박사 학위 논문
Ph.D. Dissertation

양자 내성을 가진 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

최락용 (崔洛龍 Choi, Rakyong)

한국 과학기술원
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 립 용

한국 과학 기술 원

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

최락용

위 논문은 한국과학기술원 박사학위논문으로 학위논문 심사위원회의 심사를 통과하였습니다.

2020년 06월 12일

심사위원장 김强国 (인)
심사위원 한상근 (인)
심사위원 이주영 (인)
심사위원 권대성 (인)
심사위원 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\textsuperscript{1}.

\textsuperscript{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, fabrication, 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.
최수용. 양자 내성을 가지는 레티스 문제를 이용한 키 재사용이 가능한 동적 그룹 키 교환 방식의 설계 및 분석. 전산학부. 2020년. 76+i v 폭. 지도교수: 김광조. (영문 논문)

Rakyong Choi. Key-reusable Dynamic Group Key Exchange from Lattice with Quantum Resistance. School of Computing. 2020. 76+i v pages. Advisor: Kwangjo Kim. (Text in English)

초록

최근 키 재사용 공격에 의해 Ding Key Exchange, NewHope 뿐 아니라 Peikert의 키 조정 매커니즘을 이용한 레티스 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. 본 논문에서는 기존 레티스 기반 그룹 키 교환 방식은 정적 환경에서 제안되었으며 동적 환경을 고려하지 않았다.

본 논문에서는 기존 레티스 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. 이에 본 논문에서는 파티 $P_{N-1}$의 비밀키를 얻기 위해 파티 $P_{N-2}$와 $P_{N-3}$을 조작 가능하다고 가정한다.

또한 본 논문에서는 레티스 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 변형화 시켜서 키 재사용이 가능한 동적 그룹 키 교환 방식의 설계를 Dutta-Barua 프로토콜을 이용하여 동적 환경에서 RLWE 문제를 기반으로 설계 및 편집 오라클 모델 상에서 증명하였다.

핵심 낭담 키 교환 프로토콜, 적자 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 $\Pi_{GKE}$ is key-reusable if the protocol participants of $\Pi_{GKE}$ 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
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 → N=5) ................................. 62
8.3 U.Leave procedure in DRUG (N=5 → 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-Hellman 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 \(\text{\textit{one, two}-neighbour attack}\) 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 \( \Pi_{\text{GKE}} \) is key-reusable if the protocol participants of \( \Pi_{\text{GKE}} \)
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 \( G \) of prime order \( q \) and a generator \( g \in G \), 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: \( b_i = z_{i-1}^{N-1} \cdot X_i^{N-1} \cdot X_{i+1}^{N-2} \cdots X_{i+N-2} \).
   - 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 \( a \in R_q \),
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 $[N] = \{0, 1, 2, \cdots, N - 1\}$. $[x]$ 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 $\chi$. We set $\text{Supp}(\chi) = \{\epsilon : \chi(\epsilon) \neq 0\}$ and let $\bar{E}$ be the complement of an event set $E$. $\chi_\alpha$ denotes the distribution with standard deviation $\alpha$.

A function $\mu$ 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 $\rho_{\sigma}(x) = \exp \left( -\frac{\|x\|^2}{2\sigma^2} \right)$ as the Gaussian function scaled by $\sigma$ where $x \in \mathbb{R}^m$ and let $\rho_{\sigma}(\mathbb{Z}^m) = \sum_{x \in \mathbb{Z}^m} \rho_{\sigma}(x)$. Define the $m$-dimensional discrete Gaussian distribution $D_{\mathbb{Z}^m, \sigma}(x) = \frac{\rho_{\sigma}(x)}{\rho_{\sigma}(\mathbb{Z}^m)}$ for $x \in \mathbb{Z}^m$. 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 $D_{\mathbb{Z}^n, \sigma}(x)$. We write $D_{\mathbb{Z}^m, \sigma}$ to denote sampling $m$ times from $D_{\mathbb{Z}^n, \sigma}$. We can restrict the sample domain to $\mathbb{Z}_q$, in other words, $i$-th coordinate $x_i$ of the input $x$ is in the range $\frac{-\sigma}{2} < x_i \leq \frac{\sigma}{2}$. Figure ?? shows a 2-dimensional discrete Gaussian distribution with standard deviation $\sigma = 2\sqrt{3}$.

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 $(a_i, a_s + e_i)$ 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 \(\text{Adv}^{\text{RLWE}}_{n,q,s,s_e}(\mathcal{B})\) denote the advantage of algorithm \(\mathcal{B}\) in distinguishing RLWE distribution and uniform distribution of \(R_q \times R_q\). In addition, let \(\text{Adv}^{\text{RLWE}}_{n,q,s,s_e}(t)\) be the maximum advantage of any algorithm running in time \(t\). If \(\chi = \chi_s = \chi_e\), we write \(\text{Adv}^{\text{RLWE}}_{n,q,s}(t)\) for simplicity. For the remaining paper, we set \(R, q, \chi\) and \(t\) 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 \(\chi_s\) is a secret-key distribution and \(\chi_e\) is an error distribution that are concentrated on small elements. We choose Gaussian distribution \(D_{\mathbb{Z}_q, \sigma}\) for both \(\chi_s\) and \(\chi_e\).

4. \(t\) is the number of samples given to the adversary.

### 2.4 Rényi Divergence

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

\[
\text{RD}_2(P||Q) = \sum_{x \in \text{Supp}(P)} \frac{P(x)^2}{Q(x)}.
\]

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 \(\text{Supp}(P) \subseteq \text{Supp}(Q)\), let \(E \subseteq \text{Supp}(Q)\) be an arbitrary event. We have

\[
Q(E) \geq P(E)^2 / \text{RD}_2(P||Q)
\]

Roughly, the proposition says that if \(\text{RD}_2(P||Q)\) 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 \( \beta_{\text{Rényi}} \) and \( \sigma \) with \( \beta_{\text{Rényi}} < \sigma < q \). Let \( e \in \mathbb{Z} \) satisfying \(|e| \leq \beta_{\text{Rényi}} \). Then
\[
\text{RD}_2((e + D_{2\lambda, \sigma})^m || D_{2\lambda, \sigma}^m) \leq \exp(2\pi m (\beta_{\text{Rényi}} / \sigma)^2)
\]
where \( \chi^m \) means that we sample \( m \) times independently from the distribution \( \chi \). Moreover, if we take \( \sigma = \Omega(\beta_{\text{Rényi}} \sqrt{m} / \log \lambda) \) with security parameter \( \lambda \), we deduce that \( \text{RD}_2((e + D_{2\lambda, \sigma})^m || D_{2\lambda, \sigma}^m) \leq \text{poly}(\lambda) \).

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

\[
\{(\text{rec}, k) : b \leftarrow R_q; (\text{rec}, k) \leftarrow \text{recMsg}(1^\lambda, b)\}_{\lambda \in \mathbb{N}} \text{ and }
\{(\text{rec}, k') : b \leftarrow R_q; (\text{rec}, k) \leftarrow \text{recMsg}(1^\lambda, b); k' \leftarrow \{0, 1\}^\lambda\}_{\lambda \in \mathbb{N}}
\]

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

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 \( k \), they define hint functions \( \text{sgn}_0(x) \) and \( \text{sgn}_1(x) \) from \( Z_q^\lambda \) to \( \{0, 1\}^\lambda \) as \( \text{sgn}_b(k) = (\text{sgn}_b(k[0]), \text{sgn}_b(k[1]), \ldots, \text{sgn}_b(k[\lambda - 1])) \) for \( b \in \{0, 1\} \) where
### Protocol 1: Ding Key Exchange

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_A, e_A \leftarrow R_q$</td>
<td>$s_B, e_B \leftarrow R_q$</td>
</tr>
<tr>
<td>$p_A = as_A + 2e_A$</td>
<td>$p_A$</td>
</tr>
<tr>
<td>$k_B = p_A s_B + 2e'_B$</td>
<td>$p_B = a \cdot s_B + 2e_B$</td>
</tr>
<tr>
<td>$e'_A \leftarrow R_q$</td>
<td>$p_B, \text{rec}$</td>
</tr>
<tr>
<td>$k_A = s_A p_B + 2e'_A$</td>
<td>$\text{rec} \leftarrow S(k_B)$</td>
</tr>
<tr>
<td>$sk_A \leftarrow E(k_A, \text{rec})$</td>
<td>$sk_B \leftarrow E(k_B, \text{rec})$</td>
</tr>
</tbody>
</table>

\[
\text{sgn}_0(k[i]) = \begin{cases} 
0, & k[i] \in [-\frac{1}{2}, \frac{1}{2}] \\
1, & \text{otherwise}
\end{cases}
\]

\[
\text{sgn}_1(k[i]) = \begin{cases} 
0, & k[i] \in [-\frac{1}{2} + 1, \frac{1}{2} + 1] \\
1, & \text{otherwise}
\end{cases}
\]

A signal function $S(k)$ outputs $\text{rec} = \text{sgn}_b(k)$ where $b$ is randomly chosen. For a ring element $k$ and a signal value $\text{rec}$, a reconciliation function $E(x, \sigma)$ is a deterministic function with error tolerance $\delta$ where $E(k, \text{rec}) = (k + \gamma \text{rec} \mod q)$. 

### 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., $P_j$ computes $z'_i = z_i + aH(z_i) + 2f_j$, where $H$ is a random oracle whose output is sampled from $\chi_\alpha$ and $f_j$ is sampled from $\chi_\alpha$. 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 Barna [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 $A$ 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 $z_i$ 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. [36] 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, 36]**

1. For $k = 0, 1, \cdots, q - 1$, $A$ does followings:
   (i) take $s_i = 0, e_i = 1$ and sets $pk_{A,k} = k$
   (ii) invoke the oracle $\mathcal{S}$ with $pk_{A,k}$ 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 $k = 0, 1, \cdots, q - 1$, $A$ does followings:
   (i) set $pk_{A,k} = (1 + x) \cdot k$,
   (ii) invoke the oracle $\mathcal{S}$ with $pk_{A,k}$ 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 $p_B - a \cdot s_B^i$

If $p_B - a \cdot s_B^i$ follows the discrete Gaussian distribution, then $s_B^i = s_B$, and if $p_B - a \cdot s_B^i$ looks random, then $s_B^i = -s_B$.  

---

12
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 \((P[0,1,\ldots, N-1], q, n, \chi, a)\)

1. For a party \(P_i\),
   (i) \(s_i, e_{i_0} \leftarrow \chi\);
   (ii) Compute \(p_i = as_i + 2e_{i_0} \in \mathbb{R}_q\);
   (iii) Send \(p_i\) to \(P_{i+1}\);
2. For \(P_{i+j}\), \(1 \leq j \leq N-2\),
   (i) Computes \(p_{ij} = p_{i-1+j} + 2e_{ij} \mod q\) where \(e_{ij} \leftarrow \chi\);
   (ii) Sends \(p_{ij}\) to a party \(P_{i+j+1}\);
3. For the party \(P_{N-1}\),
   (i) \(e'_{N-1} \leftarrow \chi\) and computes \(K_{N-1} = p_{0N-2} s_{N-1} + 2e'_{N-1}\);
   (ii) Computes and broadcast a signal \(rec \leftarrow S(K_{N-1})\);
   (iii) \(sk_{N-1} \leftarrow E(K_{N-1}, rec)\);
4. For a party \(P_j\) except \(P_{N-1}\),
   (i) \(e'_j \leftarrow \chi\) and computes \(K_i = p_{(i+1)N-2} s_i + 2e'_j\);
   (ii) \(sk_i \leftarrow E(K_i, rec)\);

4.1.1 One-neighbour Attack

Suppose that party \(P_{N-1}\) reuses its public key \(p_{N-1}\) and \(A\) 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 \(s_{N-1}\).

An oracle \(S\) performs the action of party \(P_{N-1}\) and the adversary \(A\) has access to this oracle to
make multiple queries. The adversary behaves like party $P_{N-2}$ and sends $p_{0,A,k} = ke_A$ instead of $p_{0,N-2}$, where $e_A$ 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 $k = 0, 1, \cdots, q - 1$, $A$ does the following:
   (i) Set $p_{0,A,k} = ke_A$;
   (ii) Invoke the oracle $S$ and obtains the signal value $rec$ from $K_{N-1} = ks_{N-1}$;
   (iii) Analyze the number of changes in each coefficient of $rec$;
   (iv) Guess an absolute value of $i$-th coefficient of $s_{N-1}$;

2. For $k = 0, 1, \cdots, q - 1$, $A$ does the following:
   (i) Set $p_{0,A,k} = (1 + x)ke_A$;
   (ii) Invoke the oracle $S$ and obtains the signal value $rec$ from $K_{N-1} = k(1 + x)s_{N-1}$ from $S$;
   (iii) Analyze relations between two adjacent coefficients $s_{N-1}[i]$ and $s_{N-1}[i + 1]$;
   (iv) Let $s'_{N-1}$ be the guess of $A$ about $s_{N-1}$. Then, from the relationship between two adjacent coefficients and their absolute values, $A$ gets either $s'_{N-1} = s_{N-1}$ or $s'_{N-1} = -s_{N-1}$;
   (v) Analyze a distribution of $p_{N-1} - as'_{N-1}$.

If $p_{N-1} - as'_{N-1}$ follows the discrete Gaussian distribution, then $s'_{N-1} = s_{N-1}$; If $p_{N-1} - as'_{N-1}$ looks random, then $s'_{N-1} = -s_{N-1}$;

For the original case with the noise, the number of queries required to recover $s_{N-1}$ increases compared to the steps above, due to the complexity involved in eliminating the effect of the noise $2e'_{N-1}$.

By repeating the procedures in Algorithm 1, we check the pattern of the noise $2e'_{N-1}$ 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 $a$, two standard deviations of Gaussian distributions $\sigma_1$ and $\sigma_2$ and a hash function $\mathcal{H}(\cdot)$ that samples from $\chi_{\sigma_1}$, 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 described in Protocol 3 [4]. We consider two security parameters $\lambda$ and $\rho$ for computational and statistical security parameters, respectively. The parameters $N, n, \sigma_1, \sigma_2, \lambda$ and $\rho$ of the protocol are required to satisfy some constraints described as follows:
\[
(N^2 + 2N) \cdot \sqrt{n} \cdot \rho^{3/2} \cdot \sigma_1^2 + (\frac{N^2}{2} + 1) \cdot \sigma_1 + (N - 2) \cdot \sigma_2 \leq \beta_{\text{Ree}}
\]
\[
2N \sqrt{n} \lambda^{3/2} \sigma_1^2 + (N - 1) \sigma_1 \leq \beta_{\text{Rényi}}
\]
\[
\sigma_2 = \Omega(\beta_{\text{Rényi}} \sqrt{n}/\log(\lambda))
\]

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:

\text{ADGK19}([P[0,1,\ldots,N-1],a,H,\sigma_1,\sigma_2])

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

1. Computes \(z_i = a \cdot s_i + e_i\) where \(s_i, e_i \leftarrow \chi_{\sigma_1}\).

2. Broadcasts \(z_i\).

(\textbf{Round 2}) For \(i = 0\) to \(N - 1\), do the following in parallel.

1. If \(i = 0\), party \(P_0\) samples \(e_i' \leftarrow \chi_{\sigma_2}\) and otherwise, party \(P_i\) samples \(e_i' \leftarrow \chi_{\sigma_1}\).

2. Each party \(P_i\) broadcasts \(X_i = (z_{i+1} - z_{i-1}) \cdot s_i + e_i'\).

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

1. Samples \(e_{N-1}' \leftarrow \chi_{\sigma_1}\) and calculates \(b_{N-1} = z_{N-2} N s_{N-1} + (N - 1) X_{N-1} + (N - 2) X_0 + \cdots + X_{N-3} + e_{N-1}'\).

2. Runs \(\text{recMsg}(\cdot)\) to output \((\text{rec}, k_{N-1}) = \text{recMsg}(b_{N-1})\).

3. Broadcasts \(\text{rec}\) and gets session key as \(s_{k_{N-1}} = H(k_{N-1})\).

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

1. Computes \(b_i = z_{i-1} N s_i + (N - 1) X_i + (N - 2) X_{i+1} + \cdots + X_{i+N-2}\).

2. Runs \(\text{recKey}(\cdot)\) to output \(k_i = \text{recKey}(b_i, \text{rec})\) and gets session key as \(s_{k_i} = H(k_i)\).

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 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, $\text{recMsg}(b_{N-1})$ outputs $\text{rec} \leftarrow S(b_{N-1})$ and $k_{N-1} \leftarrow E(b_{N-1}, \text{rec})$ and $\text{recKey}(b_i, \text{rec})$ outputs $k_i \leftarrow E(b_i, \text{rec})$.

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

We describe an attack on simplified ADGK19 where the error term $e_{N-1}'$ is not added to the key computation of $b_{N-1}$. We set an oracle $S$ that simulates party $P_{N-1}$’s action from a given input public key. On receiving $z_A$ from $A$, $S$ computes $b_{N-1}$ and outputs $(\text{rec}, k_{N-1})$ from $b_{N-1}$ according to the protocol.

For the term $b_{N-1}$, we only consider the term $z_{N-2}Ns_{N-1} + (N - 1)X_{N-1}$ since all $X_i$’s except $X_{N-1}$ and $X_{N-3}$ are fixed and $X_{N-3}$ is controlled by $A$ since $P_{N-3}$ is corrupted. Then, $z_An_sN_{N-1} + (N - 1)X_{N-1} = z_A\bar{s}_{N-1} + (N - 1)z_0s_{N-1}$ and by the assumption, $(N - 1)z_0s_{N-1}$ is also fixed. Hence, $z_{N-2}Ns_{N-1}$ is the only non-constant term of $b_{N-1}$.

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

**Step 1.** $A$ invokes the oracle $S$ with input $z_A = ke_A$ $(k = 0, 1, \cdots, q - 1)$ where $s_A$ is 0 and $e_A$ is the identity element 1 in $R_q$ so that $b_{N-1}$ becomes $kNs_{N-1} + (N - 1)X_{N-1} + (N - 2)X_0 + \cdots + X_{N-3}$. $A$ can make a correct guess of the value of $s_{N-1}[i]$ based on the number of times the signal of $\text{rec}$ for $\{0, s_{N-1}[i], \cdots, (q - 1)s_{N-1}[i]\}$ changes for each coefficient $s_{N-1}[i]$.

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

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

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

Thus, $A$ is able to determine the exact value of $s_{N-1}$ 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 $b_{N-1}$, 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 $e''_{N-1}$, 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 $\Tilde{T}_G$ of the given graph $G$. 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 $P_u$ as $l_u = d - l_{\text{root},u}$ where $l_{\text{root},u}$ is the length of a path from the node $u$ 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 $ek_{p,c}$ between two parties $P_p$ and $P_c$ as an ephemeral key.

3. We run $\mathcal{TKE}$ between level 0 party 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 $ek_{2,4}$ and $ek_{2,5}$ and compute the XORed values $ek_{2} = ek_{2,4} \oplus ek_{2,5}$.
4. Similarly, from level 1 party to level d party, we run $\mathcal{KE}$.

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

**Step S3. Key Sharing.**

1. The root node sends the encrypted common secret key $k_\varepsilon$ to its child nodes $\bar{v}_\varepsilon$ by computing $k_\varepsilon = sk_{\text{root}} \oplus epk_{p,\varepsilon}$

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

![Diagram](image1.png)

(a) common key construction example

![Diagram](image2.png)

(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.

![Join example](image)

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 \( \tilde{T}' = \{ \tilde{V} \cup \{ \tilde{v}_N \}, \tilde{E}_T \cup \{(\tilde{v}_i, \tilde{v}_N)\} \} \).

2. Between parent node \( \tilde{v}_i \) and child node \( \tilde{v}_N \), we run two-party key exchange protocol TKE to find the ephemeral key \( e_{pk_{i,N}} \) between two parties \( \tilde{v}_i \) and \( \tilde{v}_N \).

3. \( \tilde{v}_i \) sends the encrypted common secret key \( k_c = sk_{root} \oplus e_{pk_{i,N}} \) 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 \( \tilde{T}' = \{ \tilde{V} \setminus \{ \tilde{v}_i \}, \tilde{E}_T \setminus \{ 1 \leq j \leq g(\tilde{v}_i, \tilde{v}_j) \} \} \).
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 $\tilde{v}$ from the original keygen tree $\tilde{T} = (\tilde{V}, \tilde{E})$. 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 $sk'$. 

![Diagram](image)

(a) membership revocation (leaf case)

![Diagram](image)

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

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

$\text{Game}_1$. This game is identical to $\text{Game}_0$ 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 $A$. Random values are indistinguishable from the key generated by two-party KE for any PPT adversaries. Thus, the adversary cannot distinguish $\text{Game}_0$ and $\text{Game}_1$.

$\text{Game}_i$ ($2 \leq i \leq d - 1$). This game is identical to $\text{Game}_{i-1}$ 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 $A$. Random values are indistinguishable from keys generated by two-party KE for any PPT adversaries. Thus, the adversary cannot distinguish $\text{Game}_{i-1}$ and $\text{Game}_i$. 

22
**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$_{d-1}$ and Game$_d$, 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 $\mathcal{HE} = (\mathcal{HE}.\text{Gen}, \mathcal{HE}.\text{Enc}, \mathcal{HE}.\text{Eval}, \mathcal{HE}.\text{Dec})$ with the following functionality:

- **$\mathcal{HE}.\text{Gen}(n, \alpha)$ :**
  
  Given the security parameter $n$ and an auxiliary input $\alpha$, 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.

- **$\mathcal{HE}.\text{Enc}(pk, m)$ :**
  
  Given a public key $pk$ and a message $m$, this algorithm outputs a ciphertext $c$ of the message $m$.

- **$\mathcal{HE}.\text{Eval}(evk, C, c_1, \cdots, c_n)$ :**
  
  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.

- **$\mathcal{HE}.\text{Dec}(sk, c)$ :**
  
  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 algorithms that support HE efficiently such as HElib, FHEW [27, 34].

23
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 \( \tilde{G} = (\tilde{V}, \tilde{E}) \) 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 \( \tilde{G}' = (\tilde{V}', \tilde{E}') \), where \( \tilde{V}' = (\tilde{V} \setminus \{\tilde{u}, \tilde{v}\}) \cup \tilde{w} \), \( \tilde{E}' = \tilde{E} \cup \tilde{e} \), and for every \( \tilde{v} \in \tilde{V}' \), \( \tilde{v}' = f(\tilde{v}) \in \tilde{V}' \) is incident to an edge \( \tilde{e}' \in \tilde{E}' \) if and only if, the corresponding edge, \( \tilde{e} \in \tilde{E} \) is incident to \( \tilde{w} \) in \( \tilde{G} \).

**Definition 3.** (graph minor) Let \( \tilde{G} = (\tilde{V}, \tilde{E}) \) and \( \tilde{H} = (\tilde{V}', \tilde{E}') \) 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 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 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](image)

**Step 1.** Each party pre-shares the master group key \( sk \leftarrow \text{HE.Gen}(n, c) \) 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 $pk_i$, $c_i = HE.Enc(pk_i, k_i)$. (① from Figure 5.4.)

Step 4. The server computes $c = HE.Eval(evk, C, c_1, \ldots, c_g)$ 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 $k = HE.Dec(sk, c)$.

Lemma 1. [20] If underlying homomorphic encryption $HE$ 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 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](image)

**Step 1.** Each party pre-shares the master group key $sk \leftarrow \text{HE.Gen}(n, \alpha)$ with each other and the server runs two-party key exchange protocol $\mathcal{KE}$ 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 $pk_i$, $c_i = \text{HE.Enc}(pk_i, k_i)$. (2) from Figure 5.6.)
Step 4. The server computes $c = \text{HE.Eval}(evk, C, c_1, \ldots, c_g)$ with the Boolean circuit $C$ and broadcasts $c$. (② and ③ from Figure 5.6, respectively.)

Step 5. The server sends the ephemeral key $epk$ to each party encrypted by encryption scheme $SK\mathcal{E}$ 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 $k = \text{HE.Dec}(sk, c) \oplus epk$.

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 $\text{HE.Eval}(evk, C, c_1, \ldots, c_g)$ 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 $TK\mathcal{E}$ 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 $\bar{v}_i$ 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 $TK\mathcal{E}$ 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:

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

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

27
**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:
1. Find the leaf node marked as $v_i$ and its parent node $w$.

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

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

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

**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 \( C_{\text{Merge}} \) 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 \( \tilde{e}_{\text{Merge}} \) 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 \( \tilde{e}_{\text{Merge}} \) and put some Boolean operation to the node after this edge contraction.

4. Set the leaf nodes as the group members \( \tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_{g_1+g_2} \).

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(\( \tilde{v}, C \)) 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 \( \tilde{z}_{i+1} = z_{i+1} + ah_{i+1} + f_{i+1} \) and \( \tilde{z}_{i-1} = z_{i-1} + ah_{i-1} + g_{i-1} \) 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 \( \langle \tilde{z}_{i+1} - \tilde{z}_{i-1} \rangle (s_i + h_i) + c_i' \) 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 (Key reusable group 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 \( c = \sqrt{\frac{2p}{\pi \log(c)}} \) and let \( \text{bound}_p \) be the event that for all \( i \in [N] \) and all coefficients \( j \in [n] \), \( |s_i[j]|, |e_i[j]|, |e'_i[j]|, |e''_{N-1}[j]| \leq c \sigma_2 \) except \( |e'_0[j]| \leq c \sigma_2 \). Then

\[
\Pr[\text{bound}_p] \geq 1 - 2^p.
\]
Protocol 4: Krug($P \{0, 1, \cdots, N-1\}, a, \mathcal{H}, \sigma_1, \sigma_2$)

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 $s_i, e_i \leftarrow \chi_{\sigma_i};$
   2. Broadcasts $z_i$;

2 (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 $e'_i \leftarrow \chi_{\sigma_i};$
   2. Each party $P_i$ computes $\hat{z}_{i+1} = z_{i+1} + ah_{i+1} + f_{i+1}$ and $\hat{z}_{i-1} = z_{i-1} + ah_{i-1} + g_{i-1}$ where $h_i = \mathcal{H}(z_i)$ and $f_i, g_i \leftarrow \chi_{\sigma_i};$
   3. Each party $P_i$ broadcasts $X_i = (\hat{z}_{i+1} - \hat{z}_{i-1})(s_i + h_i) + c'_i$;

3 (Round 3) For a party $P_{N-1}$, do the following.
   1. Samples $e'_{N-1} \leftarrow \chi_{\sigma_1}$ and calculates
      \begin{align*}
      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};
      \end{align*}
   2. Runs recMsg($\cdot$) to output $(\text{rec}, k_{N-1}) = \text{recMsg}(b_{N-1});$
   3. Broadcasts rec and gets session key as $sk_{N-1} = \mathcal{H}(k_{N-1});$

(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 recKey($\cdot$) to output $k_i = \text{recKey}(h_i, \text{rec})$ and gets session key as $sk_i = \mathcal{H}(k_i);$

33
Proof. Since the complementary error function \( \text{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_{x}^{\infty} \exp(-t^2) dt \leq \exp(-x^2) \), we get
\[
\Pr[v \leftarrow D_{Z_{s_1}, \sigma_1}; |v| \geq \sigma \alpha + 1] \leq 2 \sum_{x=\lfloor \sigma \alpha + 1 \rfloor}^{\infty} D_{Z_{s_1}, \sigma_1}(x)
\leq \frac{2}{\sigma} \int_{\sigma \alpha}^{\infty} \exp\left(-\frac{\pi x^2}{\sigma^2}\right) dx
= \frac{2}{\pi} \int_{\sigma \sqrt{\pi} (\sigma \alpha)}^{\infty} \exp(-t^2) dt \leq \exp(-c^2 \pi).
\]
Then we have \(3nN\) samplings from \(D_{Z_{s_1}, \sigma_1}\) and \(n\) samplings from \(D_{Z_{s_2}, \sigma_2}\) in our protocol. Under the assumption that \(3nN + n \leq \exp(c^2 \pi / 2)\), we have
\[
\Pr[\text{bound}_p] = (1 - \Pr[v \leftarrow D_{Z_{s_1}, \sigma_1}; |v| \geq \alpha \sigma_1 + 1])^{3nN}(1 - \Pr[e_0 \leftarrow D_{Z_{s_2}, \sigma_2}; |v| \geq \alpha \sigma_2 + 1])^n
\geq 1 - (3nN + n) \cdot \exp(-c^2 \pi) \geq 1 - \exp(c^2 \pi / 2)
\geq 1 - 2^{-\rho}.
\]

\[\square\]

Lemma 3 (\cite{2}). Given \(\text{bound}_p\) defined in Lemma 2, let \(\text{product}_{s_i, e_j}\) be the event that for all \(\alpha\)-th coefficient, \(|(s_i \cdot e_j)| \leq \sqrt{\rho} \beta^{3/2/2} \sigma_1^2\). Then
\[
\Pr[\text{product}_{s_i, e_j} \mid \text{bound}_p] \geq 1 - n \cdot 2 \cdot 2^{-2 \rho}
\]

Proof. Note that for \(l \in [n], (s_i)_l\) denotes the \(l\)-th coefficient of \(s_i\) and we can express \(s_i = \sum_{k=0}^{n-1} (s_i)_k X^l\).
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 \(l-k \geq 0\) and \(-(e_j)_{l-k}\) otherwise. Thus, under \(\text{bound}_p\), specifically \(|(s_i)_l|, |(e_j)_l| \leq \alpha \sigma_1\) where \(c = \sqrt{\frac{2e}{\ln(\gamma)}}\), by Hoeffding’s inequality \cite{35}, we can get
\[
\Pr[|((s_i e_j)_l| \geq \gamma \mid \text{bound}_p] = \Pr\left[\left|\sum_{k=0}^{n-1} (s_i)_k (e_j)_{l-k}\right| \geq \gamma \mid \text{bound}_p\right]
\leq 2 \cdot \exp\left(-\frac{\gamma^2}{n(2c^2 \sigma_1^2)^2}\right).
\]
(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
\[
\Pr[|((s_i e_j)_l| \geq \gamma \mid \text{bound}_p] \leq 2 \cdot \exp\left(-\frac{\rho^3}{2c^4}\right) \leq 2^{-2 \rho + 1}
\]
Thus, after union all bound, we have
\[
\Pr[\text{product}_{s_i, e_j} \mid \text{bound}_p] = \Pr[\forall l, |(s_i e_j)_l| \leq \sqrt{n} \rho^{3/2} \sigma_1^2]
\geq 1 - n \cdot 2 \cdot 2^{-2 \rho}.
\]

\[\square\]

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 \(\rho\), and assume that
\[
(4N^2 + 4N - 4)\sqrt{n} \rho^{3/2} \sigma_1^2 + \left(\frac{2N - 1}{8}\sigma_1 + (N - 2)\sigma_2\right) \leq \beta_{\text{rec}}.
\]
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 $b_{N-1} - b_i \leq \beta_{\text{Rec}}$, then $k_i = k_{N-1}$. After some careful computation, we have

$$b_{N-1} - b_i = c_i^{N-1} + e_{N-2} + g_{N-2}/(s_{N-1} + h_{N-1}) - (e_{i-1} + f_{i-1})(s_i + h_i)$$

$$= \sum_{j=0}^{N-2} (N-1-j)(c_{N-1-j} - c_{i-j})$$

$$+ \sum_{j=0}^{N-2} (N-1-j)((e_j + f_j)(s_{N-1+j} + h_{N-1+j}) - (e_{i+j} + f_{i+j})(s_{i+j} + h_{i+j}))$$

$$+ \sum_{j=0}^{N-3} (N-2-j)((e_{N-1+j} + g_{N-1+j})(s_j + h_j) - (e_{i+j} + g_{i+j})(s_{i+j+1} + h_{i+j+1})).$$

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 $(2N - 1)^2/8$ terms in form of $c_i$ sampled from $\chi_{\sigma_1}$, at most $N - 2$ terms of $e_i$ sampled from $\chi_{\sigma_2}$, 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 $N^2/2$ terms in form of $f_i \cdot c_j$ and $f_i \cdot h_j$ and at most $N^2 - 1$ terms in form of $g_i \cdot e_j$ and $g_i \cdot h_j$.

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

$$\Pr[\text{product}_{\text{ALL}} \mid \text{bound}_{\rho}] \leq 2n(2(N^2 + 2N) + N^2 + (N^2 - 4)) \cdot 2^{-2p} \leq 2^{-p}$$

Let fail be the event that at least one of parties does not agree on the same key. Given a condition that

$$\{2(N^2 + 2N) + N^2 + (N^2 - 4))\sqrt{np^3/2\cdot \sigma_1} + (4N - 1)^2/8 \cdot \sigma_1 + (N - 2)\sigma_2 \leq \beta_{\text{Rec}},$$

by Lemma 2 and the above inequality we have

$$\Pr[\text{fail}] = \Pr[\text{fail} \mid \text{bound}_{\rho}] \cdot \Pr[\text{bound}_{\rho}] + \Pr[\text{fail} \mid \text{bound}_{\rho}] \cdot \Pr[\text{bound}_{\rho}]$$

$$\leq \Pr[\text{product}_{\text{ALL}} \mid \text{bound}_{\rho}] \cdot 1 + 1 \cdot \Pr[\text{bound}_{\rho}]$$

$$\leq 2 \cdot 2^{-p}.$$ 

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

Note that $e_0$ is not considered in the form of $e_i$ sampled from $\chi_{\sigma_2}$, since $e_0$ is sampled from $\chi_{\sigma_2}$. We have different number of coefficient for $e_i$ since ADGK19 included the term $e_0$ but we calculate the exact number of coefficient for $e_i$ by removing $e_0$.

6.2.2 Security Proof

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

Our goal is to show that the advantage $\text{Adv}_{\text{Krug}}^A$ of $A$ in distinguishing a pair $(T, sk)$ distributed according to $\text{Expt}_i$, with samples $(T, sk_0)$, in which $T$ is distributed the same way but $sk_0$ is a uniformly chosen key, is negligible. Also, we define $\text{Adv}_{\text{Krug}}^A(t, q_E)$ 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, \(8N\sqrt{n}\lambda^{3/2}\sigma_1^2 + (N - 1)\sigma_1 \leq \beta_{\text{B\&E}}\) and \(\sigma_2 = \Omega\left(\beta_{\text{B\&E}}/\sqrt{n}\log \lambda\right)\). Then,

\[
\text{Adv}_{\text{K}\text{E}}(t, q_E) \leq 2^{-\lambda+1} + \sqrt{\left(N \cdot \text{Adv}_{\text{K}\text{E}}(t_1) + \text{Adv}_{\text{K}\text{E}}(t_2) \right) + \frac{q_E}{2^\lambda} \cdot \exp\left(\frac{2\pi n (\beta_{\text{B\&E}} / \sigma_2)^2}{1 - 2^{\lambda-1}}\right)}
\]

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

Proof. Let Query be the event that \(A\) queries \(k_{N-1}\) to the random oracle. Since this is the only way that \(A\) can distinguish \(sk = H(k_{N-1})\) from an independent uniform value, we need to show that \(Pr_1[\text{Query}]\) is small where \(Pr_1[\cdot]\) denotes the probability of an event in \(\text{Expt}_1\).

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

\[
\begin{align*}
\text{Expt}_0 & := \begin{cases} 
  a \leftarrow R_q; s_i, e_i, f_i, g_i \leftarrow \chi_{\sigma_1} \text{ for } i \in [N]; \\
  z_i = a s_i + e_i, h_i = H(z_i) \text{ for } i \in [N]; \\
  \tilde{z}_i = z_i + a h_i + f_i, \tilde{z}_i = z_i + ah_i + g_i \text{ for } i \in [N]; \\
  e'_{0} \leftarrow \chi_{\sigma_1}; e'_{i} \leftarrow \chi_{\sigma_1} \text{ for } 1 \leq i \leq N - 1; \\
  X_i = (\tilde{z}_{i+1} - \tilde{z}_i)(s_i + h_i) + e'_i \text{ for } i \in [N]; \\
  e''_{N-1} \leftarrow \chi_{\sigma_1}; \\
  b_{N-1} = \tilde{z}_{N-2}N(s_{N-1} + h_{N-1}) + (N - 1)X_{N-1} + (N - 2)X_0 + \cdots + X_{N-3} + e''_{N-1}; \\
  (\text{rec}, k_{N-1}) = \text{recMsg}(b_{N-1}); sk = H(k_{N-1}); \\
  T = (z_0, z_1, \ldots, z_{N-1}, X_0, X_1, \ldots, X_{N-1}, \text{rec})
\end{cases} 
\end{align*}
\]

Since \(Pr[\text{A wins}] = \frac{1}{2} + \text{Adv}_{\text{K}\text{E}}(t, q_E) = Pr_0[\text{Query}] + Pr_0[\text{Query}] : \frac{1}{2},\)

\[
\text{Adv}_{\text{K}\text{E}}(t, q_E) \leq Pr_0[\text{Query}].
\]

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.

\[
\begin{align*}
\text{Expt}_1 & := \begin{cases} 
  a \leftarrow R_q; s_i, e_i, f_i, g_i \leftarrow \chi_{\sigma_1} \text{ for } i \in [N]; \\
  z_i = a s_i + e_i, h_i = H(z_i) \text{ for } i \in [N]; \\
  \tilde{z}_i = z_i + ah_i + f_i, \tilde{z}_i = z_i + ah_i + g_i \text{ for } i \in [N]; \\
  e'_{0} \leftarrow \chi_{\sigma_1}; e'_{i} \leftarrow \chi_{\sigma_1} \text{ for } 1 \leq i \leq N - 1; \\
  X'_i = -\sum_{i=1}^{N-1} X_i + e'_0; \\
  e''_{N-1} \leftarrow \chi_{\sigma_1}; \\
  b_{N-1} = \tilde{z}_{N-2}N(s_{N-1} + h_{N-1}) + (N - 1)X_{N-1} + (N - 2)X'_0 + \cdots + X_{N-3} + e''_{N-1}; \\
  (\text{rec}, k_{N-1}) = \text{recMsg}(b_{N-1}); sk = H(k_{N-1}); \\
  T = (z_0, z_1, \ldots, z_{N-1}, X'_0, X_1, \ldots, X_{N-1}, \text{rec})
\end{cases} 
\end{align*}
\]

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

\[
Pr_0[\text{Query}] \leq 2^{-\lambda+1} + \sqrt{Pr_1[\text{Query}] \cdot \frac{\exp\left(\frac{2\pi n (\beta_{\text{B\&E}} / \sigma_2)^2}{1 - 2^{\lambda-1}}\right)}{1 - 2^{-\lambda+1}}}
\]

36
using the property of Rényi divergence.

Proof. Note that we define the random variables $X_0, X'_0$ in both experiments $\text{Expt}_0$ and $\text{Expt}_1$. We define $\text{Error}$ and $\text{Main}$ as

$$\text{Error} = \sum_{i=0}^{N-1} (e_{i+1} - e_i - s_i + h_i) + \sum_{i=1}^{N-1} e'_i$$
$$\text{Main} = (\hat{z}_1 - \hat{z}_{N-1})(s_0 + h_0) - \text{Error}.$$ 

Then,

$$X_0 = \text{Main} + \text{Error} + e'_0$$
$$X'_0 = \text{Main} + e'_0,$$

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 $\text{bound}_{\text{Error}}$ be the event that for all participants $j$, $|\text{Error}[j]| \leq \beta_{\text{Rény}}$. Then,

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

Set $c = \sqrt{\frac{2\lambda}{\log \epsilon}}$ and let bound be the event that $|e'_0[j]| \leq c\sigma_2$, $|e'[j]| \leq c\sigma_1$, and $|e'_0[j]| \leq c\sigma_1$ for all $i > 0$ and $j$.

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

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

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

We use $\text{RD}_2(\text{Error} + \chi_{\sigma_2} | \chi_{\sigma_2}) \leq \exp(2\pi\nu(\beta_{\text{Rény}} / \sigma)^2)$ from the property of Rényi divergence. Thus,

$$\Pr_0[\text{Query}] \leq \Pr_0[\text{Query} | \text{bound}_{\text{Error}}] + \Pr_0[\text{bound}_{\text{Error}}] \leq \Pr_0[\text{Query} | \text{bound}_{\text{Error}}] + 2^{-\lambda+1}$$

$$\leq \sqrt{\Pr_1[\text{Query} | \text{bound}_{\text{Error}}] \cdot \exp \left( \frac{2\pi\nu(\beta_{\text{Rény}} / \sigma_2)^2}{1 - 2^{-\lambda+1}} \right) + 2^{-\lambda+1}}$$

$$\leq \sqrt{\frac{\exp \left( \frac{2\pi\nu(\beta_{\text{Rény}} / \sigma_2)^2}{1 - 2^{-\lambda+1}} \right)}{\Pr_1[\text{bound}_{\text{Error}}]}} + 2^{-\lambda+1}.$$

Lemma 5. The probability that $A$ queries $k_{N-1}$ to the random oracle in $\text{Expt}_1$ is negligible when key reconciliation mechanism is secure and our RLWE problem is hard. i.e.

$$\Pr_1[\text{Query}] \leq \left( \frac{N}{3} \right) + N \cdot \text{Adv}^{\text{RLWE}_E}_{\nu, t_1}(t_1) + \text{Adv}_{\text{RejRec}}(t_2) + \frac{qE}{2\lambda}$$

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

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.

$$
\begin{align*}
& a \leftarrow R_q, z_i \leftarrow R_q \text{ for all } i \in [N]; \\
& f_i, g_i \leftarrow \chi_{e_i}, h_i = \mathcal{H}(z_i) \text{ for } i \in [N]; \\
& \tilde{z}_i = z_i + ah_i + f_i, \tilde{z}_i = z_i + ah_i + g_i \text{ for } i \in [N]; \\
& e_0' \leftarrow \chi_{e_0}; e_i' \leftarrow \chi_{e_i} \text{ for } 1 \leq i \leq N - 1; \\
& X_i' = \sum_{i=1}^{N-1} X_i + e_0' + e_i'; \\
& \text{Expt}_2 := (x_i = (\tilde{z}_{i+1} - \tilde{z}_i)(s_i + h_i) + e_i' \text{ for } 1 \leq i \leq N - 1); (T, sk) \\
& e_{N-1}' \leftarrow \chi_{e_{N-1}}; \\
& b_N = \tilde{z}_{N-2}N(s_{N-1} + h_{N-1}) + (N - 1)X_{N-1} + \\
& (N - 2)X_0 + \cdots + X_{N-3} + e_{N-1}'; \\
& \text{(rec, } k_{N-1}) = \text{recMsg}(b_{N-1}); sk = \mathcal{H}(k_{N-1}); \\
& T = (z_0, z_1, \ldots, z_{N-1}, X_0', X_1', \ldots, X_{N-1}, \text{rec})
\end{align*}
$$

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,

$$
|\Pr_2[\text{Query}] - \Pr_1[\text{Query}]| \leq \frac{N}{3} \cdot \text{Adv}_{\eta, \phi, X, e_i}^{RLWE}(t_1),
$$

where $t_1 = t + O(N \cdot t_{ring})$ 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 $(T, sk)$ in Experiment 3, $3j + 1, 3j + 2$ and $3j + 3$ as below:

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

$$
\begin{align*}
& a \leftarrow R_q, z_i \leftarrow R_q \text{ for all } i \in [N]; \\
& \tilde{z}_0, \tilde{z}_N, \tilde{z}_{N-1} \leftarrow R_q; \\
& h_i = \mathcal{H}(z_i) \text{ for } i \in [N]; \\
& f_i \leftarrow \chi_{e_i}, \tilde{z}_i = z_i + ah_i + f_i \text{ for } i \geq 1; \\
& g_i \leftarrow \chi_{e_i}, \tilde{z}_i = z_i + ah_i + g_i \text{ for } 1 \leq i \leq N - 2; \\
& e_0' \leftarrow \chi_{e_0}; e_i' \leftarrow \chi_{e_i} \text{ for } 1 \leq i \leq N - 1; \\
& X_i' = \sum_{i=1}^{N-1} X_i + e_0'; X_1, \ldots, X_j \leftarrow R_q; \\
& \forall i \geq j, X_i = (\tilde{z}_{i+1} - \tilde{z}_i)(s_i + h_i) + e_i'; \\
& e_{N-1}' \leftarrow \chi_{e_{N-1}}; \\
& b_{N-1} = \tilde{z}_{N-2}N(s_{N-1} + h_{N-1}) + (N - 1)X_{N-1} + \\
& (N - 2)X_0 + \cdots + X_{N-3} + e_{N-1}'; \\
& \text{(rec, } k_{N-1}) = \text{recMsg}(b_{N-1}); sk = \mathcal{H}(k_{N-1}); \\
& T = (z_0, z_1, \ldots, z_{N-1}, X_0, X_1, \ldots, X_{N-1}, \text{rec})
\end{align*}
$$

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

$$
|\Pr_3[\text{Query}] - \Pr_2[\text{Query}]| \leq \text{Adv}_{\eta, \phi, X, e_i}^{RLWE}(t_1),
$$

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

38
Remark. We change $\tilde{z}_{N-1}$ 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 $\tilde{z}_{j-1}$ and $X_j$ into $\tilde{z}_{j+1} - r_j$ and $r_j s_j + e_j'$, respectively, where $r_j \leftarrow R_q$.

The rest is the same as the previous experiment.

$$
\text{Expt}_{3j+1} := \begin{cases}
    a \leftarrow R_q, z_i \leftarrow R_q \text{ for all } i \in [N]; \\
    \tilde{z}_i \leftarrow R_q \text{ for } i \leq j - 1; \tilde{z}_{N-1} \leftarrow R_q; \\
    \tilde{z}_i \leftarrow R_q \text{ for } i \leq j - 2; \tilde{z}_{j-1} = \tilde{z}_{j+1} - r_j; \\
    h_i = \mathcal{H}(z_i) \text{ for } i \in [N]; \\
    f_i \leftarrow \chi_{\sigma_i}, \tilde{z}_i = z_i + ah_i + f_i \text{ for } i \geq j + 1; \\
    g_i \leftarrow \chi_{\sigma_i}, \tilde{z}_i = z_i + ah_i + g_i \text{ for } j + 1 \leq i \leq N - 2; \\
    c_0' \leftarrow \chi_{\sigma_0}; c_j' \leftarrow \chi_{\sigma_j}; \\
    X_0' = -\sum_{i=1}^{N-1} X_i + e_0'; X_1, \ldots, X_{j-1} \leftarrow R_q; \\
    \forall i \geq j, X_i = (\tilde{z}_{i+1} - \tilde{z}_{i-1})(s_i + h_i) + e_i'; \\
    e_{N-1}' \leftarrow \chi_{\sigma_i}; \\
    b_{N-1} = z_{N-2}N(s_{N-1} + h_{N-1}) + (N - 1)X_{N-1} + \\
    (N - 2)X_0' + \cdots + X_{N-3} + e_{N-1}'; \\
    (\text{rec, } k_{N-1}) \leftarrow \text{recMsg}(b_{N-1}); sk = \mathcal{H}(k_{N-1}); \\
    T = (z_0, z_1, \ldots, z_{N-1}, X_0, X_1, \ldots, X_{N-1}, \text{rec})
\end{cases} : (T, sk)
$$

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

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

$$
\text{Expt}_{3j+2} := \begin{cases}
    a \leftarrow R_q, z_i \leftarrow R_q \text{ for all } i \in [N]; \\
    \tilde{z}_i \leftarrow R_q \text{ for } i \leq j; \tilde{z}_{N-1} \leftarrow R_q; \\
    \tilde{z}_i \leftarrow R_q \text{ for } i \leq j - 2; \tilde{z}_{j-1} = \tilde{z}_{j+1} - r_j; \tilde{z}_j \leftarrow R_q; \\
    h_i = \mathcal{H}(z_i) \text{ for } i \in [N]; \\
    f_i \leftarrow \chi_{\sigma_i}, \tilde{z}_i = z_i + ah_i + f_i \text{ for } i \geq j + 1; \\
    g_i \leftarrow \chi_{\sigma_i}, \tilde{z}_i = z_i + ah_i + g_i \text{ for } j + 1 \leq i \leq N - 2; \\
    c_0' \leftarrow \chi_{\sigma_0}; c_j' \leftarrow \chi_{\sigma_j}; \\
    X_0' = -\sum_{i=1}^{N-1} X_i + e_0'; X_1, \ldots, X_j \leftarrow R_q; \\
    \forall i \geq j + 1, X_i = (\tilde{z}_{i+1} - \tilde{z}_{i-1})(s_i + h_i) + e_i'; \\
    e_{N-1}' \leftarrow \chi_{\sigma_i}; \\
    b_{N-1} = z_{N-2}N(s_{N-1} + h_{N-1}) + (N - 1)X_{N-1} + \\
    (N - 2)X_0' + \cdots + X_{N-3} + e_{N-1}'; \\
    (\text{rec, } k_{N-1}) \leftarrow \text{recMsg}(b_{N-1}); sk = \mathcal{H}(k_{N-1}); \\
    T = (z_0, z_1, \ldots, z_{N-1}, X_0, X_1, \ldots, X_{N-1}, \text{rec})
\end{cases} : (T, sk)
$$

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

$$
|Pr_{3j+2} [\text{Query}] - Pr_{3j+1} [\text{Query}]| \leq \text{Adv}_{n, d, X, \epsilon, \delta}^{RLWE}(t_1)
$$

where $t_1$ is the time to solve RLWE problem which is the sum of $t$ and some minor overhead $O(t_{ring})$ for simulation.
Experiment 3j + 3. We replace $\tilde{z}_{j-1}$ into uniform elements in $R_q$. The rest is the same as the previous experiment.

\[
\begin{cases}
  a \leftarrow R_q; z_i \leftarrow R_q \text{ for all } i \in [N]; \\
  \tilde{z}_i, \tilde{z}_i \leftarrow R_q \text{ for } i \leq j; \tilde{z}_{N-1} \leftarrow R_q; \\
  h_i = \mathcal{H}(z_i) \text{ for } i \in [N]; \\
  f_i \leftarrow \chi_{a_i}, \tilde{z}_i = z_i + ah_i + f_i \text{ for } i \geq j + 1; \\
  g_i \leftarrow \chi_{a_i}, \tilde{z}_i = z_i + ah_i + g_i \text{ for } j + 1 \leq i \leq N - 2; \\
  e_0' \leftarrow \chi_{a_0}; e_j', \ldots, e_{N-1}' \leftarrow \chi_{a_1}; \\
  X_0' = -\sum_{i=1}^{N-1} X_i + e_0' \text{ for } i \geq j + 1, X_j \leftarrow R_q; \\
  \forall i \geq j + 1, X_i = (\tilde{z}_{i+1} - \tilde{z}_{i-1})(s_i + h_i) + e_i'; \\
  e_{N-1}' \leftarrow \chi_{a_1}; \\
  b_{N-1} = \tilde{z}_{N-2} N(s_{N-1} + h_{N-1}) + (N - 1)X_{N-1} + \\
  (N - 2)X_0' + \cdots + X_{N-3} + e_{N-1}' + (\text{rec}, k_{N-1}) = \text{recMsg}(b_{N-1}); sk = \mathcal{H}(k_{N-1}); \\
  T = (z_0, z_1, \ldots, z_{N-1}, X_0, X_1, \ldots, X_{N-1}, \text{rec})
\end{cases}
\]  

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

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

\[
\begin{cases}
  a, r_1, r_2 \leftarrow R_q; z_i \leftarrow R_q \text{ for all } i \in [N]; \\
  z_0 = r_1 + r_2, \tilde{z}_i \leftarrow R_q \text{ for } 1 \leq i \leq N - 2; \\
  \tilde{z}_{N-1} \leftarrow R_q; \\
  \tilde{z}_i \leftarrow R_q \text{ for } i \leq N - 3; \tilde{z}_{N-2} = r_2; \\
  h_i = \mathcal{H}(z_i) \text{ for } i \in [N]; \\
  f_{N-1} \leftarrow \chi_{a_{N-1}}$, $\tilde{z}_{N-1} = z_{N-1} + ah_{N-1} + f_{N-1}; \\
  e_0' \leftarrow \chi_{a_0}; e_j', \ldots, e_{N-1}' \leftarrow \chi_{a_1}; \\
  X_0' = -\sum_{i=1}^{N-1} X_i + e_0' \text{ for } i \geq j + 1, X_j \leftarrow R_q; \\
  X_{N-1} = r_1(s_{N-1} + h_{N-1}) + e_{N-1}' + (N - 2)X_0' + \cdots + X_{N-3} + e_{N-1}' + (\text{rec}, k_{N-1}) = \text{recMsg}(b_{N-1}); sk = \mathcal{H}(k_{N-1}); \\
  T = (z_0, z_1, \ldots, z_{N-1}, X_0, X_1, \ldots, X_{N-1}, \text{rec})
\end{cases}
\]  

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

Experiment 3N − 1. We replace $\tilde{z}_{N-1}, X_{N-1}$ and $b_{N-1}$ into uniform element in $R_q$. The rest is the same as the previous experiment.
\[
\text{Expt}_{3N-1} := \begin{cases} \\
\alpha \leftarrow R_q; z_i \leftarrow R_q \text{ for all } i \in [N] ; \\
\bar{z}_i, \bar{z}_i \leftarrow R_q \text{ for all } i \in [N] ; \\
h_i = \mathcal{H}(z_i) \text{ for } i \in [N] ; \\
\epsilon_0 \leftarrow \chi_{\pi z_i} ; \\
X_0' = - \sum_{i=1}^{N-1} X_i + \epsilon_0' ; \\
X_{N-1} \leftarrow R_q^* ; \\
T = (T, sk) ; \\
\text{(rec, } k_{N-1}) = \text{recMsg}(b_{N-1}) ; \\
\text{sk} = \mathcal{H}(k_{N-1}) ; \\
\end{cases} 
\]

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,
\[
|\Pr_{3N-2}[\text{Query}] - \Pr_{3N-3}[\text{Query}]| \leq \text{Adv}^{RLWE}_{n, \chi_{\pi z_i}, 3}(t_1).
\]

**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.

\[
\text{Expt}_{3N} := \begin{cases} \\
\alpha \leftarrow R_q; z_i \leftarrow R_q \text{ for all } i \in [N] ; \\
\bar{z}_i, \bar{z}_i \leftarrow R_q \text{ for all } i \in [N] ; \\
h_i = \mathcal{H}(z_i) \text{ for } i \in [N] ; \\
\epsilon_0 \leftarrow \chi_{\pi z_i} ; \\
X_0' = - \sum_{i=1}^{N-1} X_i + \epsilon_0' ; \\
X_{N-1} \leftarrow R_q^* ; \\
T = (T, sk) ; \\
\text{(rec, } k_{N-1}) = \text{recMsg}(b_{N-1}) ; \\
\text{sk} = \mathcal{H}(k'_{N-1}) ; \\
\text{rec} = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \cdots, X_{N-1}, \text{rec}) 
\end{cases} 
\]

Between Experiment 3N−1 and Experiment 3N, we replace \( k_{N-1} \) from \( \text{recMsg}(b_{N-1}) \) into random. Hence,
\[
|\Pr_{3N}[\text{Query}] - \Pr_{3N-1}[\text{Query}]| \leq \text{Adv}_{\text{KeyRec}}(t_2)
\]

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

Since adversary attacking Krug makes at most \( q_{\text{E}} \) queries to the random oracle, we have \( \Pr_1[\text{Query}] = \frac{q_{\text{E}}}{2^\lambda} \), which is negligible in \( \lambda \).

From Experiment 2 to Experiment 3N, we can conclude that
\[
\Pr_1[\text{Query}] \leq \left( \frac{N}{3} + N \right) \cdot \text{Adv}^{RLWE}_{n, \chi_{\pi z_i}, 3}(t_1) + \text{Adv}_{\text{KeyRec}}(t_2) + \frac{q_{\text{E}}}{2^\lambda}
\]
as expected. \( \square \)

From Lemmas 4 and 5, we have the result of the theorem. \( \square \)
Conditions for parameter setting

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

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

Rényi Condition: $2N \sqrt{n} \lambda^{3/2} \sigma_2^2 + (N - 1) \sigma_1 \leq \beta_{\text{Rényi}}$

$\sigma_2$ Condition 1: $\sigma_2 = \Omega(\beta_{\text{Rényi}} \cdot \sqrt{n/\log(\lambda)})$

$\sigma_2$ Condition 1: $\beta_{\text{Rényi}} < \sigma_2 < q$

$2n(N^2 + 2N) \leq 2^\rho$

$\rho = \lambda$

$|\text{Error}_j| \leq \beta_{\text{Rényi}}$

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 numbers compared 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 \( \mathcal{P} = \mathcal{P}[0, 1, \cdots, N - 1] \) 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 \( \text{sid}_P \) and \( \text{pid}_P \), 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 \( \text{sid}_{U_j} = S = \{(U_0, i_0), \cdots, (U_{l-1}, i_{l-1})\} \) and \( \text{pid}_{U_j} = U[0, 1, \cdots, l-1] \) 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

\[
S_2 = \{(U_{i_0}, i_{i_0}), \cdots, (U_{i_{l-1}}, i_{l-1})\}
\]

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:

- **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 **Send\((U, i, U[0, 1, \cdots, l-1])\)**.
• **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.

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

• **Leave(S, S_2):** This oracle models the removal of S_2 \( \subseteq S \) from S where all parties are in P. Similar to Join(S, S_1), if 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.

• **Reveal(U, i):** This oracle models the misuse of the group secret key. This query outputs the group secret key \( sk^i_U \) for a session with an instance \( (U, i) \).

• **Corrupt(U):** This oracle models (perfect) forward secrecy. This query outputs the long-term secret key of party U.

• **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 Execute, Join, Leave, Reveal, 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, Execute, Join, Leave, Reveal and 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 Reveal(U, i) or Reveal(U', j) with \( U' \in \text{pid}_U^j \),

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

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

The protocol $\text{XP}$ 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 $\text{Execute}$, $\text{Join}$, $\text{Leave}$, $\text{Send}$ oracles respectively. $\text{Adv}_{\text{XP}}^{\text{KE}}(t, q_{E})$ is the maximum advantage of any passive adversary $\mathcal{A}_p$ attacking protocol $\text{XP}$ and $\text{Adv}_{\text{XP}}^{\text{AKe}}(t, q_{E}, q_{S})$ and $\text{Adv}_{\text{XP}}^{\text{AKe}}(t, q_{E}, q_{J}, q_{L}, q_{S})$ are the maximum advantage of any active adversary $\mathcal{A}_a$ attacking protocol $\text{XP}$.

### 7.2 Unauthenticated Group Key Exchange

In the static setting, given $R_y = \mathbb{Z}_q / \langle x^3 + 1 \rangle$ 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 $\text{recMsg}$ algorithm from $\text{KeyRec}$ 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 $\text{DSig} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ where $\mathcal{K}$ is the key generation algorithm with output $(sk_i, pk_i)$ for each party, $\mathcal{S}$ outputs a signature $\delta_i$ 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 $\text{sid}_{R_i}^d$. As protocol proceeds, we build this set from partial session identity set $\text{psid}_{R_i}^d$. Initially, $\text{psid}_{R_i}^d = \{(P_i, d_i)\}$ and after completing the procedure, it becomes the full session identity set $\text{sid}_{R_i}^d$. We assume that all parties know its partner identity $\text{psid}_{R_i}^d$. 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: 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 \(\text{recMsg}()\) to output \((\text{rec}, k_{N-1}) = \text{recMsg}(b_{N-1})\);
   5. Broadcasts \(\text{rec}\) and gets session key as \(sk_{N-1} = \mathcal{H}(k_{N-1})\);

   **(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 \(\text{recKey}()\) to output \(k_i = \text{recKey}(b_i, \text{rec})\) and gets session key as \(sk_i = \mathcal{H}(k_i)\);

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

For authenticated version A.Join procedure, we consider partial session-identity as STAG protocol but we assume that \(\text{psid}^d_{P_i} = \text{psid}^b_{P_i} \cup \{(P_j, d_j) \mid j = 1 \text{ to } N-2\}\) if \(P_i (i = 0, 1, \text{ or } N \leq i \leq N + M - 1)\) verifies \(\overline{d}_1\) 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 A.Join procedure when \(\overline{s}_1\), \(\overline{z}_3\), and \(\overline{X}_1\) are delivered to group \(P\{2,\cdots,N-2\}\).

By modifying the concept of \(\text{psid}^d_{P_i}\) slightly, we can achieve a common session identity \(\text{sid}^d_{P_i} = \{(P_j, d_j) \mid j \in [N + M]\}\) for parties in \(P\{0,1,\cdots,N + M - 1\}\) while Dutta-Barua protocol only provides a common session identity \(\text{sid}^d_{P_i} = \{(U_j, d_j) \mid j \in [N]\}\) for parties in \(U\{0,1,\cdots,N-1\}\) where \(N = M + 3\).
Protocol 6: STAG($\mathcal{P}[0, 1, \ldots, N-1], a, \mathcal{H}, S, \sigma_1, \sigma_2$)

1 **(Round 1)** For each party $P_i$ for $i = 0$ to $N - 1$, do the following in parallel.
   1. Sets partial session-identity $\text{psid}_{P_i}^{d_i} = \{P_i, d_i\}$;
   2. Computes $z_i = a s_i + e_i$ where $s_i, e_i \leftarrow \chi_{\sigma_i}$;
   3. Sets $m_i = P_i \mid 1 \mid z_i$ and $\delta_i = S(m_i)$;
   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 $\delta'_i = S(m'_i)$ 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 $\text{psid}_{P_{N-1}}^{d_{N-1}} = \text{psid}_{P_{N-1}}^{d_{N-1}} \cup \{(P_j, d_j)\}$;
   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 $Y_{N-1,N-1+j} = X_{N-1+j} + Y_{N-1,N-1+j} + (j-1)$;
   5. Calculates $b_{N-1} = \sum_{j=0}^{N-1} Y_{N-1,N-1+j}$;
   6. Runs $\text{recMsg}(\cdot)$ to output $(\text{rec}, k_{N-1}) = \text{recMsg}(b_{N-1})$;
   7. Broadcasts $\text{rec}$ and gets session key as $sk_{N-1} = \mathcal{H}(k_{N-1})$;

**(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 $\text{psid}_{P_i}^{d_i} = \text{psid}_{P_i}^{d_i} \cup \{(P_j, d_j)\}$;
   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 $\text{recKey}(\cdot)$ to output $k_i = \text{recKey}(b_i, \text{rec})$ and gets session key as $sk_i = \mathcal{H}(k_i)$;
Protocol 7: U.Join\(\{P_0, P_1, \cdots, P_{N-1}\}, P_{N+1}, \cdots, P_{N+M-1}\}\)

(Round 1) Rearrange the order with a new array of \(N = M + 3\) parties

1. \(U_0 = P_0, U_1 = P_1, U_2 = P_{N-1}, \bar{s}_0 = s_0, \bar{s}_1 = s_k, \bar{s}_2 = s_{N-1}\) and for \(1 \leq i \leq N-3, U_{i+3} = P_{N-1+i}\);

2. Let \(U[0,1,\cdots,N-1]\) be a new ring that we run in (Round 2);

(Round 2) Run STUG algorithm.

1. Group \(U[0,1,\cdots,N-1]\) runs STUG;

2. \(U_i\) calculates \(\bar{z}_i\) during the 1st round of STUG and broadcasts it;

3. \(U_0\) and \(U_2\) during 1st round of STUG additionally sends \(\bar{z}_1\) and \(\bar{z}_3\) to all parties in \(P[2,\cdots,N-2]\);

4. \(U_i\) calculates \(\bar{X}_i\) during the 2nd round of STUG sends \(\bar{X}_i\) to all parties in \(P[0,\cdots,N+M-1]\);

5. After the 3rd round of STUG, \(U_{N-1}\) sends \(\bar{r}\bar{r}\bar{c}\) to all parties in \(P[0,\cdots,N+M-1]\);

(Key Computation) For party \(P_i\) (\(2 \leq i \leq N-2\)).

1. Computes \(\bar{Y}_{i,2} = \bar{X}_2 + \bar{z}_2\bar{s}_1 = \bar{X}_2 + \bar{z}_2 \cdot s_k\);

2. For \(j = 1\) to \(N-2\), computes \(\bar{Y}_{i,2+j} = \bar{X}_{2+j} + \bar{Y}_{i,2+(j-1)}\);

3. \(\bar{y}_i' = \sum_{j=0}^{N-1} \bar{Y}_{i,j}\);

4. Runs recKey(\(\cdot,\cdot\)) to output \(\bar{k}_i = \text{recKey}(\bar{h}_i, \bar{r}\bar{r}\bar{c})\) and gets session key as \(\bar{s}\bar{k}_i = \mathcal{H}(\bar{k}_i)\);
7.4.2 Leave Algorithm

Let the set of parties $P_1, P_2, \ldots, P_M$ want to leave the group $P = [0, 1, \ldots, N - 1]$. Then, the new group becomes $P' = \{P_1, P_2, \ldots, P_{i-1}, P_{i+1}, \ldots, P_{i+R}, \ldots, P_{M+R}, \ldots, N - 1\}$. 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 $P = [0, 1, \ldots, N - 1]$. E.g., if $P_{i-1}, P_{i-2}, \ldots, P_{i-(j-1)}$ are consecutive parties who want to leave, then $P_{i-L} = P_{i+R}$.

After making a new group $P'$, we simply relabel orders to make a new array $U = [0, 1, \ldots, N - M - 1]$ of the parties in the protocol and run U.Leave procedure for $U = [0, 1, \ldots, N - M - 1]$ based on the remaining parties and run STUG protocol. For authenticated version 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, U.Join, U.Leave) or (STAG, A.Join, 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 $\rho$, and assume that

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

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 $|b_{N-1} - b_i| \leq \beta_{rec}$, then $k_i = k_{N-1}$. After some tedious computation, we have

$$b_{N-1} - b_i = N\epsilon''_{N-1} + \sum_{j=0}^{N-1} (N - j)(\epsilon'_{N-1+j} - \epsilon'_{i+j})$$

$$+ \sum_{j=0}^{N-2} (N - 1 - j)(\epsilon_{N+j}s_{N-1+j} - \epsilon_{N-1+j}s_{N+j}) - (\epsilon_{i+j+1}s_{i+j} - \epsilon_{i+j}s_{i+j+1}).$$

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 $\chi_{\sigma_1}$, at most $N - 2$ terms of $e'_0$ sampled from $\chi_{\sigma_2}$, and $N$ terms of $e''_{N-1}$. Sum of these at most terms is less than Apon et al.’s terms.
Let product\textsubscript{ALL} be the event that for all terms in form of \(s_i \cdot e_j\), each coefficient of this form is bounded by \(\sqrt{n}p^{3/2}\sigma_2^2\). Under an assumption that \(2n(N - 1)/N^2 \leq 2^\rho\), by Lemma 3 we can get
\[
\Pr[\text{product\textsubscript{ALL} | bound}_p] \leq \frac{(N - 1)N}{2} \cdot n \cdot 2^{-2\rho + 1} \leq 2^{-\rho}.
\]

Denote fail by the event that at least one of parties does not agree on the same key. Given a condition that \((N - 1)N/2 \cdot \sqrt{n}p^{3/2}\sigma_1^2 + (N(N + 1)/2 + N)\sigma_1 + (N - 2)\sigma_2 \leq \beta_{\text{Rec}}\), by Lemma 2 and the above inequality we have
\[
\Pr[\text{fail}] = \Pr[\text{fail | bound}_p] \cdot \Pr[\text{bound}_p] + \Pr[\text{fail | bound}_p] \cdot \Pr[\text{bound}_p] \\
\leq \Pr[\text{product\textsubscript{ALL} | bound}_p] \cdot 1 + 1 \cdot \Pr[\text{bound}_p] \\
\leq 2 \cdot 2^{-\rho}.
\]

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

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 \(\Pr_{\text{STUG}}[\text{AbortKey}]\) of the event AbortKey that error between \(b_i\)'s exceeds \(\beta_{\text{Rec}}\) in our protocol is smaller than the probability \(\Pr_{\text{Apon}}[\text{AbortKey}]\) 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 DRAG = (STUG, U.Join, U.Leave) (or (STAG, A.Join, A.Leave)) is secure in the random oracle model based on hardness of RLWE assumption.

**Theorem 9.** For unaugmented GKE protocol STUG, \(2N\sqrt{n}p^{3/2}\sigma_1^2 + (N - 1)\sigma_1 \leq \beta_{\text{Rényi}}\) and \(\sigma_2 = \Omega\left(\beta_{\text{Rényi}}\sqrt{n/\log \lambda}\right)\). Then, we have the following:
\[
\text{Adv}_{\text{STUG}}^\text{KE}(t, q_E) \leq 2^{-\lambda + 1} + \\
\sqrt{\left(N \cdot \text{Adv}_{\text{RLWE}}^\text{RLWE}(t_1) + \text{Adv}_{\text{KeyRec}}^\text{KeyRec}(t_2) + \frac{q_E}{2^\lambda}\right) \frac{\exp\left(2\pi n(\beta_{\text{Rényi}}/\sigma_2)^2\right)}{1 - 2^{-\lambda + 1}}}.
\]

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

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

Let Query be the event that \(k_{N-1}\) is among the adversary \(A\)'s random oracle queries and \(Pr_i[\text{Query}]\) 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. \(\square\)
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.

\[
\begin{aligned}
\text{Exp}_0 := \\
&\begin{cases}
\alpha \leftarrow R_q; \sigma_i \leftarrow \chi_{\sigma_i} \text{ for } i \in [N]; \\
z_i = \alpha s_i + e_i \text{ for } i \in [N]; \\
e'_0 \leftarrow \chi_{\sigma_0} \text{ and } e'_i \leftarrow \chi_{\sigma_i} \text{ for } 1 \leq i \leq N - 1; \\
X_i = (z_{i+1} - z_{i-1})s_i + e'_i \text{ for } i \in [N]; \\
e''_{N-1} \leftarrow \chi_{\sigma_1}; \\
Y_{N-1,N-1} = X_{N-1} + z_{N-2}s_{N-1} + e''_{N-1}; \\
Y_{N-1,(N-1)+j} = X_{(N-1)+j} + Y_{N-1,N-1} + (j-1)i; \\
b_{N-1} = \sum_{j=0}^{N-1} Y_{N-1,(N-1)+j}; \\
(\text{rec, } k_{N-1}) = \text{recMsg}(b_{N-1}); s_k = H(k_{N-1}); \\
T = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \cdots, X_{N-1}, \text{rec})
\end{cases}
\end{aligned}
\]

Since \( \Pr[A \text{ wins}] = \frac{1}{2} + \text{Adv}^{KE}_{STUG}(t, q_E) = \Pr_0[\text{Query}] + \Pr_0[\text{Query}] \cdot \frac{1}{2}, \)

\[
\text{Adv}^{KE}_{STUG}(t, q_E) \leq \Pr_0[\text{Query}].
\]

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.

\[
\begin{aligned}
\text{Exp}_1 := \\
&\begin{cases}
\alpha \leftarrow R_q; \sigma_i \leftarrow \chi_{\sigma_i} \text{ for } i \in [N]; \\
z_i = \alpha s_i + e_i \text{ for } i \in [N]; \\
e'_0 \leftarrow \chi_{\sigma_0} \text{ and } e'_i \leftarrow \chi_{\sigma_i} \text{ for } 1 \leq i \leq N - 1; \\
X_i = -\sum_{i=1}^{N-1} X_i + e'_0; \\
X_i = (z_{i+1} - z_{i-1})s_i + e'_i \text{ for } 1 \leq i \leq N - 1; \\
e''_{N-1} \leftarrow \chi_{\sigma_1}; \\
Y_{N-1,N-1} = X_{N-1} + z_{N-2}s_{N-1} + e''_{N-1}; \\
Y_{N-1,(N-1)+j} = X_{(N-1)+j} + Y_{N-1,N-1} + (j-1)i; \\
b_{N-1} = \sum_{j=0}^{N-1} Y_{N-1,(N-1)+j}; \\
(\text{rec, } k_{N-1}) = \text{recMsg}(b_{N-1}); s_k = H(k_{N-1}); \\
T = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \cdots, X_{N-1}, \text{rec})
\end{cases}
\end{aligned}
\]

Lemma 6. Given two distributions of \( X_0 \) and \( X'_0 \), if we have \( 2N\sqrt{n} \lambda^3/\sigma_0^2 + (N-1)\sigma_1 \leq \beta_{\text{Rényi}} \), then

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

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. We define \( \text{Error} \) and \( \text{main} \) as \( \text{Error} = \sum_{i=0}^{N-1} (s_i e_{i+1} - s_i e_{i-1}) + \sum_{i=1}^{N-1} e'_i \) and \( \text{main} = z_1 s_0 - z_0 s_1 - \text{Error} \), respectively. Then,

\[
X_0 = \text{main} + \text{Error} + e'_0 \quad \text{and} \quad X'_0 = \text{main} + e'_0.
\]
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 $\text{bound}_{\text{Error}}$ be the event that for all participants $j$, $\text{Error}_j \leq \beta_{\text{Rényi}}$. Then,

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

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

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

$$\Pr[V_j : |\text{Error}_j| \leq 2N\sqrt{n\lambda^{3/2} \sigma_1^2} + (N - 1)\sigma_1 |\text{bound} ] \geq 1 - 4N : n : 2^{-2\lambda}$$

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

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

$$\Pr_0[\text{Query}] \leq \Pr_0[\text{Query} |\text{bound}_{\text{Error}}] + \Pr_0[\text{bound}_{\text{Error}}]$$

$$\leq \sqrt{\Pr_1[\text{Query} |\text{bound}_{\text{Error}}]} \cdot \exp \left( 2\pi n(\beta_{\text{Rényi}}/\sigma_2)^2 \right) + 2^{-\lambda+1}$$

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

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

For the rest of the proof, we will show that

$$\Pr_1[\text{Query}] \leq N \cdot \text{Adv}_{\text{RLWE}}^{\text{R}_{\alpha}}(t_1) + \text{Adv}_{\text{RLWE}}^{\text{Rec}}(t_2) + \frac{\sigma_E}{2\lambda}.$$ 

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

$$\text{Exp}_2 := $$

\[
\begin{aligned}
a, z_0 &\leftarrow R_q; \forall i \geq 1, s_i, e_i \leftarrow \chi_{\sigma_1}; \ z_i = a s_i + e_i; \\
\ e'_0 &\leftarrow \chi_{\sigma_2} \text{ and } e'_i \leftarrow \chi_{\sigma_2} \text{ for } 1 \leq i \leq N - 1; \\
X_0' &\leftarrow -\sum_{i=1}^{N-1} X_i + e'_0; \\
X_i &\leftarrow (z_{i+1} - z_i) s_i + e'_i \text{ for } 1 \leq i \leq N - 1; \\
\ e''_{N-1} &\leftarrow \chi_{\sigma_1}; \\
Y_{N-1,N-1} &\leftarrow X_{N-1} + 2z_{N-2} #_{N-1} + e''_{N-1}; \\
Y_{N-1,N-1} &\leftarrow X_{(N-1)+j} + Y_{N-1,N-1} + (j-1); \\
b_{N-1} &\leftarrow \sum_{j=0}^{N-1} Y_{N-1,N-1} + j; \\
(\text{rec}, k_{N-1}) &\leftarrow \text{recMsg}(b_{N-1}); \text{sk} = \mathcal{H}(k_{N-1}); \\
T &\leftarrow (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \cdots, X_{N-1}, \text{rec})
\end{aligned}
\]

Between Experiment 1 and Experiment 2, we replace one RLWE instance into random. Hence, $|\Pr_2[\text{Query}] - \Pr_1[\text{Query}]| \leq \text{Adv}_{\text{RLWE}}^{\text{R}_{\alpha}}(t_1).$
where \( t_1 = t + \mathcal{O}(N \cdot t_{\text{ring}}) \) and \( t_{\text{ring}} \) is the time required to perform operations in \( \mathbb{Z}_q \). Since
\[
\text{Adv}_{n,q,\chi_2,\lambda}(t_1) \leq \text{Adv}_{n,q,\chi_1,\lambda}(t_1) \leq \text{Adv}_{n,q,\chi_2,\beta}(t_1),
\]
we have \(|\Pr_2[\text{Query}] - \Pr_1[\text{Query}]| \leq \text{Adv}_{n,q,\chi_2,\beta}(t_1)\).

**Experiment 3.** We replace \( z_0 \) into \( z_2 - r_2 \) and \( X_1 \) into \( r_1 s_1 + e'_1 \) where \( r_1 \leftarrow \mathbb{Z}_q \). The rest are same as the previous experiment.

\[
\begin{align*}
\text{Exp}_3 & := \left\{ a, r_1 \leftarrow \mathbb{Z}_q; \forall i \geq 1, s_i, e_i \leftarrow \chi_{\sigma_i}; \; z_i = a s_i + e_i; \\
z_0 & = z_2 - r_1; \\
e'_0 \leftarrow \chi_{\sigma_0} \text{ and } e'_1 \leftarrow \chi_{\sigma_1} \text{ for } 1 \leq i \leq N - 1; \\
X'_0 & = -\sum_{i=1}^{N-1} X_i + e'_0; \; X_1 = r_1 s_1 + e'_1 \\
X_i & = (z_{i+1} - z_i - 1) s_i + e'_i \text{ for } 2 \leq i \leq N - 1; \\
e''_{N-1} & \leftarrow \chi_{\sigma_{N-1}}; \\
Y_{N-1,N-1} & = X_{N-1} + z_{N-2} s_{N-1} + e''_{N-1}; \\
Y_{N-1,N-1+j} & = X_{N-1+j} + Y_{N-1,N-1}(j-1); \\
b_{N-1} & = \sum_{j=0}^{N-2} Y_{N-1,N-1+j}; \\
(\text{rec}, k_{N-1}) & = \text{recMsg}(b_{N-1}); \; \text{sk} = \mathcal{H}(k_{N-1}); \\
T & = (z_0, z_1, \ldots, z_{N-1}, X_0, X_1, \ldots, X_{N-1}, \text{rec})
\right\}
\end{align*}
\]

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

**Experiment 4.** We replace \( z_1, X_1 \) into uniform element in \( \mathbb{Z}_q \). The rest are same as the previous experiment.

\[
\begin{align*}
\text{Exp}_4 & := \left\{ a, r_1 \leftarrow \mathbb{Z}_q; \forall i \geq 2, s_i, e_i \leftarrow \chi_{\sigma_i}; \; z_i = a s_i + e_i; \\
z_0 & = z_2 - r_1; \; z_1 \leftarrow \mathbb{Z}_q; \\
e'_0 \leftarrow \chi_{\sigma_0} \text{ and } e'_1 \leftarrow \chi_{\sigma_1} \text{ for } 2 \leq i \leq N - 1; \\
X'_0 & = -\sum_{i=1}^{N-1} X_i + e'_0; \; X_1 \leftarrow \mathbb{Z}_q \\
X_i & = (z_{i+1} - z_i - 1) s_i + e'_i \text{ for } 2 \leq i \leq N - 1; \\
e''_{N-1} & \leftarrow \chi_{\sigma_{N-1}}; \\
Y_{N-1,N-1} & = X_{N-1} + z_{N-2} s_{N-1} + e''_{N-1}; \\
Y_{N-1,N-1+j} & = X_{N-1+j} + Y_{N-1,N-1}(j-1); \\
b_{N-1} & = \sum_{j=0}^{N-2} Y_{N-1,N-1+j}; \\
(\text{rec}, k_{N-1}) & = \text{recMsg}(b_{N-1}); \; \text{sk} = \mathcal{H}(k_{N-1}); \\
T & = (z_0, z_1, \ldots, z_{N-1}, X_0, X_1, \ldots, X_{N-1}, \text{rec})
\right\}
\end{align*}
\]

Between Experiment 3 and Experiment 4, we replace two RLWE instances into random. Hence, \(|\Pr_4[\text{Query}] - \Pr_3[\text{Query}]| \leq \text{Adv}_{n,q,\chi_2,\beta}(t_1)\) and thus,
\[
|\Pr_4[\text{Query}] - \Pr_3[\text{Query}]| \leq \text{Adv}_{n,q,\chi_2,\beta}(t_1)
\]
\(t_1\) is the time to solve RLWE problem which is the sum of \( t \) and some minor overhead \( \mathcal{O}(t_{\text{ring}}) \) for simulation.

**Experiment 5.** We replace \( z_0 \) into uniform element in \( \mathbb{Z}_q \). The rest are same as the previous experiment.
\[
\begin{align*}
&\text{Exp}_5 := \begin{cases} 
  a \leftarrow R_q; \forall i \geq 2, s_i, e_i \leftarrow \chi_{s_i}; \ z_i = as_i + e_i; \\
  z_0, z_1 \leftarrow R_q; \\
  e'_0 \leftarrow \chi_{s_0} \quad \text{and} \quad e'_i \leftarrow \chi_{s_i} \quad \text{for} \ 2 \leq i \leq N - 1; \\
  X_i = - \sum_{i=1}^{N-1} X_i + e'_0; \ X_1 \leftarrow R_q \\
  X_i = (z_{i+1} - z_i) s_i + e'_i \quad \text{for} \ 2 \leq i \leq N - 1; \\
  \text{Pr}_5 \{\text{Query} \} = \text{Pr}_4 \{\text{Query} \}
\end{cases} \\
&\quad : (T, sk)
\end{align*}
\]

Since both \(z_0\) and \(z_2 - r_1\) are uniform, \(\text{Pr}_5 \{\text{Query} \} = \text{Pr}_4 \{\text{Query} \} \).

Similarly, we can design distribution of \((T, sk)\) 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.

\[
\begin{align*}
&\text{Exp}_{3i} := \begin{cases} 
  a, r_i \leftarrow R_q; \forall j \geq i, s_j, e_j \leftarrow \chi_{s_j}; \ z_j = as_j + e_j; \\
  z_0, \cdots, z_{i-2} \leftarrow R_q; z_{i-1} = z_{i+1} - r_i; \\
  e'_0, e'_1, \cdots, e'_{N-1} \leftarrow \chi_{s_i}; \\
  X_j = - \sum_{i=1}^{N-1} X_i + e'_0; \ X_1, \cdots, X_{i-1} \leftarrow R_q \\
  X_i = r_is_i + e'_i; \quad \forall j \geq i + 1, X_j = (z_{j+1} - z_{j-1}) s_j + e'_j; \\
  \text{Pr}_5 \{\text{Query} \} = \text{Pr}_4 \{\text{Query} \} \\
\end{cases} \\
&\quad : (T, sk)
\end{align*}
\]

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

\[
\begin{align*}
&\text{Exp}_{3i+1} := \begin{cases} 
  a, r_i \leftarrow R_q; \forall j \geq i + 1, s_j, e_j \leftarrow \chi_{s_j}; \ z_j = as_j + e_j; \\
  z_0, \cdots, z_{i-2} \leftarrow R_q; z_{i-1} = z_{i+1} - r_i, z_i \leftarrow R_q; \\
  e'_0, e'_1, \cdots, e'_{N-1} \leftarrow \chi_{s_i}; \\
  X_0 = - \sum_{i=1}^{N-1} X_i + e'_0; \ X_1, \cdots, X_{i-1} \leftarrow R_q \\
  \forall j \geq i + 1, X_j = (z_{j+1} - z_{j-1}) s_j + e'_j; \\
  \text{Pr}_5 \{\text{Query} \} = \text{Pr}_4 \{\text{Query} \} \\
\end{cases} \\
&\quad : (T, sk)
\end{align*}
\]
Experiment 3i + 2. We replace $z_{i-1}$ into uniform element in $R_q$. The rest are same as the previous experiment.

$$\begin{align*}
&\text{Exp}_{3i+2} := \\
&\begin{cases}
a \leftarrow R_q; \forall j \geq i+1, s_j, e_j \leftarrow \chi; z_j = a s_j + e_j; \\
z_0, \cdots, z_i \leftarrow R_q; \\
c'_0, \cdots, c'_{N-1} \leftarrow \chi; \\
X'_0 = -\sum_{i=1}^{N-1} X_i + c'_0; X'_1, \cdots, X'_i \leftarrow R_q; \\
\forall j \geq i+1, X'_j = (z_{j-1} - z_{j-1})s_j + c'_j; \\
c''_{N-1} \leftarrow \chi; \\
Y_{N-1,N-1} = X_{N-1} + z_{N-2} s_{N-1} + c''_{N-1}; \\
Y_{N-1,(N-1)+j} = X_{(N-1)+j} + Y_{N-1,(N-1)+(j-1)}; \\
b_{N-1} = \sum_{j=0}^{N-1} Y_{N-1,(N-1)+j}; \\
(\text{rec}, k_{N-1}) = \text{recMsg}(b_{N-1}); sk = H(k_{N-1}); \\
T = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \cdots, X_{N-1}, \text{rec})
\end{cases}
\end{align*}$$

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

$$\begin{align*}
\Pr_{3i}[\text{Query}] &= \Pr_{3i-1}[\text{Query}] \\
|\Pr_{3i+1}[\text{Query}] - \Pr_{3i}[\text{Query}]| &\leq \text{Adv}^{RLWE}_n(\alpha, \delta, t_1) \\
Pr_{3i+2}[\text{Query}] &= Pr_{3i+1}[\text{Query}]
\end{align*}$$

Experiment 3N - 3. We set $z_{N-2} = r_2, X_{N-1} = r_1 s_{N-1} + c''_{N-1}, z_0 = r_1 + r_2$ where $r_1, r_2 \leftarrow R_q$.

The rest are same as the previous experiment.

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

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 $\Pr_{3N-3}[\text{Query}] = \Pr_{3N-4}[\text{Query}].$

Experiment 3N - 2. We replace $z_{N-1}, X_{N-1}, z_{N-2} s_{N-1} + c''_{N-1}$ into uniform element in $R_q$. The rest are same as the previous experiment.
\[
\begin{align*}
\text{Exp}_{3N-2} & := \\
\{ a, r_3 \leftarrow R_q; s_{N-1}, e_{N-1} \leftarrow \chi_{\sigma_1}; z_i \leftarrow R_q \text{ for } i \in [N]; \\
e_0' \leftarrow \chi_{\sigma_1}; \\
x_i' = -\sum_{i=1}^{N-1} x_i + e_0'; x_i \leftarrow R_q \text{ for } 1 \leq i \leq N-1; \\
Y_{N-1,N-1} & = X_{N-1} + r_3; \\
Y_{N-1,(N-1)+j} & = X_{(N-1)+j} + Y_{N-1,(N-1)+(j-1)}; \\
b_{N-1} & = \sum_{j=0}^{N-1} Y_{N-1,(N-1)+j}; \\
\text{rec}, k_{N-1} & = \text{recMsg}(b_{N-1}); \quad \text{sk} = \mathcal{H}(k_{N-1}); \\
T & = (z_0, z_1, \ldots, z_{N-1}, X_0, X_1, \ldots, X_{N-1}, \text{rec}) \\
\text{Exp}_{3N} & := \\
\{ a, r_1, r_2 \leftarrow R_q; s_{N-1}, e_{N-1} \leftarrow \chi_{\sigma_1}; z_i \leftarrow R_q \text{ for } i \in [N]; \\
e_0' \leftarrow \chi_{\sigma_1}; \\
x_i' = -\sum_{i=1}^{N-1} x_i + e_0'; x_i \leftarrow R_q \text{ for } 1 \leq i \leq N-1; \\
Y_{N-1,(N-1)+j} & = X_{(N-1)+j} + Y_{N-1,(N-1)+(j-1)}; \\
b_{N-1} & \leftarrow R_q; \\
\text{rec}, k_{N-1} & = \text{recMsg}(b_{N-1}); \quad \text{sk} = \mathcal{H}(k_{N-1}); \\
T & = (z_0, z_1, \ldots, z_{N-1}, X_0, X_1, \ldots, X_{N-1}, \text{rec}) \\
\end{align*}
\]

Between Experiment 3N−3 and Experiment 3N−2, we replace three RLWE instances into random. Hence, 
\[
|\text{Pr}_3N-2[\text{Query}] - \text{Pr}_3N-3[\text{Query}]| \leq \text{Adv}_{RLWE}^{N,q,N_0,\sigma_1}(t_1)
\]

**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.

\[
\begin{align*}
\text{Exp}_{3N-1} & := \\
\{ a, r_1, r_2 \leftarrow R_q; s_{N-1}, e_{N-1} \leftarrow \chi_{\sigma_1}; z_i \leftarrow R_q \text{ for } i \in [N]; \\
e_0' \leftarrow \chi_{\sigma_1}; \\
x_i' = -\sum_{i=1}^{N-1} x_i + e_0'; x_i \leftarrow R_q \text{ for } 1 \leq i \leq N-1; \\
Y_{N-1,(N-1)+j} & \leftarrow R_q \text{ for } j \in [N]; \\
b_{N-1} & \leftarrow R_q; \\
\text{rec}, k_{N-1} & = \text{recMsg}(b_{N-1}); \quad \text{sk} = \mathcal{H}(k_{N-1}); \\
T & = (z_0, z_1, \ldots, z_{N-1}, X_0, X_1, \ldots, X_{N-1}, \text{rec}) \\
\end{align*}
\]

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

**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.

\[
\begin{align*}
\text{Exp}_N & := \\
\{ a, r_1, r_2 \leftarrow R_q; s_{N-1}, e_{N-1} \leftarrow \chi_{\sigma_1}; z_i \leftarrow R_q \text{ for } i \in [N]; \\
e_0' \leftarrow \chi_{\sigma_1}; \\
x_i' = -\sum_{i=1}^{N-1} x_i + e_0'; x_i \leftarrow R_q \text{ for } 1 \leq i \leq N-1; \\
Y_{N-1,(N-1)+j} & \leftarrow R_q \text{ for } j \in [N]; \\
b_{N-1} & \leftarrow R_q; \\
\text{rec}, k_{N-1} & = \text{recMsg}(b_{N-1}); \\
k'_{N-1} & \leftarrow \{0,1\}^\lambda; \quad \text{sk} = \mathcal{H}(k'_{N-1}); \\
T & = (z_0, z_1, \ldots, z_{N-1}, X_0, X_1, \ldots, X_{N-1}, \text{rec}) \\
\end{align*}
\]

Between Experiment 3N−1 and Experiment 3N, we replace \( k_{N-1} \) from \( \text{recMsg}(b_{N-1}) \) into random. Hence, 
\[
|\text{Pr}_{3N}[\text{Query}] - \text{Pr}_{3N-1}[\text{Query}]| \leq \text{Adv}_{\text{KeyRec}}(t_2)
\]

\( t_2 \) is the time to break KeyRec algorithm which is the sum of \( t \) and some minor overhead \( \mathcal{O}(t_{\text{reg}}) \) for simulation.
Since adversary attacking STUG makes at most \( q_e \) queries to the random oracle, we have \( \Pr_1[\text{Query}] = \frac{q_e}{2^\lambda} \), which is negligible in \( \lambda \).

From Experiment 1 to Experiment 3N, we have
\[
\Pr_1[\text{Query}] \leq N \cdot \text{Adv}^{\text{RLWE}}_{n,q_1,q_2,q_3,n_1,n_2,n_3}(t_1) + \text{Adv}_{\text{KeyRec}}(t_2) + \frac{q_e}{2^\lambda},
\]
as expected. With the Lemma 6 and \( \text{Adv}^{\text{KE}}_{\text{STUG}}(t,q_e) \leq \Pr_0[\text{Query}] \), we derive the result of the theorem.

\[\Box\]

**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:
\[
\text{Adv}^{\text{KE}}_{\text{STAG}}(t,q_e, q_s) \leq \text{Adv}^{\text{KE}}_{\text{STUG}}(t', q_e + \frac{q_s}{2}) + |P|\text{Adv}_{\text{DSig}}(t')
\]
where \( t' \leq t + (|P|q_e + q_s)t_{\text{STAG}} \) when \( t_{\text{STAG}} \) 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 \( |P|\text{Adv}_{\text{DSig}}(t') \) where \( |P| \) is the number of participants.

Suppose event Forge occurs. Then, \( \mathcal{A}' \) makes a query of the type \( \text{Send}(x, i, m) \) 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 \( \delta_m \) 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 \( P \in P \) 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 \( P \in P \), \( \mathcal{F} \) outputs this pair as a forgery. The success probability of \( \mathcal{F} \) is equal to \( \frac{\Pr_1[\text{Forge}]}{|P|} \) and hence, \( \Pr_1[\text{Forge}] \leq |P|\text{Adv}_{\text{DSig}}(t') \).

For the latter case, we claim that \( \mathcal{A} \) can simulate oracle queries of \( \mathcal{A}' \) by its own oracle queries. Suppose \( \mathcal{A}' \) makes an Execute query with an instance \( \{(P_0,i_0), \ldots , (P_k,i_k)\} \). Then, \( \mathcal{A} \) defines a set of instances \( S = \{(P_0,i_0), \ldots , (P_k,i_k)\} \) and sends \( S \) to Execute oracle to obtain a transcript \( T \) of STUG. Then, \( (S,T) \) are added to the set \( \text{tlist} \) 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 \( \text{Send}_0 \) to start a new session and the other send queries with a message/signature pair. With an instance \( \{(P_0,i_0), \ldots , (P_k,i_k)\} \), \( \text{Send}_0(P_0,i_0, (P_0,i_0) \setminus P_j) \) for each \( 0 \leq j \leq k \). \( \mathcal{A} \) defines \( S = \{(P_0,i_0), \ldots , (P_k,i_k)\} \) and sends \( S \) to Execute oracle to obtain a transcript \( T \) of STUG. Then, \( (S,T) \) is added to \( \text{tlist} \). 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 \( \text{tlist} \) 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 \( \text{Forge} \) occurs, the success probability becomes \( \frac{1}{2} \) since \( \mathcal{A} \) aborts the instance and outputs a random value.

\[
\text{Adv}^A_{\text{STUG}} = 2|\text{Pr}_{\mathcal{A}}[\text{Succ}] - \frac{1}{2}|
\]

\[
= 2 |\text{Pr}_{\mathcal{A}}[\text{Succ} \land \text{Forge}] + \text{Pr}_{\mathcal{A}}[\text{Succ} \land \text{Forge}] - \frac{1}{2}|
\]

\[
= 2 |(\text{Pr}_{\mathcal{A}}[\text{Succ}] - \text{Pr}_{\mathcal{A}}[\text{Succ} \land \text{Forge}]) + (\text{Pr}_{\mathcal{A}}[\text{Forge}] - \text{Pr}_{\mathcal{A}}[\text{Succ} \land \text{Forge}]) - \frac{1}{2}|
\]

\[
= 2 |\text{Pr}_{\mathcal{A}}[\text{Succ}] - \frac{1}{2}| + \frac{1}{2} \text{Pr}_{\mathcal{A}}[\text{Succ} \land \text{Forge}] - \frac{1}{2}
\]

\[
\geq 2 |\text{Pr}_{\mathcal{A}}[\text{Succ}] - 1| - |\text{Pr}_{\mathcal{A}}[\text{Forge}] - 2 \text{Pr}_{\mathcal{A}}[\text{Succ} \land \text{Forge}]|
\]

\[
\geq \text{Adv}^A_{\text{STAG}} - \text{Pr}_{\mathcal{A}}[\text{Forge}].
\]

Then, \( \mathcal{A} \) makes \text{Execute} queries for each \text{Execute} and \text{Send}_0 \) queries in \( \mathcal{A}' \). Since a session has at least two instances, the number of \text{Execute} queries in \( \mathcal{A} \) is at most \( q_E + \frac{q_S}{2} \). Thus,

\[
\text{Adv}^\text{AKE}_{\text{STAG}}(t, q_E, q_S) \leq \text{Adv}^\text{KE}_{\text{STUG}}(t', q_E + \frac{q_S}{2}) + |\mathcal{P}|\text{Adv}_{\text{DSq}}(t').
\]

\( \square \)

**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:

\[
\text{Adv}^\text{AKE}_{\text{DRAG}}(t, q_E, q_J, q_L, q_S) \leq \text{Adv}^\text{KE}_{\text{DRAG}}(t', q_E + \frac{q_J + q_L + q_S}{2}) + |\mathcal{P}|\text{Adv}_{\text{DSq}}(t'),
\]

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

**Proof.** Similar to Theorem 10, we separate the event \text{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 \text{Execute}, \text{Join}, \text{Leave} and \text{Send} queries from DRAG using \text{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 \( \text{Pr}[\text{Forge}] \leq |\mathcal{P}|\text{Adv}_{\text{DSq}}(t') \).

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

For \text{Send} query in \( \mathcal{A}' \), we have two special types of \text{Send} queries as the join send query \text{Send}_J \) and the leave send query \text{Send}_L, to initiate \text{Join} and \text{Leave} queries in \( \mathcal{A}' \).

We define three lists \( \text{Tist}, \text{Jlist} \) and \( \text{List} \) to store the result from \text{Execute}, \text{Join} and \text{Leave} queries as a set.

If a set \( S_J = \{(P_{k+1}, i_{k+1}), \cdots, (P_{k+j}, i_{k+j})\} \) of unused instances wants to join the group \( S = \{(P_0, i_0), \cdots, (P_k, i_k)\}, \mathcal{A}' \) sends \( \text{Send}_J(P_{j}, i_j, \{P_0, \cdots, (P_k)\}) \) for each \( k + 1 \leq j \leq k + l \) to initiate \text{Join}(S, S_J) \) query. \( \mathcal{A} \) finds whether \( S \) is in \( \text{Tist} \) with \( (S, T) \), \text{Jlist} with \( (S', S'', T) \) where \( S = S' \cup S'' \) or
List with $(S', S'', T)$ where $S = S' \setminus S''$. If nothing is found, $A$ runs Execute oracle to get a transcript $T$ and store $(S, T)$ into $\text{List}$. When transcript $T$ is found, $A$ runs Reveal oracle to obtain $sk$ and simulates a membership addition procedure $\text{A.Join}$ to obtain a transcript $T'$. Then, $(S, S_j, T')$ is added to $\text{List}$.

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

For $\text{Join}(S, S_j)$ and $\text{Leave}(S, S_L)$ queries in $\mathcal{A}'$, $A$ finds $(S, S_j, T)$ in $\text{List}$ and $(S, S_L, T)$ in $\text{List}$ 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 $\text{Forge}$ occurs, the success probability becomes $\frac{1}{2}$ since $A$ aborts the instance and outputs a random value. Then,

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

Then, $A$ makes Execute queries for each $\text{Execute}$, $\text{Send}_j$, $\text{Send}_l$, and $\text{Send}_0$ query in $\mathcal{A}'$. A session has at least two instances, and the number of Execute queries for non-special $\text{Send}$ queries is at most $\frac{q_S - q_J - q_L}{2}$. Hence, the number of Execute queries in $A$ is at most $q_E + \frac{q_S + q_J + q_L}{2}$. Thus,

$$\text{Adv}^{\mathcal{A}}_{\text{DRAG}}(t, q_E, q_J, q_L, q_S) \leq \text{Adv}^{\mathcal{A}}_{\text{STUG}}(t', q_E + \frac{q_J + q_L + q_S}{2} + |P|\text{Adv}_{\text{DSG}}(t')).$$

$\Box$
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 \( \sigma_2 = 14, 198, 340/\sqrt{2\pi}, \beta_{\text{Rec}} = 2^{29} - 1 \), and \( \beta_{\text{Renyi}} = 5, 664, 317 \) where \( \beta_{\text{Rec}} = q/8 \) as shown in TABLE 8.1, for \( N = 6 \). If we increase the number of group members in DRUG like real-world applications, both \( \sigma_2 \) and \( \beta_{\text{Renyi}} \) 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 $\sigma_2$, $\beta_{\text{R\&D}}$, $q$, and $\beta_{\text{rec}}$ into around $2^{10}$ which might become not very practical.

<table>
<thead>
<tr>
<th>$\rho$</th>
<th>$\lambda$</th>
<th>$N$</th>
<th>$n$</th>
<th>$q$</th>
<th>$\sigma_1$</th>
<th>$\sigma_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>128</td>
<td>6</td>
<td>1024</td>
<td>$2^{32} - 1$</td>
<td>$8/\sqrt{2\pi}$</td>
<td>14, 198, 340/$\sqrt{2\pi}$</td>
</tr>
</tbody>
</table>

### 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 $\text{rec}$ 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.
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 α₁ takes a longer time than random sampling from α₂. This is a result of the difference in size of σ₁ = 8/√(2π) and σ₂ = 14,198,340/√(2π). 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

<table>
<thead>
<tr>
<th>Operation</th>
<th>runtime(µsec)</th>
<th>cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random sampling from α₁</td>
<td>195</td>
<td>352,369</td>
</tr>
<tr>
<td>Random sampling from α₂</td>
<td>1,043</td>
<td>1,879,027</td>
</tr>
<tr>
<td>Ring polynomial addition</td>
<td>≈ 0</td>
<td>1,398</td>
</tr>
<tr>
<td>Ring polynomial multiplication</td>
<td>185</td>
<td>334,047</td>
</tr>
</tbody>
</table>

TABLE 8.3 shows the performance of each function. We measure the computational time for zᵢ, Xᵢ, reconcile, and the group secret key. As we can see in TABLE 8.3, computing X₀ takes longer than computing Xᵢ due to the sampling from α₂. Even though only one peer, P₀, samples values from α₂, its runtime is much longer than random samplings from α₁ or ring polynomial multiplications. Hence, random sampling from α₂ is another important factor in measuring the performance of DRUG.

Table 8.3: Average runtime and cycles of each function

<table>
<thead>
<tr>
<th>Function</th>
<th>runtime(µsec)</th>
<th>cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute zᵢ (Phase c1)</td>
<td>493</td>
<td>889,014</td>
</tr>
<tr>
<td>Compute X₀ (Phase c2)</td>
<td>1,353</td>
<td>2,435,457</td>
</tr>
<tr>
<td>Compute Xᵢ (Phase c2)</td>
<td>344</td>
<td>619,686</td>
</tr>
<tr>
<td>Compute reconcile (Phase c3.1)</td>
<td>500</td>
<td>901,395</td>
</tr>
<tr>
<td>Compute the group secret key</td>
<td>209</td>
<td>376,906</td>
</tr>
</tbody>
</table>

(Phase c3.2)

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 simplicity, 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>
<thead>
<tr>
<th>Table 8.4: Performance evaluation of U.Join and U.Leave</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Total runtime (μsec)</td>
</tr>
</tbody>
</table>
Chapter 9. Key-resuable Dynamic GKE from RLWE

9.1 Tweaked Two-neighbour Attack

If we apply Ding’s reconciliation mechanism to DRAG, recMsg(b_{N-1}) outputs rec ← S(b_{N-1}) and k_{N-1} ← E(b_{N-1}, rec) and recKey(b_i, rec) outputs k_i ← E(b_i, rec).

We assume that a public value a ∈ R_q and public keys z_i for each party P_i are fixed except P_{N-2}. A can initiate many sessions with all parties in the group and can access to the oracle S. In performing the attack, an adversary 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. A creates key pairs (s_0, z_0) of P_0 by deviating from the protocol. We denote the public key z_0 deviated by A as z_A and the corresponding secret key s_0 and error e_0 deviated by A as s_A and e_A, respectively.

We describe an attack on simplified DRAG where the error term e_{N-1}' is not added to the key computation of Y_{N-1,N-1}. We set an oracle S that simulates party P_{N-1}'s action from a given input public key. On receiving z_A from A, S computes b_{N-1} and outputs (rec, k_{N-1}) from b_{N-1} according to the protocol.

Then, Y_{N-1,N-1} = X_{N-1} + z_{N-1}s_{N-1} = z_0s_{N-1} and all Y_{N-1,N-1+j} = z_0s_{N-1} + (constant) except for Y_{N-1,1} = X_1 + Y_{N-1,0} since X_1 contains the term z_0. Hence, if 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_0s_{N-1} + (constant) for all j and b_{N-1} = \sum_{j=0}^{N-1} Y_{N-1,N-1+j} = Nz_0s_{N-1} + (constant). From the assumption, we set z_0 = ke_A and describe the attack on DRAG as follows:

Step 1. A invokes the oracle S with input z_A = ke_A (k = 0, 1, ···, q - 1) where s_A is 0 and e_A is the identity element 1 in R_q so that b_{N-1} becomes ks_{N-1} + (constant). Since kN represents all elements in Z_q, A can make a correct guess of the value of s_{N-1}[i] based on the number of times the signal of rec for \{0, s_{N-1}, ···, (q - 1)s_{N-1}\} changes for each coefficient s_{N-1}[i].

Step 2. A invokes S with input (1 + x)z_A = (1 + x)ke_A (k = 0, 1, ···, q - 1). A is able to see the key reconciliation value of (1 + x)ks_{N-1} that is the output by S. Thus, again by checking the number of signal changes, A finds values of the coefficients of (1 + x)s_{N-1}, which are s_{N-1}[0] - s_{N-1}[n - 1], s_{N-1}[1] + s_{N-1}[0], ···, s_{N-1}[n - 1] + s_{N-1}[n - 2] up to sign.

Step 3. From Steps 1 and 2, we can determine if each pair of coefficients s_{N-1}[i], s_{N-1}[i+1] have equal or opposite signs, hence narrowing down to only two possibilities such that the guess s'_{N-1} = s_{N-1} or -s_{N-1}.
Step 4. Since \( a \) and \( z_{N-1} \) are public, \( A \) computes \( z_{N-1} - as'_{N-1} \) and verifies the distribution of the result. If \( A \) correctly guesses the sign of \( s_{N-1}[0] \) and so does all the coefficients of \( s_{N-1} \), the resulting distribution of \( z_{N-1} - as'_{N-1} \) is same as the distribution of \( z_{N-1} - as_{N-1} = e_{N-1} \), which is the Gaussian distribution. Otherwise, the output becomes random and \( A \) obtain the correct \( s_{N-1} \) value by flipping the sign.

Thus, \( A \) is able to determine the exact value of \( s_{N-1} \) 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 \( A \) manipulates \( P_0 \) and \( P_1 \) instead of \( P_{N-2} \) and \( P_{N-3} \).

When the error term is added to \( Y_{N-1,N-1} \), 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 \( e'_{N-1} \), 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 \( \tilde{z}_{i+1} = z_{i+1} + ah_{i+1} + f_{i+1} \) and \( \tilde{z}_{i-1} = z_{i-1} + ah_{i-1} + g_{i-1} \) for each party \( P_i \) where \( h_i = H(z_i) \) is precomputed and \( f_i, g_i \leftarrow \chi_{\sigma_i} \). Then, \( X_i \) becomes \( (\tilde{z}_{i+1} - \tilde{z}_{i-1})(s_i + h_i) + e'_i \) 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: KR-DRAK(P[0,1,⋯,N−1],a,H,σ1,σ2)

1 (Round 1) For each party Pi for i = 0 to N − 1, do the following in parallel.
   1. Computes z_i = as_i + e_i where s_i, e_i ← σ_i;
   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 ← σ_2 and otherwise, party P_i samples e'_i ← σ_i;
   2. Each party P_i computes \( \hat{z}_i = \hat{z}_{i+1} + ah_{i+1} + f_i + 1 \) and \( \hat{z}_i = z_i + 1 + ah_i + 1 + g_i \) where
      \( h_i = H(z_i) \) and \( f_i, g_i ← x_s_i \);
   3. Each party P_i broadcasts \( X_i = (\hat{z}_i + \hat{z}_{i+1}) (s_i + h_i) + e'_i \);

(Round 3) For party P_{N−1}, do the following.
   1. Samples e'_{N−1} ← σ_1, and computes \( Y_{N−1,N−1} = X_{N−1} + \hat{z}_{N−1} (s_{N−1} + h_{N−1}) + e'_{N−1} \);
   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 recMsg() to output \( \text{rec}, k_{N−1} = \text{recMsg}(b_{N−1}) \);
   5. Broadcasts \( \text{rec} \) and gets session key as \( \text{sk}_{N−1} = H(k_{N−1}) \);

(Key Computation) For party P_i (i ≠ N − 1).
   1. Computes \( Y_{i,j} = X_i + \hat{z}_{i,j} (s_i + h_i) \);
   2. For j = 1 to N − 1, computes \( Y_{i,j} = X_{i,j} + Y_{i,i+j} \);
   3. \( b_i = \sum_{j=0}^{N−1} Y_{i,i+j} \);
   4. Runs recKey() to output \( k_i = \text{recKey}(b_i, \text{rec}) \) and gets session key as \( \text{sk}_i = H(k_i) \);
Chapter 10. Comparison with Other Protocols

A comparison between KR-DRAI 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:

<table>
<thead>
<tr>
<th>Samp</th>
<th>total number of Gaussian samplings</th>
</tr>
</thead>
<tbody>
<tr>
<td>R.Mult</td>
<td>total number of ring multiplication computed</td>
</tr>
<tr>
<td>Sign</td>
<td>total number of signatures generated</td>
</tr>
<tr>
<td>Verify</td>
<td>total number of verification</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TA</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Scalability</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Communication Round for GKE (AGKE)*</td>
<td>N</td>
<td>2</td>
<td>3(4)</td>
<td>3(3)</td>
</tr>
<tr>
<td>Computational Complexity*</td>
<td>$(N^2, N^2 - N)$</td>
<td>$(2N, 2N + 2)$</td>
<td>$(3N + 1, 2N + 1)$</td>
<td>$(6N + 1, 4N + 1, 2N, N + 2)$</td>
</tr>
<tr>
<td>Dynamic Setting</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Resistance to Key Reuse Attacks</td>
<td>X</td>
<td>N/A</td>
<td>X</td>
<td>✓</td>
</tr>
</tbody>
</table>

* 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

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


