys cannot discover any other session group keys.

5.2.4 Parcel-S Protocol

In Figure 5.5, we give a methodology to build a GKE protocol from HE called Parcel-S for static setting and Parcel-D for dynamic setting, which supports the membership events. As discussed earlier, the server delivers the ephemeral key to all group members to provide forward and backward secrecy. Then, the server broadcasts the evaluated value of ciphertexts from group members and the Boolean circuit C. Assume that we have either XOR or AND operation in C.

All group members pre-share the master secret key sk from $\mathsf{HE.Gen}$ algorithm of HE scheme $\mathcal{HE}$. A circuit C can be public in this protocol.

Under this condition, Parcel-S protocol runs as follows.

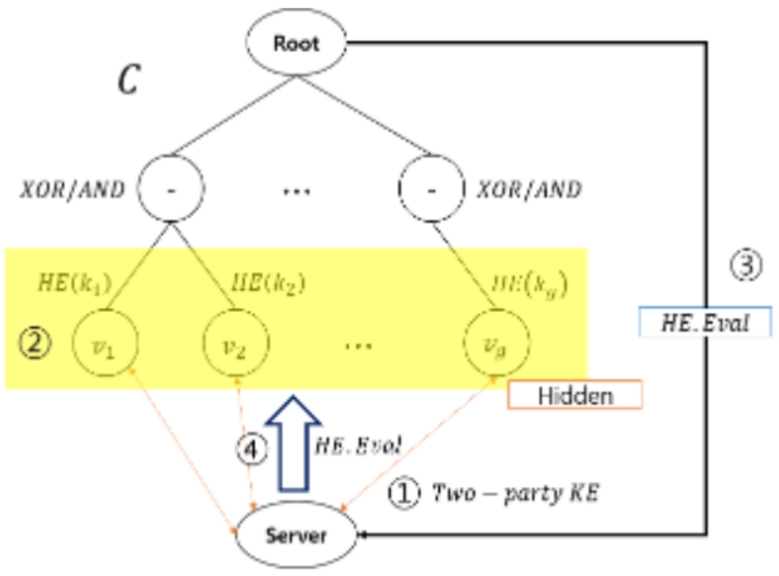


Figure 5.5: Parcel-S protocol

Step 1. Each party pre-shares the master group key $sk \leftarrow \mathsf{HE.Gen}(n, \alpha)$ with each other and the server runs two-party key exchange protocol $\mathcal{TKE}$ with each party to make the long-term secret key msk_i. (① from Figure 5.6.)

Step 2. Make the Boolean circuit C with g leaf nodes.

Step 3. Each party makes ephemeral session key k_i and encrypts it with its public key pk_i, $c_i = \mathsf{HE.Enc}(pk_i, k_i)$. (② from Figure 5.6.)

Step 4. The server computes $c = \mathsf{HE.Eval}(evk, C, c_1, \cdots, c_g)$ with the Boolean circuit C and broadcasts c. (③ and ④ from Figure 5.6, respectively.)

Step 5. The server sends the ephemeral key epk to each party encrypted by encryption scheme $\mathcal{SKE}$ with secret key msk_i.

Step 6. Each group member decrypts the evaluated value c and the ephemeral key epk. Then, each group member gets the proper session group key $k = \mathsf{HE.Dec}(sk, c) \oplus epk$.

Then, we show the correctness and security requirements as below.

Theorem 4. If underlying homomorphic encryption scheme $\mathcal{HE}$ is valid, Parcel-S protocol is correct, *i.e.*, it outputs the valid session group key for each session.

Proof. In Step 4, the server can compute the evaluated value c if $\mathcal{HE}$ is valid and $\mathsf{HE.Eval}(evk, C, c_1, \cdots, c_g)$ is well-defined. Then, each party can get the same value from XOR operation between decryption of c and ephemeral key epk. □

Theorem 5. If underlying homomorphic encryption scheme $\mathcal{HE}$ and two-party key exchange protocol $\mathcal{TKE}$ are secure, Parcel-S protocol is also secure, *i.e.*, it satisfies group key secrecy, forward/backward secrecy, and key independence.

Proof. Since $\mathcal{HE}$ is secure, each ciphertext and evaluated value are indistinguishable from random. Thus, all ciphertext c_i of the ephemeral session key k_i from party $\tilde{v}_i$ are indistinguishable from random and so does the ciphertext c of the session group key k, evaluated value of all the ciphertext $c_i's$. Hence, our construction guarantees group key secrecy.

Since the server doesn't have the information of the pre-shared secret key sk, the server cannot know the session group key k because the server doesn't know the decryption value of c. Likewise, the former(*resp.* new) group members cannot know the new(*resp.* previous) session group keys since they don't know the ephemeral key since $\mathcal{TKE}$ is secure. Thus, our protocol provides forward secrecy and backward secrecy.

Thanks to the presence of ephemeral keys, we can show the key independence as well.

□

5.2.5 Parcel-D Protocol

Membership Events

As discussed in Chapter 5.2.3, a dynamic setting needs to provide key adjustment protocols to cope with any membership changes. Parcel-D protocol includes algorithms to support the following operations:

Join$(\tilde{v}', C)$:

When a new group member $\tilde{v}'$ is added to the group to participate in the group communication, reconstruct the Boolean circuit C_{Join}.

Leave$(\tilde{v}_i, C)$:

When a group member $\tilde{v}_i$ is removed from the group communication, reconstruct the Boolean circuit C_{Leave}.

Merge$(\tilde{V}', C, C')$:

When some group $\tilde{V}'$ is merged with the current group, reconstruct the Boolean circuit C_{Merge}.

Partition$(\tilde{V}_j, C)$:

When a subset of group members $\tilde{V}_j$ are removed from the group communication, reconstruct the Boolean circuit $C_{Partition}$.

After these membership events, we run Parcel-S protocol again to get a new session group key. Note that for each membership event, the only change is the Boolean circuit. A new member can get the same master group key sk since only the existence of ephemeral keys provides forward/backward secrecy. For the remaining of this chapter, we give the detail explanation on each membership event algorithm.

Join Protocol

We assume the group has g members. The new member $\tilde{v}_{g+1}$ initiates the protocol by sending a 'join' request message to the server. If the server receives this message, the server makes the long-term secret key with $\tilde{v}_{g+1}$. Then, it determines the splitting point in the circuit. The splitting point is the shallowest rightmost node, where the joining of a new member does not increase the depth of the circuit. If the circuit is a fully balanced tree, it chooses a point with more XOR operations in the path to the root node, to minimize the complexity. In the splitting point, we put some Boolean operation like XOR or AND. A new Boolean circuit C_{Join} is constructed as follows:

1. Find the splitting point that does not increase the depth of the circuit or minimize the complexity.
2. Add two vertices to the splitting point and connect these vertices to the splitting point.
3. Put some Boolean operation to the splitting point.
4. Set the leaf nodes as the group members $\tilde{v}_1, \tilde{v}_2, \cdots, \tilde{v}_{g+1}$.

Figure 5.6 shows an example of Boolean circuit when a new group member is joined to the group.

Leave Protocol

Again, we start with g members and assume that member $\tilde{v}_i$ leaves the group. When $\tilde{v}_i$ leaves, the server first deletes the long-term secret key between the server and $\tilde{v}_i$.

From the original Boolean circuit C, the server finds the leaf node marked as $\tilde{v}_i$ and its parent node. Then, we construct a new Boolean circuit C_{Leave} as follows:

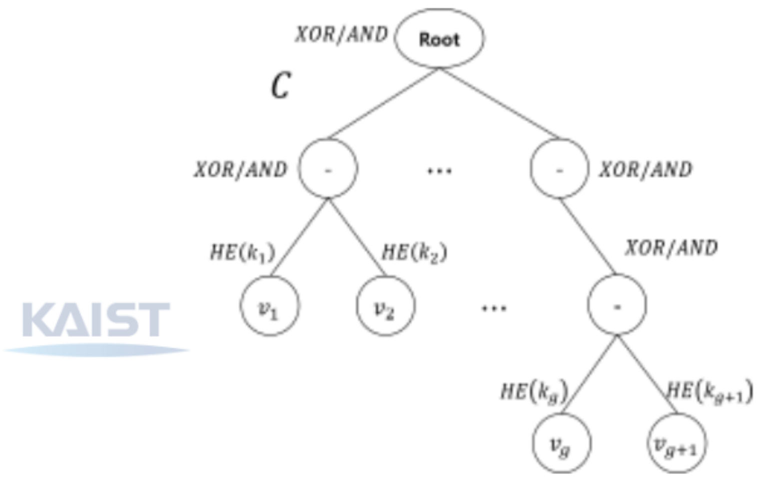


Figure 5.6: Join algorithm in Parcel-D protocol

1. Find the leaf node marked as $\tilde{v}_i$ and its parent node $\tilde{w}$.

2. Remove $\tilde{v}_i$ node and contract an edge between $\tilde{w}$ node and $\tilde{v}_{i+1}$ node, where $\tilde{v}_{i+1}$ node has the same parent node $\tilde{w}$ with $\tilde{v}_i$ node. (Without loss of generality, we may rename the group members to satisfy this condition.)

3. Rename the leaf nodes as the group members $\tilde{v}'_1, \tilde{v}'_2, \cdots, \tilde{v}'_{g-1}$.

Figure 5.7 shows an example of Boolean circuit when a group member leaves the group.

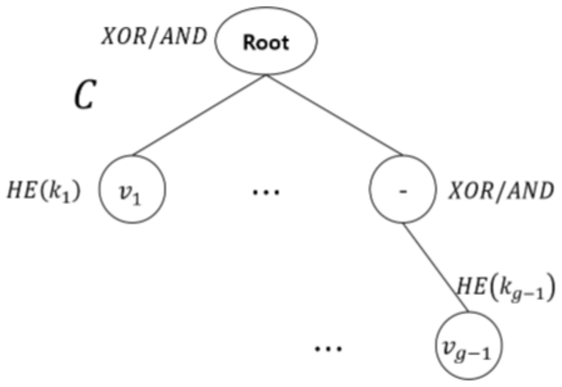


Figure 5.7: Leave algorithm in Parcel-D protocol

Merge Protocol

Network faults may partition a group into several subgroups. In the meantime, they communicate inside the subgroups only. After the network recovers, subgroups need to be merged into a single group. In this case, since all group members already exist in the group communication, we don't need to make a new long-term secret key between the server and each group member.

To build a new Boolean circuit, the server checks the connecting point in the circuit C. The connecting point is chosen similarly to the splitting node in Join algorithm. We assume that the original group $\tilde{V}$ consists of g_1 members and the merged group $\tilde{V}'$ consists of g_2 members, where $g_1 \geq g_2$. Then, a new Boolean circuit C_{Merge} is processed as follows:

1. Find the connecting point from a Boolean circuit C that does not increase the depth of the circuit or minimize the complexity.
2. Connect the edge $\tilde{e}_{Merge}$ between connecting point from C and the root node of a circuit C', which is the Boolean circuit of group $\tilde{V}'$.
3. Contract the edge $\tilde{e}_{Merge}$ and put some Boolean operation to the node after this edge contraction.
4. Set the leaf nodes as the group members $\tilde{v}_1, \tilde{v}_2, \cdots, \tilde{v}_{g_1+g_2}$.

Figure 5.8 shows an example of Boolean circuit when two groups merge.

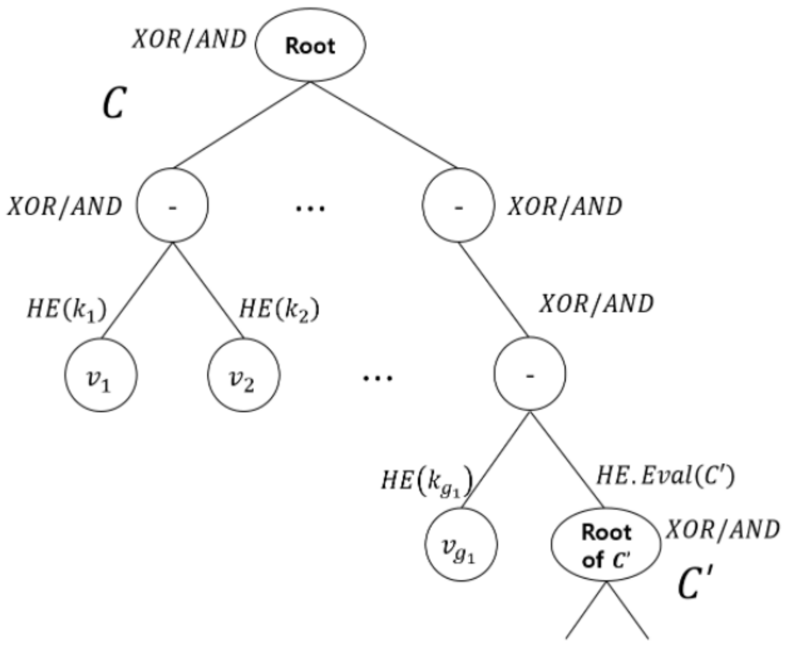


Figure 5.8: Merge algorithm in Parcel-D protocol

Partition Protocol

Assume that a network fault causes a partition of the group with g members. From the remaining member, this event seems to be a concurrent 'leave' of multiple members.

Starting from the leftmost leaf node of the Boolean circuit C, we run Leave($\tilde{v}_i, C$) if $\tilde{v} \in \tilde{V}_j$. But, instead of removing all long-term secret keys of the vertices in $\tilde{V}_j$, the server keeps those long-term secret keys in separate box.

Relation on Boolean Circuit

For all membership events, one Boolean circuit is the minor of the other Boolean circuit. For addition event like Join and Merge algorithms, an original circuit C is the minor of the new circuits C_{Join} and C_{Merge}. Similarly, for revocation event like Leave and Partition algorithms, new circuits C_{Leave} and $C_{Partition}$ are the minor of the original circuit C. With this property, we can check the validity of the circuit after each membership event.

Chapter 6. Key-reusable GKE in Static Setting

6.1 Construction of Key-reusable GKE

Given $R_q = \mathbb{Z}_q[x]/(x^n+1)$ with a prime q and $a \leftarrow R_q$, all parties calculate X_i and agree on "close" values $b_0 \approx b_1 \approx \cdots \approx b_{N-1}$ in **Round 3** and **Key Computation**. Then, the party P_{N-1} runs a key reconciliation mechanism to allow all parties to get a common value $k = k_0 = k_1 = \cdots = k_{N-1}$.

Similar to [24], a party P_i applies pasteurization techniques to z_{i+1} and z_{i-1} by two parties P_{i+1} and P_{i-1} in **Round 2**. We have two pasteurized values $\bar{z}_{i+1} = z_{i+1} + ah_{i+1} + f_{i+1}$ and $\hat{z}_{i-1} = z_{i-1} + ah_{i-1} + g_{i-1}$ for each party P_i where $h_i = \mathcal{H}(z_i)$ is precomputed and $f_i, g_i \leftarrow \chi_{\sigma_1}$. Then, X_i becomes $(\bar{z}_{i+1} - \hat{z}_{i-1})(s_i + h_i) + e'_i$ for the correctness of the protocol.

Since we only show that k is difficult to compute for a passive adversary in the security proof, we hash k using random oracle $\mathcal{H}$ to get the session group secret key sk, which is indistinguishable from random. The detailed description of our unauthenticated GKE is given in **Protocol 4**, named as Krug (<u>K</u>ey <u>reu</u>sable <u>g</u>roup key exchange).

Krug shows a counter-example that resists our two-neighbour attack since the dishonest behaviour of two parties doesn't guarantee to get any information of the secret key of the party P_{N-1} from the output of the key reconciliation function. The authenticated one which is secure against an active adversary can be derived by applying known techniques like Katz-Yung compiler [38].

6.2 Security Analysis

6.2.1 Correctness Proof

Note that group key exchange is correct when all parties agree on the same secret key. **Lemmas 2 and 3** are from Apon *et al.*'s paper.

Lemma 2 ([2]). Given s_i for all i defined in the group key exchange protocol, fix $c = \sqrt{\frac{2\rho}{\pi \log(e)}}$ and let bound_ρ be the event that for all $i \in [N]$ and all coefficients $j \in [n]$, $|s_i[j]|, |e_i[j]|, |e'_i[j]|, |e''_{N-1}[j]| \leq c\sigma_1$ except $|e'_0[j]| \leq c\sigma_2$. Then

$$\Pr[\mathsf{bound}_\rho] \geq 1 - 2^\rho.$$

Protocol 4: $\mathsf{Krug}(P\,[0,1,\cdots,N-1]\,,a,\mathcal{H},\sigma_1,\sigma_2)$

(Round 1) For each party P_i for $i = 0$ to $N-1$, do the following in parallel.

1. Computes $z_i = as_i + e_i$ where $s_i, e_i \leftarrow \chi_{\sigma_1}$;
2. Broadcasts z_i;

(Round 2) For $i = 0$ to $N-1$, do the following in parallel.

1. If $i = 0$, party P_0 samples $e'_0 \leftarrow \chi_{\sigma_2}$ and otherwise, party P_i samples $e'_i \leftarrow \chi_{\sigma_1}$;
2. Each party P_i computes $\bar{z}_{i+1} = z_{i+1} + ah_{i+1} + f_{i+1}$ and $\hat{z}_{i-1} = z_{i-1} + ah_{i-1} + g_{i-1}$ where $h_i = \mathcal{H}(z_i)$ and $f_i, g_i \leftarrow \chi_{\sigma_1}$;
3. Each party P_i broadcasts $X_i = (\bar{z}_{i+1} - \hat{z}_{i-1})\,(s_i + h_i) + e'_i$;

(Round 3) For a party P_{N-1}, do the following.

1. Samples $e''_{N-1} \leftarrow \chi_{\sigma_1}$ and calculates
   $b_{N-1} = \hat{z}_{N-2}N(s_{N-1} + h_{N-1}) + (N-1)X_{N-1} + (N-2)X_0 + \cdots + X_{N-3} + e''_{N-1}$;
2. Runs $\mathsf{recMsg}(\cdot)$ to output $(\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1})$;
3. Broadcasts rec and gets session key as $\mathsf{sk}_{N-1} = \mathcal{H}(k_{N-1})$;

(Key Computation) For each party P_i $(i \neq N-1)$.

1. Computes $b_i = \hat{z}_{i-1}N(s_i + h_i) + (N-1)X_i + (N-2)X_{i+1} + \cdots + X_{i+N-2}$;
2. Runs $\mathsf{recKey}()$ to output $k_i = \mathsf{recKey}\,(b_i, \mathsf{rec})$ and gets session key as $\mathsf{sk}_i = \mathcal{H}(k_i)$;

Proof. Since the complementary error function $erfc(x) = \frac{2}{\pi}\int_x^\infty \exp(-t^2)dt \le \exp(-x^2)$, we get

$$\begin{aligned}\Pr[v \leftarrow D_{\mathbb{Z}_q,\sigma}; |v| \ge c\sigma + 1] &\le 2 \sum_{x=\lfloor c\sigma+1 \rceil}^{\infty} D_{\mathbb{Z}_q,\sigma}(x) \\ &\le \frac{2}{\sigma}\int_{c\sigma}^{\infty} \exp(\frac{-\pi x^2}{\sigma^2})dx \\ &= \frac{2}{\pi}\int_{\frac{\sqrt{\pi}}{\sigma}(c\sigma)}^{\infty} \exp(-t^2)dt \le \exp(-c^2\pi).\end{aligned}$$

Then we have $3nN$ samplings from $D_{\mathbb{Z}_q,\sigma_1}$ and n samplings from $D_{\mathbb{Z}_q,\sigma_2}$ in our protocol. Under the assumption that $3nN + n \le \exp(c^2\pi/2)$, we have

$$\begin{aligned}\Pr[\mathsf{bound}_\rho] &= (1 - \Pr[v \leftarrow D_{\mathbb{Z}_q,\sigma_1}; |v| \ge c\sigma_1 + 1])^{3nN}(1 - \Pr[e_0' \leftarrow D_{\mathbb{Z}_q,\sigma_2}; |v| \ge c\sigma_2 + 1])^n \\ &\ge 1 - (3nN + n)\cdot \exp(-c^2\pi) \ge 1 - \exp(c^2\pi/2) \\ &\ge 1 - 2^{-\rho}.\end{aligned}$$

□

Lemma 3 ([2])**.** Given bound_ρ defined in Lemma 2, let $\mathsf{product}_{s_i,\, e_j}$ be the event that for all v-th coefficient, $|(s_i \cdot e_j)[v]| \le \sqrt{n}\rho^{3/2}\sigma_1^2$. Then

$$\Pr[\mathsf{product}_{s_i \cdot e_j} \mid \mathsf{bound}_\rho] \ge 1 - n \cdot 2 \cdot 2^{-2\rho}$$

Proof. Note that for $l \in [n]$, $(s_i)_l$ denotes the l-th coefficient of s_i and we can express $s_i = \sum_{l=0}^{n-1}(s_i)_l X^l$. Since we take $X^n + 1$ as modulus of R, $(s_i e_j)_l = \sum_{k=0}^{n-1}(s_i)_k (e_j)^*_{l-k} X^l$ where $(e_j)^*_{l-k}$ is $(e_j)_{l-k}$ if $l - k \ge 0$ and $-(e_j)_{l-k}$ otherwise. Thus, under bound_ρ, specifically $|(s_i)_l|,\ |(e_j)_l| \le c\sigma_1$ where $c = \sqrt{\frac{2\rho}{\pi\cdot\log(e)}}$, by Hoeffding's inequality [35], we can get

$$\begin{aligned}\Pr[|(s_i e_j)_l| \ge \gamma \mid \mathsf{bound}_\rho] &= \Pr\left[\left|\sum_{k=0}^{n-1}(s_i)_k(e_j)_{l-k}\right| \ge \gamma \mid \mathsf{bound}_\rho\right] \\ &\le 2\cdot\exp\left(\frac{-2\gamma^2}{n(2c^2\sigma_1^2)^2}\right).\end{aligned}$$

(Note that $(s_i)_k(e_j)_{l-k}$ is an independent random variable with mean 0 in interval $[-c^2\sigma_1^2,\ c^2\sigma_1^2]$.) If we take $\gamma = \sqrt{n}\rho^{3/2}\sigma_1^2$, then we get

$$\Pr[|(s_i e_j)_l| \ge \gamma \mid \mathsf{bound}_\rho] \le 2\cdot\exp(\frac{-\rho^3}{2c^4}) \le 2^{-2\rho+1}$$

Thus, after union all bound, we have

$$\begin{aligned}\Pr[\mathsf{product}_{s_i,\, e_j} \mid \mathsf{bound}_\rho] &= \Pr[\forall l,\ |(s_i e_j)_l| \le \sqrt{n}\rho^{3/2}\sigma_1^2] \\ &\ge 1 - n\cdot 2\cdot 2^{-2\rho}.\end{aligned}$$

□

In **Theorem 6**, we give a condition that Krug is correct. Most part of our correctness is similar to that of Apon *et al.*'s except some modification on error bound.

Theorem 6. For a fixed ρ, and assume that

$$(4N^2 + 4N - 4)\sqrt{n}\rho^{3/2}\sigma_1^2 + \{\frac{(2N-1)^2}{8} + 1\}\sigma_1 + (N-2)\sigma_2 \le \beta_{\mathsf{Rec}}.$$

Then all participants in a group have the same key except with probability at most $2^{-\rho}$.

Proof. As mentioned in Section 2.5, we will show that all parties have the same secret key except with negligible probability. To hold this, we claim that if for all $i \in [N]$ and $j \in [n]$, the j-th coefficient of $|b_{N-1} - b_i| \le \beta_{\mathsf{Rec}}$, then $k_i = k_{N-1}$. After some careful computation, we have

$$\begin{aligned}
b_{N-1} - b_i = e''_{N-1} &+ (e_{N-2} + g_{N-2})(s_{N-1} + h_{N-1}) - (e_{i-1} + f_{i-1})(s_i + h_i) \\
&+ \sum_{j=0}^{N-2} (N-1-j)(e'_{N-1+j} - e'_{i+j}) \\
&+ \sum_{j=0}^{N-2} (N-1-j)\{(e_j + f_j)(s_{N-1+j} + h_{N-1+j}) - (e_{i+1+j} + f_{i+1+j})(s_{i+j} + h_{i+j})\} \\
&+ \sum_{j=0}^{N-3} (N-2-j)\{(e_{N-1+j} + g_{N-1+j})(s_j + h_j) - (e_{i+j} + g_{i+j})(s_{i+j+1} + h_{i+j+1})\}.
\end{aligned}$$

Now observe how many terms are in $b_{N-1} - b_i$. Like [4], there are at most $N^2 + 2N$ terms in form of $s_i \cdot e_j$, at most $\frac{(2N-1)^2}{8}$ terms in form of e'_k sampled from χ_{σ_1}, at most $N-2$ terms of e'_0 sampled from χ_{σ_2}, and one term of e''_{N-1}. Also, from pasteurization, there are at most $N^2 + 2N$ terms in form of $h_i \cdot e_j$, at most $\frac{N^2}{2}$ terms in form of $f_i \cdot e_j$ and $f_i \cdot h_j$ and at most $\frac{N^2-4}{2}$ terms in form of $g_i \cdot e_j$ and $g_i \cdot h_j$.

Let $\mathsf{product_{ALL}}$ be the event that for all terms in forms of $s_i \cdot e_j$, $h_i \cdot e_j$, $f_i \cdot e_j$, $f_i \cdot h_j$, $g_i \cdot e_j$ and $g_i \cdot h_j$, each coefficient of one of these forms is bounded by $\sqrt{n}\rho^{3/2}\sigma_1^2$. Under an assumption that $2n\{2(N^2 + 2N) + N^2 + (N^2 - 4)\} \le 2^\rho$, by **Lemma 3** we can get

$$\Pr[\overline{\mathsf{product_{ALL}}} \mid \mathsf{bound}_\rho] \le 2n\{2(N^2 + 2N) + N^2 + (N^2 - 4)\} \cdot 2^{-2\rho} \le 2^{-\rho}$$

Let fail be the event that at least one of parties does not agree on the same key. Given a condition that $\{2(N^2 + 2N) + N^2 + (N^2 - 4)\}\sqrt{n}\rho^{3/2}\sigma_1^2 + (\frac{(2N-1)^2}{8} + 1)\sigma_1 + (N-2)\sigma_2 \le \beta_{\mathsf{Rec}}$, by **Lemma 2** and the above inequality we have

$$\begin{aligned}
\Pr[\mathsf{fail}] &= \Pr[\mathsf{fail} \mid \mathsf{bound}_\rho] \cdot \Pr[\mathsf{bound}_\rho] + \Pr[\mathsf{fail} \mid \overline{\mathsf{bound}_\rho}] \cdot \Pr[\overline{\mathsf{bound}_\rho}] \\
&\le \Pr[\overline{\mathsf{product_{ALL}}} \mid \mathsf{bound}_\rho] \cdot 1 + 1 \cdot \Pr[\overline{\mathsf{bound}_\rho}] \\
&\le 2 \cdot 2^{-\rho}.
\end{aligned}$$

Therefore, all parties agree on the same key except with probability $2 \cdot 2^{-\rho}$. □

Note that e'_0 is not considered in the form of e'_k sampled from χ_{σ_1} since e'_0 is sampled from χ_{σ_2}. We have different number of coefficient for e'_k since ADGK19 included the term e'_0 but we calculate the exact number of coefficient for e'_k by removing e'_0.

6.2.2 Security Proof

Let Expt_i refer to the i-th experiment where Krug is executed to obtain output a pair $(\mathsf{T}, \mathsf{sk})$ of the transcript $\mathsf{T} = (z_i, X_i, \mathsf{rec})$ and the session key sk. We provide $(\mathsf{T}, \mathsf{sk})$ to the adversary $\mathcal{A}$ and allow $\mathcal{A}$ to interact with the random oracle used when executing Krug.

Our goal is to show that the advantage $\mathsf{Adv}^{\mathsf{KE}}_{\mathsf{Krug}}(\mathcal{A})$ of $\mathcal{A}$ in distinguishing a pair $(\mathsf{T}, \mathsf{sk})$ distributed according to Expt_0 with samples $(\mathsf{T}, \mathsf{sk}_0)$, in which T is distributed the same way but sk_0 is a uniformly chosen key, is negligible. Also, we define $\mathsf{Adv}^{\mathsf{KE}}_{\mathsf{Krug}}(t, q_E)$ to be the maximum advantage of any adversary running in time t with at most q_E queries to the random oracle.

Theorem 7. For our key-reusable group key exchange Krug, $8N\sqrt{n}\lambda^{3/2}\sigma_1^2 + (N-1)\sigma_1 \le \beta_{\mathsf{Rényi}}$ and $\sigma_2 = \Omega\left(\beta_{\mathsf{Rényi}}\sqrt{n/\log\lambda}\right)$. Then,

$$\mathsf{Adv}^{\mathsf{KE}}_{\mathsf{Krug}}(t, q_E) \le 2^{-\lambda+1} + \sqrt{\left(N' \cdot \mathsf{Adv}^{RLWE}_{n,q,\chi_{\sigma_1},3}(t_1) + \mathsf{Adv}_{\mathsf{KeyRec}}(t_2) + \frac{q_E}{2^\lambda}\right) \cdot \frac{\exp\left(2\pi n(\beta_{\mathsf{Rényi}}/\sigma_2)^2\right)}{1-2^{-\lambda+1}}}$$

where $N' = \left\lceil \frac{N}{3} \right\rceil + N$, $t_1 = t + \mathcal{O}(N \cdot t_{ring}), t_2 = t + \mathcal{O}(N \cdot t_{ring})$ such that t_{ring} is the maximum time required to make operations in R_q.

Proof. Let Query be the event that $\mathcal{A}$ queries k_{N-1} to the random oracle. Since this is the only way that $\mathcal{A}$ can distinguish $\mathsf{sk} = \mathcal{H}(k_{N-1})$ from an independent uniform value, we need to show that $\Pr_i[\mathsf{Query}]$ is small where $\Pr_i[\cdot]$ denotes the probability of an event in Expt_i.

Experiment 0. This is the original experiment that is equal to the procedure of Krug.

$$\mathsf{Expt}_0 := \left\{ \begin{array}{ll} a \leftarrow R_q; s_i, e_i, f_i, g_i \leftarrow \chi_{\sigma_1} \text{ for } i \in [N]; & \\ z_i = as_i + e_i, h_i = \mathcal{H}(z_i) \text{ for } i \in [N]; & \\ \bar{z}_i = z_i + ah_i + f_i, \hat{z}_i = z_i + ah_i + g_i \text{ for } i \in [N]; & \\ e'_0 \leftarrow \chi_{\sigma_2};\ e'_i \leftarrow \chi_{\sigma_1} \text{ for } 1 \le i \le N-1; & \\ X_i = (\bar{z}_{i+1} - \hat{z}_{i-1})(s_i + h_i) + e'_i \text{ for } i \in [N]; & \\ e''_{N-1} \leftarrow \chi_{\sigma_1}; & : (\mathsf{T}, \mathsf{sk}) \\ b_{N-1} = \hat{z}_{N-2}N(s_{N-1} + h_{N-1}) + (N-1)X_{N-1} + & \\ \quad (N-2)X_0 + \cdots + X_{N-3} + e''_{N-1}; & \\ (\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1}); \mathsf{sk} = \mathcal{H}(k_{N-1}); & \\ \mathsf{T} = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \cdots, X_{N-1}, \mathsf{rec}) & \end{array} \right\}$$

$$\text{Since } \Pr[\mathcal{A} \text{ wins}] = \frac{1}{2} + \mathsf{Adv}^{\mathsf{KE}}_{\mathsf{Krug}}(t, q_E) = \Pr_0[\mathsf{Query}] + \Pr_0\left[\overline{\mathsf{Query}}\right] \cdot \frac{1}{2},$$

$$\mathsf{Adv}^{\mathsf{KE}}_{\mathsf{Krug}}(t, q_E) \le \Pr_0[\mathsf{Query}].$$

Experiment 1. We replace X_0 into $X'_0 = -\sum_{i=1}^{N-1} X_i + e'_0$. The rest is the same as the previous experiment.

$$\mathsf{Expt}_1 := \left\{ \begin{array}{ll} a \leftarrow R_q; s_i, e_i, f_i, g_i \leftarrow \chi_{\sigma_1} \text{ for } i \in [N]; & \\ z_i = as_i + e_i, h_i = \mathcal{H}(z_i) \text{ for } i \in [N]; & \\ \bar{z}_i = z_i + ah_i + f_i, \hat{z}_i = z_i + ah_i + g_i \text{ for } i \in [N]; & \\ e'_0 \leftarrow \chi_{\sigma_2};\ e'_i \leftarrow \chi_{\sigma_1} \text{ for } 1 \le i \le N-1; & \\ X'_0 = -\sum_{i=1}^{N-1} X_i + e'_0; & \\ X_i = (\bar{z}_{i+1} - \hat{z}_{i-1})(s_i + h_i) + e'_i \text{ for } 1 \le i \le N-1; & : (\mathsf{T}, \mathsf{sk}) \\ e''_{N-1} \leftarrow \chi_{\sigma_1}; & \\ b_{N-1} = \hat{z}_{N-2}N(s_{N-1} + h_{N-1}) + (N-1)X_{N-1} + & \\ \quad (N-2)X'_0 + \cdots + X_{N-3} + e''_{N-1}; & \\ (\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1}); \mathsf{sk} = \mathcal{H}(k_{N-1}); & \\ \mathsf{T} = (z_0, z_1, \cdots, z_{N-1}, X'_0, X_1, \cdots, X_{N-1}, \mathsf{rec}) & \end{array} \right\}$$

Lemma 4. Given two distributions of X_0 and X'_0, if we have $8N\sqrt{n}\lambda^{3/2}\sigma_1^2 + (N-1)\sigma_1 \le \beta_{\mathsf{Rényi}}$, then

$$\Pr_0[\mathsf{Query}] \le 2^{-\lambda+1} + \sqrt{\Pr_1[\mathsf{Query}] \cdot \frac{\exp\left(2\pi n(\beta_{\mathsf{Rényi}}/\sigma_2)^2\right)}{1-2^{-\lambda+1}}}$$

using the property of Rényi divergence.

Proof. Note that we define the random variables X_0, X_0' in both experiments Expt_0 and Expt_1. We define Error and main as

$$\begin{aligned}\mathsf{Error} &= \sum_{i=0}^{N-1}(e_{i+1} - e_{i-1} + f_{i+1} - g_{i-1})(s_i + h_i) + \sum_{i=1}^{N-1} e_i' \text{ and}\\ \mathsf{main} &= (\bar{z}_1 - \hat{z}_{N-1})(s_0 + h_0) - \mathsf{Error}.\end{aligned}$$

Then,

$$X_0 = \mathsf{main} + \mathsf{Error} + e_0' \text{ and } X_0' = \mathsf{main} + e_0',$$

where $e_0' \leftarrow \sigma_2$. We check whether Rényi divergence between two distributions of X_0 and X_0' is small using **Theorem 1**. Let $\mathsf{bound}_{\mathsf{Error}}$ be the event that for all participants j, $|\mathsf{Error}[j]| \leq \beta_{\mathsf{Rényi}}$. Then,

$$|\mathsf{Error}[j]| = \left|\left(\sum_{i=0}^{N-1}(e_{i+1} - e_{i-1} + f_{i+1} - g_{i-1})(s_i + h_i) + \sum_{i=1}^{N-1} e_i'\right)[j]\right|.$$

Set $c = \sqrt{\frac{2\lambda}{\pi \log e}}$ and let bound be the event that $|e_0'[j]| \leq c\sigma_2$, $|s_i[j]|, |e_i[j]|$, $|e''_{N-1}[j]| \leq c\sigma_1$, and $|e_i'[j]| \leq c\sigma_1$ for all $i > 0$ and j.

From **Lemmas 2 and 3** described in **Appendix A**, we have $\Pr[\mathsf{bound}] \geq 1 - 2^{-\lambda}$ and $\Pr[|(s_i e_j)[v]| \leq \sqrt{n}\lambda^{3/2}\sigma_1^2 \mid \mathsf{bound}] \geq 1 - 2^{-2\lambda+1}$. With a union bound, we have

$$\Pr\left[\forall j : |\mathsf{Error}[j]| \leq 8N\sqrt{n}\lambda^{3/2}\sigma_1^2 + (N-1)\sigma_1 \mid \mathsf{bound}\right] \geq 1 - 16Nn2^{-2\lambda}.$$

If we assume $16Nn \leq 2^{\lambda}$, we derive that $\Pr[\mathsf{bound}_{\mathsf{Error}}] \geq 1 - 2^{-\lambda+1}$.

We have $\mathrm{RD}_2(\mathsf{Error} + \chi_{\sigma_2} \| \chi_{\sigma_2}) \leq \exp(2\pi n(\beta_{\mathsf{Rényi}}/\sigma)^2)$ from the property of Rényi divergence. Thus,

$$\begin{aligned}\Pr_0[\mathsf{Query}] &\leq \Pr_0[\mathsf{Query} \mid \mathsf{bound}_{\mathsf{Error}}] + \Pr_0[\overline{\mathsf{bound}_{\mathsf{Error}}}]\\ &\leq \Pr_0[\mathsf{Query} \mid \mathsf{bound}_{\mathsf{Error}}] + 2^{-\lambda+1}\\ &\leq \sqrt{\Pr_1[\mathsf{Query} \mid \mathsf{bound}_{\mathsf{Error}}] \cdot \exp\left(2\pi n(\beta_{\mathsf{Rényi}}/\sigma_2)^2\right)} + 2^{-\lambda+1}\\ &\leq \sqrt{\Pr_1[\mathsf{Query}] \cdot \frac{\exp\left(2\pi n(\beta_{\mathsf{Rényi}}/\sigma_2)^2\right)}{\Pr_1[\mathsf{bound}_{\mathsf{Error}}]}} + 2^{-\lambda+1}\\ &\leq \sqrt{\Pr_1[\mathsf{Query}] \cdot \frac{\exp\left(2\pi n(\beta_{\mathsf{Rényi}}/\sigma_2)^2\right)}{1 - 2^{-\lambda+1}}} + 2^{-\lambda+1}.\end{aligned}$$

□

Lemma 5. The probability that $\mathcal{A}$ queries k_{N-1} to the random oracle in Expt_1 is negligible when key reconciliation mechanism is secure and our RLWE problem is hard. *i.e.*

$$\Pr_1[\mathsf{Query}] \leq (\left\lceil\frac{N}{3}\right\rceil + N) \cdot \mathsf{Adv}^{RLWE}_{n,q,\chi_{\sigma_1},3}(t_1) + \mathsf{Adv}_{\mathsf{KeyRec}}(t_2) + \frac{q_E}{2^{\lambda}}$$

where $t_1 = t + \mathcal{O}(N \cdot t_{ring})$, $t_2 = t + \mathcal{O}(N \cdot t_{ring})$ such that t_{ring} is the maximum time required to make operations in R_q.

Proof. We prove this lemma by considering a sequence of experiments.

Experiment 2. We replace all z_i's into uniform elements in R_q. The rest is the same as the previous experiment.

$$\mathsf{Expt}_2 := \left\{ \begin{array}{l} a \leftarrow R_q, z_i \leftarrow R_q \text{ for all } i \in [N]\,; \\ f_i, g_i \leftarrow \chi_{\sigma_1}, h_i = \mathcal{H}(z_i) \text{ for } i \in [N]\,; \\ \bar{z}_i = z_i + ah_i + f_i, \hat{z}_i = z_i + ah_i + g_i \text{ for } i \in [N]\,; \\ e'_0 \leftarrow \chi_{\sigma_2};\ e'_i \leftarrow \chi_{\sigma_1} \text{ for } 1 \le i \le N-1; \\ X'_0 = -\sum_{i=1}^{N-1} X_i + e'_0; \\ X_i = (\bar{z}_{i+1} - \hat{z}_{i-1})(s_i + h_i) + e'_i \text{ for } 1 \le i \le N-1; \quad : (\mathsf{T}, \mathsf{sk}) \\ e''_{N-1} \leftarrow \chi_{\sigma_1}; \\ b_{N-1} = \hat{z}_{N-2} N(s_{N-1} + h_{N-1}) + (N-1)X_{N-1} + \\ \qquad (N-2)X'_0 + \cdots + X_{N-3} + e''_{N-1}; \\ (\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1}); \mathsf{sk} = \mathcal{H}(k_{N-1}); \\ \mathsf{T} = (z_0, z_1, \cdots, z_{N-1}, X'_0, X_1, \cdots, X_{N-1}, \mathsf{rec}) \end{array} \right\}$$

Between **Experiment 1** and **Experiment 2**, we replace N RLWE instances into random. In our proof, we assume that 3 RLWE instances are changed into random at once. Then,

$$|\Pr_2[\mathsf{Query}] - \Pr_1[\mathsf{Query}]| \le \left\lceil \frac{N}{3} \right\rceil \cdot \mathsf{Adv}^{RLWE}_{n,q,\chi_{\sigma_1},3}(t_1),$$

where $t_1 = t + \mathcal{O}(N \cdot t_{ring})$ and t_{ring} is the time required to perform operations in R_q.

The rest of the proof follows the procedure of ADGK19. We design distribution of $(\mathsf{T}, \mathsf{sk})$ in Experiment $3, 3j+1, 3j+2$ and $3j+3$ as below:

Experiment 3. We replace $\bar{z}_0, \hat{z}_0$ and $\hat{z}_{N-1}$ into uniform elements in R_q. The rest is the same as the previous experiment.

$$\mathsf{Expt}_3 := \left\{ \begin{array}{l} a \leftarrow R_q, z_i \leftarrow R_q \text{ for all } i \in [N]\,; \\ \bar{z}_0, \hat{z}_0, \hat{z}_{N-1} \leftarrow R_q; \\ h_i = \mathcal{H}(z_i) \text{ for } i \in [N]\,; \\ f_i \leftarrow \chi_{\sigma_1}, \bar{z}_i = z_i + ah_i + f_i \text{ for } i \ge 1; \\ g_i \leftarrow \chi_{\sigma_1}, \hat{z}_i = z_i + ah_i + g_i \text{ for } 1 \le i \le N-2; \\ e'_0 \leftarrow \chi_{\sigma_2};\ e'_i \leftarrow \chi_{\sigma_1} \text{ for } 1 \le i \le N-1; \\ X'_0 = -\sum_{i=1}^{N-1} X_i + e'_0;\ X_1, \cdots, X_j \leftarrow R_q; \qquad : (\mathsf{T}, \mathsf{sk}) \\ \forall i \ge j,\ X_i = (\bar{z}_{i+1} - \hat{z}_{i-1})(s_i + h_i) + e'_i; \\ e''_{N-1} \leftarrow \chi_{\sigma_1}; \\ b_{N-1} = \hat{z}_{N-2} N(s_{N-1} + h_{N-1}) + (N-1)X_{N-1} + \\ \qquad (N-2)X'_0 + \cdots + X_{N-3} + e''_{N-1}; \\ (\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1}); \mathsf{sk} = \mathcal{H}(k_{N-1}); \\ \mathsf{T} = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \cdots, X_{N-1}, \mathsf{rec}) \end{array} \right\}$$

Between **Experiment 2** and **Experiment 3**, we replace three RLWE instances into random. Hence,

$$|\Pr_3[\mathsf{Query}] - \Pr_2[\mathsf{Query}]| \le \mathsf{Adv}^{RLWE}_{n,q,\chi_{\sigma_1},3}(t_1),$$

where $t_1 = t + \mathcal{O}(N \cdot t_{ring})$ and t_{ring} is the time required to perform operations in R_q.

Remark. We change $\hat{z}_{N-1}$ into uniform in Experiment 3 to control the maximum number of RLWE instances that becomes uniform in each experiment to be three.

Experiment $3j+1$. We replace $\hat{z}_{j-1}$ and X_j into $\bar{z}_{j+1}-r_j$ and $r_j s_j + e'_j$, respectively, where $r_j \leftarrow R_q$. The rest is the same as the previous experiment.

$$
\mathsf{Expt}_{3j+1} := \left\{ \begin{array}{l}
a \leftarrow R_q, z_i \leftarrow R_q \text{ for all } i \in [N]\,; \\
\bar{z}_i, \leftarrow R_q \text{ for } i \leq j-1;\ \hat{z}_{N-1} \leftarrow R_q; \\
\hat{z}_i \leftarrow R_q \text{ for } i \leq j-2;\ \hat{z}_{j-1} = \bar{z}_{j+1} - r_j; \\
h_i = \mathcal{H}(z_i) \text{ for } i \in [N]\,; \\
f_i \leftarrow \chi_{\sigma_1}, \bar{z}_i = z_i + ah_i + f_i \text{ for } i \geq j+1; \\
g_i \leftarrow \chi_{\sigma_1}, \hat{z}_i = z_i + ah_i + g_i \text{ for } j+1 \leq i \leq N-2; \\
e'_0 \leftarrow \chi_{\sigma_2};\ e'_j, \cdots, e'_{N-1} \leftarrow \chi_{\sigma_1}; \\
X'_0 = -\sum_{i=1}^{N-1} X_i + e'_0;\ X_1, \cdots, X_{j-1} \leftarrow R_q; \\
\forall i \geq j,\ X_i = (\bar{z}_{i+1} - \hat{z}_{i-1})(s_i + h_i) + e'_i; \\
e''_{N-1} \leftarrow \chi_{\sigma_1}; \\
b_{N-1} = \hat{z}_{N-2} N (s_{N-1} + h_{N-1}) + (N-1) X_{N-1} + \\
\qquad (N-2) X'_0 + \cdots + X_{N-3} + e''_{N-1}; \\
(\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1}); \mathsf{sk} = \mathcal{H}(k_{N-1}); \\
\mathsf{T} = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \cdots, X_{N-1}, \mathsf{rec})
\end{array} : (\mathsf{T}, \mathsf{sk}) \right\}
$$

Since r_j is uniform, then $\bar{z}_{j+1}-r_j$ is also uniform. Thus, **Experiment** $3j+1$ is identical to **Experiment** $3j$, *i.e.*, $\Pr_{3j+1}[\mathsf{Query}] = \Pr_{3j}[\mathsf{Query}]$.

Experiment $3j+2$. We replace $\bar{z}_j, \hat{z}_j$ and X_j into uniform element in R_q. The rest is the same as the previous experiment.

$$
\mathsf{Expt}_{3j+2} := \left\{ \begin{array}{l}
a \leftarrow R_q, z_i \leftarrow R_q \text{ for all } i \in [N]\,; \\
\bar{z}_i, \leftarrow R_q \text{ for } i \leq j;\ \hat{z}_{N-1} \leftarrow R_q; \\
\hat{z}_i \leftarrow R_q \text{ for } i \leq j-2;\ \hat{z}_{j-1} = \bar{z}_{j+1} - r_j;\ \hat{z}_j \leftarrow R_q; \\
h_i = \mathcal{H}(z_i) \text{ for } i \in [N]\,; \\
f_i \leftarrow \chi_{\sigma_1}, \bar{z}_i = z_i + ah_i + f_i \text{ for } i \geq j+1; \\
g_i \leftarrow \chi_{\sigma_1}, \hat{z}_i = z_i + ah_i + g_i \text{ for } j+1 \leq i \leq N-2; \\
e'_0 \leftarrow \chi_{\sigma_2};\ e'_j, \cdots, e'_{N-1} \leftarrow \chi_{\sigma_1}; \\
X'_0 = -\sum_{i=1}^{N-1} X_i + e'_0;\ X_1, \cdots, X_j \leftarrow R_q; \\
\forall i \geq j+1,\ X_i = (\bar{z}_{i+1} - \hat{z}_{i-1})(s_i + h_i) + e'_i; \\
e''_{N-1} \leftarrow \chi_{\sigma_1}; \\
b_{N-1} = \hat{z}_{N-2} N (s_{N-1} + h_{N-1}) + (N-1) X_{N-1} + \\
\qquad (N-2) X'_0 + \cdots + X_{N-3} + e''_{N-1}; \\
(\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1}); \mathsf{sk} = \mathcal{H}(k_{N-1}); \\
\mathsf{T} = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \cdots, X_{N-1}, \mathsf{rec})
\end{array} : (\mathsf{T}, \mathsf{sk}) \right\}
$$

Between **Experiment** $3j+1$ and **Experiment** $3j+2$, we replace three RLWE instances $\bar{z}_j, \hat{z}_j, X_j$ into random. Hence,

$$|\Pr_{3j+2}[\mathsf{Query}] - \Pr_{3j+1}[\mathsf{Query}]| \leq \mathsf{Adv}^{RLWE}_{n,q,\chi_{\sigma_1},3}(t_1)$$

where t_1 is the time to solve RLWE problem which is the sum of t and some minor overhead $\mathcal{O}(t_{ring})$ for simulation.

Experiment $3j+3$. We replace $\hat{z}_{j-1}$ into uniform elements in R_q. The rest is the same as the previous experiment.

$$\mathsf{Expt}_{3j+3} := \left\{ \begin{array}{l} a \leftarrow R_q, z_i \leftarrow R_q \text{ for all } i \in [N]\,; \\ \bar{z}_i, \hat{z}_i \leftarrow R_q \text{ for } i \leq j;\ \hat{z}_{N-1} \leftarrow R_q; \\ h_i = \mathcal{H}(z_i) \text{ for } i \in [N]\,; \\ f_i \leftarrow \chi_{\sigma_1}, \bar{z}_i = z_i + ah_i + f_i \text{ for } i \geq j+1; \\ g_i \leftarrow \chi_{\sigma_1}, \hat{z}_i = z_i + ah_i + g_i \text{ for } j+1 \leq i \leq N-2; \\ e'_0 \leftarrow \chi_{\sigma_2};\ e'_j, \cdots, e'_{N-1} \leftarrow \chi_{\sigma_1}; \\ X'_0 = -\sum_{i=1}^{N-1} X_i + e'_0;\ X_1, \cdots, X_j \leftarrow R_q; \\ \forall i \geq j+1,\ X_i = (\bar{z}_{i+1} - \hat{z}_{i-1})(s_i + h_i) + e'_i; \\ e''_{N-1} \leftarrow \chi_{\sigma_1}; \\ b_{N-1} = \hat{z}_{N-2} N(s_{N-1} + h_{N-1}) + (N-1)X_{N-1} + \\ \qquad (N-2)X'_0 + \cdots + X_{N-3} + e''_{N-1}; \\ (\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1}); \mathsf{sk} = \mathcal{H}(k_{N-1}); \\ \mathsf{T} = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \cdots, X_{N-1}, \mathsf{rec}) \end{array} \right. : (\mathsf{T}, \mathsf{sk}) \right\}$$

Since both $\bar{z}_{j+1} - r_j$ and $\hat{z}_{j-1}$ are uniform, **Experiment** $3j+3$ is identical to **Experiment** $3j+2$, *i.e.*, $\Pr_{3j+3}[\mathsf{Query}] = \Pr_{3j+2}[\mathsf{Query}]$.

Experiment $3N-2$. We set $\hat{z}_{N-2} = r_2, X_{N-1} = r_1(s_{N-1} + h_{N-1}) + e'_{N-1}, \bar{z}_0 = r_1 + r_2$ where $r_1, r_2 \leftarrow R_q$. The rest is the same as the previous experiment.

$$\mathsf{Expt}_{3N-2} := \left\{ \begin{array}{l} a, r_1, r_2 \leftarrow R_q; z_i \leftarrow R_q \text{ for all } i \in [N]\,; \\ \bar{z}_0 = r_1 + r_2, \bar{z}_i \leftarrow R_q \text{ for } 1 \leq i \leq N-2; \\ \hat{z}_{N-1} \leftarrow R_q; \\ \hat{z}_i \leftarrow R_q \text{ for } i \leq N-3;\ \hat{z}_{N-2} = r_2; \\ h_i = \mathcal{H}(z_i) \text{ for } i \in [N]\,; \\ f_{N-1} \leftarrow \chi_{\sigma_1}, \bar{z}_{N-1} = z_{N-1} + ah_{N-1} + f_{N-1}; \\ e'_0 \leftarrow \chi_{\sigma_2};\ e'_{N-1} \leftarrow \chi_{\sigma_1}; \\ X'_0 = -\sum_{i=1}^{N-1} X_i + e'_0;\ X_1, \cdots, X_{N-2} \leftarrow R_q; \\ X_{N-1} = r_1(s_{N-1} + h_{N-1}) + e'_{N-1}; \\ e''_{N-1} \leftarrow \chi_{\sigma_1}; \\ b_{N-1} = r_2 N(s_{N-1} + h_{N-1}) + (N-1)X_{N-1} + \\ \qquad (N-2)X'_0 + \cdots + X_{N-3} + e''_{N-1}; \\ (\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1}); \mathsf{sk} = \mathcal{H}(k_{N-1}); \\ \mathsf{T} = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \cdots, X_{N-1}, \mathsf{rec}) \end{array} \right. : (\mathsf{T}, \mathsf{sk}) \right\}$$

Since r_1, r_2 are uniform, so does $\bar{z}_0 = r_1 + r_2$. For both **Experiment** $3N-3$ and **Experiment** $3N-2$, $\hat{z}_{N-2}$ and $\bar{z}_0$ are uniform. Then, we have $\Pr_{3N-2}[\mathsf{Query}] = \Pr_{3N-3}[\mathsf{Query}]$.

Experiment $3N-1$. We replace $\bar{z}_{N-1}, X_{N-1}$ and b_{N-1} into uniform element in R_q. The rest is the same as the previous experiment.

$$\mathsf{Expt}_{3N-1} := \left\{ \begin{array}{l} a \leftarrow R_q; z_i \leftarrow R_q \text{ for all } i \in [N]\,; \\ \bar{z}_i, \hat{z}_i \leftarrow R_q \text{ for all } i \in [N]\,; \\ h_i = \mathcal{H}(z_i) \text{ for } i \in [N]\,;\ e_0' \leftarrow \chi_{\sigma_2}; \\ X_0' = -\sum_{i=1}^{N-1} X_i + e_0';\ X_1, \cdots, X_{N-1} \leftarrow R_q; \\ b_{N-1} \leftarrow R_q \\ (\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1}); \mathsf{sk} = \mathcal{H}(k_{N-1}); \\ \mathsf{T} = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \cdots, X_{N-1}, \mathsf{rec}) \end{array} : (\mathsf{T}, \mathsf{sk}) \right\}$$

Between **Experiment** $3N-2$ and **Experiment** $3N-1$, we replace three RLWE instances $\bar{z}_{N-1}, X_{N-1}, b_{N-1}$ into random. Hence,

$$|\mathrm{Pr}_{3N-2}[\mathsf{Query}] - \mathrm{Pr}_{3N-3}[\mathsf{Query}]| \leq \mathsf{Adv}_{n,q,\chi_{\sigma_1},3}^{RLWE}(t_1).$$

Experiment $3N$. We replace k_{N-1} into uniform element k'_{N-1} in $\{0,1\}^\lambda$. The rest is the same as the previous experiment.

$$\mathsf{Expt}_{3N} := \left\{ \begin{array}{l} a \leftarrow R_q; z_i \leftarrow R_q \text{ for all } i \in [N]\,; \\ \bar{z}_i, \hat{z}_i \leftarrow R_q \text{ for all } i \in [N]\,; \\ h_i = \mathcal{H}(z_i) \text{ for } i \in [N]\,;\ e_0' \leftarrow \chi_{\sigma_2}; \\ X_0' = -\sum_{i=1}^{N-1} X_i + e_0';\ X_1, \cdots, X_{N-1} \leftarrow R_q; \\ b_{N-1} \leftarrow R_q \\ (\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1}); \\ k'_{N-1} \leftarrow \{0,1\}^\lambda; \mathsf{sk'} = \mathcal{H}(k'_{N-1}); \\ \mathsf{T} = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \cdots, X_{N-1}, \mathsf{rec}) \end{array} : (\mathsf{T}, \mathsf{sk}) \right\}$$

Between **Experiment** $3N-1$ and **Experiment** $3N$, we replace k_{N-1} from $\mathsf{recMsg}(b_{N-1})$ into random. Hence,

$$|\mathrm{Pr}_{3N}[\mathsf{Query}] - \mathrm{Pr}_{3N-1}[\mathsf{Query}]| \leq \mathsf{Adv}_{\mathsf{KeyRec}}(t_2)$$

where t_2 is the time to break KeyRec algorithm which is the sum of t and some minor overhead $\mathcal{O}(t_{ring})$ for simulation.

Since adversary attacking Krug makes at most q_E queries to the random oracle, we have $\mathrm{Pr}_1[\mathsf{Query}] = \frac{q_E}{2^\lambda}$, which is negligible in λ.

From **Experiment 2** to **Experiment** $3N$, we can conclude that

$$\mathrm{Pr}_1[\mathsf{Query}] \leq (\left\lceil \frac{N}{3} \right\rceil + N) \cdot \mathsf{Adv}_{n,q,\chi_{\sigma_1},3}^{RLWE}(t_1) + \mathsf{Adv}_{\mathsf{KeyRec}}(t_2) + \frac{q_E}{2^\lambda}$$

as expected. □

From **Lemmas 4 and 5**, we have the result of the theorem.

□

Conditions for parameter setting

To guarantee correctness and security of our instantiation, parameters $N, n, \sigma_1, \sigma_2, \lambda, \rho, \beta_{Rec}$ and $\beta_{\text{Rényi}}$ are also satisfied the following conditions:

$$\begin{aligned}
\text{Reconciliation Condition:} \quad & (N^2 + 2N) \cdot \sqrt{n}\rho^{3/2}\sigma_1^2 + (\frac{N^2}{2} + 1)\sigma_1 + (N-2)\sigma_2 \leq \beta_{\text{Rec}} \\
\text{Rényi Condition:} \quad & 2N\sqrt{n}\lambda^{3/2}\sigma_1^2 + (N-1)\sigma_1 \leq \beta_{\text{Rényi}} \\
\sigma_2 \text{ Condition 1:} \quad & \sigma_2 = \Omega(\beta_{\text{Rényi}} \cdot \sqrt{n/log(\lambda)} \\
\sigma_2 \text{ Condition 1:} \quad & \beta_{\text{Rényi}} < \sigma_2 < q \\
& 2n(N^2 + 2N) \leq 2^{\rho} \\
& \rho = \lambda \\
& |\mathsf{Error}_j| \leq \beta_{\text{Rényi}}
\end{aligned}$$

We compute some parameters under an assumption that $N = 10$ participants comprise a group and want to share the group key. Under $n = 512$, $N = 10$, $\lambda = 128$, $\rho = 128$, and $\sigma_1 = 2\sqrt{8}$, we have $\beta_{\text{Rec}} = 293,602,943.6875$, $\beta_{\text{Rényi}} = 20,971,572$, $\sigma_2 = 20,971,573$, and $q = 1,565,882,369$. These parameters are really big number compare to $q = 12289$ used in NewHope, and are not optimized yet.

Chapter 7. Dynamic GKE from RLWE

7.1 Security Model

We describe the adversary model by Bresson *et al.* [13], which is suitable for our dynamic authenticated GKE protocol since their model covers authenticated GKE with a dynamic setting with a weak corruption model.

Let $\mathcal{P} = P[0, 1, \cdots, N-1]$ be a set of N parties. Any subset of $\mathcal{P}$ wishes to establish a group secret key. We identify the execution of protocols for (authenticated) GKE or addition/deletion of a party or a set of parties as different sessions. We assume that the adversary never participates as a party in the protocol.

This adversary model allows concurrent execution of the protocol. The interaction between adversary $\mathcal{A}$ and the protocol participants only happens via oracle queries.

We denote a set of session identity and partner identity of the party P as sid^i_P and pid^i_P, respectively. For an instance $(U_j, i_j) \in S$ where U_j is the j-th party and i_j is the counter value that counts the number of queries that U_j is requested by the given protocol or the adversary, we define $\mathsf{sid}^{i_j}_{U_j} = S = \{(U_0, i_0), \cdots, (U_{l-1}, i_{l-1})\}$ and $\mathsf{pid}^{i_j}_{U_j} = U[0, 1, \cdots, l-1]$ when $U[0, 1, \cdots, l-1]$ wish to agree on a group secret key.

Let S, S_1, and S_2 be three sets of instances defined as:

$S = \{(U_0, i_0), \cdots, (U_{l-1}, i_{l-1})\}$,

$S_1 = \{(U_l, i_l), \cdots, (U_{l+k-1}, i_{l+k-1})\}$, and

$S_2 = \{(U_{l_0}, i_{l_0}), \cdots, (U_{l_{j-1}}, i_{l_{j-1}})\}$

where $U[0, 1, \cdots, l+k-1]$ is any non-empty subset of $\mathcal{P}$.

We assume that the adversary has full control over all communications in the network. All information that the adversary can get is written in a transcript since a transcript consists of all public information flowing across the network. The following oracles describe adversary's interaction with the protocol participants:

- $\mathsf{Send}(U, i, m)$: This oracle models an active attack where the adversary has full control of communication. The output is the reply by (U, i) upon the receipt of message m. The adversary can initiate the protocol with partners $U[0, 1, \cdots, l-1]$ where $l \leq N$, by invoking $\mathsf{Send}(U, i, U[0, 1, \cdots, l-1])$.

- $\mathsf{Execute}(S)$: This oracle models passive attacks where the attacker eavesdrops on an honest execution of the protocol and outputs the transcript of the execution. A transcript consists of all messages exchanged.

- $\mathsf{Join}(S, S_1)$: This oracle models the addition of S_1 to S, where all parties in S and S_1 are in $\mathcal{P}$. For S, $\mathsf{Execute}$ oracle has already been queried. The output is a transcript generated by the honest execution of the membership addition procedure. If $\mathsf{Execute}(S)$ is not preprocessed, the adversary gets no output.

- $\mathsf{Leave}(S, S_2)$: This oracle models the removal of $S_2 \subseteq S$ from S where all parties are in $\mathcal{P}$. Similar to $\mathsf{Join}(S, S_1)$, if $\mathsf{Execute}(S)$ is not preprocessed, the adversary gets no output. Otherwise, the membership deletion procedure is invoked. The adversary obtains the transcript from the honest execution of the membership deletion procedure.

- $\mathsf{Reveal}(U, i)$: This oracle models the misuse of the group secret key. This query outputs the group secret key sk_U^i for a session with an instance (U, i).

- $\mathsf{Corrupt}(U)$: This oracle models (perfect) forward secrecy. This query outputs the long-term secret key of party U.

- $\mathsf{Test}(U,\, i)$: We can query this oracle only once during the adversary's execution. A bit $b \in \{0, 1\}$ is chosen uniformly at random. The adversary gets sk if $b = 1$ and a random group secret key sk' if $b = 0$. This oracle checks the adversary's ability to distinguish a real group secret key from random.

An adversary who can access for $\mathsf{Execute}$, Join, Leave, Reveal, $\mathsf{Corrupt}$ and Test oracles is considered "passive" while an "active" adversary has full access to the above-mentioned oracles including Send oracle. (For a static case, Join or Leave queries do not need to be considered.)

The adversary can ask Send, $\mathsf{Execute}$, Join, Leave, Reveal and $\mathsf{Corrupt}$ queries several times, but Test query is asked for only once for a fresh instance. An instance (U, i) is *fresh* if none of the following occurs:

(1) the adversary queried $\mathsf{Reveal}(U, i)$ or $\mathsf{Reveal}(U', j)$ with $U' \in \mathsf{pid}_U^i$,

(2) the adversary queried $\mathsf{Corrupt}(U')$ (with $U' \in \mathsf{pid}_U^i$) before a query of the form $\mathsf{Send}(U, i, \star)$ or $\mathsf{Send}(U', j, \star)$ where $U' \in \mathsf{pid}_U^i$.

The adversary outputs a guess b'. Then, the adversary wins the game if $b = b'$ where b is the bit chosen from Test oracle.

Let Succ denote the event that the adversary $\mathcal{A}$ wins the game for a protocol XP. We define $\mathsf{Adv}_{\mathcal{A},\mathsf{XP}} := |2 \cdot \Pr[\mathsf{Succ}] - 1|$ to be the advantage that adversary $\mathcal{A}$ has in attacking the protocol XP.

The protocol XP provides *secure unauthenticated/authenticated GKE* (KE/AKE) security if there is no polynomial time passive adversary $\mathcal{A}_p$ and active adversary $\mathcal{A}_a$ with a non-negligible advantage, respectively.

Let t be the running time for $\mathcal{A}$ and q_E, q_J, q_L, q_S be the number of queries to Execute, Join, Leave, Send oracles respectively. $\mathsf{Adv}_{\mathsf{XP}}^{\mathsf{KE}}(t, q_E)$ is the maximum advantage of any passive adversary $\mathcal{A}_p$ attacking protocol XP and $\mathsf{Adv}_{\mathsf{XP}}^{\mathsf{AKE}}(t, q_E, q_S)$ and $\mathsf{Adv}_{\mathsf{XP}}^{\mathsf{AKE}}(t, q_E, q_J, q_L, q_S)$ are the maximum advantage of any active adversary $\mathcal{A}_a$ attacking protocol XP.

7.2 Unauthenticated Group Key Exchange

In the static setting, given $R_q = \mathbb{Z}_q[x]/(x^n+1)$ and $a \leftarrow R_q$, all parties calculate the partial numbers X_i and $Y_{i,j}$ and agree on "close" values $b_0 \approx b_1 \approx \cdots \approx b_{N-1}$ after the second round. Then, party P_{N-1} runs recMsg algorithm from KeyRec to allow all parties to get a common value $k = k_0 = k_1 = \cdots = k_{N-1}$.

Since we only show that k is difficult to compute for a passive adversary in the security proof, we hash k using random oracle $\mathcal{H}$ to get the session group secret key sk, which is indistinguishable from random. More detail description of unauthenticated GKE is given in **Protocol 5**.

7.3 Authenticated Group Key Exchange

To authenticate the unauthenticated one in Section 7.2, we use a digital signature scheme $\mathsf{DSig} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ where $\mathcal{K}$ is the key generation algorithm with output (sk_i, pk_i) for each party, $\mathcal{S}$ outputs a signature δ_i for a message m_i, and $\mathcal{V}$ outputs whether the input signature is valid or not.

Following Dutta-Barua protocol [28], at the start of the session, P_i doesn't need to know the entire session identity set $\mathsf{sid}_{P_i}^{d_i}$. As protocol proceeds, we build this set from partial session identity set $\mathsf{psid}_{P_i}^{d_i}$. Initially, $\mathsf{psid}_{P_i}^{d_i} = \{(P_i, d_i)\}$ and after completing the procedure, it becomes the full session identity set $\mathsf{sid}_{P_i}^{d_i}$. We assume that all parties know its partner identity $\mathsf{pid}_{P_i}^{d_i}$. We give a detail description as below.

7.4 Dynamic Group Key Exchange

7.4.1 Join Algorithm

If we assume that there are M parties in the set $P[N, N+1, \cdots, N+M-1]$ who wish to join the group $P[0, 1, \cdots, N-1]$ who already shared the common secret key sk, we make a new ring that consists of three parties P_0, P_1, P_{N-1} from $P[0, 1, \cdots, N-1]$ and all parties from the set $P[N, N+1, \cdots, N+M-1]$. P_1

Protocol 5: $\mathsf{STUG}(P\,[0,1,\cdots,N-1]\,,a,\mathcal{H},\sigma_1,\sigma_2)$

```
1 (Round 1) For each party $P_i$ for $i = 0$ to $N-1$, do the following in parallel.
      1. Computes $z_i = as_i + e_i$ where $s_i, e_i \leftarrow \chi_{\sigma_1}$;
      2. Broadcasts $z_i$;
  (Round 2) For $i = 0$ to $N-1$, do the following in parallel.
      1. If $i = 0$, party $P_0$ samples $e_0' \leftarrow \chi_{\sigma_2}$ and otherwise, party $P_i$ samples $e_i' \leftarrow \chi_{\sigma_1}$;
      2. Each party $P_i$ broadcasts $X_i = (z_{i+1} - z_{i-1})\,s_i + e_i'$;
  (Round 3) For party $P_{N-1}$ only.
      1. Samples $e''_{N-1} \leftarrow \chi_{\sigma_1}$ and computes $Y_{N-1,N-1} = X_{N-1} + z_{N-2}s_{N-1} + e''_{N-1}$;
      2. For $j = 1$ to $N-1$, computes $Y_{N-1,(N-1)+j} = X_{(N-1)+j} + Y_{N-1,(N-1)+(j-1)}$;
      3. Calculates $b_{N-1} = \sum_{j=0}^{N-1} Y_{N-1,(N-1)+j}$;
      4. Runs $\mathsf{recMsg}()$ to output $(\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1})$;
      5. Broadcasts $\mathsf{rec}$ and gets session key as $\mathsf{sk}_{N-1} = \mathcal{H}(k_{N-1})$;
  (Key Computation) For party $P_i$ $(i \neq N-1)$.
      1. Computes $Y_{i,i} = X_i + z_{i-1}s_i$;
      2. For $j = 1$ to $N-1$, computes $Y_{i,i+j} = X_{i+j} + Y_{i,i+(j-1)}$;
      3. $b_i = \sum_{j=0}^{N-1} Y_{i,i+j}$;
      4. Runs $\mathsf{recKey}()$ to output $k_i = \mathsf{recKey}\,(b_i, \mathsf{rec})$ and gets session key as $\mathsf{sk}_i = \mathcal{H}(k_i)$;
```

chooses the original session key sk as his ephemeral key $\overline{s}_1$. We give a full description of $\mathsf{U.Join}$ protocol for unauthenticated GKE as below.

For authenticated version $\mathsf{A.Join}$ procedure, we consider partial session-identity as STAG protocol but we assume that $\mathsf{psid}_{P_i}^{d_i} = \mathsf{psid}_{P_i}^{d_i} \bigcup \{\{(P_j, d_j) \mid j = 1 \text{ to } N-2\}$ if $P_i (i = 0, 1, \text{ or } N \leq i \leq N+M-1)$ verifies $\overline{\delta}_1'$ of $\overline{m}_1'$. We assume this since the ephemeral keys $\overline{s}_1$ and $\overline{z}_1$ are from the session key sk among the group $P\,[0,1,\cdots,N-1]$.

Signature generation and verification happen by switching STUG protocol into STAG protocol, also these operations happen in Round 2 of $\mathsf{A.Join}$ procedure when $\overline{z}_1$, $\overline{z}_3$, and $\overline{X}_i$ are delivered to group $P\,[2,\cdots,N-2]$.

By modifying the concept of $\mathsf{psid}_{P_i}^{d_i}$ slightly, we can achieve a common session identity $\mathsf{sid}_{P_i}^{d_i} = \{(P_j, d_j) \mid j \in [N+M]\}$ for parties in $P[0,1,\cdots,N+M-1]$ while Dutta-Barua protocol only provides a common session identity $\mathsf{sid}_{U_i}^{d_i} = \{(U_j, d_j) \mid j \in [\overline{N}]\}$ for parties in $U[0,1,\cdots,\overline{N}-1]$ where $\overline{N} = M+3$.

Protocol 6: $\mathsf{STAG}(P[0,1,\cdots,N-1], a, \mathcal{H}, \mathcal{S}, \sigma_1, \sigma_2)$

```
(Round 1) For each party $P_i$ for $i = 0$ to $N-1$, do the following in parallel.
    1. Sets partial session-identity $\mathsf{psid}_{P_i}^{d_i} = \{P_i, d_i\}$;
    2. Computes $z_i = as_i + e_i$ where $s_i, e_i \leftarrow \chi_{\sigma_1}$;
    3. Sets $m_i = P_i \mid 1 \mid z_i$ and $\delta_i = \mathcal{S}(m_i)$;
    4. Broadcasts $m_i \mid \delta_i$;
(Round 2) For each party $P_i$ for $i = 0$ to $N-1$, do the following in parallel.
    1. Verifies $\delta_{i-1}$ of $m_{i-1}$ and $\delta_{i+1}$ of $m_{i+1}$ and proceeds only if both signatures are valid (Otherwise, aborts);
    2. If $i = 0$, party $P_0$ samples $e'_0 \leftarrow \chi_{\sigma_2}$ and otherwise, party $P_i$ samples $e'_i \leftarrow \chi_{\sigma_1}$;
    3. Computes $X_i = (z_{i+1} - z_{i-1})\, s_i + e'_i$;
    4. Sets $m'_i = P_i \mid 2 \mid X_i \mid d_i$ and $\delta'_i = \mathcal{S}(m'_i)$ and broadcasts $m'_i \mid \delta'_i$;
(Round 3) For party $P_{N-1}$ only.
    1. Verifies all $\delta'_j$ of $m'_j$ where $j \neq N-1$ and proceeds only if both signatures are valid (Otherwise, aborts);
    2. Extracts $d_j$ from $m'_j$ and sets $\mathsf{psid}_{P_{N-1}}^{d_{N-1}} = \mathsf{psid}_{P_{N-1}}^{d_{N-1}} \bigcup \{(P_j, d_j)\}$;
    3. Samples $e''_{N-1} \leftarrow \chi_{\sigma_1}$ and computes $Y_{N-1,N-1} = X_{N-1} + z_{N-2}s_{N-1} + e''_{N-1}$;
    4. For $j = 1$ to $N-1$, computes $Y_{N-1,(N-1)+j} = X_{(N-1)+j} + Y_{N-1,(N-1)+(j-1)}$;
    5. Calculates $b_{N-1} = \sum_{j=0}^{N-1} Y_{N-1,(N-1)+j}$;
    6. Runs $\mathsf{recMsg}(\cdot)$ to output $(\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1})$;
    7. Broadcasts $\mathsf{rec}$ and gets session key as $\mathsf{sk}_{N-1} = \mathcal{H}(k_{N-1})$;
(Key Computation) For party $P_i$ $(i \neq N-1)$.
    1. Verifies all $\delta'_j$ of $m'_j$ where $j \neq i$ and proceeds only if both signatures are valid (Otherwise, aborts);
    2. Extracts $d_j$ from $m'_j$ and sets $\mathsf{psid}_{P_i}^{d_i} = \mathsf{psid}_{P_i}^{d_i} \bigcup \{(P_j, d_j)\}$;
    3. Computes $Y_{i,i} = X_i + z_{i-1}s_i$;
    4. For $j = 1$ to $N-1$, computes $Y_{i,i+j} = X_{i+j} + Y_{i,i+(j-1)}$;
    5. $b_i = \sum_{j=0}^{N-1} Y_{i,i+j}$;
    6. Runs $\mathsf{recKey}()$ to output $k_i = \mathsf{recKey}(b_i, \mathsf{rec})$ and gets session key as $\mathsf{sk}_i = \mathcal{H}(k_i)$;
```

Protocol 7: $\mathsf{U.Join}(P[0,1,\cdots,N-1], P[N,N+1,\cdots,N+M-1])$

```
(Round 1) Rearrange the order with a new array of $\overline{N} = M+3$ parties
    1. $U_0 = P_0, U_1 = P_1, U_2 = P_{N-1}, \overline{s}_0 = s_0, \overline{s}_1 = \mathsf{sk}, \overline{s}_2 = s_{N-1}$ and for $1 \le i \le \overline{N}-3, U_{i+3} = P_{N-1+i}$;
    2. Let $U[0,1,\cdots,\overline{N}-1]$ be a new ring that we run in (Round 2);
(Round 2) Run STUG algorithm.
    1. Group $U[0,1,\cdots,\overline{N}-1]$ runs STUG;
    2. $U_i$ calculates $\overline{z}_i$ during the 1st round of STUG and broadcasts it;
    3. $U_0$ and $U_2$ during 1st round of STUG additionally sends $\overline{z}_1$ and $\overline{z}_3$ to all parties in
       $P[2,\cdots,N-2]$;
    4. $U_i$ calculates $\overline{X}_i$ during the 2nd round of STUG sends $\overline{X}_i$ to all parties in $P[0,\cdots,N+M-1]$;
    5. After the 3rd round of STUG, $U_{\overline{N}-1}$ sends $\overline{\mathsf{rec}}$ to all parties in $P[0,\cdots,N+M-1]$;
(Key Computation) For party $P_i$ $(2 \le i \le N-2)$.
    1. Computes $\overline{Y}_{i,2} = \overline{X}_2 + \overline{z}_2\overline{s}_1 = \overline{X}_2 + \overline{z}_2 \cdot \mathsf{sk}$;
    2. For $j = 1$ to $\overline{N}-2$, computes $\overline{Y}_{i,2+j} = \overline{X}_{2+j} + \overline{Y}_{i,2+(j-1)}$;
    3. $b'_i = \sum_{j=0}^{\overline{N}-1} \overline{Y}_{i,j}$;
    4. Runs $\mathsf{recKey}(\cdot,\cdot)$ to output $\overline{k}_i = \mathsf{recKey}(\overline{b}_i, \overline{\mathsf{rec}})$ and gets session key as $\overline{\mathsf{sk}}_i = \mathcal{H}(\overline{k}_i)$;
```

7.4.2 Leave Algorithm

Let the set of parties $P_{l_1}, P_{l_2}, \cdots P_{l_M}$ want to leave the group $P[0, 1, \cdots, N-1]$. Then, the new group becomes $P' = P[0, \cdots, l_1 - L] \cup P[l_1 + R, \cdots, l_2 - L] \cup \cdots \cup P[l_M + R, \cdots, N-1]$. Instead of $l_i - 1$ and $l_i + 1$, we use $l_i - L$ and $l_i + R$ since there might be consecutive parties who want to leave the group $P[0, 1, \cdots, N-1]$. *e.g.*, if $P_l, P_{l-1}, P_{l-2}, \cdots, P_{l-(j-1)}$ are consecutive parties who want to leave, then $P_{l-L} = P_{l-j}$.

After making a new group P', we simply relabel orders to make a new array $U[0, 1, \cdots, N-M-1]$ of the parties in the protocol and run $\mathsf{U.Leave}$ procedure for $U[0, 1, \cdots, N-M-1]$ based on the remaining parties and run STUG protocol. For authenticated version $\mathsf{A.Leave}$ procedure, we simply apply STAG protocol instead of STUG algorithm.

Our dynamic (un-)authenticated GKE protocol DRAG consists of three procedures, either (STUG, $\mathsf{U.Join}, \mathsf{U.Leave}$) or ($\mathsf{STAG}$, $\mathsf{A.Join}, \mathsf{A.Leave}$), as a subroutine.

7.5 Security Analysis

In this section, we check the correctness of our protocol and give a full security proof using the security model by Bresson *et al.* [13]. Our proof techniques is based on Apon *et al.*'s protocol [4] and Dutta-Barua protocol [28].

7.5.1 Correctness Proof

In **Theorem 8**, we give a condition that our GKE is correct. Most part of our correctness proof follow Apon *et al.*'s correctness proof but there are some modification on error bound.

Theorem 8. For a fixed ρ, and assume that

$$(N-1)N/2 \cdot \sqrt{n}\rho^{3/2}\sigma_1^2 + (N(N+1)/2 + N)\,\sigma_1 + (N-2)\sigma_2 \leq \beta_{\mathsf{Rec}}.$$

Then all participants in a group have the same key except with probability at most $2^{-\rho+1}$.

Proof. As mentioned in Section 2.5, we will show that all parties have the same secret key except with negligible probability. To hold this, we claim that if for all $i \in [N]$ and $j \in [n]$ j-th coefficient of $|b_{N-1} - b_i| \leq \beta_{\mathsf{Rec}}$, then $k_i = k_{N-1}$. After some tedious computation, we have

$$\begin{aligned} b_{N-1} - b_i = N e''_{N-1} &+ \sum_{j=0}^{N-1} (N-j)(e'_{N-1+j} - e'_{i+j}) \\ &+ \sum_{j=0}^{N-2} (N-1-j)\{(e_{N+j}s_{N-1+j} - e_{N-1+j}s_{N+j}) - (e_{i+j+1}s_{i+j} - e_{i+j}s_{i+j+1})\}. \end{aligned}$$

Now observe how many terms are in $b_{N-1} - b_i$. There are at most $(N-1)N/2$ terms in form of $s_i \cdot e_j$, at most $N(N+1)/2$ terms in form of e'_k sampled from χ_{σ_1}, at most $N-2$ terms of e'_0 sampled from χ_{σ_2}, and N terms of e''_{N-1}. Sum of these at most terms is less than Apon *et al.*'s terms.

Let $\mathsf{product}_{\mathsf{ALL}}$ be the event that for all terms in form of $s_i \cdot e_j$, each coefficient of this form is bounded by $\sqrt{n}\rho^{3/2}\sigma_1^2$. Under an assumptiont that $2n(N-1)N/2 \leq 2^{\rho}$, by Lemma 3 we can get

$$\Pr[\overline{\mathsf{product}_{\mathsf{ALL}}} \mid \mathsf{bound}_{\rho}] \leq \frac{(N-1)N}{2} \cdot n \cdot 2^{-2\rho+1} \leq 2^{-\rho}$$

Denote fail by the event that at least one of parties does not agree on the same key. Given a condition that $(N-1)N/2 \cdot \sqrt{n}\rho^{3/2}\sigma_1^2 + (N(N+1)/2 + N)\sigma_1 + (N-2)\sigma_2 \leq \beta_{\mathsf{Rec}}$, by Lemma 2 and the above inequality we have

$$\begin{aligned}
\Pr[\mathsf{fail}] &= \Pr[\mathsf{fail} \mid \mathsf{bound}_{\rho}] \cdot \Pr[\mathsf{bound}_{\rho}] + \Pr[\mathsf{fail} \mid \overline{\mathsf{bound}_{\rho}}] \cdot \Pr[\overline{\mathsf{bound}_{\rho}}] \\
&\leq \Pr[\overline{\mathsf{product}_{\mathsf{ALL}}} \mid \mathsf{bound}_{\rho}] \cdot 1 + 1 \cdot \Pr[\overline{\mathsf{bound}_{\rho}}] \\
&\leq 2 \cdot 2^{-\rho}.
\end{aligned}$$

Therefore, all parties agree on the same secret key except with probability $2 \cdot 2^{-\rho}$. □

From the result of **Theorem 8**, the number of error terms in our protocol is smaller than Apon *et al.*'s protocol. Then, the probability $\Pr_{\mathsf{STUG}}[\mathsf{AbortKey}]$ of the event $\mathsf{AbortKey}$ that error between b_i's exceeds β_{Rec} in our protocol is smaller than the probability $\Pr_{\mathsf{Apon}}[\mathsf{AbortKey}]$ in Apon *et al.*'s protocol. Thus, our protocol has higher probability to have the common secret key between protocol participants.

7.5.2 Security Proof

We write **Theorems 9, 10 and 11** to show that our dynamic key exchange protocol $\mathsf{DRAG} = (\mathsf{STUG}, \mathsf{U.Join}, \mathsf{U.Leave})$ (or $(\mathsf{STAG}, \mathsf{A.Join}, \mathsf{A.Leave})$) is secure in the random oracle model based on hardness of RLWE assumption.

Theorem 9. For unauthenticated GKE protocol STUG, $2N\sqrt{n}\lambda^{3/2}\sigma_1^2 + (N-1)\sigma_1 \leq \beta_{\mathsf{Rényi}}$ and $\sigma_2 = \Omega\left(\beta_{\mathsf{Rényi}}\sqrt{n/\log\lambda}\right)$. Then, we have the following:

$$\mathsf{Adv}^{\mathsf{KE}}_{\mathsf{STUG}}(t, q_E) \leq 2^{-\lambda+1} + \sqrt{\left(N \cdot \mathsf{Adv}^{RLWE}_{n,q,\chi_{\sigma_1},3}(t_1) + \mathsf{Adv}_{\mathsf{KeyRec}}(t_2) + \frac{q_E}{2^{\lambda}}\right) \cdot \frac{\exp\left(2\pi n(\beta_{\mathsf{Rényi}}/\sigma_2)^2\right)}{1 - 2^{-\lambda+1}}}$$

where $t_1 = t + \mathcal{O}(N \cdot t_{ring}), t_2 = t + \mathcal{O}(N \cdot t_{ring})$ such that t_{ring} is the maximum time required to make operations in R_q.

Proof. Let $\mathcal{A}$ be an adversary that breaks the protocol STUG. From this, we construct an adversary $\mathcal{B}$ that solves RLWE problem with non-negligible advantage. Since we do not have any long-term secret key in our protocol STUG, $\mathsf{Corrupt}$ can be ignored and the protocol achieves the forward secrecy.

Let Query be the event that k_{N-1} is among the adversary $\mathcal{A}$'s random oracle queries and $\Pr_i[\mathsf{Query}]$ be the probability of Query in Experiment i.

Then, by a sequence of experiments, we show that an efficient adversary who queries the random oracle in Ideal experiment with at most negligible probability can query the random oracle in Real experiment. For Ideal experiment, the input k_{N-1} is chosen uniformly random while k_{N-1} is chosen by the honest execution of STUG in Real experiment. We continue the proof in Appendix C.

□

Proof. We present Experiment 0 and Experiment 1 as an example. **Experiment** 0. This is the original experiment that is equal to the procedure of STUG.

$$\mathsf{Exp}_0 := \left\{ \begin{array}{l} a \leftarrow R_q; s_i, e_i \leftarrow \chi_{\sigma_1} \text{ for } i \in [N]; \\ z_i = as_i + e_i \text{ for } i \in [N]; \\ e'_0 \leftarrow \chi_{\sigma_2} \text{ and } e'_i \leftarrow \chi_{\sigma_1} \text{ for } 1 \le i \le N-1; \\ X_i = (z_{i+1} - z_{i-1})s_i + e'_i \text{ for } i \in [N]; \\ e''_{N-1} \leftarrow \chi_{\sigma_1}; \\ Y_{N-1,N-1} = X_{N-1} + z_{N-2}s_{N-1} + e''_{N-1}; \\ Y_{N-1,(N-1)+j} = X_{(N-1)+j} + Y_{N-1,(N-1)+(j-1)}; \\ b_{N-1} = \sum_{j=0}^{N-1} Y_{N-1,(N-1)+j}; \\ (\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1}); \mathsf{sk} = \mathcal{H}(k_{N-1}); \\ \mathsf{T} = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \cdots, X_{N-1}, \mathsf{rec}) \end{array} : (\mathsf{T}, \mathsf{sk}) \right\}$$

Since $\Pr[\mathcal{A} \text{ wins}] = \dfrac{1}{2} + \mathsf{Adv}^{\mathsf{KE}}_{\mathsf{STUG}}(t, q_E) = \Pr_0[\mathsf{Query}] + \Pr_0\left[\overline{\mathsf{Query}}\right] \cdot \dfrac{1}{2}$,

$$\mathsf{Adv}^{\mathsf{KE}}_{\mathsf{STUG}}(t, q_E) \le \Pr_0[\mathsf{Query}].$$

Experiment 1. We replace X_0 into $X'_0 = -\sum_{i=1}^{N-1} X_i + e'_0$. The rest are same as the previous experiment.

$$\mathsf{Exp}_1 := \left\{ \begin{array}{l} a \leftarrow R_q; s_i, e_i \leftarrow \chi_{\sigma_1} \text{ for } i \in [N]; \\ z_i = as_i + e_i \text{ for } i \in [N]; \\ e'_0 \leftarrow \chi_{\sigma_2} \text{ and } e'_i \leftarrow \chi_{\sigma_1} \text{ for } 1 \le i \le N-1; \\ X'_0 = -\sum_{i=1}^{N-1} X_i + e'_0; \\ X_i = (z_{i+1} - z_{i-1})s_i + e'_i \text{ for } 1 \le i \le N-1; \\ e''_{N-1} \leftarrow \chi_{\sigma_1}; \\ Y_{N-1,N-1} = X_{N-1} + z_{N-2}s_{N-1} + e''_{N-1}; \\ Y_{N-1,(N-1)+j} = X_{(N-1)+j} + Y_{N-1,(N-1)+(j-1)}; \\ b_{N-1} = \sum_{j=0}^{N-1} Y_{N-1,(N-1)+j}; \\ (\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1}); \mathsf{sk} = \mathcal{H}(k_{N-1}); \\ \mathsf{T} = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \cdots, X_{N-1}, \mathsf{rec}) \end{array} : (\mathsf{T}, \mathsf{sk}) \right\}$$

Lemma 6. Given two distributions of X_0 and X'_0, if we have $2N\sqrt{n}\lambda^{3/2}\sigma_1^2 + (N-1)\sigma_1 \le \beta_{\mathsf{Rényi}}$, then

$$\Pr_0[\mathsf{Query}] \le 2^{-\lambda+1} + \sqrt{\Pr_1[\mathsf{Query}] \cdot \frac{\exp\left(2\pi n(\beta_{\mathsf{Rényi}}/\sigma_2)^2\right)}{1 - 2^{-\lambda+1}}}$$

using the property of Rényi divergence.

Proof. Note that we may define the random variables X_0, X'_0 in both experiments Real and Dist_1. We define Error and main as $\mathsf{Error} = \sum_{i=0}^{N-1}(s_ie_{i+1} - s_ie_{i-1}) + \sum_{i=1}^{N-1} e'_i$ and $\mathsf{main} = z_1s_0 - z_0s_1 - \mathsf{Error}$, respectively. Then,

$$X_0 = \mathsf{main} + \mathsf{Error} + e'_0 \text{ and } X'_0 = \mathsf{main} + e'_0$$

where $e'_0 \leftarrow \sigma_2$. We check whether Rényi divergence between two distributions of X_0 and X'_0 is small using Theorem 1. Let $\mathsf{bound}_{\mathsf{Error}}$ be the event that for all participants j, $\mathsf{Error}_j \leq \beta_{\mathsf{Rényi}}$. Then,

$$|\mathsf{Error}_j| = \left| \left(\sum_{i=0}^{N-1} (s_i e_{i+1} - s_i e_{i-1}) + \sum_{i=1}^{N-1} e'_i \right)_j \right|.$$

Set $c = \sqrt{\frac{2\lambda}{\pi \log e}}$ and let bound be the event that $|(e'_0)_j| \leq c\sigma_2$, $|(s_i)_j|, |(e_i)_j|$, $|(e''_{N-1})_j| \leq c\sigma_1$, and $|(e'_i)_j| \leq c\sigma_1$ for all $i > 0$ and j.

From Lemmas 2 and 3, we have $\Pr[\mathsf{bound}] \geq 1 - 2^{-\lambda}$ and $\Pr[|(s_i e_j)_v| \leq \sqrt{n}\lambda^{3/2}\sigma_1^2 \mid \mathsf{bound}] \geq 1 - 2^{-2\lambda+1}$. With a union bound, we have

$$\Pr\left[\forall j : |\mathsf{Error}_j| \leq 2N\sqrt{n}\lambda^{3/2}\sigma_1^2 + (N-1)\sigma_1 \mid \mathsf{bound}\right] \geq 1 - 4N \cdot n \cdot 2^{-2\lambda}$$

. If we assume $4Nn \leq 2^{\lambda}$, we derive that $\Pr[\mathsf{bound}_{\mathsf{Error}}] \geq 1 - 2^{-\lambda+1}$.

We have $\mathrm{RD}_2(\mathsf{Error} + \chi_{\sigma_2} \| \chi_{\sigma_2}) \leq \exp(2\pi n(\beta_{\mathsf{Rényi}}/\sigma)^2)$ from Theorem 1. Thus,

$$\begin{aligned}
\Pr_0[\mathsf{Query}] &\leq \Pr_0[\mathsf{Query} \mid \mathsf{bound}_{\mathsf{Error}}] + \Pr_0\left[\overline{\mathsf{bound}_{\mathsf{Error}}}\right] \\
&\leq \Pr_0[\mathsf{Query} \mid \mathsf{bound}_{\mathsf{Error}}] + 2^{-\lambda+1} \\
&\leq \sqrt{\Pr_1[\mathsf{Query} \mid \mathsf{bound}_{\mathsf{Error}}] \cdot \exp(2\pi n(\beta_{\mathsf{Rényi}}/\sigma_2)^2)} + 2^{-\lambda+1} \\
&\leq \sqrt{\Pr_1[\mathsf{Query}] \cdot \frac{\exp(2\pi n(\beta_{\mathsf{Rényi}}/\sigma_2)^2)}{\Pr_1[\mathsf{bound}_{\mathsf{Error}}]}} + 2^{-\lambda+1} \\
&\leq \sqrt{\Pr_1[\mathsf{Query}] \cdot \frac{\exp(2\pi n(\beta_{\mathsf{Rényi}}/\sigma_2)^2)}{1 - 2^{-\lambda+1}}} + 2^{-\lambda+1}
\end{aligned}$$

From second to third inequality, we use the property that Rényi divergence is bounded. □

For the rest of the proof, we will show that

$$\Pr_1[\mathsf{Query}] \leq N \cdot \mathsf{Adv}^{RLWE}_{n,q,\chi_{\sigma_1},3}(t_1) + \mathsf{Adv}_{\mathsf{KeyRec}}(t_2) + \frac{q_E}{2^{\lambda}}.$$

Experiment 2. We replace z_0 into uniform element in R_q. The rest are same as the previous experiment.

$$\mathsf{Exp}_2 := \left\{ \begin{array}{ll} a, z_0 \leftarrow R_q; \forall i \geq 1,\ s_i, e_i \leftarrow \chi_{\sigma_1};\ z_i = as_i + e_i; & \\ e'_0 \leftarrow \chi_{\sigma_2} \text{ and } e'_i \leftarrow \chi_{\sigma_1} \text{ for } 1 \leq i \leq N-1; & \\ X'_0 = -\sum_{i=1}^{N-1} X_i + e'_0; & \\ X_i = (z_{i+1} - z_{i-1})s_i + e'_i \text{ for } 1 \leq i \leq N-1; & \\ e''_{N-1} \leftarrow \chi_{\sigma_1}; & : (\mathsf{T}, \mathsf{sk}) \\ Y_{N-1,N-1} = X_{N-1} + z_{N-2}s_{N-1} + e''_{N-1}; & \\ Y_{N-1,(N-1)+j} = X_{(N-1)+j} + Y_{N-1,(N-1)+(j-1)}; & \\ b_{N-1} = \sum_{j=0}^{N-1} Y_{N-1,(N-1)+j}; & \\ (\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1}); \mathsf{sk} = \mathcal{H}(k_{N-1}); & \\ \mathsf{T} = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \cdots, X_{N-1}, \mathsf{rec}) & \end{array} \right\}$$

Between Experiment 1 and Experiment 2, we replace one RLWE instance into random. Hence, $|\Pr_2[\mathsf{Query}] - \Pr_1[\mathsf{Query}]| \leq \mathsf{Adv}^{RLWE}_{n,q,\chi_{\sigma_1},1}(t_1)$

where $t_1 = t + \mathcal{O}(N \cdot t_{ring})$ and t_{ring} is the time required to perform operations in R_q. Since $\mathsf{Adv}^{RLWE}_{n,q,\chi_{\sigma_1},1}(t_1) \leq \mathsf{Adv}^{RLWE}_{n,q,\chi_{\sigma_1},2}(t_1) \leq \mathsf{Adv}^{RLWE}_{n,q,\chi_{\sigma_1},3}(t_1)$, we have $|\Pr_2[\mathsf{Query}] - \Pr_1[\mathsf{Query}]| \leq \mathsf{Adv}^{RLWE}_{n,q,\chi_{\sigma_1},3}(t_1)$.

Experiment 3. We replace z_0 into $z_2 - r_1$ and X_1 into $r_1 s_1 + e'_1$ where $r_1 \leftarrow R_q$. The rest are same as the previous experiment.

$$\mathsf{Exp}_3 := \left\{ \begin{array}{ll} a, r_1 \leftarrow R_q; \forall i \geq 1,\ s_i, e_i \leftarrow \chi_{\sigma_1};\ z_i = a s_i + e_i; & \\ z_0 = z_2 - r_1; & \\ e'_0 \leftarrow \chi_{\sigma_2} \text{ and } e'_i \leftarrow \chi_{\sigma_1} \text{ for } 1 \leq i \leq N-1; & \\ X'_0 = -\sum_{i=1}^{N-1} X_i + e'_0;\ X_1 = r_1 s_1 + e'_1 & \\ X_i = (z_{i+1} - z_{i-1}) s_i + e'_i \text{ for } 2 \leq i \leq N-1; & \\ e''_{N-1} \leftarrow \chi_{\sigma_1}; & : (\mathsf{T}, \mathsf{sk}) \\ Y_{N-1,N-1} = X_{N-1} + z_{N-2} s_{N-1} + e''_{N-1}; & \\ Y_{N-1,(N-1)+j} = X_{(N-1)+j} + Y_{N-1,(N-1)+(j-1)}; & \\ b_{N-1} = \sum_{j=0}^{N-1} Y_{N-1,(N-1)+j}; & \\ (\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1}); \mathsf{sk} = \mathcal{H}(k_{N-1}); & \\ \mathsf{T} = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \cdots, X_{N-1}, \mathsf{rec}) & \end{array} \right\}$$

Since both z_0 and $z_2 - r_1$ are uniform, $\Pr_3[\mathsf{Query}] = \Pr_2[\mathsf{Query}]$.

Experiment 4. We replace z_1, X_1 into uniform element in R_q. The rest are same as the previous experiment.

$$\mathsf{Exp}_4 := \left\{ \begin{array}{ll} a, r_1 \leftarrow R_q; \forall i \geq 2,\ s_i, e_i \leftarrow \chi_{\sigma_1};\ z_i = a s_i + e_i; & \\ z_0 = z_2 - r_1;\ z_1 \leftarrow R_q; & \\ e'_0 \leftarrow \chi_{\sigma_2} \text{ and } e'_i \leftarrow \chi_{\sigma_1} \text{ for } 2 \leq i \leq N-1; & \\ X'_0 = -\sum_{i=1}^{N-1} X_i + e'_0;\ X_1 \leftarrow R_q & \\ X_i = (z_{i+1} - z_{i-1}) s_i + e'_i \text{ for } 2 \leq i \leq N-1; & \\ e''_{N-1} \leftarrow \chi_{\sigma_1}; & : (\mathsf{T}, \mathsf{sk}) \\ Y_{N-1,N-1} = X_{N-1} + z_{N-2} s_{N-1} + e''_{N-1}; & \\ Y_{N-1,(N-1)+j} = X_{(N-1)+j} + Y_{N-1,(N-1)+(j-1)}; & \\ b_{N-1} = \sum_{j=0}^{N-1} Y_{N-1,(N-1)+j}; & \\ (\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1}); \mathsf{sk} = \mathcal{H}(k_{N-1}); & \\ \mathsf{T} = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \cdots, X_{N-1}, \mathsf{rec}) & \end{array} \right\}$$

Between Experiment 3 and Experiment 4, we replace two RLWE instances into random. Hence, $|\Pr_4[\mathsf{Query}] - \Pr_3[\mathsf{Query}]| \leq \mathsf{Adv}^{RLWE}_{n,q,\chi_{\sigma_1},2}(t_1)$ and thus,

$$|\Pr_4[\mathsf{Query}] - \Pr_3[\mathsf{Query}]| \leq \mathsf{Adv}^{RLWE}_{n,q,\chi_{\sigma_1},3}(t_1)$$

t_1 is the time to solve RLWE problem which is the sum of t and some minor overhead $\mathcal{O}(t_{ring})$ for simulation.

Experiment 5. We replace z_0 into uniform element in R_q. The rest are same as the previous experiment.

$$\mathsf{Exp}_5 := \left\{ \begin{array}{ll} a \leftarrow R_q; \forall i \geq 2,\ s_i, e_i \leftarrow \chi_{\sigma_1};\ z_i = as_i + e_i; & \\ z_0, z_1 \leftarrow R_q; & \\ e'_0 \leftarrow \chi_{\sigma_2} \text{ and } e'_i \leftarrow \chi_{\sigma_1} \text{ for } 2 \leq i \leq N-1; & \\ X'_0 = -\sum_{i=1}^{N-1} X_i + e'_0;\ X_1 \leftarrow R_q & \\ X_i = (z_{i+1} - z_{i-1})s_i + e'_i \text{ for } 2 \leq i \leq N-1; & \\ e''_{N-1} \leftarrow \chi_{\sigma_1}; & : (\mathsf{T}, \mathsf{sk}) \\ Y_{N-1,N-1} = X_{N-1} + z_{N-2}s_{N-1} + e''_{N-1}; & \\ Y_{N-1,(N-1)+j} = X_{(N-1)+j} + Y_{N-1,(N-1)+(j-1)}; & \\ b_{N-1} = \sum_{j=0}^{N-1} Y_{N-1,(N-1)+j}; & \\ (\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1}); \mathsf{sk} = \mathcal{H}(k_{N-1}); & \\ \mathsf{T} = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \cdots, X_{N-1}, \mathsf{rec}) & \end{array} \right\}$$

Since both z_0 and $z_2 - r_1$ are uniform, $\Pr_5[\mathsf{Query}] = \Pr_4[\mathsf{Query}]$.

Similarly, we can design distribution of $(\mathsf{T}, \mathsf{sk})$ in Experiment $3i, 3i+1, 3i+2$ as below:

Experiment $3i$. We replace z_{i-1} into $z_{i+1} - r_i$ and X_i into $r_i s_i + e'_i$ where $r_i \leftarrow R_q$. The rest are same as the previous experiment.

$$\mathsf{Exp}_{3i} := \left\{ \begin{array}{ll} a, r_i \leftarrow R_q;\ \forall j \geq i,\ s_j, e_j \leftarrow \chi_{\sigma_1};\ z_j = as_j + e_j; & \\ z_0, \cdots, z_{i-2} \leftarrow R_q,\ z_{i-1} = z_{i+1} - r_i; & \\ e'_0, \leftarrow \chi_{\sigma_2} \text{ and } e'_i, \cdots, e'_{N-1} \leftarrow \chi_{\sigma_1}; & \\ X'_0 = -\sum_{i=1}^{N-1} X_i + e'_0;\ X_1, \cdots, X_{i-1} \leftarrow R_q; & \\ X_i = r_i s_i + e'_i; & \\ \forall j \geq i+1,\ X_j = (z_{j+1} - z_{j-1})s_j + e'_j; & : (\mathsf{T}, \mathsf{sk}) \\ e''_{N-1} \leftarrow \chi_{\sigma_1}; & \\ Y_{N-1,N-1} = X_{N-1} + z_{N-2}s_{N-1} + e''_{N-1}; & \\ Y_{N-1,(N-1)+j} = X_{(N-1)+j} + Y_{N-1,(N-1)+(j-1)}; & \\ b_{N-1} = \sum_{j=0}^{N-1} Y_{N-1,(N-1)+j}; & \\ (\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1}); \mathsf{sk} = \mathcal{H}(k_{N-1}); & \\ \mathsf{T} = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \cdots, X_{N-1}, \mathsf{rec}) & \end{array} \right\}$$

Experiment $3i+1$. We replace z_i, X_i into uniform element in R_q. The rest are same as the previous experiment.

$$\mathsf{Exp}_{3i+1} := \left\{ \begin{array}{ll} a, r_i \leftarrow R_q;\ \forall j \geq i+1,\ s_j, e_j \leftarrow \chi_{\sigma_1};\ z_j = as_j + e_j; & \\ z_0, \cdots, z_{i-2} \leftarrow R_q; z_{i-1} = z_{i+1} - r_i,\ z_i \leftarrow R_q; & \\ e'_0, \leftarrow \chi_{\sigma_2} \text{ and } e'_i, \cdots, e'_{N-1} \leftarrow \chi_{\sigma_1}; & \\ X'_0 = -\sum_{i=1}^{N-1} X_i + e'_0;\ X_1, \cdots, X_i \leftarrow R_q; & \\ \forall j \geq i+1,\ X_j = (z_{j+1} - z_{j-1})s_j + e'_j; & \\ e''_{N-1} \leftarrow \chi_{\sigma_1}; & : (\mathsf{T}, \mathsf{sk}) \\ Y_{N-1,N-1} = X_{N-1} + z_{N-2}s_{N-1} + e''_{N-1}; & \\ Y_{N-1,(N-1)+j} = X_{(N-1)+j} + Y_{N-1,(N-1)+(j-1)}; & \\ b_{N-1} = \sum_{j=0}^{N-1} Y_{N-1,(N-1)+j}; & \\ (\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1}); \mathsf{sk} = \mathcal{H}(k_{N-1}); & \\ \mathsf{T} = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \cdots, X_{N-1}, \mathsf{rec}) & \end{array} \right\}$$

Experiment $3i+2$. We replace z_{i-1} into uniform element in R_q. The rest are same as the previous experiment.

$$\mathsf{Exp}_{3i+2} := \left\{ \begin{array}{ll} a \leftarrow R_q;\ \forall j \geq i+1,\ s_j, e_j \leftarrow \chi_{\sigma_1};\ z_j = as_j + e_j; & \\ z_0, \cdots, z_i \leftarrow R_q; & \\ e'_0, \leftarrow \chi_{\sigma_2} \text{ and } e'_i, \cdots, e'_{N-1} \leftarrow \chi_{\sigma_1}; & \\ X'_0 = -\sum_{i=1}^{N-1} X_i + e'_0;\ X_1, \cdots, X_i \leftarrow R_q; & \\ \forall j \geq i+1,\ X_j = (z_{j+1} - z_{j-1})s_j + e'_j; & \\ e''_{N-1} \leftarrow \chi_{\sigma_1}; & : (\mathsf{T}, \mathsf{sk}) \\ Y_{N-1,N-1} = X_{N-1} + z_{N-2}s_{N-1} + e''_{N-1}; & \\ Y_{N-1,(N-1)+j} = X_{(N-1)+j} + Y_{N-1,(N-1)+(j-1)}; & \\ b_{N-1} = \sum_{j=0}^{N-1} Y_{N-1,(N-1)+j}; & \\ (\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1}); \mathsf{sk} = \mathcal{H}(k_{N-1}); & \\ \mathsf{T} = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \cdots, X_{N-1}, \mathsf{rec}) & \end{array} \right\}$$

With similar argument of Experiment 3, 4 and 5, we have

$$\begin{aligned} &\Pr_{3i}[\mathsf{Query}] = \Pr_{3i-1}[\mathsf{Query}] \\ &|\Pr_{3i+1}[\mathsf{Query}] - \Pr_{3i}[\mathsf{Query}]| \leq \mathsf{Adv}^{RLWE}_{n,q,\chi_{\sigma_1},3}(t_1) \\ &\Pr_{3i+2}[\mathsf{Query}] = \Pr_{3i+1}[\mathsf{Query}] \end{aligned}$$

Experiment $3N-3$. We set $z_{N-2} = r_2, X_{N-1} = r_1 s_{N-1} + e'_{N-1}, z_0 = r_1 + r_2$ where $r_1, r_2 \leftarrow R_q$. The rest are same as the previous experiment.

$$\mathsf{Exp}_{3N-3} := \left\{ \begin{array}{ll} a, r_1, r_2 \leftarrow R_q; s_{N-1}, e_{N-1} \leftarrow \chi_{\sigma_1}; z_0 = r_1 + r_2; & \\ z_i \leftarrow R_q \text{ for } 1 \leq i \leq N-3; z_{N-2} = r_2; & \\ z_{N-1} = as_{N-1} + e_{N-1}; e'_0 \leftarrow \chi_{\sigma_2}; e'_{N-1} \leftarrow \chi_{\sigma_1}; & \\ X'_0 = -\sum_{i=1}^{N-1} X_i + e'_0; X_i \leftarrow R_q \text{ for } 1 \leq i \leq N-2; & \\ X_{N-1} = r_1 s_{N-1} + e'_{N-1}; e''_{N-1} \leftarrow \chi_{\sigma_1}; & : (\mathsf{T}, \mathsf{sk}) \\ Y_{N-1,N-1} = X_{N-1} + z_{N-2}s_{N-1} + e''_{N-1}; & \\ Y_{N-1,(N-1)+j} = X_{(N-1)+j} + Y_{N-1,(N-1)+(j-1)}; & \\ b_{N-1} = \sum_{j=0}^{N-1} Y_{N-1,(N-1)+j}; & \\ (\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1}); \mathsf{sk} = \mathcal{H}(k_{N-1}); & \\ \mathsf{T} = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \cdots, X_{N-1}, \mathsf{rec}) & \end{array} \right\}$$

Since r_1, r_2 are uniform, so does $z_0 = r_1 + r_2$. For both Experiment $3N-4$ and $3N-3$, z_{N-2} and z_0 are uniform. Then, we have $\Pr_{3N-3}[\mathsf{Query}] = \Pr_{3N-4}[\mathsf{Query}]$.

Experiment $3N-2$. We replace $z_{N-1}, X_{N-1}, z_{N-2}s_{N-1} + e''_{N-1}$ into uniform element in R_q. The rest are same as the previous experiment.

$$\mathsf{Exp}_{3N-2} := \left\{ \begin{array}{ll} a, r_3 \leftarrow R_q; s_{N-1}, e_{N-1} \leftarrow \chi_{\sigma_1}; z_i \leftarrow R_q \text{ for } i \in [N]\,; & \\ e'_0 \leftarrow \chi_{\sigma_2}; & \\ X'_0 = -\sum_{i=1}^{N-1} X_i + e'_0; X_i \leftarrow R_q \text{ for } 1 \leq i \leq N-1; & \\ Y_{N-1,N-1} = X_{N-1} + r_3; & : (\mathsf{T}, \mathsf{sk}) \\ Y_{N-1,(N-1)+j} = X_{(N-1)+j} + Y_{N-1,(N-1)+(j-1)}; & \\ b_{N-1} = \sum_{j=0}^{N-1} Y_{N-1,(N-1)+j}; & \\ (\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1}); \mathsf{sk} = \mathcal{H}(k_{N-1}); & \\ \mathsf{T} = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \cdots, X_{N-1}, \mathsf{rec}) & \end{array} \right\}$$

Between Experiment $3N-3$ and Experiment $3N-2$, we replace three RLWE instances into random. Hence,

$$|\Pr_{3N-2}[\mathsf{Query}] - \Pr_{3N-3}[\mathsf{Query}]| \leq \mathsf{Adv}^{RLWE}_{n,q,\chi_{\sigma_1},3}(t_1)$$

Experiment $3N-1$. We replace $Y_{N-1,N-1}, Y_{N-1,(N-1)+j}, b_{N-1}$ into uniform element in R_q. The rest are same as the previous experiment.

$$\mathsf{Exp}_{3N-1} := \left\{ \begin{array}{ll} a, r_1, r_2 \leftarrow R_q; s_{N-1}, e_{N-1} \leftarrow \chi_{\sigma_1}; z_i \leftarrow R_q \text{ for } i \in [N]\,; & \\ e'_0 \leftarrow \chi_{\sigma_2}; & \\ X'_0 = -\sum_{i=1}^{N-1} X_i + e'_0; X_i \leftarrow R_q \text{ for } 1 \leq i \leq N-1; & \\ Y_{N-1,(N-1)+j} \leftarrow R_q \text{ for } j \in [N]\,; & : (\mathsf{T}, \mathsf{sk}) \\ b_{N-1} \leftarrow R_q; & \\ (\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1}); \mathsf{sk} = \mathcal{H}(k_{N-1}); & \\ \mathsf{T} = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \cdots, X_{N-1}, \mathsf{rec}) & \end{array} \right\}$$

For both Experiment $3N-2$ and Experiment $3N-1$, $Y_{N-1,N-1}, Y_{N-1,(N-1)+j}$, and b_{N-1} are all uniform since r_3 is uniform in Experiment $3N-2$. Then, we have $\Pr_{3N-1}[\mathsf{Query}] = \Pr_{3N-2}[\mathsf{Query}]$.

Experiment $3N$. We replace k_{N-1} into uniform element k'_{N-1} in $\{0,1\}^\lambda$. The rest are same as the previous experiment.

$$\mathsf{Exp}_{3N} := \left\{ \begin{array}{ll} a, r_1, r_2 \leftarrow R_q; s_{N-1}, e_{N-1} \leftarrow \chi_{\sigma_1}; z_i \leftarrow R_q \text{ for } i \in [N]\,; & \\ e'_0 \leftarrow \chi_{\sigma_2}; & \\ X'_0 = -\sum_{i=1}^{N-1} X_i + e'_0; X_i \leftarrow R_q \text{ for } 1 \leq i \leq N-1; & \\ Y_{N-1,(N-1)+j} \leftarrow R_q \text{ for } j \in [N]\,; & : (\mathsf{T}, \mathsf{sk}) \\ b_{N-1} \leftarrow R_q; & \\ (\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1}); & \\ k'_{N-1} \leftarrow \{0,1\}^\lambda; \mathsf{sk'} = \mathcal{H}(k'_{N-1}); & \\ \mathsf{T} = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \cdots, X_{N-1}, \mathsf{rec}) & \end{array} \right\}$$

Between Experiment $3N-1$ and Experiment $3N$, we replace k_{N-1} from $\mathsf{recMsg}(b_{N-1})$ into random. Hence,

$$|\Pr_{3N}[\mathsf{Query}] - \Pr_{3N-1}[\mathsf{Query}]| \leq \mathsf{Adv}_{\mathsf{KeyRec}}(t_2)$$

t_2 is the time to break KeyRec algorithm which is the sum of t and some minor overhead $\mathcal{O}(t_{ring})$ for simulation.

Since adversary attacking STUG makes at most q_E queries to the random oracle, we have $\Pr_1[\mathsf{Query}] = \frac{q_E}{2^\lambda}$, which is negligible in λ.

From Experiment 1 to Experiment $3N$, we have

$$\Pr_1[\mathsf{Query}] \le N \cdot \mathsf{Adv}^{RLWE}_{n,q,\chi_{\sigma_1},3}(t_1) + \mathsf{Adv}_{\mathsf{KeyRec}}(t_2) + \frac{q_E}{2^\lambda}.$$

as expected. With the Lemma 6 and $\mathsf{Adv}^{\mathsf{KE}}_{\mathsf{STUG}}(t, q_E) \le \Pr_0[\mathsf{Query}]$, we derive the result of the theorem.

$\square$

Theorem 10. The authenticated GKE protocol STAG described in Section 7.3 is secure against active adversary under RLWE assumption, achieves forward secrecy and satisfies the following:

$$\mathsf{Adv}^{\mathsf{AKE}}_{\mathsf{STAG}}(t, q_E, q_S) \le \mathsf{Adv}^{\mathsf{KE}}_{\mathsf{STUG}}(t', q_E + \frac{q_S}{2}) + |\mathcal{P}|\mathsf{Adv}_{\mathsf{DSig}}(t')$$

where $t' \le t + (|\mathcal{P}|q_E + q_S)t_{\mathsf{STAG}}$ when t_{STAG} is the time required for execution of STAG by any one of the protocol participants.

Proof. From an adversary $\mathcal{A}'$ that attacks STAG, we construct an adversary $\mathcal{A}$ that attacks STUG. We divide the event Succ that $\mathcal{A}'$ wins the security game defined in Chapter 7.1 into the one that $\mathcal{A}'$ can forge a signature and the one that $\mathcal{A}'$ cannot forge a signature.

For the former case, we claim that the probability of event Forge that the adversary can forge a signature is bounded by $|\mathcal{P}|\mathsf{Adv}_{\mathsf{DSig}}(t')$ where $|\mathcal{P}|$ is the number of participants.

Suppose event Forge occurs. Then, $\mathcal{A}'$ makes a query of the type $\mathsf{Send}(V, i, m)$ where m is either of the form $m = P_i \mid 1 \mid z_i$ or of the form $m = P_i \mid 2 \mid X_i \mid d_i$ with $\mathcal{V}(pk_{P_i}, m, \delta_m) = 1$ where δ_m is not output by any instance of P_i on the message m.

We construct an algorithm $\mathcal{F}$ that forges a signature for a signature scheme DSig using $\mathcal{A}'$. Given a public key pk, $\mathcal{F}$ chooses a random party $P \in \mathcal{P}$ and sets $pk_P = pk$. The other public/secret keys are honestly generated. Then, $\mathcal{F}$ simulates all queries of $\mathcal{A}'$ and obtains the proper signatures with respect to pk_P from its signing oracle. If $\mathcal{A}'$ outputs a valid message/signature pair with respect to $pk_P = pk$ for any party $P \in \mathcal{P}$, $\mathcal{F}$ outputs this pair as a forgery. The success probability of $\mathcal{F}$ is equal to $\dfrac{\Pr[\mathsf{Forge}]}{|\mathcal{P}|}$ and hence, $\Pr[\mathsf{Forge}] \le |\mathcal{P}|\mathsf{Adv}_{\mathsf{DSig}}(t')$.

For the latter case, we claim that $\mathcal{A}$ can simulate oracle queries of $\mathcal{A}$ by its own oracles. Suppose $\mathcal{A}'$ makes an $\mathsf{Execute}$ query with an instance $\{(P_0, i_0), \cdots, (P_k, i_k)\}$. Then, $\mathcal{A}$ defines a set of instances $S = \{(P_0, i_0), \cdots, (P_k, i_k)\}$ and sends S to $\mathsf{Execute}$ oracle to obtain a transcript T of STUG. Then, (S, T) are added to the set tlist which stores session identity/transcript pairs. $\mathcal{A}$ outputs a transcript T' of STAG by expanding T and returns T' to $\mathcal{A}'$.

For Send query in $\mathcal{A}'$, we have two types of Send queries since for each instance (P_0, i_0), there is a first send query Send_0 to start a new session and the other send queries with a message/signature pair. With an instance $\{(P_0, i_0), \cdots, (P_k, i_k)\}$, $\mathsf{Send}_0(P_j, i_j\{P_0, \cdots, (P_k)\} \setminus P_j)$ for each $0 \le j \le k$. $\mathcal{A}$ defines $S = \{(P_0, i_0), \cdots, (P_k, i_k)\}$ and sends S to $\mathsf{Execute}$ oracle to obtain a transcript T of STUG. Then, (S, T) is added to tlist. For other send queries, $\mathcal{A}$ verifies the query according to Protocol 6. If the verifications fail, $\mathcal{A}$ aborts an instance (P_j, i_j). Otherwise, $\mathcal{A}$ finds (S, T) from tlist such that $(P_j, i_j) \in S$ and from T, it finds the appropriate message with respect to (P_j, i_j) in $\mathcal{A}'$ and returns the public information to $\mathcal{A}'$.

For Reveal and Test queries in $\mathcal{A}'$, if a session is terminated properly with an instance (P_j, i_j), T' is well defined. Then, $\mathcal{A}$ finds (S, T) where $(P_j, i_j) \in S$ and runs Reveal and Test queries for a transcript T. Then, the result is sent to $\mathcal{A}'$.

If Forge occurs, the success probability becomes $\frac{1}{2}$ since $\mathcal{A}$ aborts the instance and outputs a random value.

$$\begin{aligned}
\mathsf{Adv}^{\mathcal{A}}_{\mathsf{STUG}} &= 2|\Pr_{\mathcal{A}}[\mathsf{Succ}] - \frac{1}{2}| \\
&= 2\ |\Pr_{\mathcal{A}'}[\mathsf{Succ} \wedge \overline{\mathsf{Forge}}] \\
&\quad + \Pr_{\mathcal{A}'}[\mathsf{Succ} \wedge \mathsf{Forge}] - \frac{1}{2}| \\
&= 2\ |(\Pr_{\mathcal{A}'}[\mathsf{Succ}] - \Pr_{\mathcal{A}'}[\mathsf{Succ} \wedge \mathsf{Forge}]) \\
&\quad + (\Pr_{\mathcal{A}'}[\mathsf{Succ} \mid \mathsf{Forge}]\ \Pr_{\mathcal{A}'}[\mathsf{Forge}]) - \frac{1}{2}| \\
&= 2\ |\Pr_{\mathcal{A}'}[\mathsf{Succ}] - \Pr_{\mathcal{A}'}[\mathsf{Succ} \wedge \mathsf{Forge}] \\
&\quad + \frac{1}{2}\ \Pr_{\mathcal{A}'}[\mathsf{Forge}] - \frac{1}{2}| \\
&\geq 2\ |\Pr_{\mathcal{A}'}[\mathsf{Succ}] - 1| \\
&\quad - |\Pr_{\mathcal{A}'}[\mathsf{Forge}] - 2\ \Pr_{\mathcal{A}'}[\mathsf{Succ} \wedge \mathsf{Forge}]| \\
&\geq \mathsf{Adv}^{\mathcal{A}'}_{\mathsf{STAG}} - \Pr_{\mathcal{A}'}[\mathsf{Forge}].
\end{aligned}$$

Then, $\mathcal{A}$ makes $\mathsf{Execute}$ queries for each $\mathsf{Execute}$ and Send_0 queries in $\mathcal{A}'$. Since a session has at least two instances, the number of $\mathsf{Execute}$ queries in $\mathcal{A}$ is at most $q_E + \frac{q_S}{2}$. Thus,

$$\mathsf{Adv}^{\mathsf{AKE}}_{\mathsf{STAG}}(t, q_E, q_S) \leq \mathsf{Adv}^{\mathsf{KE}}_{\mathsf{STUG}}(t', q_E + \frac{q_S}{2}) + |\mathcal{P}|\mathsf{Adv}_{\mathsf{DSig}}(t').$$

□

Theorem 11. The dynamic authenticated GKE protocol DRAG described in Section 7.4 is secure against active adversary under RLWE assumption, achieves forward secrecy and satisfies the following:

$$\mathsf{Adv}^{\mathsf{AKE}}_{\mathsf{DRAG}}(t, q_E, q_J, q_L, q_S) \leq \mathsf{Adv}^{\mathsf{KE}}_{\mathsf{DRAG}}(t', q_E + \frac{q_J + q_L + q_S}{2}) + |\mathcal{P}|\mathsf{Adv}_{\mathsf{DSig}}(t')$$

where $t' \leq t + (|\mathcal{P}|q_E + q_J + q_L + q_S)t_{\mathsf{DRAG}}$ when t_{DRAG} is the time required for execution of DRAG by any one of the protocol participants.

Proof. Similar to Theorem 10, we separate the event Succ that adversary $\mathcal{A}'$ wins into two cases: one with forging a signature and the other without forging. Then, we design how to answer $\mathsf{Execute}$, Join, Leave and Send queries from DRAG using $\mathsf{Execute}$ queries from STUG.

From an adversary $\mathcal{A}'$ which attacks DRAG, we construct an adversary $\mathcal{A}$ who attacks STUG.

For the former case, the proof is the same as Theorem 3 and we obtain that $\Pr[\mathsf{Forge}] \leq |\mathcal{P}|\mathsf{Adv}_{\mathsf{DSig}}(t')$.

For the latter case, we claim that $\mathcal{A}$ can simulate all oracle queries of $\mathcal{A}'$ by its own oracles. $\mathsf{Execute}$ query in $\mathcal{A}'$ can be returned as the same procedure of the proof of Theorem 3.

For Send query in $\mathcal{A}'$, we have two special types of Send queries as the join send query Send_J and the leave send query Send_L to initiate Join and Leave queries in $\mathcal{A}'$.

We define three lists Tlist, Jlist and Llist to store the result from $\mathsf{Execute}$, Join and Leave queries as a set.

If a set $S_J = \{(P_{k+1}, i_{k+1}), \cdots, (P_{k+l}, i_{k+l})\}$ of unused instances wants to join the group $S = \{(P_0, i_0), \cdots, (P_k, i_k)\}$, $\mathcal{A}'$ sends $\mathsf{Send}_J(P_j, i_j, \{P_0, \cdots, (P_k)\})$ for each $k+1 \leq j \leq k+l$ to initiate $\mathsf{Join}(S, S_J)$ query. $\mathcal{A}$ finds whether S is in Tlist with (S, T), Jlist with (S', S'', T) where $S = S' \cup S''$ or

Llist with (S', S'', T) where $S = S' \setminus S''$. If nothing is found, $\mathcal{A}$ runs $\mathsf{Execute}$ oracle to get a transcript T and store (S, T) into Tlist. When transcript T is found, $\mathcal{A}$ runs Reveal oracle to obtain sk and simulates a membership addition procedure $\mathsf{A.Join}$ to obtain a transcript T'. Then, (S, S_J, T') is added to Jlist.

Similarly, when a set S_L of unused instances wants to leave the group $S = \{(P_0, i_0), \cdots, (P_k, i_k)\}$, $\mathcal{A}'$ sends Send_L for each party in S_L to initiate $\mathsf{Leave}(S, S_L)$ query. Then, (S, S_L, T') is added to Llist.

For $\mathsf{Join}(S, S_J)$ and $\mathsf{Leave}(S, S_L)$ queries in $\mathcal{A}'$, $\mathcal{A}$ finds (S, S_J, T) in Jlist and (S, S_J, T) in Llist for a transcript T, respectively. Then, the result is sent to $\mathcal{A}'$.

Both Reveal and Test queries in $\mathcal{A}'$ can be returned with the same procedure in the proof of Theorem 3.

If Forge occurs, the success probability becomes $\frac{1}{2}$ since $\mathcal{A}$ aborts the instance and outputs a random value. Then,

$$\mathsf{Adv}^{\mathcal{A}}_{\mathsf{STUG}} \geq \mathsf{Adv}^{\mathcal{A}'}_{\mathsf{DRAG}} - \Pr_{\mathcal{A}'}[\mathsf{Forge}].$$

Then, $\mathcal{A}$ makes $\mathsf{Execute}$ queries for each $\mathsf{Execute}$, Send_J, Send_L, and Send_0 query in $\mathcal{A}'$. A session has at least two instances, and the number of $\mathsf{Execute}$ queries for non-special Send queries is at most $\frac{q_S - q_J - q_L}{2}$. Hence, the number of $\mathsf{Execute}$ queries in $\mathcal{A}$ is at most $q_E + \frac{q_S + q_J + q_L}{2}$. Thus,

$$\begin{aligned} &\mathsf{Adv}^{\mathsf{AKE}}_{\mathsf{DRAG}}(t, q_E, q_J, q_L, q_S) \leq \\ &\qquad \mathsf{Adv}^{\mathsf{KE}}_{\mathsf{STUG}}(t', q_E + \frac{q_J + q_L + q_S}{2}) + |\mathcal{P}|\mathsf{Adv}_{\mathsf{DSig}}(t'). \end{aligned}$$

□

Chapter 8. Implementation

We instantiate and implement DRUG for two purposes: to provide the proof of concept and check the performance of our protocol. Since DRUG is a 3-round GKE protocol from RLWE assumption with a generic key reconciliation mechanism, we instantiate DRUG with one of the previous key reconciliation mechanisms whose implementation is open in public domain. We implement STUG considering the network topology and membership addition/deletion procedures based on the source code of STUG. Then, we analyze the test results from our implementation. Pseudocode of our implementation is given in Appendix D. The full reference source code is available in our GitHub address (https://github.com/hansh17/DRAGKE).

8.1 Instantiation

8.1.1 Restrictions on the Parameters

To instantiate DRUG, we should consider the restrictions for choosing parameters. There are three restrictions from security analysis and performance requirements.

Considering Theorems 1 and 2, we set a statistical security parameter $\rho = 256$ that is related to the correctness and a computational security parameter $\lambda = 128$ that ensures level 1 NIST security that is as hard as breaking AES128. $\lambda = 128$ is selected for a practical reason since $\lambda = 256$, which ensures level 5 NIST security that is as hard as breaking AES256, requires more than a day to make a precomputed table for sampling.

8.1.2 Parameter Selection

Some ring parameters, such as the dimension n and the modulus q, are highly dependent on the implementation of the basic ring operations. Generally, n can be a power of two as cyclotomic rings are used for RLWE. q can be any integers since decisional RLWE is hard over a prime cyclotomic ring with any modulus.

We chose the key reconciliation mechanism of Bos *et al.*'s protocol [12] where $n = 1024$, $q = 2^{32} - 1$, and $\sigma_1 = 8/\sqrt{2\pi}$ that provides both reasonable security level and practical implementation result. To maximize the number of group members N in DRUG with those parameters, we select other parameters as $\sigma_2 = 14,198,340/\sqrt{2\pi}$, $\beta_{\mathsf{Rec}} = 2^{29} - 1$, and $\beta_{\mathsf{Rényi}} = 5,664,317$ where $\beta_{\mathsf{Rec}} = q/8$ as shown in TABLE 8.1, for $N = 6$. If we increase the number of group members in DRUG like real-world applications, both σ_2 and $\beta_{\mathsf{Rényi}}$ will increase to around 2^{25} while other parameters are fixed, for N is around 2^5. As we increase the number into hundreds, we have to increase parameters σ_2, $\beta_{\mathsf{Rényi}}$, q, and β_{rec} into around 2^{40} which might become not very practical.

Table 8.1: Parameter choice

ρ	λ	N	n	q	σ_1	σ_2
256	128	6	1024	$2^{32}-1$	$8/\sqrt{2\pi}$	$14,198,340/\sqrt{2\pi}$

8.2 Network Topology

In our parameter settings, up to six peers can participate in DRUG. Each peer broadcasts the computed intermediate values to all other peers in DRUG.

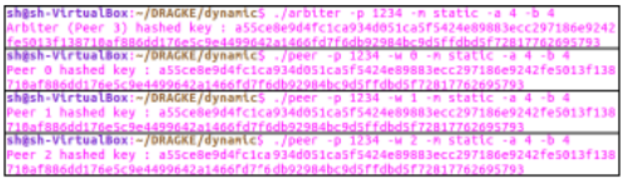
To make broadcasting easy, we deployed an arbiter-based network for communication. Different from TA who has a dedicated role in a protocol, an arbiter is a designated party who all group members have agreed on in the initialization phase that does broadcasting, being independent of the protocol. The role of an arbiter is similar to a public bulletin board, except that the arbiter actively broadcasts the received message to the other peers while the public bulletin board is queried by the peers. Since any peer can behave as an arbiter in a fully connected network, we pick P_{N-1} as the arbiter for simple implementation.

In summary, the roles of an arbiter are as follows: i) participates in a protocol ii) receives public information such as z_i or X_i from a peer in DRUG, and broadcasts the information to the other peers iii) runs the key reconciliation mechanism and broadcasts rec to other peers (since we selected P_{N-1} as the arbiter).

The roles of peers ($P_0, P_1, \cdots$ and P_{N-2}) are as follows: i) participates in a protocol ii) calculates and sends public information to the arbiter iii) calculates the group secret key from the data received from the arbiter.

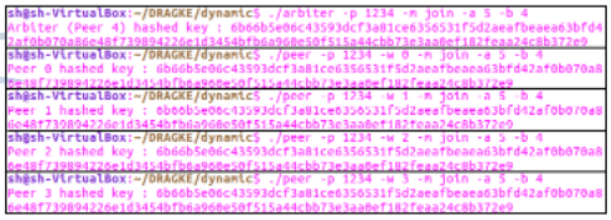
8.3 Our Tests

We conducted three tests to verify correctness (Test 0) and performance (Tests 1 and 2). The test environment is as follows: Intel(R) CPU i5-8250, RAM 8GB, and OS Ubuntu v16.04.5 LTS. Since we use a virtual machine, only a partial power of the computer is used for performance evaluation. We use two processors with a 100% execution cap for CPU and 4GB for RAM. gcc v5.4.0 is used as a compiler with $-O3$ optimizations.



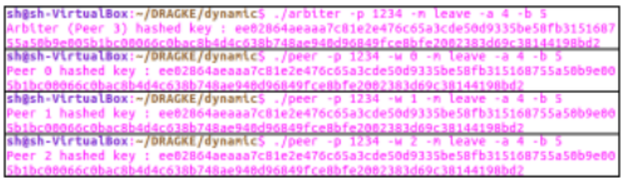
```
sh@sh-VirtualBox:~/DRAGKE/dynamic$ ./arbiter -p 1234 -m static -a 4 -b 4
Arbiter (Peer 3) hashed key : a55ce8e9d4fc1ca934d051ca5f5424e89883ecc297186e9242
fe5013f138710af886dd176e5c9e4499642a1466fd7f6db92984bc9d5ffdbd5f72817762695793
sh@sh-VirtualBox:~/DRAGKE/dynamic$ ./peer -p 1234 -w 0 -m static -a 4 -b 4
Peer 0 hashed key : a55ce8e9d4fc1ca934d051ca5f5424e89883ecc297186e9242fe5013f138
710af886dd176e5c9e4499642a1466fd7f6db92984bc9d5ffdbd5f72817762695793
sh@sh-VirtualBox:~/DRAGKE/dynamic$ ./peer -p 1234 -w 1 -m static -a 4 -b 4
Peer 1 hashed key : a55ce8e9d4fc1ca934d051ca5f5424e89883ecc297186e9242fe5013f138
710af886dd176e5c9e4499642a1466fd7f6db92984bc9d5ffdbd5f72817762695793
sh@sh-VirtualBox:~/DRAGKE/dynamic$ ./peer -p 1234 -w 2 -m static -a 4 -b 4
Peer 2 hashed key : a55ce8e9d4fc1ca934d051ca5f5424e89883ecc297186e9242fe5013f138
710af886dd176e5c9e4499642a1466fd7f6db92984bc9d5ffdbd5f72817762695793
```

Figure 8.1: Group secret key of STUG (N=4)



```
sh@sh-VirtualBox:~/DRAGKE/dynamic$ ./arbiter -p 1234 -m join -a 5 -b 4
Arbiter (Peer 4) hashed key : 6b66b5e06c43593dcf3a81ce6356531f5d2aeafbeaea63bfd4
2af0b070a86e48f739894226e1d3454bfb6a968e50f515a44cbb73e3aa8ef182feaa24c8b372e9
sh@sh-VirtualBox:~/DRAGKE/dynamic$ ./peer -p 1234 -w 0 -m join -a 5 -b 4
Peer 0 hashed key : 6b66b5e06c43593dcf3a81ce6356531f5d2aeafbeaea63bfd42af0b070a8
6e48f739894226e1d3454bfb6a968e50f515a44cbb73e3aa8ef182feaa24c8b372e9
sh@sh-VirtualBox:~/DRAGKE/dynamic$ ./peer -p 1234 -w 1 -m join -a 5 -b 4
Peer 1 hashed key : 6b66b5e06c43593dcf3a81ce6356531f5d2aeafbeaea63bfd42af0b070a8
6e48f739894226e1d3454bfb6a968e50f515a44cbb73e3aa8ef182feaa24c8b372e9
sh@sh-VirtualBox:~/DRAGKE/dynamic$ ./peer -p 1234 -w 2 -m join -a 5 -b 4
Peer 2 hashed key : 6b66b5e06c43593dcf3a81ce6356531f5d2aeafbeaea63bfd42af0b070a8
6e48f739894226e1d3454bfb6a968e50f515a44cbb73e3aa8ef182feaa24c8b372e9
sh@sh-VirtualBox:~/DRAGKE/dynamic$ ./peer -p 1234 -w 3 -m join -a 5 -b 4
Peer 3 hashed key : 6b66b5e06c43593dcf3a81ce6356531f5d2aeafbeaea63bfd42af0b070a8
6e48f739894226e1d3454bfb6a968e50f515a44cbb73e3aa8ef182feaa24c8b372e9
```

Figure 8.2: U.Join procedure in DRUG (N=4 → N=5)



```
sh@sh-VirtualBox:~/DRAGKE/dynamic$ ./arbiter -p 1234 -m leave -a 4 -b 5
Arbiter (Peer 3) hashed key : ee02864aeaaa7c81e2e476c65a3cde50d9335be58fb3151687
55a58b9e805b1bc08066c0bac8b4d4c638b748ae948d96849fce8bfe2082383d69c38144198bd2
sh@sh-VirtualBox:~/DRAGKE/dynamic$ ./peer -p 1234 -w 0 -m leave -a 4 -b 5
Peer 0 hashed key : ee02864aeaaa7c81e2e476c65a3cde50d9335be58fb315168755a58b9e80
5b1bc08066c0bac8b4d4c638b748ae948d96849fce8bfe2082383d69c38144198bd2
sh@sh-VirtualBox:~/DRAGKE/dynamic$ ./peer -p 1234 -w 1 -m leave -a 4 -b 5
Peer 1 hashed key : ee02864aeaaa7c81e2e476c65a3cde50d9335be58fb315168755a58b9e80
5b1bc08066c0bac8b4d4c638b748ae948d96849fce8bfe2082383d69c38144198bd2
sh@sh-VirtualBox:~/DRAGKE/dynamic$ ./peer -p 1234 -w 2 -m leave -a 4 -b 5
Peer 2 hashed key : ee02864aeaaa7c81e2e476c65a3cde50d9335be58fb315168755a58b9e80
5b1bc08066c0bac8b4d4c638b748ae948d96849fce8bfe2082383d69c38144198bd2
```

Figure 8.3: U.Leave procedure in DRUG (N=5 → N=4)

8.3.1 Test 0: Verifying the Success of Key Exchange

In Test 0, we verified that peers can successfully exchange the group secret key using our implementation. We built two executable programs; one for the arbiter and the other for peers who are not an arbiter. These executable programs support all three modes (static, join and leave) with the −m option. An index of the peer and the number of members before and after the dynamic operations can be provided.

Our programs are configured to run on a local environment to exclude network latencies while measuring time on the subsequent tests. Each peer is deployed in one independent process, and communication between peers and the arbiter is done by sockets toward the localhost. Small modifications would allow programs to communicate through an actual network; in the actual network, the values such as the peer indices should be determined before a protocol is given as the argument. In Fig. 8.1, we can observe that a group of four outputs the same value as the agreed group secret key. Figs. 8.2 and 8.3 demonstrate that the U.Join and U.Leave procedures were done without errors.

8.3.2 Test 1: Performance Check on Components

In Test 1, we checked the runtime and cycles of each operation and function of our implementation. TABLE 8.2 shows the performance of each operation on average after running 200 times.

As we can see in TABLE 8.2, random sampling from χ_{σ_2} takes a longer time than random sampling from χ_{σ_1}. This is a result of the difference in size of $\sigma_1 = 8/\sqrt{2\pi}$ and $\sigma_2 = 14,198,340/\sqrt{2\pi}$. The ring polynomial addition running time is almost 0, but the ring polynomial multiplication running time is 185 μsec. Therefore ring polynomial multiplication and random sampling is an important factor in performance as we stated in Chapter 10.

Table 8.2: Average runtime and cycles of each operation

Operation	runtime(μsec)	cycles
Random sampling from χ_{σ_1}	195	352,369
Random sampling from χ_{σ_2}	1,043	1,879,027
Ring polynomial addition	≈ 0	1,398
Ring polynomial multiplication	185	334,047

TABLE 8.3 shows the performance of each function. We measure the computational time for z_i, X_i, reconcile, and the group secret key. As we can see in TABLE 8.3, computing X_0 takes longer than computing X_i due to the sampling from χ_{σ_2}. Even though only one peer, P_0, samples values from χ_{σ_2}, its runtime is much longer than random samplings from χ_{σ_1} or ring polynomial multiplications. Hence, random sampling from χ_{σ_2} is another important factor in measuring the performance of DRUG.

Table 8.3: Average runtime and cycles of each function

Function	runtime(μsec)	cycles
Compute z_i (**Phase c1**)	493	889,014
Compute X_0 (**Phase c2**)	1,353	2,435,457
Compute X_i (**Phase c2**)	344	619,686
Compute reconcile (**Phase c3.1**)	500	901,395
Compute the group secret key (**Phase c3.2**)	209	376,906

To claim that the performance of DRUG is reasonable compared to other previous key exchange protocols, we compare our protocol with the previously standardized two-party key exchange protocols like Diffie-Hellman key exchange (DH), MQV [42], and their elliptic curve variants (ECDH, ECMQV). For the sake of simplicty, we limit the comparison with the Crypto++ library [1]. Given the 128-bit security level, key size should be 3072 bit for DH and MQV and 283 bit for elliptic curve variants. The runtime of our protocol takes around 3 msec which is comparable to the runtime of Diffie-Hellman-like key exchange protocols, which takes around 2 to 3 msec. Our protocol is faster than elliptic curve variants whose runtime requires around 6 to 8 msec. We increase the security without losing efficiency by adopting RLWE setting since most of current key exchange protocols can be totally broken by quantum adversaries.

8.3.3 Test 2: Performance Check on Dynamic Operations

In Test 2, we checked the total runtime of U.Join and U.Leave procedures. The measurement was performed when the number of group members changes from four to five and five to four. The whole procedure of GKE is measured, *i.e.*, the protocol ends when all peers calculate a group secret key. Since the calculation in each phase is performed in parallel, we add the longest time for calculation in each phase.

In TABLE 8.4, U.Join procedure takes 3,168 μsec and U.Leave procedure takes 2,956 μsec, which is reasonable for practical use.

Table 8.4: Performance evaluation of U.Join and U.Leave

	U.Join	U.Leave
Total runtime (μsec)	3,168	2,956

Chapter 9. Key-resuable Dynamic GKE from RLWE

9.1 Tweaked Two-neighbour Attack

If we apply Ding's reconciliation mechanism to DRAG, $\mathsf{recMsg}(\boldsymbol{b}_{N-1})$ outputs $\mathsf{rec} \leftarrow S(\boldsymbol{b}_{N-1})$ and $\boldsymbol{k}_{N-1} \leftarrow E(\boldsymbol{b}_{N-1}, \mathsf{rec})$ and $\mathsf{recKey}(\boldsymbol{b}_i, \mathsf{rec})$ outputs $\boldsymbol{k}_i \leftarrow E(\boldsymbol{b}_i, \mathsf{rec})$.

We assume that a public value $\boldsymbol{a} \in R_q$ and public keys $\boldsymbol{z}_i$ for each party P_i are fixed except P_{N-2}. $\mathcal{A}$ can initiate many sessions with all parties in the group and can access to the oracle $\mathcal{S}$. In performing the attack, an adversary $\mathcal{A}$ plays the role of two parties P_0 and P_1 instead of P_{N-2} and P_{N-3} in the original two-neighbour attack. $\mathcal{A}$ creates key pairs $(\boldsymbol{s}_0, \boldsymbol{z}_0)$ of P_0 by deviating from the protocol. We denote the public key $\boldsymbol{z}_0$ deviated by $\mathcal{A}$ as $\boldsymbol{z}_{\mathcal{A}}$ and the corresponding secret key $\boldsymbol{s}_0$ and error $\boldsymbol{e}_0$ deviated by $\mathcal{A}$ as $\boldsymbol{s}_{\mathcal{A}}$ and $\boldsymbol{e}_{\mathcal{A}}$, respectively.

We describe an attack on simplified DRAG where the error term $\boldsymbol{e}''_{N-1}$ is not added to the key computation of $\boldsymbol{Y}_{N-1,N-1}$. We set an oracle $\mathcal{S}$ that simulates party P_{N-1}'s action from a given input public key. On receiving $\boldsymbol{z}_{\mathcal{A}}$ from $\mathcal{A}$, S computes $\boldsymbol{b}_{N-1}$ and outputs $(\mathsf{rec}, \boldsymbol{k}_{N-1})$ from $\boldsymbol{b}_{N-1}$ according to the protocol.

Then, $Y_{N-1,N-1} = X_{N-1} + z_{N-2}s_{N-1} = z_0 s_{N-1}$ and all $Y_{N-1,N-1+j} = z_0 s_{N-1} + (\text{constant})$ except for $Y_{N-1,1} = X_1 + Y_{N-1,0}$ since X_1 contains the term z_0. Hence, if $\mathcal{A}$ can control party P_1 to output X_1 to be some constant value, we obtain that $Y_{N-1,N-1+j} = z_0 s_{N-1} + (\text{constant})$ for all j and $b_{N-1} = \sum_{j=0}^{N-1} Y_{N-1,N-1+j} = N z_0 s_{N-1} + (\text{constant})$. From the assumption, we set $z_0 = k e_{\mathcal{A}}$ and describe the attack on DRAG as follows:

Step 1. $\mathcal{A}$ invokes the oracle $\mathcal{S}$ with input $\boldsymbol{z}_{\mathcal{A}} = k\boldsymbol{e}_{\mathcal{A}}$ $(k = 0, 1, \cdots, q-1)$ where $\boldsymbol{s}_{\mathcal{A}}$ is 0 and $\boldsymbol{e}_{\mathcal{A}}$ is the identity element 1 in R_q so that $\boldsymbol{b}_{N-1}$ becomes $kN\boldsymbol{s}_{N-1} + (\text{constant})$. Since kN represents all elements in $\mathbb{Z}_q$, $\mathcal{A}$ can make a correct guess of the value of $\boldsymbol{s}_{N-1}[i]$ based on the number of times the signal of rec for $\{0, \boldsymbol{s}_{N-1}, \cdots, (q-1)\boldsymbol{s}_{N-1}\}$ changes for each coefficient $\boldsymbol{s}_{N-1}[i]$.

Step 2. $\mathcal{A}$ invokes $\mathcal{S}$ with input $(1+x)\boldsymbol{z}_{\mathcal{A}} = (1+x)k\boldsymbol{e}_{\mathcal{A}}$ $(k = 0, 1, \cdots, q-1)$. $\mathcal{A}$ is able to see the key reconciliation value of $(1+x)k\boldsymbol{s}_{N-1}$ that is the output by $\mathcal{S}$. Thus, again by checking the number of signal changes, $\mathcal{A}$ finds values of the coefficients of $(1+x)\boldsymbol{s}_{N-1}$, which are $\boldsymbol{s}_{N-1}[0] - \boldsymbol{s}_{N-1}[n-1], \boldsymbol{s}_{N-1}[1] + \boldsymbol{s}_{N-1}[0], ..., \boldsymbol{s}_{N-1}[n-1] + \boldsymbol{s}_{N-1}[n-2]$ up to sign.

Step 3. From **Steps 1** and **2**, we can determine if each pair of coefficients $\boldsymbol{s}_{N-1}[i]$, $\boldsymbol{s}_{N-1}[i+1]$ have equal or opposite signs, hence narrowing down to only two possibilities such that the guess $\boldsymbol{s}'_{N-1} = \boldsymbol{s}_{N-1}$ or $-\boldsymbol{s}_{N-1}$.

Step 4. Since $\boldsymbol{a}$ and $\boldsymbol{z}_{N-1}$ are public, $\mathcal{A}$ computes $\boldsymbol{z}_{N-1} - \boldsymbol{a}\boldsymbol{s}'_{N-1}$ and verifies the distribution of the result. If $\mathcal{A}$ correctly guesses the sign of $\boldsymbol{s}_{N-1}[0]$ and so does all the coefficients of $\boldsymbol{s}_{N-1}$, the resulting distribution of $\boldsymbol{z}_{N-1} - \boldsymbol{a}\boldsymbol{s}'_{N-1}$ is same as the distribution of $\boldsymbol{z}_{N-1} - \boldsymbol{a}\boldsymbol{s}_{N-1} = \boldsymbol{e}_{N-1}$, which is the Gaussian distribution. Otherwise, the output becomes random and $\mathcal{A}$ obtain the correct $\boldsymbol{s}_{N-1}$ value by flipping the sign.

Thus, $\mathcal{A}$ is able to determine the exact value of $\boldsymbol{s}_{N-1}$ without any ambiguity at the end of the execution when P_{N-1} reuses the same key for several executions. The success of the attack also shows the significance of the role of the key reconciliation function in the group key exchange protocol. The above step is exactly the same as two-neighbour attack except that $\mathcal{A}$ manipulates P_0 and P_1 instead of P_{N-2} and P_{N-3}.

When the error term is added to $\boldsymbol{Y}_{N-1,N-1}$, there are some frequent changes in the signal value at the boundary values. But, similar to one-neighbour attack in Ding's GKE protocol, we check the pattern of the noise $\boldsymbol{e}''_{N-1}$, by repeating all steps in two-neighbour attack, which is from Gaussian distribution and get the secret key P_{N-1}.

9.2 Security-enhanced Dynamic GKE from RLWE

Similar to Krug, a party P_i applies pasteurization techniques to z_{i+1} and z_{i-1} by two parties P_{i+1} and P_{i-1} in **Round 2**. We have two pasteurized values $\bar{z}_{i+1} = z_{i+1} + ah_{i+1} + f_{i+1}$ and $\hat{z}_{i-1} = z_{i-1} + ah_{i-1} + g_{i-1}$ for each party P_i where $h_i = \mathcal{H}(z_i)$ is precomputed and $f_i, g_i \leftarrow \chi_{\sigma_1}$. Then, X_i becomes $(\bar{z}_{i+1} - \hat{z}_{i-1})(s_i + h_i) + e'_i$ for the correctness of the protocol. The detailed description of key-reusable dynamic GKE is given in **Protocol 8**, named as KR-DRAG (Key Reusable Dynamic constant-Round Authenticated Group key exchange).

KR-DRAG shows a counter-example that resists our tweaked two-neighbour attack since the dishonest behaviour of two parties doesn't guarantee to get any information of the secret key of the party P_{N-1} from the output of the key reconciliation function. The authenticated one which is secure against an active adversary can be derived by applying known techniques like Katz-Yung compiler [38].

Protocol 8: $\mathsf{KR\text{-}DRAG}(P\,[0,1,\cdots,N-1]\,,a,\mathcal{H},\sigma_1,\sigma_2)$

```
(Round 1) For each party $P_i$ for $i = 0$ to $N-1$, do the following in parallel.
    1. Computes $z_i = as_i + e_i$ where $s_i, e_i \leftarrow \chi_{\sigma_1}$;
    2. Broadcasts $z_i$;
(Round 2) For $i = 0$ to $N-1$, do the following in parallel.
    1. If $i = 0$, party $P_0$ samples $e'_0 \leftarrow \chi_{\sigma_2}$ and otherwise, party $P_i$ samples $e'_i \leftarrow \chi_{\sigma_1}$;
    2. Each party $P_i$ computes $\bar{z}_{i+1} = z_{i+1} + ah_{i+1} + f_{i+1}$ and $\hat{z}_{i-1} = z_{i-1} + ah_{i-1} + g_{i-1}$ where
       $h_i = \mathcal{H}(z_i)$ and $f_i, g_i \leftarrow \chi_{\sigma_1}$;
    3. Each party $P_i$ broadcasts $X_i = (\bar{z}_{i+1} - \hat{z}_{i-1})\,(s_i + h_i) + e'_i$;
(Round 3) For party $P_{N-1}$, do the following.
    1. Samples $e''_{N-1} \leftarrow \chi_{\sigma_1}$ and computes $Y_{N-1,N-1} = X_{N-1} + \hat{z}_{N-2}(s_{N-1} + h_{N-1}) + e''_{N-1}$;
    2. For $j = 1$ to $N-1$, computes $Y_{N-1,(N-1)+j} = X_{(N-1)+j} + Y_{N-1,(N-1)+(j-1)}$;
    3. Calculates $b_{N-1} = \sum_{j=0}^{N-1} Y_{N-1,(N-1)+j}$;
    4. Runs $\mathsf{recMsg}()$ to output $(\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1})$;
    5. Broadcasts $\mathsf{rec}$ and gets session key as $\mathsf{sk}_{N-1} = \mathcal{H}(k_{N-1})$;
(Key Computation) For party $P_i$ $(i \neq N-1)$.
    1. Computes $Y_{i,i} = X_i + \hat{z}_{i-1}(s_i + h_i)$;
    2. For $j = 1$ to $N-1$, computes $Y_{i,i+j} = X_{i+j} + Y_{i,i+(j-1)}$;
    3. $b_i = \sum_{j=0}^{N-1} Y_{i,i+j}$;
    4. Runs $\mathsf{recKey}()$ to output $k_i = \mathsf{recKey}\,(b_i, \mathsf{rec})$ and gets session key as $\mathsf{sk}_i = \mathcal{H}(k_i)$;
```

Chapter 10. Comparison with Other Protocols

A comparison between KR-DRAG and other previously known lattice-based GKE protocols [4,26,64] is given in TABLE 10.1. For computational complexity, we ignore ring addition/deletion, or scalar multiplication due to its relatively smaller computing power, and only consider the following:

Samp	total number of Gaussian samplings
R.Mult	total number of ring multiplication computed
Sign	total number of signatures generated
Verify	total number of verification

Table 10.1: Comparison with other lattice-based (authenticated) GKE protocols

Method	DXL12.G [26]	YMZ15 [64]	ADGK19 [4]	KR-DRAG
TA	X	✓	X	X
Scalability	X	✓	✓	✓
Communication Round for GKE (AGKE)*	N	2	3(4)	3(3)
Computational Complexity* (Samp, R.Mult, Sign, Verify)	$(N^2, N^2 - N, \cdot, \cdot)$	$(2N, 2N+2, \cdot, \cdot)$	$(3N+1, 2N+1, 2N, 2N)$	$(6N+1, 4N+1, 2N, N+2)$
Dynamic Setting	X	X	X	✓
Resistance to Key Reuse Attacks	X	N/A	X	✓

* N is the number of protocol participants in GKE protocol

From TABLE 10.1, we define a round as the number of interactions where one party sends their message to another.

DXL12.G requires $N-1$ rounds to have N approximately agreed ring elements and one round to obtain the group secret key by key reconciliation. For each party, there are N Gaussian samplings (one secret sampling and $N-1$ error samplings) and $N-1$ ring multiplications. YMZ15 provides the minimum communication rounds but Yang *et al.*'s protocol has TA which causes more security issues such as a single point of failure. Moreover, this protocol does one more computation for the secure sketch, which requires huge computing power. Neither DXL12.G and YMZ15 specify a digital signature scheme in the paper.

For ADGK19 and KR-DRAG, both provide scalability without TA. KR-DRAG remains three rounds to make the group secret key of authenticated GKE while ADGK19 requires four rounds to apply Katz-Yung compiler. The number of Gaussian sampling and ring multiplications are $3N+1$ and $2N+1$, respectively, for ADGK19. However, KR-DRAG requires more Gaussian sampling and ring multiplications for pasteurization technique. KR-DRAG expects a smaller number of signature verification since we only verify the signatures from the neighbourhood.

Chapter 11. Concluding Remarks

In this paper, we suggest a new type of key reuse attacks called {one, two}-neighbour attack against previously known lattice-based GKE protocols with the key reconciliation mechanism. Instead of a trivial countermeasure to refresh the public key in each session, we suggest a counter-example for our attack in both static and dynamic setting using pasteurization technique. Then, we compare our protocol with other lattice-based GKE protocols in the open literature. Assuming the hardness of RLWE assumption and key reconciliation mechanism, we provide a concrete security analysis of Krug, DRAG, and KR-DRAG in the random oracle model.

As future work, we will optimize our group key exchange protocol as key sharing can be done within a second even if the number of group members are more than hundreds. Then, we plan to check the security in the quantum-accessible random oracle model where the adversary has an access to the quantum oracle provided by an external challenger, instead of a direct access to the quantum oracle.

Bibliography

[1] *Crypto++ Library 8.2.* https://www.cryptopp.com/.

[2] S. Akleylek and K. Seyhan. "A Probably Secure Bi-GISIS Based Modified AKE Scheme With Reusable Keys". *IEEE Access*, 8:26210–26222, 2020.

[3] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. "Post-quantum Key Exchange – A New Hope". In *USENIX Security Symposium*, pages 327–343, 2016.

[4] D. Apon, D. Dachman-Soled, H. Gong, and J. Katz. "Constant-Round Group Key Exchange from the Ring-LWE Assumption." In *International Conference on Post-Quantum Cryptography*, pages 189–205. Springer, 2019.

[5] H. Baan, S. Bhattacharya, S. R. Fluhrer, O. Garcia-Morchon, T. Laarhoven, R. Rietman, M.-J. O. Saarinen, L. Tolhuizen, and Z. Zhang. "Round5: Compact and Fast Post-Quantum Public-Key Encryption." In *International Conference on Post-Quantum Cryptography*, pages 83–102. Springer, 2019.

[6] W. Barker, W. Polk, and M. Souppaya. *Getting Ready for Post-Quantum Cryptography: Explore Challenges Associated with Adoption and Use of Post-Quantum Cryptographic Algorithms.* NIST cybersecurity white paper (draft), https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.05262020-draft.pdf.

[7] A. Bauer, H. Gilbert, G. Renault, and M. Rossi. "Assessment of the key-reuse resilience of newhope". In *Cryptographers' Track at the RSA Conference*, pages 272–292. Springer, 2019.

[8] A. Bogdanov, S. Guo, D. Masny, S. Richelson, and A. Rosen. "On the hardness of learning with rounding over small modulus". In *Theory of Cryptography Conference*, pages 209–224. Springer, 2016.

[9] D. Boneh, D. Glass, D. Krashen, K. Lauter, S. Sharif, A. Silverberg, M. Tibouchi, and M. Zhandry. "Multiparty Non-Interactive Key Exchange and More From Isogenies on Elliptic Curves". *arXiv preprint arXiv:1807.03038*, 2018.

[10] J. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko, A. Raghunathan, and D. Stebila. "Frodo: Take off the ring! practical, quantum-secure key exchange from LWE". In *ACM SIGSAC Conference on Computer and Communications Security*, pages 1006–1018. ACM, 2016.

[11] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé. “CRYSTALS–Kyber: a CCA-secure module-lattice-based KEM”. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 353–367. IEEE, 2018.

[12] J. W. Bos, C. Costello, M. Naehrig, and D. Stebila. “Post-quantum key exchange for the TLS protocol from the ring learning with errors problem”. In *IEEE Symposium on Security and Privacy*, pages 553–570. IEEE, 2015.

[13] E. Bresson, O. Chevassut, and D. Pointcheval. “Provably authenticated group Diffie-Hellman key exchange – the dynamic case”. In *International Conference on the Theory and Application of Cryptology and Information Security – ASIACRYPT 2001*, pages 290–309. Springer, 2001.

[14] E. Bresson, O. Chevassut, and D. Pointcheval. “Dynamic group Diffie-Hellman key exchange under standard assumptions”. In *International Conference on the Theory and Applications of Cryptographic Techniques – EUROCRYPT 2002*, pages 321–336. Springer, 2002.

[15] E. Bresson, O. Chevassut, D. Pointcheval, and J.-J. Quisquater. “Provably authenticated group Diffie-Hellman key exchange”. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 255–264. ACM, 2001.

[16] M. Burmester and Y. Desmedt. “A secure and efficient conference key distribution system”. In *Workshop on the Theory and Application of of Cryptographic Techniques – EUROCRYPT 1994*, pages 275–286. Springer, 1994.

[17] M. Burmester and Y. Desmedt. “A secure and scalable group key exchange system”. *Information Processing Letters*, 94(3):137–143, 2005.

[18] J. H. Cheon, D. Kim, J. Lee, and Y. Song. “Lizard: Cut Off the Tail! A Practical Post-quantum Public-Key Encryption from LWE and LWR”. In *International Conference on Security and Cryptography for Networks*, pages 160–177. Springer, 2018.

[19] P. Choi, J.-H. Kim, and D. K. Kim. “Fast and Power-Analysis Resistant Ring Lizard Crypto-Processor Based on the Sparse Ternary Property”. *IEEE Access*, 7:98684–98693, 2019.

[20] R. Choi and K. Kim. “A Novel Non-Interactive Multi-party Key Exchange from Homomorphic Encryption”. In *ProvSec Workshop 2018*, 2018.

[21] W. Diffie and M. Hellman. “New directions in cryptography”. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[22] J. Ding, S. Alsayigh, J. Lancrenon, R. Saraswathy, and M. Snook. “Provably secure password authenticated key exchange based on RLWE for the post-quantum world”. In *Cryptographers' Track at the RSA Conference*, pages 183–204. Springer, 2017.

[23] J. Ding, S. Alsayigh, R. Saraswathy, S. Fluhrer, and X. Lin. “Leakage of signal function with reused keys in RLWE key exchange”. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2017.

[24] J. Ding, P. Branco, and K. Schmitt. “Key Exchange and Authenticated Key Exchange with Reusable Keys Based on RLWE Assumption”. *IACR Cryptology ePrint Archive 2019/665*, 2019.

[25] J. Ding, S. Fluhrer, and R. Saraswathy. “Complete attack on RLWE key exchange with reused keys, without signal leakage”. In *Australasian Conference on Information Security and Privacy*, pages 467–486. Springer, 2018.

[26] J. Ding, X. Xie, and X. Lin. “A Simple Provably Secure Key Exchange Scheme Based on the Learning with Errors Problem”. *IACR Cryptology ePrint Archive 2012/688*, 2012.

[27] L. Ducas and D. Micciancio. “FHEW: bootstrapping homomorphic encryption in less than a second”. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 617–640. Springer, 2015.

[28] R. Dutta and R. Barua. “Constant round dynamic group key agreement”. In *International Conference on Information Security*, pages 74–88. Springer, 2005.

[29] R. Dutta and R. Barua. “Provably secure constant round contributory group key agreement in dynamic setting”. *IEEE Transactions on Information Theory*, 54(5):2007–2025, 2008.

[30] S. R. Fluhrer. “Cryptanalysis of ring-LWE based key exchange with key share reuse.” *IACR Cryptology ePrint Archive 2016/085*, 2016.

[31] X. Gao, J. Ding, L. Li, and J. Liu. “Practical randomized RLWE-based key exchange against signal leakage attack”. *IEEE Transactions on Computers*, 67(11):1584–1593, 2018.

[32] X. Gao, J. Ding, L. Li, and J. Liu. “Practical randomized RLWE-based key exchange against signal leakage attack”. *IEEE Transactions on Computers*, 67(11):1584–1593, 2018.

[33] X. Gao, J. Ding, J. Liu, and L. Li. “Post-Quantum Secure Remote Password Protocol from RLWE Problem”. In *International Conference on Information Security and Cryptology*, pages 99–116. Springer, 2017.

[34] S. Halevi and V. Shoup. “Algorithms in HElib”. In *International Cryptology Conference*, pages 554–571. Springer, 2014.

[35] W. Hoeffding. Probability inequalities for sums of bounded random variables. In *The Collected Works of Wassily Hoeffding*, pages 409–426. Springer, 1994.

[36] M. Just and S. Vaudenay. “Authenticated multi-party key agreement”. In *International Conference on the Theory and Application of Cryptology and Information Security – ASIACRYPT 1996*, pages 36–49. Springer, 1996.

[37] J. Katz and J. S. Shin. “Modeling insider attacks on group key-exchange protocols”. In *Proceedings of the 12th ACM conference on Computer and Communications Security*, pages 180–189. ACM, 2005.

[38] J. Katz and M. Yung. “Scalable protocols for authenticated group key exchange”. In *Annual International Cryptology Conference – CRYPTO 2003*, pages 110–125. Springer, 2003.

[39] H.-J. Kim, S.-M. Lee, and D. H. Lee. “Constant-round authenticated group key exchange for dynamic groups”. In *International Conference on the Theory and Application of Cryptology and Information Security – ASIACRYPT 2004*, pages 245–259. Springer, 2004.

[40] Y. Kim, A. Perrig, and G. Tsudik. “Simple and fault-tolerant key agreement for dynamic collaborative groups”. In *Proceedings of the 7th ACM conference on Computer and Communications Security*, pages 235–244. ACM, 2000.

[41] Y. Kim, A. Perrig, and G. Tsudik. “Tree-based group key agreement”. *ACM Transactions on Information and System Security (TISSEC)*, 7(1):60–96, 2004.

[42] H. Krawczyk. “HMQV: A High-Performance Secure Diffie-Hellman Protocol (Extended Abstract)”. In *Annual International Cryptology Conference–CRYPTO 2005*, pages 546–566. Springer, 2005.

[43] A. Langlois, D. Stehlé, and R. Steinfeld. “GGHLite: More efficient multilinear maps from ideal lattices”. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques – EUROCRYPT 2014*, pages 239–256. Springer, 2014.

[44] E. Lee, Y.-S. Kim, J.-S. No, M. Song, and D.-J. Shin. “Modification of Frodokem Using Gray and Error-Correcting Codes”. *IEEE Access*, 7:179564–179574, 2019.

[45] C.-H. Li and J. Pieprzyk. “Conference key agreement from secret sharing”. In *Australasian Conference on Information Security and Privacy*, pages 64–76. Springer, 1999.

[46] C. Liu, Z. Zheng, and G. Zou. "Key reuse attack on newhope key exchange protocol". In *International Conference on Information Security and Cryptology*, pages 163–176. Springer, 2018.

[47] V. Lyubashevsky, C. Peikert, and O. Regev. "On ideal lattices and learning with errors over rings". In *Annual International Conference on the Theory and Applications of Cryptographic Techniques – EUROCRYPT 2010*, pages 1–23. Springer, 2010.

[48] A. J. Menezes, J. Katz, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography.* CRC press, 1996.

[49] R. F. Olimid. "Provable Secure Constant-Round Group Key Agreement Protocol Based on Secret Sharing". In *International Joint Conference SOCO'13-CISIS'13-ICEUTE'13*, pages 489–498. Springer, 2014.

[50] C. Peikert. "Lattice cryptography for the internet". In *International Workshop on Post-Quantum Cryptography*, pages 197–219. Springer, 2014.

[51] Z. Qikun, G. Yong, Z. Quanxin, W. Ruifang, and T. Yu-An. "A dynamic and cross-domain authentication asymmetric group key agreement in telemedicine application". *IEEE Access*, 6:24064–24074, 2018.

[52] Z. Qikun, L. Yongjiao, G. Yong, Z. Chuanyang, L. Xiangyang, and Z. Jun. "Group key agreement protocol based on privacy protection and attribute authentication". *IEEE Access*, 7:87085–87096, 2019.

[53] Y. Qin, C. Cheng, and J. Ding. "A Complete and Optimized Key Mismatch Attack on NIST Candidate NewHope." *IACR Cryptology ePrint Archive 2019/435*, 2019.

[54] R. L. Rivest, L. Adleman, and M. L. Dertouzos. "On data banks and privacy homomorphisms". In *Foundations of secure computation 4.11*, pages 169–180, 1978.

[55] M.-J. O. Saarinen. "HILA5: On reliability, reconciliation, and error correction for Ring-LWE encryption". In *International Conference on Selected Areas in Cryptography*, pages 192–212. Springer, 2017.

[56] M. Seo, S. Kim, D. H. Lee, and J. H. Park. "EMBLEM:(R) LWE-based key encapsulation with a new multi-bit encoding method". *International Journal of Information Security*, pages 1–17, 2019.

[57] P. W. Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In *Proceedings 35th annual symposium on Foundations of Computer Science*, pages 124–134. IEEE, 1994.

[58] M. Steiner, G. Tsudik, and M. Waidner. "Diffie-Hellman key distribution extended to group communication". In *Proceedings of the 3rd ACM conference on Computer and Communications Security*, pages 31–37. ACM, 1996.

[59] M. Steiner, G. Tsudik, and M. Waidner. "CLIQUES: A new approach to group key agreement". In *Distributed Computing Systems, 1998. Proceedings. 18th International Conference on*, pages 380–387. IEEE, 1998.

[60] M. Steiner, G. Tsudik, and M. Waidner. "Key agreement in dynamic peer groups". *IEEE Transactions on Parallel and Distributed Systems*, 11(8):769–780, 2000.

[61] S. Streit and F. De Santis. "Post-quantum key exchange on ARMv8-A: A new hope for NEON made simple". *IEEE Transactions on Computers*, 67(11):1651–1662, 2017.

[62] T. Van Erven and P. Harremos. "Rényi divergence and Kullback-Leibler divergence". *IEEE Transactions on Information Theory*, 60(7):3797–3820, 2014.

[63] K. Wang and H. Jiang. "Analysis of Two Countermeasures Against the Signal Leakage Attack". In *International Conference on Cryptology in Africa*, pages 370–388. Springer, 2019.

[64] X. Yang, W. Ma, and C. Zhang. "Group authenticated key exchange schemes via learning with errors". *Security and Communication Networks*, 8(17):3142–3156, 2015.

[65] J. Zhang, Z. Zhang, J. Ding, M. Snook, and Ö. Dagdelen. "Authenticated key exchange from ideal lattices". In *Annual International Conference on the Theory and Applications of Cryptographic Techniques – EUROCRYPT 2015*, pages 719–751. Springer, 2015.