Authenticated Tree-based R-LWE Group Key Exchange

Hector Hougaard[1] and Atsuko Miyaji[2]

[1]*The author is affiliated with the Graduate School of Engineering, Osaka University, Suita-shi, 565-0871 Japan.*
[2]*The author is affiliated with the Graduate School of Engineering and the Center for Quantum Information and Quantum Biology, Osaka University, Suita-shi, 565-0871 Japan.*
The author is also affiliated with Japan Advanced Institute of Science and Technology, Nomi-shi, 923-1292 Japan.
Email: hector@cy2sec.comm.eng.osaka-u.ac.jp

We present the first constant round, multicast, authenticated tree-based R-LWE group key exchange protocol with logarithmic communication and memory complexity. Our protocol achieves post-quantum security through a reduction to a Diffie-Hellman-like analogue to the decisional R-LWE problem. We also present a sequential version with constant memory complexity but a logarithmic number of rounds and communication complexity.

Keywords: Authenticated; Group Key Exchange; Post-Quantum; R-LWE; Key Distribution

1. INTRODUCTION

Ring-learning-with-errors (R-LWE) is a cryptograpic approach for post-quantum cryptography, a candidate for maintaining security against the upcoming quantum computers while using classical computers to enact the cryptographic protocols. As is, it has been applied to fundamental cryptography such as two party key exchanges, see for example Ding, Xie, and Lin [1] (with Peikert's reconcilliation tweak [2]). Apon, Dachman-Soled, Gong, and Katz [3] at PQCrypto 2019, and Choi, Hong, and Kim [4] from 2020, both manage to extend the applicability further by showing how to create authenticated group key exchanges (AGKEs) from the decisional R-LWE problem [5]. Their schemes are interesting generalizations of Diffie-Hellman based group key exchanges (GKEs), one by Burmester and Desmedt [6] and one by Dutta and Barua [7]. However, for n parties, their protocols both have linear order communication complexity. Group key exchanges have applicability in modern scenarios where many parties need to communicate securely, such as group chats and online gaming. These scenarios often require group encryption, especially if there is money or sensitive information involved. Furthermore, for private persons with limited hardware capabilities, fast group key exchanges with low communication complexity are preferred, so it would be beneficial to reduce the complexity if possible.

In this paper, we generalize a tree-based scheme, also by Burmester and Desmedt [8], by combining it with the Ding et al [1] key exchange using Peikert's tweak [2], and call it the tree-based R-LWE GKE (tree-R-LWE-GKE). It is a constant round protocol with logarithmic order communication and memory complexity. We also generalize a sequential version, the peer-to-peer tree-based R-LWE GKE (P2P-tree-R-LWE-GKE). P2P-tree-R-LWE-GKE achieves constant communication and memory complexity but logarithmic order round complexity. The security of these protocols reduces to a Diffie-Hellman-like version of the decisional R-LWE problem, shown by Bos, Costello, Naehrig, and Stebila [9] to have comparable security to the decisional R-LWE problem. These GKEs were presented at ICICS 2020 [10] in a preliminary version. We extend these results by creating a compiler that turns both of our previous protocols into AGKEs, while maintaining logarithmic order complexities. The reason for considering such a compiler is that the standard approach for turning a GKE into an AGKE, by using a Katz-Yung compiler [11], forces the compiled protocol to have at least linear order complexity, which we seek to avoid. In order to do this, we modify the compiler given by Desmedt, Lange, and Burmester [12], and prove the security in a stronger security model than the original, namely the G-CK$^+$ model of Suzuki and Yoneyama [13], by a simple reduction to the Diffie-Hellman-like problem given by Bos et al [9]. The specific differences between our compiler and the one given in [12] are explained in our concluding remarks.

The organization of our paper is as follows. In Sect. 2, we give our notation and explain both our security definitions in Sect. 2.1 and R-LWE specific definitions in Sect. 2.2. In Sect. 3, we state our tree-based R-LWE group key exchange (tree-R-LWE-GKE) and our sequential tree-based R-LWE group key exchange (P2P-tree-R-LWE-GKE) protocol. The authenticated compiler for our GKEs is given in Sect. 4. We compare our AGKEs to other post-quantum R-LWE AGKEs in Sect. 5. We give our concluding remarks in Sect. 6.

2. PRELIMINARIES

We begin by defining notation and notions that we will use in our treatment of GKEs.

On notation, if χ is a probability distribution over a set S, then $s \xleftarrow{R} \chi$ denotes sampling an element $s \in S$ according to the distribution χ. We use the notation $s \xleftarrow{R} S$ to denote element $s \in S$ being chosen uniformly at random from S. If $\texttt{Algo}$ is an algorithm, then we let $y \leftarrow \texttt{Algo}(x)$ denote the output y from the algorithm, given input x. If the algorithm is probabilistic and uses some randomness to choose its output, we may draw attention to this by using the notation $y \xleftarrow{R} \texttt{Algo}(x)$.

In this paper, (post-quantum) hard means that there is no polynomial-time (quantum) algorithm that can solve a given problem, except with negligible probability.

2.1. Security Model

For the security of our AGKE protocols, we consider the G-CK$^+$ model of Suzuki and Yoneyama [13]. We reformulate their description to using our notation and note where we add to the definition to fit the specific purposes of this article.

Consider a finite set of parties $\mathfrak{P} = \{\mathcal{P}_0, \ldots, \mathcal{P}_\eta\}$ modeled by probabilistic polynomial-time (PPT) Turing machines with security parameter λ, usually given in unary as 1^λ. A party $\mathcal{P}_i$ generates its own static secret key SSK_i and static public key SPK_i, where the public key is linked with $\mathcal{P}_i$'s identity in some public system like a public key infrastructure (PKI). For a party $\mathcal{P}_i$, we denote the ephemeral secret (public) key by ESK_i (EPK_i).

2.1.1. Session

Any subset $\{\mathcal{P}_{i_1}, \ldots, \mathcal{P}_{i_n}\} \subseteq \mathfrak{P}$, where $2 \leq n \leq \eta$, can, at any time, decide to invoke a new GKE protocol. We call such an invocation a *session*, managed by a tuple $(\Pi, \mathcal{P}_{i_l}, \{\mathcal{P}_{i_1}, \ldots, \mathcal{P}_{i_n}\})$, where Π is the protocol identifier and $\mathcal{P}_{i_l}$ is the party identifier.[3] Without loss of generality, we hereafter suppose that $\mathcal{P}_{i_l} = \mathcal{P}_l$. $\mathcal{P}_i$ outputs EPK_i and receives $EPK_{i'}$ from $\mathcal{P}_{i'}$ from all relevant (necessary) parties and outputs the session key.[4]

When $\mathcal{P}_i$ is the i-th party of a session, we may define the *session id* $\mathsf{sid} = (\Pi, \mathcal{P}_i, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}, EPK_i)$. We call $\mathcal{P}_i$ the *owner* of sid, if the second coordinate of sid is $\mathcal{P}_i$, and a *peer* of sid if it is not. A session is said to be *completed* if its owner computes the session key. We say that $(\Pi, \mathcal{P}_{i'}, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\})$ is a *matching session* of $(\Pi, \mathcal{P}_i, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\})$, where $i' \neq i$.

[3] Suzuki and Yoneyama [13] define their sessions with a 'role' for a party, which may be indexed differently from the party index, as well as a corresponding 'player' definition. In our protocols, the role of a party is determined by the placement in the double-tree (see Sect. 3), which in turn is determined by the index of the party, which can be altered as needed, hence the role is uniquely determined by the party index. We therefore remove this 'role' (and 'player') from our definition of session.

[4] Suzuki and Yoneyama [13] assume that each party receives public keys from all other parties, but this forces any GKE to have at least linear order in n, which we aim to avoid, hence we have altered the model slightly. In the end, parties only need as many keys as are relevant or necessary to compute the session key, which our alteration highlights.

2.1.2. Adversary

We consider an adversary $\mathcal{A}$, modeled as a (not necessarily classical) PPT Turing machine that controls all communication, including session activation and registration of parties. It does so using the following queries:

SEND$(\mathcal{P}_i, m)$**:** $\mathcal{P}_i$ is the receiver and the message has the form $(\Pi, \mathcal{P}_i, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}, Init)$ if it is an initializing message for session activation, and includes $(\Pi, \mathcal{P}_{i'}, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}, EPK_{i'})$ otherwise. Note that these messages may be whatever $\mathcal{A}$ requires.

ESTABLISH$(\mathcal{P}_i, SPK_i)$**:** A new party is added to $\mathfrak{P}$. Note that $\mathcal{A}$ is not required to prove possession of the corresponding SSK_i. If a party is registered by such a query, then the party is called *dishonest*, if not, the party is called *honest*.

As an addition to the G-CK$^+$ model, we consider two different adversary types: *fair* and *manipulating*. Both types of adversary may send an initializing message to a subset of parties to start a GKE protocol between them, but the *fair adversary* may not create, omit, nor manipulate any message between parties, honestly relaying messages between parties as per the protocol. The *manipulating adversary* has no such restrictions. This is to mimic the difference between GKEs and authenticated GKEs.

The adversary is further given access to the following attack queries.

SESSIONREVEAL(sid)**:** Reveals the session key of a session sid to the adversary, only if the session is completed.

STATEREVEAL(sid)**:** Reveals to the adversary the session state of the owner of the sid if the session is not yet completed, i.e. the session key has yet to be established. The session state contains all ephemeral secret keys and intermediate computation results except for immediately erased information, but it does not contain the static secret key. The protocol specifies what the session state contains.

STATICREVEAL$(\mathcal{P}_i)$**:** Reveals the static secret key of party $\mathcal{P}_i$ to $\mathcal{A}$.

EPHEMERALREVEAL(sid)**:** Reveals to the adversary all ephemeral secret keys of the owner of sid if the session is not yet completed. This does not reveal other state information such that an adversary might trivially win.

DEFINITION 2.1. *Let* $\mathsf{sid}^* = (\Pi, \mathcal{P}_i, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}, EPK_i)$ *be a completed session between honest parties* $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, *owned by* $\mathcal{P}_i$. *If a matching session exists, then let* $\overline{\mathsf{sid}^*}$ *be a matching session of* sid^*. *We say that* sid^* *is* ***fresh*** *if none of the following is true:*

1. $\mathcal{A}$ *queried* SESSIONREVEAL(sid^*), *or* SESSIONREVEAL$(\overline{\mathsf{sid}^*})$ *for any* $\overline{\mathsf{sid}^*}$ *if* $\overline{\mathsf{sid}^*}$ *exists; or*
2. $\overline{\mathsf{sid}^*}$ *exists, and* $\mathcal{A}$ *queried either* STATEREVEAL(sid^*) *or* STATICREVEAL$(\overline{\mathsf{sid}^*})$*; or*

3. $\overline{\mathsf{sid}^*}$ *does not exist, and* $\mathcal{A}$ *queried* StateReveal(sid^*)*; or*
4. $\mathcal{A}$ *queried both* StaticReveal($\mathcal{P}_i$) *and* EphemeralReveal(sid^*)*; or*
5. $\overline{\mathsf{sid}^*}$ *exists (the owner of* $\overline{\mathsf{sid}^*}$ *is* $\mathcal{P}_{i'}$*), and* $\mathcal{A}$ *queried both* StaticReveal($\mathcal{P}_{i'}$) *and* EphemeralReveal($\overline{\mathsf{sid}^*}$)*; or*
6. $\overline{\mathsf{sid}^*}$ *does not exist, and* $\mathcal{A}$ *queried* StaticReveal($\mathcal{P}_{i'}$) *for any intended peer* $\mathcal{P}_{i'}$ *of* $\mathcal{P}_i$.

Otherwise, we call the session ***exposed****.*

Initially, $\mathcal{A}$ is given a set of honest users and makes use of its attack queries as described above. Eventually, $\mathcal{A}$ makes the following query.

Test(sid^*): Issues the final test. Once the adversary decides that they have enough data, they query the Test oracle for a challenge. A random bit b is generated; if $b = 1$ then the adversary is given the session key, otherwise they receive a random key from the key space.

Before and after Test(sid^*) is issued, the adversary is allowed to make adaptive queries, issuing oracle queries with the condition that it cannot expose the test session. By the definition of freshness, the Test query requires that sid^* is completed. Eventually, $\mathcal{A}$ guesses a bit b'. We let $\mathsf{Succ}_{\mathcal{A}}(\Pi)$ be the event that $\mathcal{A}$ guesses $b' = b$, i.e. guesses the Test bit b correctly, and define the advantage

$$\mathsf{Adv}(\mathcal{A}) = \left| Pr[\mathsf{Succ}_{\mathcal{A}}(\Pi)] - \frac{1}{2} \right|.$$

In this model, we define (authenticated) GKE security to be the following.

Definition 2.2. *A group key exchange is said to be a* ***secure group key exchange (GKE)*** *in the G-CK$^+$ security model if for any fair PPT adversary* $\mathcal{A}$*,*

1. If two honest parties complete matching sessions, these sessions produce the same session key as output, except with at most negligible probability;
2. $\mathsf{Adv}(\mathcal{A})$ *is negligible in security parameter* 1^λ *for the test session* sid^*

Definition 2.3. *A group key exchange is said to be a* ***secure authenticated group key exchange (AGKE)*** *in the G-CK$^+$ security model if for any manipulating PPT adversary* $\mathcal{A}$*,*

1. If two honest parties complete matching sessions, these sessions produce the same session key as output, except with at most negligible probability;
2. $\mathsf{Adv}(\mathcal{A})$ *is negligible in security parameter* 1^λ *for the test session* sid^*

Note that, for the sake of forward secrecy, the StaticReveal query is considered. If a group key exchange is secure in this model, it is said to have *forward security* or be *forward-secure*, i.e. revealing static keys does not expose previous session keys.

2.1.3. Signatures

In this section, we also provide a definition of a (digital) signature scheme as we will assume the existence of one for our authenticated group key exchange later.

Definition 2.4 ([14]). *A* ***(digital) signature scheme****,* Π_{sign}*, consists of three PPT algorithms* $(\mathtt{Gen}, \mathtt{Sign}, \mathtt{Vrfy})$ *such that:*

1. The ***key-generation algorithm*** $\mathtt{Gen}$ *takes as input a security parameter* 1^λ *and outputs a pair of keys* $(\mathsf{sign}, \mathsf{vrfy})$*, the* ***signing key*** *(or private key) and* ***verification key*** *(or public key), respectively. We assume that* sign *and* vrfy *each has length at least* λ*, and that* λ *can be determined from* sign *or* vrfy*.*
2. The ***signing algorithm*** $\mathtt{Sign}$ *takes as input a signing key* sign *and a message* m *from some message space (that may depend on* vrfy*). It outputs a signature* σ*, and we write this as* $\sigma \leftarrow \mathtt{Sign}_{\mathsf{sign}}(m)$*.*
3. The deterministic ***verification algorithm*** $\mathtt{Vrfy}$ *takes as input a verification key* vrfy*, a message* m*, and a signature* σ*. It outputs a bit* b*, with* $b = 1$ *meaning* ***valid*** *and* $b = 0$ *meaning* ***invalid****. We write this as* $b \leftarrow \mathtt{Vrfy}_{\mathsf{vrfy}}(m, \sigma)$*.*

For correctness, it is required that except with negligible probability over $(\mathsf{sign}, \mathsf{vrfy})$ *output by* $\mathtt{Gen}(1^\lambda)$*, it holds that* $1 \leftarrow \mathtt{Vrfy}_{\mathsf{vrfy}}(m, \mathtt{Sign}_{\mathsf{sign}}(m))$ *for every (legal) message* m*.*

Definition 2.5 (Security of Signatures[14]). *The signature experiment* $\mathsf{Sig-forge}_{\mathcal{A},\Pi_{sign}}(\lambda)$*:*

1. $\mathtt{Gen}(1^\lambda)$ *is run to obtain keys* $(\mathsf{sign}, \mathsf{vrfy})$*.*
2. Adversary $\mathcal{A}$ *is given* vrfy *and access to an oracle* $\mathtt{Sign}_{\mathsf{sign}}(\cdot)$*. The adversary then outputs* (m, σ)*. Let* Q *denote the set of queries that* $\mathcal{A}$ *asked its oracle.*
3. $\mathcal{A}$ ***succeeds*** *if and only if* (1) $1 \leftarrow \mathtt{Vrfy}_{\mathtt{vrfy}}(m, \sigma)$ *and* (2) $m \notin Q$*. In this case, the output of the experiment is defined to be* 1*.*

A signature scheme $\Pi_{sign}(\mathtt{Gen}, \mathtt{Sign}, \mathtt{Vrfy})$ *is* ***existentially unforgeable under an adaptive chosen-message attack****, or just* ***secure****, if for all PPT adversaries* $\mathcal{A}$*, there is a negligible function* $negl$ *such that:*

$$\mathsf{Succ}_{\mathcal{A}}(\Pi_{sign}) = Pr[\mathsf{Sig-forge}_{\mathcal{A},\Pi_{sign}}(\lambda) = 1] \leq negl(\lambda).$$

2.2. R-LWE Key Exchange and Hard Problem

Let $\mathbb{Z}$ be the ring of integers and denote $[N] = \{0, 1, \ldots, N-1\}$. In this paper, we set $R = \mathbb{Z}[X]/(\Phi(X))$ where $\Phi(X) = X^m + 1$ for $m = 2^l$, for some $l \in \mathbb{Z}_+$. We let q be a positive integer defining the quotient ring $R_q = R/qR \cong \mathbb{Z}_q[X]/(\Phi[X])$, where $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$.

Bos, Costello, Naehrig, Stebila [9] give a Diffie-Hellman-like definition of indistinguishability that takes key reconciliation into consideration and show how the R-LWE key exchange reduces to this new problem. They also show that this new problem is hard, if the decisional R-LWE problem is hard.

Definition 2.6 (Decisional R-LWE (D-R-LWE) Problem; [9], Definition 1). *Let m, R, q and R_q be as above. Let χ be a distribution over R_q and let $s \xleftarrow{R} \chi$. Define $O_{\chi,s}$ as an oracle that does the following:*

1. *Sample $a \xleftarrow{R} R_q$ and $e \xleftarrow{R} \chi$,*
2. *Return $(a, as + e) \in R_q \times R_q$.*

*The **decisional R-LWE problem** for m, q, χ is to distinguish $O_{\chi,s}$ from an oracle that returns uniformly random samples from $R_q \times R_q$.*

Note 2.1. The above D-R-LWE problem is given in its *normal* form, i.e. s is chosen from the error distribution as opposed to uniformly at random from R_q. See [15, Lemma 2.24] for a proof that this problem is as hard as choosing s uniformly at random from R_q.

In order to introduce the R-LWE key exchange, which is the chosen basis for our group key exchange, we must first define Peikert's key reconciliation mechanism, which requires some background. Let $\lceil \cdot \rfloor$ denote the rounding function: $\lceil x \rfloor = z$ for $z \in \mathbb{Z}$ and $x \in [z - 1/2, z + 1/2)$.

Definition 2.7 ([9], Definition 2). *Let q be a positive integer. Define the **modular rounding** function*

$$\lceil \cdot \rfloor_{q,2} : \mathbb{Z}_q \to \mathbb{Z}_2, \qquad x \mapsto \lceil x \rfloor_{q,2} = \left\lceil \tfrac{2}{q} x \right\rfloor \mod 2,$$

*and the **cross-rounding** function*

$$\langle \cdot \rangle_{q,2} : \mathbb{Z}_q \to \mathbb{Z}_2, \qquad x \mapsto \langle x \rangle_{q,2} = \left\lfloor \tfrac{4}{q} x \right\rfloor \mod 2.$$

Both functions are extended to elements of R_q coefficient-wise: for $f = f_{m-1}X^{m-1} + \cdots + f_1 X + f_0 \in R_q$, define

$$\lceil f \rfloor_{q,2} = \left(\lceil f_{m-1} \rfloor_{q,2}, \lceil f_{m-2} \rfloor_{q,2}, \ldots, \lceil f_0 \rfloor_{q,2} \right),$$
$$\langle f \rangle_{q,2} = \left(\langle f_{m-1} \rangle_{q,2}, \langle f_{m-2} \rangle_{q,2}, \ldots, \langle f_0 \rangle_{q,2} \right).$$

*We also define the **randomized doubling** function*

$$\mathtt{dbl} : \mathbb{Z}_q \to \mathbb{Z}_{2q}, \qquad x \mapsto \mathtt{dbl}(x) = 2x - e,$$

where e is sampled from $\{-1, 0, 1\}$ with probabilities $p_{-1} = p_1 = \frac{1}{4}$ and $p_0 = \frac{1}{2}$.

The doubling function may be applied to elements in R_q by applying it to each of the coefficients, as done with the rounding functions. Such an application of the doubling function results in a polynomial in R_{2q}. The reason for considering such a doubling function is that it allows for odd q in the key exchange protocol.

The following lemma shows that the rounding of the doubling function on a uniformly random element in $\mathbb{Z}_q$ results in a uniformly random element in $\mathbb{Z}_{2q}$.

Lemma 2.1 ([9], Lemma 1). *For odd q, if $v \in \mathbb{Z}_q$ is uniformly random and $\overline{v} \xleftarrow{R} \mathtt{dbl}(v) \in \mathbb{Z}_{2q}$, then, given $\langle \overline{v} \rangle_{2q,2}$, $\lceil \overline{v} \rfloor_{2q,2}$ is uniformly random.*

We may now define Peikert's reconciliation function, $\mathtt{rec}(\cdot)$, which recovers $\lceil v \rfloor_{q,2}$ from an element $w \in \mathbb{Z}_q$ that is "close" to the original $v \in \mathbb{Z}_q$, given only w and the cross-rounding of v.

Definition 2.8. *Define sets $I_0 = \{0, 1, \ldots, \lceil \frac{q}{2} \rfloor - 1\}$ and $I_1 = \{-\lceil \frac{q}{2} \rfloor, \ldots, -1\}$. Let $E = [-\frac{q}{4}, \frac{q}{4})$, then*

$$\mathtt{rec}: \quad \mathbb{Z}_{2q} \times \mathbb{Z}_2 \to \mathbb{Z}_2, \qquad (w, b) \mapsto \begin{cases} 0, & \text{if } w \in I_b + E \mod 2q, \\ 1, & \text{otherwise}. \end{cases}$$

Reconciliation of a polynomial in R_q is done coefficient-wise so the following lemma allows us to reconcile two polynomials in R_q that are close to each other, by considering the coefficients of the two polynomials.

Lemma 2.2 ([9], Lemma 2). *For odd q, let $v = w + e \in \mathbb{Z}_q$ for $w, e \in \mathbb{Z}_q$ such that $2e \pm 1 \in E \pmod q$. Let $\overline{v} = \mathtt{dbl}(v)$, then $\mathtt{rec}(2w, \langle \overline{v} \rangle_{2q,2}) = \lceil \overline{v} \rfloor_{2q,2}$.*

Finally, we define the R-LWE key exchange.

Protocol 2.2 (R-LWE Key Exchange). Let m, R, q, R_q and χ be as in the D-R-LWE problem (Definition 2.6). Given R_q, $\mathtt{ParaGen}$ outputs a uniformly random $a \xleftarrow{R} R_q$. Parties $\mathcal{P}_0$ and $\mathcal{P}_1$ generate a two-party key exchange protocol Π as follows:

Setup: For input R_q, $\mathtt{ParaGen}$ outputs to each party $\mathcal{P}_i$ the public parameter $a \xleftarrow{R} \mathtt{ParaGen}(R_q)$.

Publish$_1$: Each party $\mathcal{P}_i$ chooses $s_i, e_i \xleftarrow{R} \chi$ as their secret key and error key, respectively, computes their public key $b_i = as_i + e_i \in R_q$, and sends their public key b_i to party $\mathcal{P}_{1-i}$.

Publish$_2$: Party $\mathcal{P}_1$, upon receiving b_0 from $\mathcal{P}_0$, chooses a new error key $e_1' \xleftarrow{R} \chi$, computes $v = b_0 s_1 + e_1' \in R_q$, and uses the randomized doubling function on v to receive $\overline{v} \xleftarrow{R} \mathtt{dbl}(v) \in R_{2q}$. Using the cross-rounding function, $\mathcal{P}_1$ computes $c = \langle \overline{v} \rangle_{2q,2} \in \{0,1\}^m$ and sends c to $\mathcal{P}_0$.

KeyGen: In order to generate the shared key, party $\mathcal{P}_0$ uses the reconciliation function to output $k_{1,0} \leftarrow \mathtt{rec}(2b_1 s_0, c) \in \{0,1\}^m$. Party $\mathcal{P}_1$ simply computes $k_{0,1} = \lceil \overline{v} \rfloor_{2q,2} \in \{0,1\}^m$.

Except with negligible probability $k_{0,1} = k_{1,0} = \kappa$, i.e. this protocol satisfies correctness.

The security of the above protocol reduces to a decisional hardness problem that Bos, Costello, Naehrig, Stebila [9] dub the decision Diffie-Hellman-like (DDH-like) problem. We give a reformulation of the DDH-like problem definition from [9, Definition 3] for ease of proof later, although the two are equivalent.

Definition 2.9 (Decision Diffie-Hellman-like (DDH–like) Problem). *Let m, R, q, R_q, χ be D-R-LWE parameters. Given a tuple sampled with probability $1/2$ from one of the following two distributions:*

- (a, b_0, b_1, c, κ), *where* $a \xleftarrow{R} R_q$, $s_0, s_1, e_0, e_1, e_1' \xleftarrow{R} \chi$, $b_i = as_i + e_i \in R_q$ *for* $i = 0, 1$, $v = b_0 s_1 + e_1'$, $\overline{v} \xleftarrow{R} \mathtt{dbl}(v)$, $c = \langle \overline{v} \rangle_{2q,2}$, *and* $k = \lceil \overline{v} \rfloor_{2q,2} \in \{0,1\}^m$,
- (a, b_0, b_1, c, κ), *where* a, b_0, b_1, c *are as above and* $\kappa \xleftarrow{R} \{0,1\}^m$,

determine from which distribution the tuple is sampled.

THEOREM 2.3 (HARDNESS OF DDH-LIKE PROBLEM; [9], THEOREM 1). *Let m be a parameter, q an odd integer, and χ a distribution on R_q. If the D-R-LWE problem for m, R, q, R_q, χ is hard, then the DDH-like problem for m, R, q, R_q, χ is also hard.*

3. TREE-BASED R-LWE GROUP KEY EXCHANGE

From a set of parties, $\mathfrak{P}$, we consider a subset of $n \geq 2$ parties $\{\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_{n-1}\}$, re-indexing if need be, that generate a shared key. The GKE arranges the parties in a *double tree*, i.e. two trees connected at the root. We assume the parties are arranged in ascending order from the top-leftmost root, going right, and continuing down the tree level-wise, starting with $\mathcal{P}_0$ and $\mathcal{P}_1$ at the roots (see Figure 1). Parties are assumed to only appear once in the tree.

Excepting the leaves of the tree, each party $\mathcal{P}_i$ has a parent $\mathsf{par}(i)$, and a set of children $j.\mathsf{cld}(i)$ for $j = 1, 2, \ldots, l_i$ where $0 \leq l_i \leq n-2$ is the amount of children of $\mathcal{P}_i$, which are all considered the *neighbours* of $\mathcal{P}_i$ (see Figure 2). For all $\mathcal{P}_i$ that are leaves we have $l_i = 0$, i.e. the lowest rungs of each branch have no children. The set $\mathsf{ancestors}(i)$ is the set of indexes of all ancestors of a party $\mathcal{P}_i$, including i but having removed 0 and 1. Parties $\mathcal{P}_0$ and $\mathcal{P}_1$ are parents of each other.

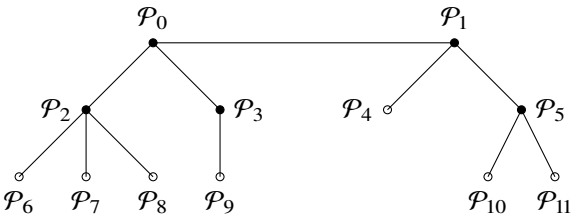

FIGURE 1: Possible double tree graph configuration for $n = 12$

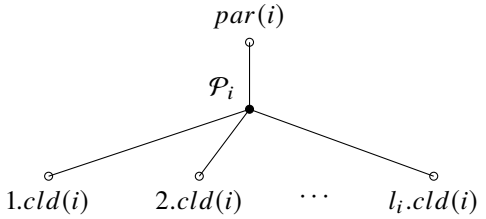

FIGURE 2: The neighbours of $\mathcal{P}_i$: parent and children.

In the following, *multicast* means that a party only sends a message to a discrete subset of all potential parties.

For security reasons to be discussed in Note 3.2, we distinguish a set of *root parties* during the protocol. For use in our AGKE compiler in Sect. 4, we add the *session name* to the GKE protocol parameter generation. The session name is a unique session name.

PROTOCOL 3.1 (TREE-BASED R-LWE GROUP KEY EXCHANGE). The **tree-based R-LWE Group Key Exchange (tree-R-LWE-GKE)** protocol for n parties, Π_n, takes as input parameters m, R, q, R_q and χ, as in the DDH-like problem (Definition 2.9), and outputs a session key $K \in \{0,1\}^m$.

The *parameter generator* algorithm, `ParaGen`, takes as input the parameter R_q and the number of parties, n. The algorithm outputs a tuple consisting of a uniformly random $a \xleftarrow{R} R_q$, a double tree for the n parties, Γ, and a unique session name, sN.

The parties $\mathcal{P}_i$ for $i = 0, 1, \ldots, n-1$ generate a group key exchange protocol Π_n as follows:

Setup: For the input R_q and the number of parties, n, the algorithm outputs to each party $\mathcal{P}_i$ the tuple:

$$\mathsf{params} := (a, \Gamma, sN) \leftarrow \mathtt{ParaGen}(R_q, n),$$

where sN is the unique session name. Set the root parties to be $\mathfrak{R} := \{\mathcal{P}_0, \mathcal{P}_1\}$.

Publish$_1$: Given params, each $\mathcal{P}_i$ chooses random secret keys $s_i, e_i, e_i' \xleftarrow{R} \chi$ and computes a public key $b_i = as_i + e_i$. $\mathcal{P}_i$ then multicasts its public key to its neighbours (parent and l_i children).

Publish$_{2a}$: Upon receiving the public key $b_{\mathsf{par}(i)}$ from its parent, $\mathcal{P}_{\mathsf{par}(i)}$, $\mathcal{P}_i$ generates the value $v_i = b_{\mathsf{par}(i)} s_i + e_i'$. Using the randomized doubling function on this value, $\mathcal{P}_i$ finds $\overline{v}_i \xleftarrow{R} \mathtt{dbl}(v_i) \in R_{2q}$. Using the cross-rounding function, $\mathcal{P}_i$ then computes

$$c_i = \langle \overline{v}_i \rangle_{2q,2} \in \{0,1\}^m,$$

the reconciliation key for its parent, which $\mathcal{P}_i$ sends to said parent, $\mathcal{P}_{\mathsf{par}(i)}$.

We assume, without loss of generality, that $\mathcal{P}_1$ generates c_1 and sends it to $\mathcal{P}_0$, while $\mathcal{P}_0$ generates no reconciliation key c_0.

Publish$_{2b}$: Upon receiving the respective reconciliation keys $c_{j.\mathsf{cld}(i)}$ from its l_i children, and also using the value $\overline{v}_i$, $\mathcal{P}_i$ computes the shared keys $k_{\mathsf{par}(i),i}$ and $k_{j.\mathsf{cld}(i),i}$ for each $j \in \{1, \ldots, l_i\}$:

$$k_{\mathsf{par}(i),i} = \lceil \overline{v}_i \rfloor_{2q,2} \in \{0,1\}^m,$$
$$k_{j.\mathsf{cld}(i),i} \leftarrow \mathtt{rec}(2b_{j.\mathsf{cld}(i)} s_i, c_{j.\mathsf{cld}(i)}) \in \{0,1\}^m,$$

for $j \in \{1, \ldots, l_i\}$, where $\lceil \cdot \rfloor$ is the modular rounding function and `rec` is the reconciliation function.

Again, without loss of generality, $\mathcal{P}_1$ sets $k_{0,1} = \lceil \overline{v_0} \rfloor_{2q,2} \in \{0,1\}^m$ while $\mathcal{P}_0$ computes $k_{1,0} \leftarrow \mathtt{rec}(2b_1 s_0, c_1) \in \{0,1\}^m$.

Publish$_3$: Each $\mathcal{P}_i$ with children (this excepts the leaves of Γ) computes

$$x_{j.\mathsf{cld}(i)} = k_{\mathsf{par}(i),i} \oplus k_{j.\mathsf{cld}(i),i},$$

and multicasts this value to its respective descendants, for each $j \in \{1, \ldots, l_i\}$.

KeyGen: Each $\mathcal{P}_i$ computes the session key

$$K_i = k_{\mathsf{par}(i),i} \oplus \bigoplus_{h \in \mathsf{ancestors}(i)} x_h = K.$$

Proposition 3.1 (Correctness). *Except with negligible probability, each party in the tree-based Ring-LWE group key exchange protocol (Protocol 3.1) computes the same session key $K = k_{0,1}$.*

Proof. This can be seen by induction. By key reconciliation, except with negligible probability, $K_0 = K_1 = k_{0,1}$. Assume that $K_{\mathsf{par}(i)} = K$ then, as

$$K_{\mathsf{par}(i)} = k_{\mathsf{par}(\mathsf{par}(i)),\mathsf{par}(i)} \oplus \bigoplus_{h \in \mathsf{ancestors}(\mathsf{par}(i))} x_h,$$

except with negligible probability, we have that, except with negligible probability,

$$\begin{aligned} K_i &= k_{\mathsf{par}(i),i} \oplus \bigoplus_{h \in \mathsf{ancestors}(i)} x_h \\ &= k_{\mathsf{par}(i),i} \oplus \left(k_{\mathsf{par}(\mathsf{par}(i)),\mathsf{par}(i)} \oplus k_{i,\mathsf{par}(i)}\right) \\ &\qquad \oplus \bigoplus_{h \in \mathsf{ancestors}(\mathsf{par}(i))} x_h \\ &= K_{\mathsf{par}(i)} = K. \end{aligned}$$

□

The session key of the group is simply the shared key of the root parties $\mathcal{P}_0$ and $\mathcal{P}_1$.

Note 3.2. In order to prove that our tree-R-LWE-GKE protocol is secure, we show a reduction to the DDH-like problem from Definition 2.9. However, the session key is equal to the (secret) shared key between $\mathcal{P}_0$ and $\mathcal{P}_1$ so the indistinguishability of the session key depends on the indistinguishability of the secret key between $\mathcal{P}_0$ and $\mathcal{P}_1$. With the state reveal and ephemeral reveal queries simulating any attack that leaks the ephemeral keys, we are forced to assume that such queries are never asked of either $\mathcal{P}_0$ or $\mathcal{P}_1$ (the root parties; see Setup description in Protocol 3.1) as their shared key could then be easily derived and the indistinguishability trivially broken. We therefore assume that neither the static secret keys of the root parties in all sessions nor their ephemeral secret keys in the test session, are revealed by any attack query by the adversary. This additional assumption still allows for strong corruptions of ephemeral secrets such that it remains a strong model, stronger than the security models given by Bresson et al [16] and Katz and Yung [11] that do not address (strong) corruptions of ephemeral secrets, however, it is slightly weaker than both the MSU model (also called the G-eCK$^+$ model), the Bresson-Manulis model [17], and the model which we consider in this paper, G-CK$^+$.

In our extra assumption, we also consider key reuse attacks such as the signal leakage attack of [18] and the attack of [19]. However, the signal leakage attack [18] requires $2q$ queries (where q is from R_q), so we could simply require that there be less than $2q-1$ children per node. The attack in [19] is independent of q but only succeeds on a shared key mismatch, which would make the GKE and AGKE protocols fail. As keys are newly generated in each instance of the GKE and AGKE protocols, and as there is forward secrecy (this will be shown), both attacks fail to break our protocols. Furthermore, all the key reuse attacks we looked at required a manipulating adversary who could initiate as many key exchanges as needed, but in our GKE we assume a fair adversary, and in both our GKEs and AGKEs, the number of two-party key exchanges per party per session is limited by the number of children per node, eliminating such attacks.

This addition would also be needed to prove the security of the original Desmedt-Lange-Burmester [12] GKE and authenticated GKE compiler in our security model. This comes from the fact that they do not consider security against active insiders, which the current model does (and more).

Theorem 3.3. *Under the assumption that the DDH-like problem (Definition 2.9) is hard and that the root parties are secure from EphemeralReveal queries,[5] the tree-R-LWE-GKE protocol given in Protocol 3.1 is a secure GKE (with forward security) in the G-CK$^+$ security model.*

Proof. We must show that the protocol in Protocol 3.1 satisfies the security notion given in Definition 2.2. The first requirement is satisfied by the correctness shown in Proposition 3.1.

For the second requirement, assume that there exists a (not necessarily classical) polynomial-time adversary $\mathcal{A}$, allowed polynomially-many classical queries, with non-negligible advantage $\mathsf{Adv}(\mathcal{A}) = \varepsilon$ (see Definition 2.2 for this notation). We build a polynomial-time distinguisher $\mathcal{D}$, allowed polynomially-many classical queries, for the DDH-like problem in Algorithm 1.

As an analysis of our distinguishing algorithm, we note the following.

For every session, except the ℓ-th, $\mathcal{D}$ simulates the tree-R-LWE-GKE protocol to $\mathcal{A}$, choosing new random secret keys for each party in each session and simulating all communication through $\mathcal{A}$. As all randomness is generated anew for each session and there are no long-term keys, all sessions are independently generated. Hence, any attack on any other session does not reveal anything about the ℓ-th session except through repetition of secret keys, which happens with negligible probability.

For the ℓ-th session, using the public information for $\mathcal{P}_0$, namely b_0, $\mathcal{D}$ simulates R-LWE key exchange (Definition 2.2) with the secret keys $s_{j.\mathsf{cld}(0)}, e_{j.\mathsf{cld}(0)}, e'_{j.\mathsf{cld}(0)}$

[5] There are no static keys in our GKE, hence no need to assume that the StaticReveal queries are not issued to the root parties. StateReveal queries in the test case are excluded by the freshness definition.

Algorithm 1 DDH-like distinguisher, $\mathcal{D}$.

Input: $(m, R, q, R_q, \chi, a, b_0, b_1, c, \kappa)$ as in the DDH-like problem.

1: $\ell \xleftarrow{R} \{1, \ldots, \Lambda\}$, where Λ is an upper bound on the number of sessions activated by $\mathcal{A}$ in any interaction.
2: Invoke $\mathcal{A}$ and simulate protocol to $\mathcal{A}$, except for the ℓ-th activated protocol session.
3: For the ℓ-th session:
4: Set $\texttt{params} := (a, \Gamma, \mathfrak{s})$, where Γ is an n-party double tree and $\mathfrak{s}$ is the session name.
5: Set $b'_0 = b_0, b'_1 = b_1$ and $c_1 = c$. Choose $(s_i, e_i, e'_i) \xleftarrow{R} \chi^3$ for $i = 2, \ldots, n-1$ and set $b'_i = as_i + e_i$. Set $v_i = b_{\mathsf{par}(i)} s_i + e'_i$, generate $\overline{v}_i \xleftarrow{R} \texttt{dbl}(v_i) \in R_{2q}$ and compute $c_i = \langle \overline{v}_i \rangle_{2q,2} \in \{0,1\}^m$. Simulate multicasting for each $\mathcal{P}_i$ along with identifying information $(\mathcal{P}_i, \mathfrak{s})$.
6: Set

$$x'_{j.\mathsf{cld}(0)} := \kappa \oplus k_{0,j.\mathsf{cld}(0)}, \quad \forall j \in \{1, 2, \ldots, l_0\},$$
$$x'_{j.\mathsf{cld}(1)} := \kappa \oplus k_{1,j.\mathsf{cld}(1)}, \quad \forall j \in \{1, 2, \ldots, l_1\},$$
$$x'_{j.\mathsf{cld}(i)} := k_{\mathsf{par}(i),i} \oplus k_{j.\mathsf{cld}(i),i}, \ \forall j \in \{1, 2, \ldots, l_i\},$$

for $i \geq 2$ where $\mathcal{P}_i$ is not a leaf in Γ.
7: **if** the ℓ-th session is chosen by $\mathcal{A}$ as the test session **then**
8: Provide $\mathcal{A}$ as the answer to the test query,
9: $d \leftarrow \mathcal{A}$'s output
10: **else**
11: $d \xleftarrow{R} \{0,1\}$.

Output: d

of party $\mathcal{P}_{j.\mathsf{cld}(0)}$, obtaining the shared key $k_{0,j.\mathsf{cld}(0)} = k_{j.\mathsf{cld}(0),0}$, except with negligible probability. Likewise, using the public information for $\mathcal{P}_1$, namely b_1, $\mathcal{D}$ simulates R-LWE key exchange with the secret keys $s_{j.\mathsf{cld}(1)}, e_{j.\mathsf{cld}(1)}, e'_{j.\mathsf{cld}(1)}$ of party $\mathcal{P}_{1.\mathsf{cld}(1)}$, obtaining $k_{1,j.\mathsf{cld}(1)} = k_{j.\mathsf{cld}(1),1}$, except with negligible probability. All other shared keys may be computed in polynomial-time as the secret keys for $\mathcal{P}_i$ are known for $i = 2, \ldots, n-1$.

As the s_i, e_i, e'_i are chosen uniformly at random for $i \geq 2$, the distribution of the $b'_i, x'_{j.\mathsf{cld}(i)}$ in Algorithm 1 are identical to that in a tree-R-LWE-GKE instance.

The transcript given to $\mathcal{A}$ by $\mathcal{D}$ is

$$(b'_0, \ldots, b'_{n-1}, x'_{1.\mathsf{cld}(0)}, x'_{2.\mathsf{cld}(0)}, \ldots, \ldots, x'_{l_0.\mathsf{cld}(0)}, x'_{1.\mathsf{cld}(1)}, \ldots, x'_{(l_{n-1}).\mathsf{cld}(n-1)}),$$

where we assign a blank value for the x' value when there is no child.

In each session, all attack queries are answered fully and honestly, except in the ℓ-th session, where state reveal and ephemeral reveal attack queries to the root parties force the distinguisher to end the experiment and output $d \xleftarrow{R} \{0,1\}$. As the ℓ-th session is chosen as the test session with probability $1/\Lambda$ and per our assumption the ephemeral reveal query is not queried to the root parties in the test case, the experiment cannot end this way every time.

If the ℓ-th session is the test session and κ is a valid tree-R-LWE-GKE session key, then $\kappa = k_{0,1}$, i.e. (a, b_0, b_1, c, κ) is indeed a valid DDH-like tuple, where $\kappa = \lceil \overline{v}_0 \rfloor_{2q,2}$.

If the ℓ-th session is *not* the test session, then $\mathcal{D}$ outputs a random bit, i.e. it has advantage 0. However, if the test session *is* the ℓ-th session, which happens with probability $1/\Lambda$, then $\mathcal{A}$ will succeed with advantage ε. Hence, the final advantage of the DDH-like distinguisher $\mathcal{D}$ is ε/Λ, which is non-negligible. □

COROLLARY 3.1. *Assuming the DDH-like problem is post-quantum hard and that the root parties are secure from* EPHEMERALREVEAL *queries, tree-R-LWE-GKE is a post-quantum secure group key exchange with forward security.*

We now give a peer-to-peer (sequential) version of our tree-R-LWE-GKE, calling it the **P2P-tree-R-LWE-GKE** protocol. As it is sequential, relying on a party to generate the session key before sending a final message to its children, the number of rounds is bounded by the length of the double tree, while the communication and memory complexity become constant. The protocol differs from the tree-R-LWE-GKE protocol after the $Publish_{2b}$ step. It achieves the same level of security as the tree-R-LWE-GKE protocol through an analogous argument, which we therefore omit, and summarize this in the following theorem.

PROTOCOL 3.4 (PEER-TO-PEER TREE-BASED R-LWE GKE). The **peer-to-peer R-LWE group key exchange (P2P-tree-R-LWE-GKE)** protocol for n parties, Π_n, takes as input parameters m, R, q, R_q and χ, as in the DDH-like problem (Definition 2.9), and outputs a session key $K \in \{0,1\}^m$.

The **parameter generator** algorithm, `ParaGen`, takes as input the parameter R_q and the number of parties, n. The algorithm outputs a tuple consisting of a uniformly random $a \xleftarrow{R} R_q$, a double tree for the n parties, Γ, and a unique session name, sN.

The parties $\mathcal{P}_i$ for $i = 0, 1, \ldots, n-1$ generate a group key exchange protocol Π_n as follows:

Setup: For the input R_q and the number of parties, n, the algorithm outputs to each party $\mathcal{P}_i$ the tuple:

$$\mathsf{params} := (a, \Gamma, sN) \leftarrow \texttt{ParaGen}(R_q, n),$$

where sN is the unique session name. Set the root parties to be $\mathfrak{R} := \{\mathcal{P}_0, \mathcal{P}_1\}$.

Publish$_1$: Given params, each $\mathcal{P}_i$ chooses random secret keys $s_i, e_i, e'_i \xleftarrow{R} \chi$ and computes a public key $b_i = as_i + e_i$. $\mathcal{P}_i$ then multicasts its public key to its neighbours (parent and l_i children).

Publish$_{2a}$: Upon receiving the public key $b_{\mathsf{par}(i)}$ from its parent, $\mathcal{P}_{\mathsf{par}(i)}$, $\mathcal{P}_i$ generates the value $v_i = b_{\mathsf{par}(i)} s_i + e'_i$. Using the randomized doubling function on this value, $\mathcal{P}_i$ finds $\overline{v}_i \xleftarrow{R} \texttt{dbl}(v_i) \in R_{2q}$. Using the cross-rounding function, $\mathcal{P}_i$ then computes $c_i = \langle \overline{v}_i \rangle_{2q,2} \in \{0,1\}^m$, the reconciliation key for its parent, which $\mathcal{P}_i$ sends to said parent, $\mathcal{P}_{\mathsf{par}(i)}$.

We assume, without loss of generality, that $\mathcal{P}_1$ generates c_1 and sends it to $\mathcal{P}_0$, while $\mathcal{P}_0$ generates no reconciliation key c_0.

Publish$_{2b}$: Upon receiving the respective reconciliation keys $c_{\mathsf{j.cld}(i)}$ from its l_i children, and also using the value $\overline{v}_i$, $\mathcal{P}_i$ computes the shared keys $k_{\mathsf{par}(i),i}$ and $k_{j.\mathsf{cld}(i),i}$ for each $j \in \{1,\ldots,l_i\}$:

$$k_{\mathsf{par}(i),i} = \lceil \overline{v}_i \rfloor_{2q,2} \in \{0,1\}^m,$$
$$k_{j.\mathsf{cld}(i),i} \leftarrow \mathtt{rec}(2b_{j.\mathsf{cld}(i)}s_i, c_{j.\mathsf{cld}(i)}) \in \{0,1\}^m,$$

for $j \in \{1,\ldots,l_i\}$, where $\lceil\cdot\rfloor$ is the modular rounding function and $\mathtt{rec}$ is the reconciliation function.

Again, without loss of generality, $\mathcal{P}_1$ sets $k_{0,1} = \lceil \overline{v_0} \rfloor_{2q,2} \in \{0,1\}^m$ while $\mathcal{P}_0$ computes $k_{1,0} \leftarrow \mathtt{rec}(2b_1 s_0, c_1) \in \{0,1\}^m$.

Publish$_{3a}$: Parties $\mathcal{P}_0$ and $\mathcal{P}_1$ have already computed the same session key $K = k_{1,0} = k_{0,1}$ (except with negligible probability) and send $x_{j.\mathsf{cld}(0)} = K \oplus k_{j.\mathsf{cld}(0),0}$, respectively $x_{j.\mathsf{cld}(1)} = K \oplus k_{j.\mathsf{cld}(1),1}$, to their respective children, for $j \in \{1,\ldots,l_0\}$, respectively $j \in \{1,\ldots,l_1\}$.

KeyGen and Publish$_{3b}$: Upon receiving $x_{\mathsf{par}(i)}$, $\mathcal{P}_i$ computes the session key

$$K_i = x_{\mathsf{par}(i)} \oplus k_{\mathsf{par}(i),i}.$$

Every party $\mathcal{P}_i$ with children (this excepts the leaves of Γ), then computes $x_{j.\mathsf{cld}(i)} = K_i \oplus k_{j.\mathsf{cld}(i),i}$ and multicasts this to its j-th child, for each $j \in \{1,\ldots,l_i\}$.

Theorem 3.5. *Assuming the DDH-like problem is post-quantum hard and the root parties are secure from EphemeralReveal queries, P2P-tree-R-LWE-GKE (Protocol 3.4) is a post-quantum secure group key exchange with forward security.*

4. NEW AUTHENTICATED GKE BASED ON R-LWE

As is known, it is possible to combine a constant-round GKE and a signature scheme to create a constant-round AGKE using a Katz-Yung compiler [11]. Such an AGKE would have computational complexity $O(n)$ however, as each party needs to verify $O(n)$ signatures. Instead, we propose an adjusted version of the Katz-Yung compiler with computational complexity equivalent to the underlying GKE, broadly based on one given in Desmedt, Lange, and Burmester [12]. The main reason for this complexity reduction is that our compiler only requires receiving signatures from the parties it interacts with, as opposed to all parties involved in the group key exchange.

In our compiler, each party $\mathcal{P}_i$ generates static signing and verification keys. For each instantiation of the underlying GKE protocol, each party then generates session-specific randomness in the form of a random nonce and uses this nonce, along with the party ID, session name, and message counter, for communication.

In Tree-R-LWE-GKE, assuming a balanced tree, each user only needs to exchange a logarithmic number of messages in order to compute the session key, which the compiler below preserves.

Protocol 4.1 (**Authenticated Tree-R-LWE-GKE**). We assume there exists a set of all parties $\mathfrak{P}$. Consider the following protocol, which uses a secure signature scheme, Π_{sign}, and the forward-secure Tree-R-LWE-GKE, Π_{tree}, where the latter is assumed to be instantiated on the parameters m, R, q, R_q, and χ as in the D-R-LWE problem (Problem 2.6). We consider the corresponding parameter generating algorithms: $\mathtt{Gen}_{sign}$ and $\mathtt{Gen}_{tree}$. $\mathtt{Gen}_{sign}$ takes as input a security parameter λ_{sign} and outputs parameters required for secure signing. $\mathtt{Gen}_{tree}$ takes as input R_q and a number of participants, n, and outputs the public parameters for Tree-R-LWE-GKE.

Setup$_{\mathbf{sign}}$: For a security parameter λ_{sign}, the algorithm outputs to all of the parties in $\mathfrak{P}$ the parameters for the chosen signature scheme:

$$\mathsf{params}_{sign} \xleftarrow{R} \mathsf{Gen}_{sign}(1^{\lambda_{sign}}).$$

Key$_{\mathbf{sign}}$: Given params_{sign}, each party $\mathcal{P}_i \in \mathfrak{P}$ generates the signing/verification keys $(\mathsf{sign}_i, \mathsf{vrfy}_i)$. These are static (long-term) keys.

Setup$_{\mathbf{\Pi}}$: Let $\mathfrak{P}_n = \{\mathcal{P}_0,\ldots,\mathcal{P}_{n-1}\} \subset \mathfrak{P}$ be a set of n parties wishing to do a GKE (re-indexing if need be) and let gid be their group identifier (of size $O(\log_k n)$). Each $\mathcal{P}_i \in \mathfrak{P}_n$ chooses an ephemeral random nonce $\eta_i \in \{0,1\}^{\lambda_{sign}}$ for the session.

ParaGen: For the parameter R_q and the number of parties, n, the algorithm outputs to all the parties in $\mathfrak{P}_n$ the Π_{tree} parameters:

$$(a, \Gamma, sN) \xleftarrow{R} \mathsf{Gen}_{tree}(R_q, n),$$

the tuple of the public value a, the double tree Γ, and the unique session name sN, as prescribed by the tree-R-LWE-GKE protocol.

Set the root parties to be $\mathfrak{R} := \{\mathcal{P}_0, \mathcal{P}_1\}$.

Ini$_{\mathbf{sign}}$: Let $\mathsf{rel}_i^{\mathsf{sid}} = \{\mathcal{P}_{1'}, \mathcal{P}_{2'}, \ldots, \mathcal{P}_{t_i'}\}$ denote the set of all ancestors, party $\mathcal{P}_0$ or $\mathcal{P}_1$ (depending on which side of the double-tree $\mathcal{P}_i$ is on), and the l_i children of $\mathcal{P}_i$ in session sid. The size of this set depends, in particular, on the amount of ancestors. Each $\mathcal{P}_i$ computes $\sigma \leftarrow \mathtt{Sign}_{\mathsf{sign}_i}(\mathcal{P}_i|sN|0|\eta_i)$ and multicasts $\mathcal{P}_i|sN|0|\eta_i|\sigma$ to its parent and all its descendants. After positive verification, each party $\mathcal{P}_i$ stores the session specific information as $\mathsf{info}_i^{\mathsf{sid}} = (\mathsf{gid}|sN|\mathcal{P}_{1'}|\eta_{1'}|\ldots|\mathcal{P}_{t_i'}|\eta_{t_i'})$, as a part of the state information.

GKE: The tree-R-LWE-GKE protocol, Π_{Tree}, is executed with the following changes:

- Whenever party $\mathcal{P}_i$ is supposed to multicast a message m as part of the protocol, the party computes $\sigma \leftarrow \mathtt{Sign}_{\mathsf{sign}_i}(\mathcal{P}_i|sN|j|m|\eta_i)$, where j is the message number, and multicasts the string $\mathcal{P}_i|sN|j|m|\sigma$.[6]
- Upon receiving $\mathcal{P}_*|sN|j|m|\sigma$, party $\mathcal{P}_i$ checks that:
  1. $\mathcal{P}_* \in \mathsf{rel}_i$,
  2. j is the next expected sequence number for messages from $\mathcal{P}_*$,
  3. $1 \leftarrow \mathtt{Vrfy}_{\mathsf{vrfy}_*}(\mathcal{P}_*|sN|j|m|\eta_*, \sigma)$.

  If any of these are untrue, the session is aborted, i.e. $\mathcal{P}_i$ does not complete the session, wiping its state. Otherwise, $\mathcal{P}_i$ continues as it would in Π_{tree} and uses m.

KeyGen: Each non-aborted session computes the session key as in Π_{tree}.

THEOREM 4.2. *Assuming that the signature scheme Π_{sign} is (post-quantum) secure, the DDH-like problem is (post-quantum) hard, and the root parties are secure from* STATICREVEAL *and* EPHEMERALREVEAL *queries (as in Note 3.2), the authenticated tree-R-LWE-GKE given in Protocol 4.1 is a (post-quantum) secure authenticated group key exchange in the G-CK$^+$ security model (Definition 2.3), with forward security.*

Proof. As a message is utilized in a session as it would in Π_{tree} if the message is verified, by the correctness of tree-R-LWE-GKE, and as we assume that Π_{sign} is secure (and thereby has correctness), the first requirement in Definition 2.3 is satisfied by Protocol 4.1.

For the second requirement, assume that there exists a (not necessarily classical) polynomial-time adversary $\mathcal{A}$ with non-negligible advantage $\mathsf{Adv}(\mathcal{A}) = \varepsilon$. We use the adversary $\mathcal{A}$ to construct a distinguisher $\mathcal{D}$ for the DDH-like problem, which we give in Algorithm 2.

There are three ways $\mathcal{D}$ may succeed: By forging a signature for a party in the TEST case, by reusing a signature, or by distinguishing a correct session key from random. The success probability of the first is bounded by the advantage of an adversary to forge a signature, i.e. negligible. The second is bounded by a combinatorial consideration of the amount of sent messages and protocol executions compared to the space of random nonces. The third is bounded by the advantage of an adversary to distinguish between a Tree-R-LWE-GKE session key and a random key in the same space, which by Theorem 3.3 is negligible.

DDH-like distinguisher $\mathcal{D}$ generates correctly distributed signing/verification keys and random nonces to be used in sessions, and using them, honestly simulates all communication requests in the sessions (recall that the adversary $\mathcal{A}$ is in charge of all communications in our model).

[6]At most this requires one re-signing of a message when a party is required to send the tree-R-LWE-GKE protocol x value to a child (with which it has already exchanged more than one previous message) and its other descendants (with which it has sent at most one previous message).

Algorithm 2 DDH-like distinguisher, $\mathcal{D}$.

Input: (a, b_0, b_1, c, κ)

1: $\ell \xleftarrow{R} \{1, \ldots, \Lambda\}$, where Λ is an upper bound on the number of sessions activated by $\mathcal{A}$ in any interaction.
2: Invoke $\mathcal{A}$ and simulate protocol to $\mathcal{A}$:
3: Generate static signature/verification keys for each party $\mathcal{P}_i \in \mathfrak{P}$.
4: For every session except the ℓ-th: Generate ephemeral random nonces and simulate group key exchange protocol Π_{tree} as prescribed by Protocol 4.1.
5: For the ℓ-th session, using signature and verification keys and random nonces as prescribed by Protocol 4.1 for message sending/multicasting:
6: Set $\mathtt{params}_{tree} = (a, \Gamma, \mathfrak{s})$, where Γ is an n-party binary graph and $\mathfrak{s}$ is the session name.
7: Set $b'_0 = b_0, b'_1 = b_1$ and $c_1 = c$. Choose $(s_i, e_i, e'_i) \xleftarrow{R} \chi^3$ for $i = 2, \ldots, n-1$ and set $b'_i = as_i + e_i$. Set $v_i = b_{\mathsf{par}(i)}s_i + e'_i$, generate $\bar{v}_i \xleftarrow{R} \mathtt{dbl}(v_i) \in R_{2q}$ and compute $c_i = \langle \bar{v}_i \rangle_{2q,2} \in \{0,1\}^m$.
8: Set

$$x'_{j.\mathsf{cld}(0)} := \kappa \oplus k_{0,j.\mathsf{cld}(0)}, \quad \forall j \in \{1, 2, \ldots, l_0\},$$
$$x'_{j.\mathsf{cld}(1)} := \kappa \oplus k_{1,j.\mathsf{cld}(1)}, \quad \forall j \in \{1, 2, \ldots, l_1\},$$
$$x'_{j.\mathsf{cld}(i)} := k_{\mathsf{par}(i),i} \oplus k_{j.\mathsf{cld}(i),i}, \ \forall j \in \{1, 2, \ldots, l_i\},$$

for $i \geq 2$ where $\mathcal{P}_i$ is not a leaf in Γ.
9: **if** the ℓ-th session is chosen by $\mathcal{A}$ as the test session **then**
10: Provide $\mathcal{A}$ as the answer to the distinguishing query,
11: $d \leftarrow \mathcal{A}$'s output
12: **else**
13: $d \xleftarrow{R} \{0,1\}$.

Output: d

For any session $\neq \ell$, $\mathcal{D}$ may answer any query by $\mathcal{A}$ entirely.

For the ℓ-th session, the distributions are as in the tree-R-LWE-GKE protocol, giving us the session key distributed as in the DDH-like problem. We therefore wish to show that the forging and repeating attacks succeed with negligible probability such that any distinguishing attack must occur on the key exchange part with non-negligible probability, i.e. we may build a DDH-like distinguisher.

Let SignForge be the event that $\mathcal{A}$ can output a new, valid message/signature pair with respect to the public verification key vrfy_* of some party $\mathcal{P}_*$ *before* querying STATICREVEAL$(\mathcal{P}_*)$. The probability of this event occurring is bounded by the total number of parties times the success probability of forging the signature, $\mathsf{Succ}_{\mathcal{A}}(\Pi_{sign})$, which per assumption was hard, i.e. the probability is negligible. In other words,

$$\Pr[\mathsf{SignForge}] \leq |\mathfrak{P}| \cdot Succ_{\mathcal{A}}(\Pi_{sign}),$$

such that the probability that SignForge occurs, is negligible.

Let Repeat be the event that a nonce used by any

party in response to a Send query was previously used by that party. Recall that the adversary $\mathcal{A}$ is in charge of all communications. If we ignore the uniqueness of the session names, sN, which will only force the Send queries considered to be from the same execution of the protocol, then the probability of this event occurring is bounded by the maximum number of Send queries in a session and the amount of protocol executions, in the sense that

$$\Pr[\mathsf{Repeat}] \leq \frac{\nu(\nu+\Lambda)}{2^{\lambda_{sign}}},$$

where ν is the maximum number of Send queries queried by the adversary in a single execution of the protocol, and Λ is as in Algorithm 2. As both ν and Λ are polynomially-bounded, this probability is negligible.

Let the sum of these two negligible probabilities, for $\mathsf{SignForge}$ and Repeat, be denoted as negl.

In each session, all attack queries are answered fully and honestly, except in the ℓ-th session, where state reveal, ephemeral reveal, and static reveal attack queries to the root parties force the distinguisher to end the experiment and output $d \xleftarrow{R} \{0,1\}$. As the ℓ-th session is chosen as the test session with probability $1/\Lambda$, and per our assumption that the static and ephemeral reveal queries are not queried to the root parties as in Note 3.2, this outcome cannot occur every time.

If the test session is *not* the ℓ-th session, then $\mathcal{D}$ outputs a random bit, i.e. has advantage 0. If the test session *is* the ℓ-th session, which happens with probability $1/\Lambda$, then $\mathcal{A}$ will succeed with advantage ε. Hence, the final advantage of the DDH-like distinguisher $\mathcal{D}$ is $(\varepsilon - \mathsf{negl})/\Lambda$, which is non-negligible.

Forward security comes from the forward security of Π_{tree} combined with the ineffectiveness of replay attacks. □

The above compiler can be instantiated using the P2P-tree-R-LWE-GKE as the underlying protocol instead of tree-R-LWE-GKE, with only a slight change to the Ini_{sign} step. The change is that party $\mathcal{P}_i$ still computes the signature $\sigma \leftarrow \mathtt{Sign}_{\mathsf{sign}_i}(\mathcal{P}_i|sN|0|\eta_i)$ but only multicasts $\mathcal{P}_i|sN|0|\eta_i|\sigma$ to its *parent and its children*, as opposed to all the descendants. The session specific information is also reduced to

$$\mathsf{info}_i^{\mathsf{sid}} = \mathsf{gid}|sN|\mathcal{P}_{\mathsf{par}(i)}|\eta_{\mathsf{par}(i)}|\mathcal{P}_{1.\mathsf{cld}(i)}|\eta_{1.\mathsf{cld}(i)}|\cdots \cdots|\mathcal{P}_{l_i.\mathsf{cld}(i)}|\eta_{l_i.\mathsf{cld}(i)}.$$

By an analogous proof as that for authenticated tree-R-LWE-GKE, we get the following theorem.

Theorem 4.3. *Assuming the signature scheme Π_{sign} is (post-quantum) secure and that the DDH-like problem is (post-quantum) hard, authenticated P2P-tree-R-LWE-GKE is a (post-quantum) secure AGKE in the G-CK$^+$ security model, with forward security.*

5. COMPARISON

In this section, we compare our AGKEs with other post-quantum R-LWE GKEs: Apon, Dachman-Soled, Gong, and Katz [3] and Choi, Hong, and Kim [4]. Apon et al [3] generalize a Diffie-Hellman based GKE construction by Burmester and Desmedt [6] into the R-LWE setting. Choi et al [4] generalize another Diffie-Hellman based GKE by Dutta and Barua [7] into the R-LWE setting. Both papers arrange the parties in a ring structure, letting $\mathcal{P}_n = \mathcal{P}_0, \mathcal{P}_{n+1} = \mathcal{P}_1$, etc., and achieve post-quantum R-LWE n-party AGKE protocols with communication and memory complexity $O(n)$.

We choose to consider our auth. tree-R-LWE-GKE and auth. P2P-tree-R-LWE-GKE having binary double trees as their graphs, which are double trees where each party (excepting leaves) has exactly 2 children. Due to the signature initialization, there is a communication overhead in both AGKEs of at most $O(\log_2 n)$. This gives auth. tree-R-LWE-GKE a constant number of rounds with communication and memory complexity $\log_2(n)$, while the values are more or less reversed for auth. P2P-tree-R-LWE-GKE.

We evaluate the AGKEs in the following aspects: the number of rounds, the communication complexity, and the number of values needed to compute the session key, i.e. the memory complexity. The number of *rounds* is taken to be the maximum number of times any party must wait for information from other parties in order to proceed. The *communication complexity* considers the maximum number of broadcast/multicast messages received by any party in one call of the protocol[7]. The *memory complexity* takes into account the maximum number of values stored until the session key computation. Table 1 shows these parameters for our selected AGKEs.

For auth. tree-R-LWE-GKE, we have four rounds, as the signatures are initialized in the first round, R-LWE public keys are exchanged in the second round, reconciliation keys in the third round, and the exclusive-or of shared keys in the fourth round. The multicast values received are the signature initialization values of all ancestors and each child, R-LWE public keys of the parent and each child, a reconciliation key from the parent, as well as one exclusive-or sum from each ancestor. The values stored until the session key computation consist of one exclusive-or sum from each ancestor as well as the R-LWE key shared with the parent.

The auth. tree-R-LWE-GKE protocol and the related auth. P2P-tree-R-LWE-GKE differ greatly in the number of rounds, communication complexity, and values needed to generate the session key. We note that for both protocols, the overall smallest number of operations per party is obtained for a binary double tree. That said, depending on the structure of the network and the computational power of the parties involved, among other factors, it may be beneficial to select one protocol over the other and to arrange the double tree as needed.

Parameter Constraints. Beyond the parameter constraints required for the hardness of the R-LWE problem,

[7] In doing so, we assume that broadcasting/multicasting a message does not depend on the number of receivers but that receiving l messages means that the receiver incurs a cost of l, even if all messages are received in a single round. The reason for this is that it takes into account that receiving messages requires being online and also storing said messages while broadcasting/multicasting is usually a one-time operation.

TABLE 1: Comparision overview of R-LWE based AGKEs.

Protocol	Rounds	Communication	Memory
Apon et al [3]	4	$O(n)$	$O(n)$
Choi et al [4]	3	$O(n)$	$O(n)$
Auth. tree-R-LWE-GKE	4	$O(\log_2 n)$	$O(\log_2 n)$
Auth. P2P-tree-R-LWE-GKE	$O(\log_2 n)$	$O(\log_2 n)$	2

the parameters of [3] and [4] (including the number of parties) are required to satisfy further bounds set by the key reconciliation and Rényi bounds, for correctness and security. Fixing the ring, noise distributions, and security parameters therefore limits the amount of parties their protocols can support, while our security proof sets no further constraints on our parameters and our correctness bound makes the amount of parties inconsequential (see below). Although our protocol does not have constraints other than those required for the hardness of the DDH-like problem, the advantage for an adversary in solving the DDH-like problem is less than the sum of the advantages of solving two instances of the D-R-LWE problem (see [9, Theorem 1]), meaning that our R-LWE parameters must be adjusted accordingly. For example, [9] suggest $n = 1024, q = 2^{32} - 1, \sigma = 8/\sqrt{2\pi}$ to achieve statistical 128-bit classical security, giving theoretical 64-bit post-quantum security, assuming Grover's algorithm corresponds to a square-root speed up to the search problem. Using these parameters and Proposition 2 of [9], we find that the failure rate of our two AGKE protocols are equivalent and bounded by the probability of at least one party having the wrong session key: $n \cdot 2^{-2^{14}}$.

6. CONCLUDING REMARKS

We gave a compiler for our tree-based R-LWE GKEs, relying on the hardness of the DDH-like problem and the security of the signature scheme employed in the compiler. Our protocols give us versatile post-quantum R-LWE n-party AGKEs that, when balanced with 2 children per node, in one case achieves constant round complexity with communication and memory complexity $O(\log_2 n)$, and in the other case, constant memory complexity with round and communication complexity $O(\log_2 n)$.

Initially, our GKE (and thereby the P2P version and AGKE versions) incorporates the generation of a double-tree and a session name, sN, into the protocol creation algorithm. The AGKE compiler of Section 4 is a variant of the Desmedt-Lange-Burmester compiler [7], where $\mathsf{info}_i^{\mathsf{sid}}$ (called direct_U^j in [7]) is no longer included in the signature of messages, but sN instead. The reason is that $\mathsf{info}_i^{\mathsf{sid}}$ has at least length $O(\log n)$ while the session name, sN, may be much shorter, which is an improvement without detracting from the security (see the Repeat event argument in the proof of Theorem 23). In fact, [7] do not use a session name as we do and do not sign their random nonces when multicasting them to the relevant parties, which leaves them open to attacks such as replay attacks.

We remark that R-LWE AGKEs of [3] and [4] have high communication and memory complexity, but possibly other benefits, due to their integration of the R-LWE two-party key exchange mechanics into the protocol steps, unlike ours, which requires each pair of parent and child to complete a R-LWE key exchange before proceeding. It may be possible to improve our key exchange by likewise integrating R-LWE key exchange principles into the tree structure but we have not considered the possibility. In any case, as our protocols are tree-based, they benefit from being able to structure the tree according to processing power or memory capabilities. We further remark that signatures might not be required for authentication if we employ an authenticated key exchange as the underlying KE, such as the one proposed by Zhang et al [20], however our purpose in this article was to construct a compiler from the basic R-LWE KE.

In conclusion, the low communication and memory complexity in our protocol, the versatility of tree-based constructions, along with the added security benefit from reducing to the indistinguishability of a single instance of Ding, Xie, and Lin's R-LWE key exchange with Peikert's tweak, makes our protocols highly competitive R-LWE based post-quantum AGKEs.

DATA AVAILABILITY STATEMENT

No new data were generated or analysed in support of this research.

ACKNOWLEDGEMENTS

This work is partially supported by enPiT (Education Network for Practical Information Technologies) at MEXT, and Innovation Platform for Society 5.0 at MEXT.

REFERENCES

[1] Ding, J., Xie, X., and Lin, X. (2012). A simple provably secure key exchange scheme based on the learning with errors problem. Cryptology ePrint Archive, Report 2012/688.

[2] Peikert, C. (2014) Lattice cryptography for the internet. *Post-Quantum Cryptography*, pp. 197–219. Springer International Publishing.

[3] Apon, D., Dachman-Soled, D., Gong, H., and Katz, J. (2019) Constant-round group key exchange from the Ring-LWE assumption. *PQCrypto*, pp. 189–205. Springer.

[4] Choi, R., Hong, D., and Kim, K. (2020). Constant-round dynamic group key exchange from RLWE assumption. Cryptology ePrint Archive, Report 2020/035.

[5] Lyubashevsky, V., Peikert, C., and Regev, O. (2013) On ideal lattices and learning with errors over rings. *J. ACM*, **60**, 43–78.

[6] Burmester, M. and Desmedt, Y. (1995) A secure and efficient conference key distribution system. *Advances in Cryptology — EUROCRYPT'94*, pp. 275–286. Springer Berlin Heidelberg.

[7] Dutta, R. and Barua, R. (2005) Constant round dynamic group key agreement. *Information Security*, pp. 74–88. Springer Berlin Heidelberg.

[8] Burmester, M. and Desmedt, Y. (1997) Efficient and secure conference-key distribution. *Proceedings of the International Workshop on Security Protocols*, pp. 119–129. Springer-Verlag.

[9] Bos, J. W., Costello, C., Naehrig, M., and Stebila, D. (2015) Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. *IEEE Symposium on Security and Privacy*, pp. 553–570. IEEE Computer Society.

[10] Hougaard, H. B. and Miyaji, A. (2020) Tree-based ring-lwe group key exchanges with logarithmic complexity. In Meng, W., Gollmann, D., Jensen, C. D., and Zhou, J. (eds.), *Information and Communications Security - 22nd International Conference, ICICS 2020, Copenhagen, Denmark, August 24-26, 2020, Proceedings*, Lecture Notes in Computer Science, **12282**, pp. 91–106. Springer.

[11] Katz, J. and Yung, M. (2007) Scalable protocols for authenticated group key exchange. *J. Cryptol.*, **20**, 85–113.

[12] Desmedt, Y., Lange, T., and Burmester, M. (2007) Scalable authenticated tree based group key exchange for ad-hoc groups. *Financial Cryptography and Data Security*, pp. 104–118. Springer Berlin Heidelberg.

[13] Suzuki, K. and Yoneyama, K. (2014) Exposure-resilient one-round tripartite key exchange without random oracles. *IEICE Transactions*, **97-A**, 1345–1355.

[14] Katz, J. and Lindell, Y. (2015) *Introduction to Modern Cryptography*, 2 edition. CRC Press.

[15] Lyubashevsky, V., Peikert, C., and Regev, O. (2013) A toolkit for Ring-LWE cryptography. *Advances in Cryptology – EUROCRYPT 2013*. Springer Berlin Heidelberg.

[16] Bresson, E., Chevassut, O., Pointcheval, D., and Quisquater, J.-J. (2001) Provably authenticated group Diffie-Hellman key exchange. *Proceedings of the 8th ACM Conference on Computer and Communications Security* 255–264. Association for Computing Machinery.

[17] Bresson, E. and Manulis, M. (2008) Securing group key exchange against strong corruptions. *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security*, New York, NY, USA ASIACCS '08 249–260. Association for Computing Machinery.

[18] Ding, J., Alsayigh, S., Saraswathy, R. V., Fluhrer, S. R., and Lin, X. (2017) Leakage of signal function with reused keys in rlwe key exchange. *ICC*, pp. 1–6. IEEE.

[19] Ding, J., Fluhrer, S. R., and RV, S. (2018) Complete attack on RLWE key exchange with reused keys, without signal leakage. In Susilo, W. and Yang, G. (eds.), *Information Security and Privacy - 23rd Australasian Conference, ACISP 2018, Wollongong, NSW, Australia, July 11-13, 2018, Proceedings*, Lecture Notes in Computer Science, **10946**, pp. 467–486. Springer.

[20] Zhang, J., Zhang, Z., Ding, J., Snook, M., and Dagdelen, Ö. (2015) Authenticated key exchange from ideal lattices. *EUROCRYPT (2)*, pp. 719–751. Springer.