IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 10, 2020, accepted April 21, 2020, date of publication May 8, 2020, date of current version June 2, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2993296

Design and Implementation of Constant-Round Dynamic Group Key Exchange from RLWE

RAKYONG CHOI[1], DONGYEON HONG[2], SEONGHO HAN[2], SEUNGGEUN BAEK[2], WOOSEOK KANG[1], AND KWANGJO KIM[1,2], (Member, IEEE)

[1]School of Computing, KAIST, Daejeon 34141, South Korea
[2]Graduate School of Information Security, School of Computing, KAIST, Daejeon 34141, South Korea

Corresponding author: Rakyong Choi (thepride@kaist.ac.kr)

This work was supported in part by the Institute for Information and Communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) under Grant 2017-0-00555, and in part by the Towards Provable-secure Multi-party Authenticated Key Exchange Protocol based on Lattices in a Quantum World).

ABSTRACT Group Key Exchange (GKE) is required for secure group communication with high confidentiality. In particular, a trusted authority can handle issues that happen by the malicious actions of group members, but it is expensive to deploy and not suitable in a dynamic setting where the network requires frequent membership status changes. To overcome these issues, we designed yet another quantum-resistant constant-round GKE based on lattice without a trusted authority based on Apon *et al.*'s protocol (PQCrypto 2019) by modifying their key computation phase. Then, we describe the novel dynamic authenticated GKE (called DRAG) with membership addition/deletion procedures in Ring Learning with Errors (RLWE) setting, while the former ones are built from Diffie-Hellman problem. Under the specific adversary who can leak the long-term secret key from the party, we suggest a rigorous proof of DRAG in the random oracle model based on the hardness assumption of RLWE problem and the property of Rényi divergence. As a proof of concept, implementation details are described to meet level 1 NIST security. Our implementation is reasonable for practical use since the total runtime to get a group secret key takes about 3 msec and it can be considered as a reference implementation of other quantum-resistant GKEs.

INDEX TERMS Authenticated group key exchange, key establishment, lattice-based cryptography, post quantum cryptography, ring learning with errors (RLWE).

I. 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 [1]. 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.

The associate editor coordinating the review of this manuscript and approving it for publication was Yanjiao Chen.

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 [1]. 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 [2]–[20].

It is well-known that quantum algorithms like Shor's algorithm [21] 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. Since quantum computers may be realized in a decade or so in a best-case scenario, the National Institute of Standards and Technology (NIST) has been soliciting standard post-quantum cryptographic algorithms such as key exchange, encryption, and signature schemes. (Authenticated) GKE protocol is out of the scope of this competition. Beyond NIST post-quantum algorithm standardization, there are a few publications on post-quantum GKE protocols. Ding *et al.* [22] constructed the first lattice-based GKE protocol and Yang *et al.* [15] and Apon *et al.* [16] 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. Our goal is to design a novel post-quantum constant-round dynamic GKE protocol from Ring Learning with Errors (RLWE) without a TA and provide implementation details with pseudocode as a proof of concept that can be a reference implementation of other quantum-resistant group key exchange protocols since there are no public-domain implementation result to our understanding.

The organization of this paper is as follows: We introduce the basic terminologies including notations and definitions used to design our protocol in Chapter II. Previous works and a security model of GKE are given in Chapter III and IV, respectively. Our key idea to design a novel dynamic GKE protocol is discussed in Chapter V and a full description with a detailed procedure is given in Chapter VI. Then, we give a rigorous security proof of our dynamic (authenticated) GKE protocol in Chapter VII. We compare our dynamic authenticated GKE protocol with the previously known lattice-based GKE protocols and give implementation details in Chapters VIII and IX, respectively. Finally, the conclusion is summarized in Chapter X with future work.

II. PRELIMINARIES

A. NOTATIONS AND DEFINITIONS

Let $\mathbb{Z}$ be the set of integers and $[N] = \{0, 1, 2, \cdots N-1\}$. For a set A, $x_i \leftarrow A$ denotes a uniformly random sampling of $x_i \in A$. Let $\chi(E)$ stand for the probability of a set E of events to occur under distribution χ. We set $\mathrm{Supp}(\chi) = \{\epsilon : \chi(\epsilon) \neq 0\}$ and let $\bar{E}$ be the complement of an event set E. Let $f(a, b)$ be a function f on a and b. A function $g(x)$ is negligible when $g(x) \leq x^{-c}$ for all $c > c_0$ when c_0 is large enough.

Given a polynomial p, $(p)_j$ denotes the j-th coefficient of p. $\log(x)$ and $\exp(x)$ are notations for $\log_2(x)$ and e^x, respectively. $P[0, 1, \cdots, k] = \{P_0, P_1, \cdots, P_k\}$ means an array of parties where P_i denotes the i-th party of a group. λ is a computational security parameter used as a security level of a cryptographic primitive and ρ denotes a statistical security parameter used in Gaussian distributions.

B. RING LEARNING WITH ERRORS

We set public parameters R, q, χ, l for RLWE problem as below:

1) $R = \mathbb{Z}[x]/(f(x))$ is a polynomial ring where $f(x)$ is an irreducible polynomial,
2) q is a positive integer modulus defining a quotient ring $R_q = R/qR = \mathbb{Z}_q[x]/(f(x))$,
3) $\chi = (\chi_s; \chi_e)$ is a pair of noise distributions over R_q where χ_s is a secret-key distribution and χ_e is an error distribution concentrated on small elements,
4) and u is the number of samples that are given to the adversary.

Decisional RLWE problem [23] is defined as to distinguish whether samples $\{a_i, b_i\}$ are either (1) RLWE instances, *i.e.*, $b_i = a_i s + e_i \in R_q$ with uniform $a_i \in R_q$, secret key $s \leftarrow \chi_s$ and error $e \leftarrow \chi_e$ or (2) uniformly sampled from $R_q \times R_q$.

$\mathbf{Adv}^{RLWE}_{n,q,\chi_s,\chi_e,u}(\mathcal{B})$ denotes the advantage of adversary $\mathcal{B}$ for decisional RLWE and $\mathbf{Adv}^{RLWE}_{n,q,\chi_s,\chi_e,u}(t)$ denotes the maximum advantage of any adversary $\mathcal{B}$ in time t. If $\chi = \chi_s = \chi_e$, we write $\mathbf{Adv}^{RLWE}_{n,q,\chi,u}$ for simplicity.

C. RÉNYI DIVERGENCE

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

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

Rényi divergence measures the closeness of two probability distributions and widely used in cryptographic research [23]–[26]. We introduce some important results related to Rényi divergence.

Proposition 1 [26]: For discrete distributions P and Q with $\mathrm{Supp}(P) \subseteq \mathrm{Supp}(Q)$, let $E \subseteq \mathrm{Supp}(Q)$ be an arbitrary event. We have

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

Intuitively, the proposition says that if $\mathrm{RD}_2(P\|Q)$ is bounded by some polynomial, then any event set E that occurs with negligible probability $Q(E)$ under Q also occurs with negligible probability $P(E)$ under P.

Lemma 1 [26]: Let $\nu, q, \lambda \in \mathbb{Z}$ and fix a bound $\beta_{\mathsf{Rényi}}$ and σ with $\beta_{\mathsf{Rényi}} < \sigma < q$. Let $e \in \mathbb{Z}$ satisfying $|e| \leq \beta_{\mathsf{Rényi}}$. Then,

$$\mathrm{RD}_2((e + D_{\mathbb{Z},\sigma})^{\nu} || D^{\nu}_{\mathbb{Z},\sigma}) \leq \exp(2\pi\nu(\beta_{\mathsf{Rényi}}/\sigma)^2)$$

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

D. GENERIC KEY RECONCILIATION MECHANISM

The concept of a key reconciliation mechanism was first introduced by Ding *et al.* [22] to handle error between two approximately agreed ring elements in their lattice-based key exchange protocol. Since then, this mechanism has been widely used in several publications on lattice-based two-party key exchange protocol [27]–[31].

Based on Apon *et al.*'s paper [16], we describe a generic key reconciliation mechanism that is performed between two parties in the key computation phase.

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

A key exchange protocol works correctly when two participants have the same shared secret key (*i.e.* $k = k'$). To hold this equality, b and b' have to be sufficiently close. In particular, if $b - b'$ is bound by some value β_{Rec}, two participants share the same shared secret key from the output of KeyRec except with negligible probability.

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

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

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

III. PREVIOUS WORKS

A. CONSTANT-ROUND GROUP KEY EXCHANGE

Burmester and Desmedt [7] 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 [32] 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) [5] 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.* [33] suggested a two-round authenticated GKE protocol for an ad-hoc network, in which no TA is involved. In their protocol, XOR operation is introduced into the generation of the group secret key to reduce the computational cost of each group member. For a dynamic setting, the computation and communication overheads of each group member rely on the number of joining/leaving members rather than relying on the number of previous group members.

Dutta and Barua [6], [34] proposed a two-round authenticated GKE protocol (hereinafter, DB05), which is constructed by combining a variant of BD94 and a signature scheme modified from [5]. 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.

B. SECURITY MODEL OF GROUP KEY EXCHANGE

Bresson *et al.* [2] 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 [5] revised this model to compile unauthenticated GKE protocol into authenticated GKE protocol. They proved the security of BD94 [7] 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 [35] 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.* [3], [4] suggested two formal security models for authenticated GKE protocols depending on the power of corruption and the presence of MA security.

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

C. LATTICE-BASED KEY EXCHANGE

Ding *et al.* [22] suggested the first lattice-based key exchange protocol in 2012 by modifying Diffie-Hellman key exchange protocol [36] into RLWE setting. Following this research, numerous publications [15], [16], [27]–[31], [37]–[48] 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, Diffie *et al.* [36] suggested the natural extension to GKE protocol (hereinafter, DXL12.G) based on their key exchange protocol using the GKE compiler by Bresson *et al.* [3]. After that, Yang *et al.* [15] 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.* [16] 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 [7].

IV. SECURITY MODEL

We describe the adversary model by Bresson *et al.* [3], which is suitable for our dynamic authenticated GKE protocol since their model covers authenticated GKE with a dynamic setting with a weak corruption model.

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

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

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

$$S = \{(U_0, i_0), \cdots, (U_{l-1}, i_{l-1})\},$$
$$S_1 = \{(U_l, i_l), \cdots, (U_{l+k-1}, i_{l+k-1})\}, \text{ and}$$
$$S_2 = \{(U_{l_0}, i_{l_0}), \cdots, (U_{l_{j-1}}, i_{l_{j-1}})\}$$

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

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

- $\mathsf{Send}(U, i, m)$: This oracle models an active attack where the adversary has full control of communication. The output is the reply by (U, i) upon the receipt of message m. The adversary can initiate the protocol with partners $U[0, 1, \cdots, l-1]$ where $l \leq N$, by invoking $\mathsf{Send}(U, i, U[0, 1, \cdots, l-1])$.
- $\mathsf{Execute}(S)$: This oracle models passive attacks where the attacker eavesdrops on an honest execution of the protocol and outputs the transcript of the execution. A transcript consists of all messages exchanged.
- $\mathsf{Join}(S, S_1)$: This oracle models the addition of S_1 to S, where all parties in S and S_1 are in $\mathcal{P}$. For S, $\mathsf{Execute}$ oracle has already been queried. The output is a transcript generated by the honest execution of the membership addition procedure. If $\mathsf{Execute}(S)$ is not preprocessed, the adversary gets no output.
- $\mathsf{Leave}(S, S_2)$: This oracle models the removal of $S_2 \subseteq S$ from S where all parties are in $\mathcal{P}$. Similar to $\mathsf{Join}(S, S_1)$, if $\mathsf{Execute}(S)$ is not preprocessed, the adversary gets no output. Otherwise, the membership deletion procedure is invoked. The adversary obtains the transcript from the honest execution of the membership deletion procedure.
- $\mathsf{Reveal}(U, i)$: This oracle models the misuse of the group secret key. This query outputs the group secret key sk_U^i for a session with an instance (U, i).
- $\mathsf{Corrupt}(U)$: This oracle models (perfect) forward secrecy. This query outputs the long-term secret key of party U.
- $\mathsf{Test}(U, i)$: We can query this oracle only once during the adversary's execution. A bit $b \in \{0, 1\}$ is chosen uniformly at random. The adversary gets sk if $b = 1$ and a random group secret key sk' if $b = 0$. This oracle checks the adversary's ability to distinguish a real group secret key from random.

An adversary who can access for $\mathsf{Execute}$, Join, Leave, Reveal, $\mathsf{Corrupt}$ and Test oracles is considered "passive" while an "active" adversary has full access to the

above-mentioned oracles including **Send** oracle. (For a static case, **Join** or **Leave** queries do not need to be considered.)

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

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

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

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

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

V. METHODOLOGY

Before giving a detailed description of our dynamic GKE protocol based on RLWE, we give a key idea to modify ADGK19 into a dynamic GKE protocol. For that, we check the relationship between BD94 and DB05. Given two public parameters, a group $\mathbb{G}$ of prime order q and a generator $g \in \mathbb{G}$, BD94 and DB05 in a static setting are described as below:

- **Phase a1**[KeyGen]
  Each party P_i generates a "random" value $r_i \in \mathbb{Z}_q$ as his/her secret key and broadcasts his/her public key $z_i = g^{r_i}$ to all other parties.
- **Phase a2**[BroadIntValue]
  Each party P_i calculates an intermediate value $X_i = (z_{i+1}/z_{i-1})^{r_i}$ and broadcasts X_i to all other parties.
- **Phase a3**[KeyComp]
  - BD94:
    Each party P_i outputs $b_i = {z_{i-1}}^{Nr_i} \cdot {X_i}^{N-1} \cdot {X_{i+1}}^{N-2} \cdots X_{i+N-2}$ as a shared key.
  - DB05:
    Each party P_i calculates $Y_{i,i+1} = X_{i+1} {z_{i+1}}^{r_i}$ and $Y_{i,i+j} = X_{i+j} Y_{i,i+(j-1)}$ for $j = 2$ to $N-1$, then outputs $b_i = \prod_{j=0}^{N-1} Y_{i,i+j}$ as a shared key.

All elements after computation are in $\mathbb{G}$ since a group is closed under addition and scalar multiplication. **Phases a1** and **a2** can be interpreted as key generation phase and intermediate value broadcasting phase, respectively. After computation, each party P_i receives the value z_j or X_j where $i \neq j$.

For key computation phase in **Phase a3**, DB05 requires pre-computation on the value $Y_{i,j}$ for each P_i, which results in a simpler expression for each b_i. The group secret key b_i becomes $g^{\sum_{i=0}^{N-1} r_i r_{i+1}}$ for both BD94 and DB05.

Similarly, by modifying the key computation phase of ADGK19, we show that yet another unauthenticated GKE protocol can be designed and also extended to dynamic GKE protocol. Given two public parameters, a ring R_q and a ring element $a \leftarrow R_q$, ADGK19 and our basic construction for static GKE protocol are described as below:

- **Phase b1**[KeyGen]
  Each party P_i generates a "small" secret value $s_i \in R_q$ as his/her secret key and "small" noise $e_i \in R_q$ and broadcasts his/her public key $z_i = as_i + e_i$ to all other parties.
- **Phase b2**[BroadIntValue]
  Each party P_i calculates an intermediate value $X_i = (z_{i+1} - z_{i-1})\, s_i + e_i'$ where "small" noise $e_i' \in R_q$ is a "small" noise and broadcasts X_i to all other parties.
- **Phase b3**[KeyComp]
  - ADGK19:
    Each party P_i outputs $b_i = N z_{i-1} s_i + (N-1) X_i + (N-2) X_{i+1} + \cdots + X_{i+N-2}$.
  - Our basic construction:
    Each party P_i calculates $Y_{i,i} = X_i + z_{i-1} s_i$ and $Y_{i,i+j} = X_{i+j} + Y_{i,i+(j-1)}$ for $j = 1$ to $N-1$, and then outputs $b_i = \sum_{j=0}^{N-1} Y_{i,i+j}$.

All elements after computation are in R_q since a ring is closed under addition and ring multiplication. For key computation phase in **Phase b3**, our basic construction requires pre-computation on the value $Y_{i,j}$ with a simpler expression for each b_i like DB05.

Like two-party key exchange protocol from lattice, the output b_i will be $a \prod_{i=0}^{N-1} s_i s_{i+1} + \mathsf{err}_i$ for both ADGK19 and our basic construction with an error err_i. However, similar to the two-party case, we can obtain the shared key by a key reconciliation mechanism when err_i is small enough.

VI. OUR DYNAMIC (AUTHENTICATED) GKE

In this chapter, our (authenticated) GKE protocol with static and dynamic membership is described by replacing modular exponentiations and multiplications in DB05 into ring multiplication and ring addition in RLWE setting.

For the static setting, we set $\mathsf{KeyRec} = (\mathsf{recMsg}, \mathsf{recKey})$ as a subroutine. Note that there are two security parameters for security analysis, λ and ρ, where λ is used for security proof and ρ is used for correctness check.

A. UNAUTHENTICATED STATIC GKE

In the static setting, given $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ and $a \leftarrow R_q$, all parties calculate the intermediate values X_i and $Y_{i,j}$ and agree on "close" values $b_0 \approx b_1 \approx \cdots \approx b_{N-1}$ after

key computation phase (**Phase b3**). Then, party P_{N-1} runs **recMsg** from **KeyRec** to allow all parties to get a common value $k = k_0 = k_1 = \cdots = k_{N-1}$.

Since we only show that k is difficult to compute for a passive adversary in the security proof, we need to hash k using random oracle $\mathcal{H}$ to get the group secret key **sk**, which is indistinguishable from random. A detailed description of our unauthenticated GKE, called **STUG** (**ST**atic constant-round **U**nauthenticated **G**KE), is in **Protocol** 1.

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

Phase c1[**KeyGen**]

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

Phase c2[**BroadIntValue**]

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 $X_i = (z_{i+1} - z_{i-1})\, s_i + e'_i$

Phase c3.1[**KeyComp.**P_{N-1}] For party P_{N-1},

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 **recMsg**() to output $(\mathbf{rec}, k_{N-1}) = \mathbf{recMsg}(b_{N-1})$
5) Broadcasts **rec** and gets the group secret key as $\mathbf{sk}_{N-1} = \mathcal{H}(k_{N-1})$

Phase c3.2[**KeyComp.**P_i] 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 **recKey**() to output $k_i = \mathbf{recKey}(b_i, \mathbf{rec})$ and gets the group secret key as $\mathbf{sk}_i = \mathcal{H}(k_i)$

B. AUTHENTICATED STATIC GKE

To design authenticated GKE called **STAG** (**ST**atic constant-round **A**uthenticated **G**KE), we need to add a digital signature scheme $\mathsf{DSig} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ where $\mathcal{K}$ is the key generation with output (sk_i, pk_i) for each party P_i, $\mathcal{S}$ outputs a signature δ_i for a message m_i, and $\mathcal{V}$ outputs whether the input signature is valid.

Following DB05 [6], at the start of the session, P_i does not need to know the entire session identity set $\mathsf{sid}_{P_i}^{d_i}$. As the protocol proceeds, we build this set from partial session identity set $\mathsf{psid}_{P_i}^{d_i}$. Initially, $\mathsf{psid}_{P_i}^{d_i} = \{(P_i, d_i)\}$ and after completing the procedure, $\mathsf{psid}_{P_i}^{d_i}$ becomes equal to the full session identity set $\mathsf{sid}_{P_i}^{d_i}$. We assume that all parties know their partner identity $\mathsf{pid}_{P_i}^{d_i}$.

From **STUG**, we concatenate a broadcasting message $m_i = P_i \mid 1 \mid z_i$ and its signature $\delta_i = \mathcal{S}(m_i)$ in **Phase d1** and $m'_i = P_i \mid 2 \mid X_i \mid d_i$ and $\delta'_i = \mathcal{S}(m'_i)$ in **Phase d2**. Then, each party checks the validity of the signature before proceeding to the next phase. This extension can guarantee that a message is delivered without any modification. A detailed description of **STAG** is given in **Protocol** 2.

Protocol 2 STAG$(P[0, 1, \cdots, N-1], a, \mathcal{H}, \mathcal{S}, \sigma_1, \sigma_2)$

Phase d1[**KeyGen**]

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

1) Sets partial session-identity $\mathbf{psid}_{P_i}^{d_i} = \{P_i, d_i\}$
2) Computes $z_i = as_i + e_i$ where $s_i, e_i \leftarrow \chi_{\sigma_1}$
3) Sets $m_i = P_i \mid 1 \mid z_i$ and $\delta_i = \mathcal{S}(m_i)$
4) Broadcasts $m_i \mid \delta_i$

Phase d2[**BroadIntValue**]

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

1) Verifies δ_{i-1} of m_{i-1} and δ_{i+1} of m_{i+1} and proceeds only if both signatures are valid (otherwise, aborts)
2) If $i = 0$, party P_0 samples $e'_0 \leftarrow \chi_{\sigma_2}$ and otherwise, party P_i samples $e'_i \leftarrow \chi_{\sigma_1}$
3) Computes $X_i = (z_{i+1} - z_{i-1})\, s_i + e'_i$
4) Sets $m'_i = P_i \mid 2 \mid X_i \mid d_i$ and $\delta'_i = \mathcal{S}(m'_i)$ and broadcasts $m'_i \mid \delta'_i$

Phase d3.1[**KeyComp.**P_{N-1}] For party P_{N-1},

1) Verifies all δ'_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 $\mathbf{psid}_{P_{N-1}}^{d_{N-1}} = \mathbf{psid}_{P_{N-1}}^{d_{N-1}} \bigcup \{(P_j, d_j)\}$
3) Samples $e''_{N-1} \leftarrow \chi_{\sigma_1}$ and computes $Y_{N-1,N-1} = X_{N-1} + z_{N-2}s_{N-1} + e''_{N-1}$
4) For $j = 1$ to $N-1$, computes $Y_{N-1,(N-1)+j} = X_{(N-1)+j} + Y_{N-1,(N-1)+(j-1)}$
5) Calculates $b_{N-1} = \sum_{j=0}^{N-1} Y_{N-1,(N-1)+j}$
6) Runs **recMsg**$(\cdot)$ to output $(\mathbf{rec}, k_{N-1}) = \mathbf{recMsg}(b_{N-1})$
7) Broadcasts **rec** and gets the group secret key as $\mathbf{sk}_{N-1} = \mathcal{H}(k_{N-1})$

Phase d3.2[**KeyComp.**P_i] For party P_i $(i \neq N-1)$,

1) Verifies all δ'_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 $\mathbf{psid}_{P_i}^{d_i} = \mathbf{psid}_{P_i}^{d_i} \bigcup \{(P_j, d_j)\}$
3) Computes $Y_{i,i} = X_i + z_{i-1}s_i$
4) For $j = 1$ to $N-1$, computes $Y_{i,i+j} = X_{i+j} + Y_{i,i+(j-1)}$
5) $b_i = \sum_{j=0}^{N-1} Y_{i,i+j}$
6) Runs **recKey**() to output $k_i = \mathbf{recKey}(b_i, \mathbf{rec})$ and gets the group secret key as $\mathbf{sk}_i = \mathcal{H}(k_i)$

C. DYNAMIC GKE

1) MEMBERSHIP ADDITION PROCEDURE

In DB05, membership addition can be described as a ring topology that consists of joining members and three previously agreed group members $P_0, P_1,$ and P_{N-1}, while a new secret key of P_1 is the group secret key from the previously agreed group.

We cannot directly adopt this technique in our membership addition procedure U.Join since the group secret key sk does not belong to R_q. Instead, we define another function $\mathcal{H}_1$, which outputs a value from the distribution χ_{σ_1}, and apply $\mathcal{H}_1(\mathsf{sk}) \in R_q$ as a new secret key of $U_1 = P_1$.

If 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 group secret key sk, we can make a new ring that consists of three parties P_0, P_1, P_{N-1} from previously agreed parties $P[0, 1, \cdots, N-1]$ and all parties from the set $P[N, N+1, \cdots, N+M-1]$. P_1 chooses the previously agreed group secret key sk as ephemeral key $\bar{s}_1$.

After making the new ring topology, we follow the procedure of STUG to make a new shared key. A detailed description of U.Join is given in **Procedure** 1.

For authenticated membership addition procedure A.Join, the extended definition for partial session-identity is given as $\mathsf{psid}_{P_i}^{d_i} = \mathsf{psid}_{P_i}^{d_i} \bigcup \{(P_j, d_j) \mid j = 1 \text{ to } N-2\}$ if $P_i (i = 0, 1, \text{ or } N \leq i \leq N+M-1)$ verifies $\bar{\delta}_1'$ of $\overline{m}_1'$. This extension is clear since the ephemeral keys $\bar{s}_1$ and $\bar{z}_1$ are from the group secret key sk of the previously-agreed group $P[0, 1, \cdots, N-1]$. Then, we achieve a common session identity $\mathsf{sid}_{P_i}^{d_i} = \{(P_j, d_j) \mid j \in [N+M]\}$ for parties in $P[0, 1, \cdots, N+M-1]$ while Dutta-Barua only provides a common session identity $\mathsf{sid}_{U_i}^{d_i} = \{(U_j, d_j) \mid j \in [\overline{N}]\}$ for parties in $U[0, 1, \cdots, \overline{N}-1]$ where $\overline{N} = M+3$.

Since the signature generation and verification of A.Join follow the same procedure as STAG, we omit the detailed description of A.Join.

2) MEMBERSHIP DELETION PROCEDURE

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

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

Our dynamic GKE protocols DRUG (**D**ynamic constant-**R**ound **U**nauthenticated **G**KE) and DRAG (**D**ynamic constant-**R**ound **A**uthenticated **G**KE) consist of three procedures, (STUG, U.Join, U.Leave) and (STAG, A.Join, A.Leave), respectively.

Procedure 1 U.Join$(P[0, 1, \cdots, N-1], P[N, N+1, \cdots, N+M-1])$

Phase e1[**RingGen**]

Rearrange the order with a new array of $\overline{N} = M+3$ parties

1) $U_0 = P_0, U_1 = P_1, U_2 = P_{N-1}, \bar{s}_0 = s_0, \bar{s}_1 = \mathcal{H}_1(\mathsf{sk}), \bar{s}_2 = s_{N-1}$ and for $1 \leq i \leq \overline{N}-3$, $U_{i+3} = P_{N-1+i}$
2) Let $U[0, 1, \cdots, \overline{N}-1]$ be a new ring that becomes an input of **STUG** protocol in **Phase e2**

Phase e2[**KeyComp.New**]

Run **STUG** protocol for the ring structure with the members from $U[0, 1, \cdots, \overline{N}-1]$

1) Group $U[0, 1, \cdots, \overline{N}-1]$ runs **STUG**
2) U_i calculates $\bar{z}_i$ during key generation phase (**Phase c1**) of **STUG** and broadcasts $\bar{z}_i$
3) U_0 and U_2 sends $\bar{z}_1$ and $\bar{z}_3$ to all parties in $P[2, \cdots, N-2]$
4) U_i calculates $\overline{X}_i$ during intermediate value broadcasting phase (**Phase c2**) of **STUG** and sends $\overline{X}_i$ to all parties in $P[0, \cdots, N+M-1]$
5) After the key computation phase (**Phases c3.1** and **c3.2**) of **STUG**, $U_{\overline{N}-1}$ sends $\overline{\mathbf{rec}}$ to all parties in $P[0, \cdots, N+M-1]$

Phase e3[**KeyComp.Prev**]

For party P_i $(2 \leq i \leq N-2)$,

1) Computes $\overline{Y}_{i,2} = \overline{X}_2 + \bar{z}_2 \bar{s}_1 = \overline{X}_2 + \bar{z}_2 \cdot \mathcal{H}_1(\mathsf{sk})$
2) For $j = 1$ to $\overline{N}-2$, computes $\overline{Y}_{i,2+j} = \overline{X}_{2+j} + \overline{Y}_{i,2+(j-1)}$
3) $b_i' = \sum_{j=0}^{\overline{N}-1} \overline{Y}_{i,j}$
4) Runs $\mathbf{recKey}(\cdot, \cdot)$ to output $\bar{k}_i = \mathbf{recKey}(\bar{b}_i, \overline{\mathbf{rec}})$ and gets the group secret key as $\overline{\mathbf{sk}}_i = \mathcal{H}(\bar{k}_i)$

VII. SECURITY

In this chapter, we check the correctness of our protocol and give a full security proof using the security model by Bresson *et al.* [3]. Our proof techniques are used similarly to ADGK19 [16] and DB05 [6].

We adopt "unpredictability-based" security analysis (*i.e.*, given the transcript, it is infeasible to determine the real group secret key) instead of the "indistinguishability-based" one (*i.e.*, given the transcript, the real group secret key should be indistinguishable from random) to apply the characteristic of bounded Rényi divergence.

But instead of applying Katz-Yung compiler for authenticated GKE with active adversary $\mathcal{A}_{\mathsf{AKE}}$, the security model of Bresson *et al.* [3] is considered to give a full security analysis of the dynamic case. Hence, our authenticated GKE protocol also achieves forward secrecy, and is almost fully symmetric and constant-round without additional rounds to achieve AKE security, compared to ADGK19.

A. CORRECTNESS PROOF

The correctness of GKE protocol is guaranteed if all parties agree on the group secret key. In Theorem 1, we give a condition that our GKE is correct. Most parts of our correctness proof follow the correctness proof of ADGK19 but there is some modification on error bound.

Note that padding the signature to the broadcasting messages does not change the size of the error and both membership addition and deletion procedures use the ring topology and the computation process is exactly the same as **STUG**. Hence, if we show the correctness of **STUG**, it is obvious that our dynamic and authenticated protocols satisfy the correctness condition.

Lemmas 2 and 3 are restated from ADGK19 [16].

Lemma 2 [16]: Given s_i for all i defined in the group key exchange protocol, fix $c = \sqrt{\frac{2\rho}{\pi \log(e)}}$ and let bound_ρ be the event that for all $i \in [N]$ and all coordinate $j \in [n]$, $|(s_i)_j|, |(e_i)_j|, |(e'_i)_j|, |(e''_{N-1})_j| \leq c\sigma_1$ except $|(e'_0)_j| \leq c\sigma_2$. Then

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

Proof: Since the complementary error function satisfies $erfc(x) = \frac{2}{\pi}\int_x^\infty \exp(-t^2)dt \leq \exp(-x^2)$,

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

Then, there are $3nN$ samplings from $D_{\mathbb{Z}_q,\sigma_1}$ and n samplings from $D_{\mathbb{Z}_q,\sigma_2}$ in **STUG**. Under the assumption that $3nN + n \leq \exp(c^2\pi/2)$,

$$\begin{aligned}
\Pr[\mathsf{bound}_\rho] &= (1 - \Pr[v \leftarrow D_{\mathbb{Z}_q,\sigma_1}; |v| \geq c\sigma_1 + 1])^{3nN} \\
&\quad \cdot (1 - \Pr[e'_0 \leftarrow D_{\mathbb{Z}_q,\sigma_2}; |v| \geq c\sigma_2 + 1])^{n} \\
&\geq 1 - (3nN + n) \cdot \exp(-c^2\pi) \\
&\geq 1 - \exp(c^2\pi/2) \\
&\geq 1 - 2^{-\rho}.
\end{aligned}$$

■

Lemma 3 [16]: Given bound_ρ defined in Lemma 2, let $\mathsf{product}_{s_i,\, e_j}$ be the event that for all v-th coordinate, $\left|(s_i \cdot e_j)_v\right| \leq \sqrt{n}\rho^{3/2}\sigma_1^2$. Then

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

Proof: Note that for $l \in [n]$, $(s_i)_l$ denotes the l-th coefficient of s_i and $s_i = \sum_{l=0}^{n-1}(s_i)_l X^l$. Since we take $X^n + 1$ as modulus of R, $(s_i e_j)_l = \sum_{k=0}^{n-1}(s_i)_k (e_j)^*_{l-k} X^l$ where $(e_j)^*_{l-k}$ is $(e_j)_{l-k}$ if l-k $\geq$ 0 and $-(e_j)_{l-k}$ otherwise. Thus, under bound_ρ, specifically $|(s_i)_l|, \ |(e_j)_l| \leq c\sigma_1$ where $c = \sqrt{\frac{2\rho}{\pi \cdot \log(e)}}$, by Hoeffding's inequality [49],

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

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

$$\Pr[|(s_i e_j)_l| \geq \gamma \mid \mathsf{bound}_\rho] \leq 2 \cdot \exp(\tfrac{-\rho^3}{2\,c^4}) \leq 2^{-2\rho+1}.$$

Thus, with a union bound,

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

■

Theorem 1: For a fixed ρ, and assume that

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

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

Proof: We claim that all parties calculate the same value except with negligible probability. To hold this, $k_i = k_{N-1}$ holds for all $i \in [N]$ and $j \in [n]$ j-th coefficient of $|b_{N-1} - b_i| \leq \beta_{\mathsf{Rec}}$. After very careful computation,

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

Now observe how many terms are in $b_{N-1} - b_i$. There are at most $(N-1)N/2$ terms in the form of $s_i \cdot e_j$, at most $N(N+1)/2$ terms in the form of e'_k sampled from χ_{σ_1}, at most $N-2$ terms of e'_0 sampled from χ_{σ_2}, and N terms of e''_{N-1}.

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

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

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

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

by Lemma 2 and the above inequality. Therefore, all parties agree on the group secret key except with a probability $2 \cdot 2^{-\rho}$. ∎

From the result of Theorem 1, the number of error terms in **STUG** is smaller than ADGK19. Then, the probability $\Pr_{\mathsf{STUG}}[\mathsf{AbortKey}]$ of the event **AbortKey** that error between b_i's exceeds β_{Rec} in **STUG** is smaller than the probability $\Pr_{\mathsf{Apon}}[\mathsf{AbortKey}]$ in ADGK19. Thus, **STUG** has a higher probability to output the agreed group secret key between protocol participants and so does **DRAG**.

B. SECURITY PROOF

Theorems 2, 3 and 4 show that our dynamic key exchange protocols **DRUG** = (**STUG**, **U.Join**, **U.Leave**) and **DRAG** = (**STAG**, **A.Join**, **A.Leave**) are secure in the random oracle model based on hardness of RLWE assumption. All proofs are given in Appendices A, B, and C.

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

$$\mathsf{Adv}^{\mathsf{KE}}_{\mathsf{STUG}}(t, q_E) \leq 2^{-\lambda+1} + \sqrt{\mathsf{Adv}_{\mathsf{Exp\text{-}1}} \cdot \frac{\exp\left(2\pi n(\beta_{\mathsf{Rényi}}/\sigma_2)^2\right)}{1-2^{-\lambda+1}}}$$

where $\mathsf{Adv}_{\mathsf{Exp\text{-}1}} = N \cdot \mathsf{Adv}^{RLWE}_{n,q,\chi_{\sigma_1},3}(t_1) + \mathsf{Adv}_{\mathsf{KeyRec}}(t_2) + \frac{q_E}{2^\lambda}$, $t_1 = t + \mathcal{O}(N \cdot t_{ring})$, and $t_2 = t + \mathcal{O}(N \cdot t_{ring})$ such that t_{ring} is the maximum time required to make operations in R_q.

Proof: See Appendix A. ∎

For Theorem 3, q_E and q_S are the maximum number of **Execute** and **Send** queries, respectively, that an adversary for **STAG** may ask for.

Theorem 3: The authenticated GKE protocol **STAG** is secure against an active adversary under RLWE assumption, achieves forward secrecy and satisfies the following:

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

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

Proof: See Appendix B. ∎

For Theorem 4, q_E, q_S, q_J and q_L are the maximum number of **Execute**, **Send**, **Join** and **Leave** queries that an adversary for **DRAG** may ask for.

Theorem 4: The dynamic authenticated GKE protocol **DRAG** is secure against an active adversary under RLWE assumption, achieves forward secrecy and satisfies the following:

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

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

Proof: See Appendix C. ∎

TABLE 1. Comparison with other lattice-based (authenticated) GKE protocols.

Method	DXL12.G [22]	YMZ15 [15]	ADGK19 [16]	DRAG
TA	X	✓	X	X
Scalability	X	✓	✓	✓
Communication Round for GKE (AGKE)*	N	2	3(4)	3(3)
Computational Complexity* (Samp, R.Mult, Sign, Verify)	$(N^2, N^2-N, \cdot, \cdot)$	$(2N, 2N+2, \cdot, \cdot)$	$(3N+1, 2N+1, 2N, 2N)$	$(3N+1, 2N+1, 2N, N+2)$
Dynamic Setting	X	X	X	✓

* N is the number of protocol participants in GKE protocol

VIII. COMPARISON WITH OTHER PROTOCOLS

A comparison between **DRAG** and other previously known lattice-based GKE protocols [15], [16], [22] is given in TABLE 1. For computational complexity, we ignore ring addition/deletion, or scalar multiplication due to its relatively smaller computing power, and only consider the following:

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

From TABLE 1, we define a round as the number of interactions where one party sends their message to another.

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

For ADGK19 and **DRAG**, both provide scalability without TA. **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 both protocols. However, **DRAG** expects a smaller number of signature verification since we only verify the signatures from the neighborhood.

By modifying modular exponentiation and multiplication in DB05 into ring multiplication and ring addition in RLWE setting, we design the first lattice-based dynamic group key exchange protocol in the open literature, to the best of our knowledge.

TABLE 2. Parameter choice.

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

IX. 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).

A. INSTANTIATION

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.

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 (BCNS) [28] where $n = 1024, q = 2^{32}-1$, and $\sigma_1 = 8/\sqrt{2\pi}$ for the reasonable level of security. To maximize the number of group members in DRUG, N, we select other parameters as $N = 6$, $\sigma_2 = 14,198,340/\sqrt{2\pi}$, $\beta_{\mathsf{Rec}} = 2^{29}-1$, and $\beta_{\mathsf{Rényi}} = 5,664,317$ where $\beta_{\mathsf{Rec}} = q/8$, as shown in TABLE 2.

B. NETWORK TOPOLOGY

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

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

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

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

C. 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.

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. 1, we can observe that a group of four outputs the same value as the agreed group secret key. Figs. 2 and 3 demonstrate that the U.Join and U.Leave procedures were done without errors.

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 3 shows the performance of each operation on average after running 200 times.

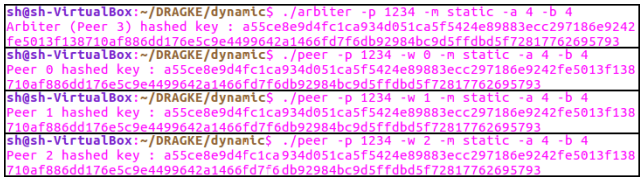


FIGURE 1. Group secret key of STUG (N = 4).

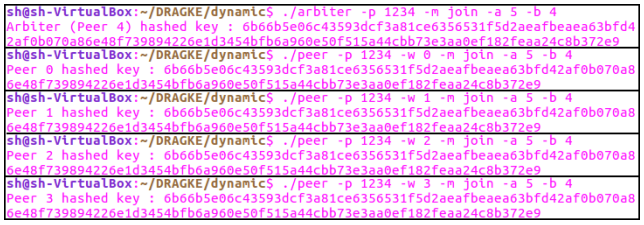


FIGURE 2. U.Join procedure in DRUG (N = 4 → N = 5).

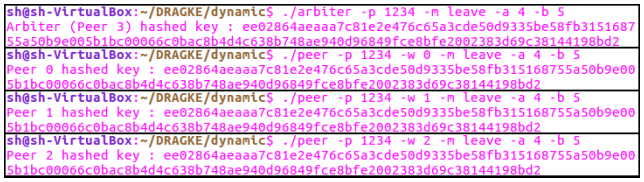


FIGURE 3. U.Leave procedure in DRUG (N = 5 → N = 4).

TABLE 3. Average runtime and cycles of each operation.

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

As we can see in TABLE 3, random sampling from χ_{σ_2} takes a longer time than random sampling from χ_{σ_1}. This is a result of the difference in size of $\sigma_1 = 8/\sqrt{2\pi}$ and $\sigma_2 = 14,198,340/\sqrt{2\pi}$. The ring polynomial addition running time is almost 0, but the ring polynomial multiplication running time is 185 μsec. Therefore ring polynomial multiplication and random sampling is an important factor in performance as we stated in Chapter VIII.

TABLE 4 shows the performance of each function. We measure the computational time for z_i, X_i, reconcile, and the group secret key. As we can see in TABLE 4, computing X_0 takes longer than computing X_i due to the sampling from χ_{σ_2}. Even though only one peer, P_0, samples values from χ_{σ_2}, its runtime is much longer than random samplings from χ_{σ_1} or ring polynomial multiplications. Hence, random sampling from χ_{σ_2} is another important factor in measuring the performance of DRUG.

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 [50], and their elliptic curve variants (ECDH, ECMQV). For the sake of simplicty, we limit the comparison with the Crypto++ library [51]. 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.

TABLE 4. Average runtime and cycles of each function.

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

TABLE 5. Performance evaluation of U.Join and U.Leave.

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

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

X. CONCLUSION AND FUTURE WORK

In this paper, we deal with the novel construction of quantum-resistant constant-round dynamic authenticated GKE protocol. Former dynamic GKE protocols rely on number-theoretic problems such as Diffie-Hellman problem which is vulnerable to quantum computing attacks. To make a quantum-resistant dynamic GKE protocol, underlying number-theoretic problems should be replaced by other quantum-resistant ones like RLWE problem.

The main contributions of this paper are as follows:

- We designed a novel constant-round dynamic authenticated GKE protocol from RLWE with quantum resistance.

- DRAG is theoretically more efficient than the previous constant-round authenticated GKE protocol from RLWE in regard to the computational complexities and communication round.
- We provided implementation details with pseudocode, as a proof of concept, to meet level 1 NIST security (128-bit AES security) in Ubuntu v16.04.5 LTS.

After checking the distinctive features between BD94 and DB05 in a static unauthenticated setting, we found that they differ only in key computation phase, Also, DB05 supports a group to agree on the group secret key with a dynamic setting. Similarly, we designed yet another quantum-resistant constant-round GKE protocol in a static setting by modifying the key computation phase of ADGK19. Then, a novel construction of a quantum-resistant constant-round dynamic authenticated GKE protocol is proposed in a dynamic setting with the quantum-resistant digital signature with the existential unforgeability under chosen message attack. For membership addition/deletion procedures, our protocol is built from RLWE setting, while the former ones are from Diffie-Hellman problem.

We consider the adversary model by Bresson *et al.* [3] that covers a dynamic authenticated GKE protocol with a weak corruption model and suggested a rigorous security proof in the random oracle model. Compared to ADGK19, our dynamic authenticated GKE protocol, DRAG, requires fewer computational complexities and a smaller communication round for authenticated GKE.

Our implementation is claimed to be practical with suitable parameter selection since the total runtime to get the group secret key takes about 3 msec. This result can be considered as a reference implementation of other quantum-resistant GKE protocols.

As future work, the vulnerability of DRAG to key reuse attacks [52]–[54] is required to be addressed. Another challenge is to reduce the size of q for better performance.

.

APPENDIX A PROOF OF THEOREM 2

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

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

Then, by a sequence of experiments, we show that an efficient adversary who queries the random oracle in Ideal experiment can query the random oracle in Exp_0 experiment. For Ideal experiment, the input k_{N-1} is chosen at uniformly random while k_{N-1} is chosen by the honest execution of STUG in Exp_0 and Exp_1 experiments.

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

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

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

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

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

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

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

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

using the property of Rényi divergence.

Proof: Let X_0, X_0' be the random variables in experiments Exp_0 and Exp_1, respectively. Error and main can be defined as

$$\mathsf{Error} = \sum_{i=0}^{N-1}(s_i e_{i+1} - s_i e_{i-1}) + \sum_{i=1}^{N-1} e_i'$$

and

$$\mathsf{main} = z_1 s_0 - z_{N-1} s_0 - \mathsf{Error},$$

respectively. Then,

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

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

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

Set $c = \sqrt{\frac{2\lambda}{\pi \log e}}$ and let bound be the event that $\left|(e_0')_j\right| \le c\sigma_2$, $\left|(s_i)_j\right|, \left|(e_i)_j\right|, \left|(e_{N-1}'')_j\right| \le c\sigma_1$, and $\left|(e_i')_j\right| \le c\sigma_1$ for all $i > 0$ and j.

From Lemmas 2 and 3, $\Pr[\mathsf{bound}] \ge 1 - 2^{-\lambda}$ and $\Pr[\left|(s_i e_j)_v\right| \le \sqrt{n}\lambda^{3/2}\sigma_1^2 \mid \mathsf{bound}] \ge 1 - 2^{-2\lambda+1}$.

With a union bound,

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

Assuming $4Nn \le 2^{\lambda}$, we derive that $\Pr[\mathsf{bound}_{\mathsf{Error}}] \ge 1 - 2^{-\lambda+1}$ and $\mathrm{RD}_2(\mathsf{Error} + \chi_{\sigma_2} \| \chi_{\sigma_2}) \le \exp(2\pi n(\beta_{\mathsf{Rényi}}/\sigma)^2)$ from Lemma 1. Thus,

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

From second to third inequality, the property that Rényi divergence is bounded is used. ■

For the rest of the proof, we will show that

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

Experiment 2: Replace z_0 with the uniform element in R_q. The rest are the same as the previous experiment.

$$\mathsf{Exp}_2 := \left\{\begin{array}{ll}
a, z_0 \leftarrow R_q; & \\
s_i, e_i \leftarrow \chi_{\sigma_1} \text{ and } z_i = a s_i + e_i & \\
\quad \text{for } 1 \le i \le N-1; & \\
e_0' \leftarrow \chi_{\sigma_2}; & \\
e_i' \leftarrow \chi_{\sigma_1} \text{ for } 1 \le i \le N-1; & \\
X_0' = -\sum_{i=1}^{N-1} X_i + e_0'; & : (\mathsf{T}, \mathsf{sk}) \\
X_i = (z_{i+1} - z_{i-1}) s_i + e_i' \text{ for} & \\
\quad 1 \le i \le N-1; & \\
e_{N-1}'' \leftarrow \chi_{\sigma_1}; & \\
Y_{N-1,N-1} = X_{N-1} & \\
\quad + z_{N-2} s_{N-1} + e_{N-1}''; & \\
Y_{N-1,(N-1)+j} = X_{(N-1)+j} & \\
\quad + Y_{N-1,(N-1)+(j-1)}; & \\
b_{N-1} = \sum_{j=0}^{N-1} Y_{N-1,(N-1)+j}; & \\
(\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1}); & \\
\mathsf{sk} = \mathcal{H}(k_{N-1}); & \\
\mathsf{T} = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, & \\
\quad \cdots, X_{N-1}, \mathsf{rec}) &
\end{array}\right\}$$

Between Experiment 1 and Experiment 2, one RLWE instance is replaced by a random value. Hence,

$$\left|\Pr_2[\mathsf{Query}] - \Pr_1[\mathsf{Query}]\right| \le \mathsf{Adv}^{RLWE}_{n,q,\chi_{\sigma_1},1}(t_1)$$

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

Since

$$\begin{aligned}
\mathsf{Adv}^{RLWE}_{n,q,\chi_{\sigma_1},1}(t_1) &\le \mathsf{Adv}^{RLWE}_{n,q,\chi_{\sigma_1},2}(t_1) \\
&\le \mathsf{Adv}^{RLWE}_{n,q,\chi_{\sigma_1},3}(t_1), \\
\left|\Pr_2[\mathsf{Query}] - \Pr_1[\mathsf{Query}]\right| &\le \mathsf{Adv}^{RLWE}_{n,q,\chi_{\sigma_1},3}(t_1).
\end{aligned}$$

Experiment 3: Replace z_0 into $z_2 - r_1$ and X_1 into $r_1 s_1 + e_1'$ where $r_1 \leftarrow R_q$. The rest are the same as the

previous experiment.

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

Since both z_0 and $z_2 - r_1$ are uniform,

$$\Pr_3\left[\mathsf{Query}\right] = \Pr_2\left[\mathsf{Query}\right].$$

Experiment 4: Replace z_1, X_1 with the uniform element in R_q. The rest are the same as the previous experiment.

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

Between Experiment 3 and Experiment 4, two RLWE instances are replaced by two independent random values. Hence,

$$\begin{aligned} \left|\Pr_4\left[\mathsf{Query}\right] - \Pr_3\left[\mathsf{Query}\right]\right| &\le \mathsf{Adv}^{RLWE}_{n,q,\chi_{\sigma_1},2}(t_1) \\ &\le \mathsf{Adv}^{RLWE}_{n,q,\chi_{\sigma_1},3}(t_1) \end{aligned}$$

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

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

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

Since both z_0 and $z_2 - r_1$ are uniform,

$$\Pr_5\left[\mathsf{Query}\right] = \Pr_4\left[\mathsf{Query}\right].$$

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

Experiment 3j: Replace z_{j-1} with $z_{j+1} - r_i$ and X_i into $r_j s_j + e'_i$ where $r_j \leftarrow R_q$. The rest are the same as the previous experiment.

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

Experiment $3j+1$: Replace z_j, X_j with the uniform element in R_q. The rest are the same as the previous experiment.

$$\mathsf{Exp}_{3j+1} := \left\{ \begin{array}{l} a, r_j \leftarrow R_q; \\ s_i, e_i \leftarrow \chi_{\sigma_1} \text{ and } z_i = as_i + e_i \\ \quad \text{for } j+1 \le i \le N-1; \\ z_0, \cdots, z_{j-2}, z_j \leftarrow R_q; \\ z_{j-1} = z_{j+1} - r_j; \\ e'_0 \leftarrow \chi_{\sigma_2}; \\ e'_i \leftarrow \chi_{\sigma_1} \text{ for } j+1 \le i \le N-1; \\ X'_0 = -\sum_{i=1}^{N-1} X_i + e'_0; \\ X_1, \cdots, X_j \leftarrow R_q; \\ X_i = (z_{i+1} - z_{i-1})s_i + e'_i \text{ for} \\ \quad j+1 \le i \le N-1; \\ e''_{N-1} \leftarrow \chi_{\sigma_1}; \\ Y_{N-1,N-1} = X_{N-1} \\ \quad + z_{N-2}s_{N-1} + e''_{N-1}; \\ Y_{N-1,(N-1)+j} = X_{(N-1)+j} \\ \quad + Y_{N-1,(N-1)+(j-1)}; \\ b_{N-1} = \sum_{j=0}^{N-1} Y_{N-1,(N-1)+j}; \\ (\mathsf{rec}, k_{N-1}) = \mathsf{recMsg}(b_{N-1}); \\ \mathsf{sk} = \mathcal{H}(k_{N-1}); \\ \mathsf{T} = (z_0, z_1, \cdots, z_{N-1}, X_0, X_1, \\ \quad \cdots, X_{N-1}, \mathsf{rec}) \end{array} : (\mathsf{T}, \mathsf{sk}) \right\}$$

Experiment $3j+2$: Replace z_{j-1} with the uniform element in R_q. The rest are the same as the previous experiment.

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

With the similar argument of Experiments 3, 4 and 5, the following inequalities are satisfied:

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

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

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

Since r_1, r_2 are uniform, so is $z_0 = r_1 + r_2$. For both Experiment $3N-4$ and $3N-3$, z_{N-2} and z_0 are uniform. Then,

$$\Pr_{3N-3}\left[\mathsf{Query}\right] = \Pr_{3N-4}\left[\mathsf{Query}\right].$$

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

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

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

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

Experiment $3N-1$: Replace $Y_{N-1,N-1}, Y_{N-1,(N-1)+j}, b_{N-1}$ with the uniform element in R_q. The rest are the same

as the previous experiment.

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

For both Experiment $3N-2$ and Experiment $3N-1$, $Y_{N-1,N-1}, Y_{N-1,(N-1)+j}$, and b_{N-1} are all uniform since r_3 is uniform in Experiment $3N-2$. Then,

$$\Pr_{3N-1}\left[\mathsf{Query}\right] = \Pr_{3N-2}\left[\mathsf{Query}\right].$$

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

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

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

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

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

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

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

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

as expected.

With the Lemma 4 and $\mathsf{Adv}^{\mathsf{KE}}_{\mathsf{STUG}}(t, q_E) \le \Pr_0\left[\mathsf{Query}\right]$, we derive the result of the theorem. ∎

APPENDIX B PROOF OF THEOREM 3

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 IV into the one that $\mathcal{A}'$ can forge a signature and the one that $\mathcal{A}'$ cannot forge a signature.

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

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

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

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

For Send query in $\mathcal{A}'$, we have two types of Send queries since for each instance (P_0, i_0), there is a first send query Send_0 to start a new session and the other send queries with a message/signature pair. With an instance $\{(P_0, i_0), \cdots, (P_k, i_k)\}$, $\mathsf{Send}_0(P_j, i_j\{P_0, \cdots, (P_k)\} \setminus P_j)$ for each $0 \le j \le k$. $\mathcal{A}$ defines $S = \{(P_0, i_0), \cdots, (P_k, i_k)\}$ and sends S to $\mathsf{Execute}$ oracle to obtain a transcript T of STUG. Then, (S, T) is added to tlist. For other send queries, $\mathcal{A}$ verifies the query according to Protocol 2. If the verifications fail, $\mathcal{A}$ aborts an instance (P_j, i_j). Otherwise, $\mathcal{A}$ finds (S, T) from tlist such that $(P_j, i_j) \in S$ and from T, it finds the appropriate message with respect to (P_j, i_j) in $\mathcal{A}'$ and returns the public information to $\mathcal{A}'$.

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

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

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

Pseudocode 1 Calculating Public key z_i

function calc_pk(a)
  ▷ (**Phase c1**) Creating RLWE public key z_i
  $s \leftarrow$ sample1() ▷ Sampling from χ_{σ_1}
  $e \leftarrow$ sample1() ▷ Sampling from χ_{σ_1}
  $result \leftarrow a * s + e$
  return s, $result$
end function

Pseudocode 2 Calculate Augmented Public key X_i

function calc_aug_pk(i, N, z, s)
  ▷ (**Phase c2**) Calculating X_i
  $e' \leftarrow$ null
  if $i = N-1$ **then** ▷ Peer $N-1$
    $e' \leftarrow$ sample1() ▷ Sampling from χ_{σ_1}
    $result \leftarrow z_0 - z_{N-2}$
  else if $i = 0$ **then** ▷ Peer 0
    $e' \leftarrow$ sample2() ▷ Sampling from χ_{σ_2}
    $result \leftarrow z_1 - z_{N-1}$
  else ▷ Peers 1 to $N-2$
    $e' \leftarrow$ sample1() ▷ Sampling from χ_{σ_1}
    $result \leftarrow z_{i+1} - z_{i-1}$
  end if
  $result \leftarrow result * s + e'$
  return $result$
end function

Pseudocode 3 Calculate Reconciliation **rec** and sk_{N-1}

function calc_recon(N, z, X, s)
  ▷ (**Phase c3.1**) Calculate sk_{N-1}, rec for the peer $N-1$
  $e'' \leftarrow$ sample1() ▷ Sampling from χ_{σ_1}
  $tmp \leftarrow z_{N-2} * s + X_{N-1} + e''$
  $Y_{N-1} \leftarrow tmp$
  for $j = 0$ to $N-2$ **do**
    $tmp \leftarrow tmp + X_j$
    $Y_j \leftarrow tmp$
  end for
  $result \leftarrow 0$
  for $j = 0$ to $N-1$ **do**
    $result \leftarrow result + Y_i$
  end for
  rec, $k_{N-1} \leftarrow$ **recMsg**($result$)
  return rec, $\mathcal{H}(k_{N-1})$
end function

Pseudocode 4 Calculate the Group Secret key sk_i of **U.Join** for Joining Members

function calc_session_key $sk_i(i, N, X, s,$ **rec**)
  ▷ (**Phase c3.2**) Calculating sk_i
  $tmp \leftarrow z_{\mathrm{mod}(N+i-1,N)} * s$
  $tmp2 \leftarrow X_i$
  for $j = 0$ to $N-1$ **do**
    $tmp \leftarrow tmp + tmp2$
    $Y_{\mathrm{mod}(i+j,N)} \leftarrow tmp$
    $tmp2 \leftarrow X_{\mathrm{mod}(i+j+1,N)}$
  end for
  $result \leftarrow 0$
  for $j = 0$ to $N-1$ **do**
    $tmp \leftarrow Y_j$
    $result \leftarrow result + tmp$
  end for
  return $\mathcal{H}$(**recKey**($result$, **rec**))
end function

Pseudocode 5 Calculate the Group Secret key sk_i of **U.Join** for the Previous Group Members

function calc_remain_session_key($i, N, X, s,$ **rec**)
  ▷ (**Phase e3**) Calc. sk_i (join, peers 2 to $N-2$)
  $tmp \leftarrow z_2 * s_1$
  $tmp2 \leftarrow X_2$
  for $j = 0$ to $N-1$ **do**
    $tmp \leftarrow tmp + tmp2$
    $Y_{\mathrm{mod}(2+j,N)} \leftarrow tmp$
    $tmp2 \leftarrow X_{\mathrm{mod}(3+j,N)}$
  end for
  $result \leftarrow 0$
  for $j = 0$ to $N-1$ **do**
    $tmp \leftarrow Y_j$
    $result \leftarrow result + tmp$
  end for
  return $\mathcal{H}$(**recKey**($result$, **rec**))
end function

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

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

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

■

APPENDIX C PROOF OF THEOREM 4

Proof: Similar to Theorem 3, we separate the event Succ that adversary $\mathcal{A}'$ wins into two cases: one with forging a signature and the other without forging. Then, we design how to answer $\mathsf{Execute}$, Join, Leave and Send queries from DRAG using $\mathsf{Execute}$ queries from STUG.

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

Pseudocode 6 Arbiter for **STUG** and **U.Leave**

procedure Arbiter_leave_or_static(N_2, N_1)
  ▷ N_2 peers after leave, N_1 peers before leave
  ▷ N_1 is not used in this procedure.
  establish connections with peers
  assign an index i of each peer
  $k \leftarrow N_2$
  $s_{k-1}, z_{k-1} \leftarrow$ calc_pk (s)
  $step \leftarrow 0$
  while $step < 4$ **do**
    for $i = 0$ to $N_2 - 2$ **do**
      receive index of peer i
      if rec is not calculated and X is available **then**
        sk_{N_2-1}, **rec** $\leftarrow$ calc_recon(k, z, X, s_{k-1})
      end if
      send step of peer i
      if $step = 1$ **then**
        $X_{k-1} \leftarrow$ calc_aug_pk($k-1, k, z, s_{k-1}$)
      end if
      if $step = 0$ **then**
        receive public key z_i
        break
      else if $step = 1$ **then**
        send all public keys z
        receive augmented pub key X_i
        break
      else if $step = 2$ **then**
        send all augmented public keys X
        break
      else ▷ $step = 3$
        send **rec**
      end if
      save that the step is completed with peer i
      if the step is completed with all peers **then**
        $step \leftarrow step + 1$
      end if
    end for
  end while
  print sk_{N_2-1}
end procedure

Pseudocode 7 Peer for **STUG** and **U.Leave**

procedure Peer_leave_or_static(N_2, N_1)
  ▷ N_2 peers after leave, N_1 peers before leave
  ▷ N_1 is not used in this procedure.
  establish connection to the arbiter
  $k \leftarrow N_2$
  while $step < 4$ **do**
    send index of peer i
    receive step
    if $step = 0$ **then**
      $s_i, z_i \leftarrow$ calc_public_key(a)
      send the public key z_i
    else if $step = 1$ **then**
      receive all public keys z
      $X_i \leftarrow$ calc_aug_pk(i, k, z, s_i)
      send the augmented public key X_i
    else if $step = 2$ **then**
      receive all augmented public keys X
    else ▷ $step = 3$
      receive **rec**
      $sk_i \leftarrow$ calc_session_key(i, k, s_i, **rec**)
    end if
  end while
  print sk_i
end procedure

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

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

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

We define three lists **Tlist**, **Jlist** and **Llist** to store the result from **Execute**, **Join** and **Leave** queries as a set.

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

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

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

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

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

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

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

Pseudocode 8 Arbiter for **U.Join**

```
procedure arbiter_join(N_2, N_1)
                ▷ N_2 peers after join, N_1 peers before join
    Establish connections with peers
    index ← the array of indices of peers
    for i = 2 to N_1 − 2 do ▷ The 3 peers performing join
        index[i] ← −1
    end for
    k ← N_2 − N_1 + 3
    s_{k−1}, z_{k−1} ← calc_pk (s)
    step ← 0
    while step < 4 do
        for i = 0 to N_2 − 2 do
            receive index of peer i
            if rec is not calculated and X is available then
                sk_{N_2−1}, rec ← calc_recon(k, z, X, s_{k−1})
            end if
            send step of peer i
            if step = 1 then
                X_{k−1} ← calc_aug_pk(k − 1, k, z, s_{k−1})
            end if
            if step = 0 then
                if index[i] = −1 then
                    break
                end if
                receive public key z_{index[i]}
                if i = 1 then
                    receive secret key s_1
                end if
                break
            else if step = 1 then
                send all public keys z
                if index[i] = −1 then
                    break
                end if
                receive augmented public key X_{index[i]}
                break
            else if step = 2 then
                send all augmented public keys X
                break
            else                                    ▷ step = 3
                send rec
                if index[i] = −1 then
                    send secret key s_1
                end if
            end if
            save that the step is completed with peer i
            if the current step is completed with all peers
    then
                step ← step + 1
            end if
        end for
    end while
    print sk_{N_2−1}
end procedure
```

Pseudocode 9 Peer for **U.Join**

```
procedure peer_join(N_2, N_1)
                ▷ N_2 peers after join, N_1 peers before join
    establish connection to the arbiter
    j ← received index[i] on the arbiter side
    k ← N_2 − N_1 + 3
    while step < 4 do
        send index of peer i
        receive step
        if step = 0 then
            if 2 <= i <= N_1 − 2 then
                continue        ▷ Existing peers do nothing
            end if
            s_j, z_j ← calc_public_key(a)
            send the public key z_j
            if i = 1 then
                send the secret key s_1
            end if
        else if step = 1 then
            receive all public keys z
            if 2 <= i <= N_1 − 2 then
                continue        ▷ Existing peers do nothing
            end if
            X_j ← calc_aug_pk(j, k, z, s_j)
            send the augmented public key X_j
        else if step = 2 then
            receive all augmented public keys X
        else                                        ▷ step = 3
            receive rec
            if 2 <= i <= N_1 − 2 then
                receive the secret key s_1
                sk_i ←
    calc_remain_session_key(i, k, X, s_1, rec)
            else
                sk_i ← calc_session_key(j, k, s_j, rec)
            end if
        end if
    end while
    print sk_i
end procedure
```

Execute queries in $\mathcal{A}$ is at most $q_E + \frac{q_S + q_J + q_L}{2}$. Thus,

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

■

APPENDIX D PSEUDOCODE OF THE IMPLEMENTATION

See Pseudocode 1–9.

REFERENCES

[1] A. J. Menezes, J. Katz, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. Boca Raton, FL, USA: CRC Press, 1996.

[2] E. Bresson, O. Chevassut, D. Pointcheval, and J.-J. Quisquater, "Provably authenticated group Diffie–Hellman key exchange," in *Proc. 8th ACM Conf. Comput. Commun. Secur. (CCS)*. Philadelphia, PA, USA: ACM, 2001, pp. 255–264.

[3] E. Bresson, O. Chevassut, and D. Pointcheval, "Provably authenticated group Diffie–Hellman key exchange—The dynamic case," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur. (ASIACRYPT)*. Gold Coast, QLD, Australia: Springer, 2001, pp. 290–309.

[4] E. Bresson, O. Chevassut, and D. Pointcheval, "Dynamic group Diffie–Hellman key exchange under standard assumptions," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn. (EUROCRYPT)*. Amsterdam, The Netherlands: Springer, 2002, pp. 321–336.

[5] J. Katz and M. Yung, "Scalable protocols for authenticated group key exchange," in *Proc. Annu. Int. Cryptol. Conf. (CRYPTO)*. Santa Barbara, CA, USA: Springer, 2003, pp. 110–125.

[6] R. Dutta and R. Barua, "Constant round dynamic group key agreement," in *Proc. Int. Conf. Inf. Secur.* Singapore: Springer, 2005, pp. 74–88.

[7] M. Burmester and Y. Desmedt, "A secure and efficient conference key distribution system," in *Proc. Workshop Theory Appl. Cryptograph. Techn. (EUROCRYPT)*. Perugia, Italy: Springer, 1994, pp. 275–286.

[8] M. Burmester and Y. Desmedt, "A secure and scalable group key exchange system," *Inf. Process. Lett.*, vol. 94, no. 3, pp. 137–143, May 2005.

[9] M. Steiner, G. Tsudik, and M. Waidner, "Diffie–Hellman key distribution extended to group communication," in *Proc. 3rd ACM Conf. Comput. Commun. Secur. (CCS)*. New Delhi, India: ACM, 1996, pp. 31–37.

[10] M. Steiner, G. Tsudik, and M. Waidner, "CLIQUES: A new approach to group key agreement," in *Proc. 18th Int. Conf. Distrib. Comput. Syst.* Amsterdam, The Netherlands: IEEE Press, 1998, pp. 380–387.

[11] M. Steiner, G. Tsudik, and M. Waidner, "Key agreement in dynamic peer groups," *IEEE Trans. Parallel Distrib. Syst.*, vol. 11, no. 8, pp. 769–780, Aug. 2000.

[12] Y. Kim, A. Perrig, and G. Tsudik, "Simple and fault-tolerant key agreement for dynamic collaborative groups," in *Proc. 7th ACM Conf. Comput. Commun. Secur. (CCS)*. Athens, Greece: ACM, 2000, pp. 235–244.

[13] Y. Kim, A. Perrig, and G. Tsudik, "Tree-based group key agreement," *ACM Trans. Inf. Syst. Secur.*, vol. 7, no. 1, pp. 60–96, 2004.

[14] D. Boneh, D. Glass, D. Krashen, K. Lauter, S. Sharif, A. Silverberg, M. Tibouchi, and M. Zhandry, "Multiparty non-interactive key exchange and more from isogenies on elliptic curves," 2018, *arXiv:1807.03038*. [Online]. Available: http://arxiv.org/abs/1807.03038

[15] X. Yang, W. Ma, and C. Zhang, "Group authenticated key exchange schemes via learning with errors," *Secur. Commun. Netw.*, vol. 8, no. 17, pp. 3142–3156, Nov. 2015.

[16] D. Apon, D. Dachman-Soled, H. Gong, and J. Katz, "Constant-round group key exchange from the Ring-LWE assumption.," in *Proc. Int. Conf. Post-Quantum Cryptogr.* Chongqing, China: Springer, 2019, pp. 189–205.

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

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

[19] C.-H. Li and J. Pieprzyk, "Conference key agreement from secret sharing," in *Proc. Australas. Conf. Inf. Secur. Privacy*. Wollongong, NSW, Australia: Springer, 1999, pp. 64–76.

[20] R. F. Olimid, "Provable secure constant-round group key agreement protocol based on secret sharing," in *Proc. Int. Joint Conf. (SOCO-CISIS-ICEUTE)*. Salamanca, Spain: Springer, 2014, pp. 489–498.

[21] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proc. 35th Annu. Symp. Found. Comput. Sci.* Santa Fe, NM, USA: IEEE Press, 1994, pp. 124–134.

[22] J. Ding, X. Xie, and X. Lin, "A simple provably secure key exchange scheme based on the learning with errors problem," IACR Cryptol. ePrint Arch., Tech. Rep. 2012/688, 2012.

[23] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn. (EUROCRYPT)*. French Riviera, France: Springer, 2010, pp. 1–23.

[24] T. van Erven and P. Harremos, "Rényi divergence and kullback-leibler divergence," *IEEE Trans. Inf. Theory*, vol. 60, no. 7, pp. 3797–3820, Jul. 2014.

[25] A. Langlois, D. Stehlé, and R. Steinfeld, "GGHLite: More efficient multilinear maps from ideal lattices," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn. (EUROCRYPT)*. Copenhagen, Denmark: Springer, 2014, pp. 239–256.

[26] A. Bogdanov, S. Guo, D. Masny, S. Richelson, and A. Rosen, "On the hardness of learning with rounding over small modulus," in *Proc. Theory Cryptogr. Conf.* Tel Aviv, Israel: Springer, 2016, pp. 209–224.

[27] C. Peikert, "Lattice cryptography for the Internet," in *Proc. Int. Workshop Post-Quantum Cryptogr.* Waterloo, ON, Canada: Springer, 2014, pp. 197–219.

[28] J. W. Bos, C. Costello, M. Naehrig, and D. Stebila, "Post-quantum key exchange for the TLS protocol from the ring learning with errors problem," in *Proc. IEEE Symp. Secur. Privacy*. San Jose, CA, USA: Springer, May 2015, pp. 553–570.

[29] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, "Post-quantum key exchange—A new hope," in *Proc. USENIX Secur. Symp.* Austin, TX, USA: USENIX, 2016, pp. 327–343.

[30] J. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko, A. Raghunathan, and D. Stebila, "Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.* Vienna, Austria: ACM, Oct. 2016, pp. 1006–1018.

[31] J. Zhang, Z. Zhang, J. Ding, M. Snook, and O. Dagdelen, "Authenticated key exchange from ideal lattices," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn. (EUROCRYPT)*. Sofia, Bulgaria: Springer, 2015, pp. 719–751.

[32] M. Just and S. Vaudenay, "Authenticated multi-party key agreement," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur. (ASIACRYPT)*. Kyongju, South Korea: Springer, 1996, pp. 36–49.

[33] H.-J. Kim, S.-M. Lee, and D. H. Lee, "Constant-round authenticated group key exchange for dynamic groups," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur. (ASIACRYPT)*. Jeju Island, South Korea: Springer, 2004, pp. 245–259.

[34] R. Dutta and R. Barua, "Provably secure constant round contributory group key agreement in dynamic setting," *IEEE Trans. Inf. Theory*, vol. 54, no. 5, pp. 2007–2025, May 2008.

[35] J. Katz and J. S. Shin, "Modeling insider attacks on group key-exchange protocols," in *Proc. 12th ACM Conf. Comput. Commun. Secur. (CCS)*. Alexandria, VA, USA: ACM, 2005, pp. 180–189.

[36] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theory*, vol. IT-22, no. 6, pp. 644–654, Nov. 1976.

[37] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehle, "CRYSTALS–Kyber: A CCA-secure module-lattice-based KEM," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*. London, U.K.: IEEE Press, Apr. 2018, pp. 353–367.

[38] H. Baan, S. Bhattacharya, S. R. Fluhrer, O. Garcia-Morchon, T. Laarhoven, R. Rietman, M.-J. O. Saarinen, L. Tolhuizen, and Z. Zhang, "Round5: Compact and fast post-quantum public-key encryption," in *Proc. Int. Conf. Post-Quantum Cryptogr.* Chongqing, China: Springer, 2019, pp. 83–102.

[39] J. H. Cheon, D. Kim, J. Lee, and Y. Song, "Lizard: Cut off the tail! A practical post-quantum public-key encryption from LWE and LWR," in *Proc. Int. Conf. Secur. Cryptogr. Netw.* Amalfi, Italy: Springer, 2018, pp. 160–177.

[40] M.-J. O. Saarinen, "HILA5: On reliability, reconciliation, and error correction for ring-LWE encryption," in *Proc. Int. Conf. Sel. Areas Cryptogr.* Ottawa, ON, Canada: Springer, 2017, pp. 192–212.

[41] M. Seo, S. Kim, D. H. Lee, and J. H. Park, "EMBLEM:(R) LWE-based key encapsulation with a new multi-bit encoding method," *Int. J. Inf. Secur.*, pp. 1–17, Jul. 2019, doi: 10.1007/s10207-019-00456-9.

[42] J. Ding, S. Alsayigh, J. Lancrenon, R. Saraswathy, and M. Snook, "Provably secure password authenticated key exchange based on RLWE for the post-quantum world," in *Proc. Cryptographers Track RSA Conf.* San Francisco, CA, USA: Springer, 2017, pp. 183–204.

[43] X. Gao, J. Ding, J. Liu, and L. Li, "Post-quantum secure remote password protocol from RLWE problem," in *Proc. Int. Conf. Inf. Secur. Cryptol.* Xi'an, China: Springer, 2017, pp. 99–116.

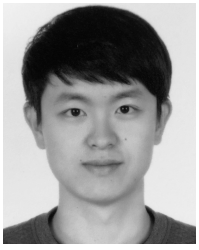
[44] E. Lee, Y.-S. Kim, J.-S. No, M. Song, and D.-J. Shin, "Modification of frodokem using gray and error-correcting codes," *IEEE Access*, vol. 7, pp. 179564–179574, 2019.

[45] S. Akleylek and K. Seyhan, "A probably secure bi-GISIS based modified AKE scheme with reusable keys," *IEEE Access*, vol. 8, pp. 26210–26222, 2020.

[46] P. Choi, J.-H. Kim, and D. K. Kim, "Fast and power-analysis resistant ring lizard crypto-processor based on the sparse ternary property," *IEEE Access*, vol. 7, pp. 98684–98693, 2019.
[47] S. Streit and F. De Santis, "Post-quantum key exchange on ARMv8-A: A new hope for NEON made simple," *IEEE Trans. Comput.*, vol. 67, no. 11, pp. 1651–1662, Nov. 2018.
[48] X. Gao, J. Ding, L. Li, and J. Liu, "Practical randomized RLWE-based key exchange against signal leakage attack," *IEEE Trans. Comput.*, vol. 67, no. 11, pp. 1584–1593, Nov. 2018.
[49] W. Hoeffding, "Probability inequalities for sums of bounded random variables," in *The Collected Works of Wassily Hoeffding*. Springer, 1994, pp. 409–426.
[50] H. Krawczyk, "HMQV: A high-performance secure Diffie–Hellman protocol (extended abstract)," in *Proc. Annu. Int. Cryptol. Conf.–CRYPTO*. Santa Barbara, CA, USA: Springer, 2005, pp. 546–566.
[51] *Crypto++ Library 8.2*. Accessed: Apr. 1–9. [Online]. Available: https://www.cryptopp.com/
[52] J. Ding, S. Alsayigh, R. V. Saraswathy, S. Fluhrer, and X. Lin, "Leakage of signal function with reused keys in RLWE key exchange," in *Proc. IEEE Int. Conf. Commun. (ICC)*. Paris, France: IEEE Press, May 2017, pp. 1–6.
[53] J. Ding, S. Fluhrer, and R. Saraswathy, "Complete attack on RLWE key exchange with reused keys, without signal leakage," in *Proc. Australas. Conf. Inf. Secur. Privacy*. Wollongong, NSW, Australia: Springer, 2018, pp. 467–486.
[54] C. Liu, Z. Zheng, and G. Zou, "Key reuse attack on newhope key exchange protocol," in *Proc. Int. Conf. Inf. Secur. Cryptol.* Seoul, South Korea: Springer, 2018, pp. 163–176.

RAKYONG CHOI received the B.S. and M.S. degrees from the Department of Mathematical Sciences, Korea Advanced Institute of Science and Technology (KAIST), South Korea, in 2011 and 2013, respectively, where he is currently pursuing the Ph.D. degree with the School of Computing.

His current research interests include lattice-based cryptography, homomorphic cryptographic schemes, and their applications.

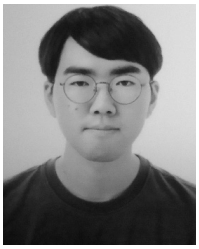


DONGYEON HONG received the B.S. degree from the Department of Mathematics, Kyungpook National University, Daegu, South Korea, in 2018, and the M.S. degree from the Graduate School of Information Security, Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2020.

His research interests include lattice-based cryptography and homomorphic encryption.

SEONGHO HAN received the B.S. degree from the Department of Mathematical Sciences and the M.S. degree from the Graduate School of Information Security, Korea Advanced Institute of Science and Technology (KAIST), South Korea, in 2016 and 2020, respectively.

SEUNGGEUN BAEK received the B.S. degree from the School of Computing, Korea Advanced Institute of Science and Technology (KAIST), South Korea, in 2018, where he is currently pursuing the master's degree with the Graduate School of Information Security.



WOOSEOK KANG is currently pursuing the B.S. degree with the School of Computing, Korea Advanced Institute of Science and Technology (KAIST), South Korea.

KWANGJO KIM (Member, IEEE) received the B.Sc. and M.Sc. degrees in electronic engineering from Yonsei University, Seoul, South Korea, in 1980 and 1983, respectively, and the Ph.D. degree from the Division of Electrical and Computer Engineering, Yokohama National University, Yokohama, Japan, in 1991.

He worked at the Electronics and Telecommunications Research Institute (ETRI), from 1979 to 1997, as a head in coding section. He was a Visiting Professor with the Massachusetts Institute of Technology, Cambridge, MA, USA, and the University of California at San Diego, La Jolla, CA, USA, in 2005, and the Khalifa University of Science, Technology and Research, Abu Dhabi, UAE, in 2012, and an Education Specialist with the Bandung Institute of Technology, Bandung, Indonesia, in 2013. He is currently a Full Professor with the School of Computing and the Graduate School of Information Security, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, the Korean representative to IFIP TC-11, and the honorable President of the Korea Institute of Information Security and Cryptography (KIISC). His current research interests include the theory of cryptology, information security, and their applications.

Dr. Kim served as a Board Member of the International Association for Cryptologic Research (IACR), from 2000 to 2004, the Chairperson of the Asiacrypt Steering Committee, from 2005 to 2008, and the President of KIISC, in 2009. He is a Fellow of IACR, a member of IEICE and ACM, and a member of IACR Fellow Selection Committee. He serves as an Editor-in-Chief of online journal *Cryptograph* and an Editor of *Journal of Mathematical Cryptology*. Moreover, he serves as a General Chair of Asiacrypt2020 and PQCrypto2021 which will be held in Daejeon, South Korea, in 2020 and 2021, respectively.