Authenticated Key Exchange from Ideal Lattices

Jiang Zhang[1], Zhenfeng Zhang[1]*, Jintai Ding[2]*, Michael Snook[2]

1. Chinese Academy of Sciences; 2. University of Cincinnati

*Corresponding Authors

Abstract

Authenticated key exchange (AKE) protocols, such as IKE and SSL/TLS, have been widely used to ensure secure communication over the Internet. We present in this paper a practical and provably secure AKE protocol from ideal lattices, which is conceptually simple and has similarities to the Diffie-Hellman based protocols such as HMQV (CRYPTO 2005) and OAKE (CCS 2013). Our protocol does not rely on other cryptographic primitives—in particular, it does not use signatures—simplifying the protocol and resting the security solely on the hardness of the ring learning with errors (RLWE) problem. The security is proven in a version of the Bellare-Rogaway model, with enhancements to capture weak Perfect Forward Secrecy. We also present concrete choices of parameters for different security levels. A proof-of-concept implementation shows our protocol is a practical candidate post-quantum key exchange protocol.

1. Introduction

Key exchange (KE) is a fundamental cryptographic primitive, allowing two parties to securely generate a common secret key over an insecure network. Because symmetric cryptographic tools (e.g. AES) are reliant on both parties having a shared key in order to securely transmit data, KE is one of the most used cryptographic tools in building secure communication protocols (e.g. SSL/TLS, IPSec, SSH). Following the introduction of the Diffie-Hellman (DH) protocol [13], cryptographers have devised a wide selection of KE protocols with various use-cases. One such class is Authenticated Key Exchange (AKE), a class of KE protocols where each party is able to verify the other's identity, so that an adversary cannot impersonate one party in the conversation.

For an AKE protocol, each party has two corresponding *static keys*: a *static secret key* and a corresponding *static public key*. The static public key is certified to belong to its owner using a public key infrastructure or ID-based infrastructure. For each run of the protocol, the parties involved generate *ephemeral secret keys* and use these to generate *ephemeral public keys* that they exchange. Then all of the keys are used along with the transcripts of the session to create a shared *session state*, which is then passed to a *key derivation function* to obtain the final session key. Intuitively, such a protocol is secure if no Probabilistic Polynomial Time (PPT) adversary is able to extract any information about the session key from the publicly exchanged messages. More formally, Bellare and Rogaway [4] introduced an indistinguishability-based security model for AKE, the BR model, which captures basic notions such as *known key security* and *impersonation resistance*. In 2001, Canetti and Krawczyk [7] presented a refined model, the CK model, that also accounts for scenarios in which the adversary is able to obtain information about a static secret key or a session state other than the state of the target session. One common weakness these models share is their inability to ensure that a protocol satisfies Perfect Forward Secrecy (PFS), the property that an adversary cannot compromise session keys after a completed session, even if it obtains the parties' static secret keys (e.g., via a heartbleed attack[1]). As shown in [30], no two-pass AKE protocol based on public-key authentication can achieve PFS. Thus, the notion of weak PFS (wPFs) is usually considered for two-pass AKE protocols, which states that the session key of an honestly run session cannot be compromised if the static keys are compromised after the session is finished [30].

1.1 Previous Work

Since Diffie and Hellman introduced their celebrated KE protocol in [13], a number of (A)KE protocols have been proposed. One approach for achieving authentication in KE protocols is to explicitly authenticate the exchanged messages between the involved parities by using some cryptographic primitives (e.g., signatures, MAC etc.), which usually incurs additional computation and communication costs with respect to the basic KE protocol, and complicates the understanding of the KE protocol. This includes several well-known protocols such as IKE [25, 28], SIGMA [29], SSL [16], TLS [12, 23, 31, 36], and so on. Another line of

[1] http://heartbleed.com/

designing AKEs follows the idea of MQV [26, 37] (which has been standardized by ISO/IEC and IEEE, and recommended by NIST and NSA Suite B) by making good of the algebraic structure of DH problems to achieve implicit authentication, e.g., HMQV [30] and OAKE [46]. All the above AKEs are based on classic hard problems, such as factoring, the RSA problem, and the computational/decision DH problem. Since these hard problems are vulnerable to quantum computers [44] as we are moving into the era of quantum computing, it is very appealing to find other counterparts based on problems believed to be resistant to quantum attacks. For instance, post-quantum AKE is considered of high priority by NIST [8]. Due to the potential benefits of lattice-based construction such as asymptotic efficiency, conceptual simplicity, worst-case hardness assumptions, it makes perfect sense to build lattice-based AKEs.

As far as we know, there are four papers focused on designing (A)KEs from lattices [14, 17, 18, 27, 42]. Katz and Vaikuntanathan [27] proposed the first password-based authenticated key exchange that can be proven secure based on the LWE assumption in the standard model. The three papers [17, 18, 42] followed generic transformations from *key encapsulation mechanisms* to AKEs by explicitly using signatures to provide authentication, thus suffered additional overheads. Recently, Ding et al. [14] proposed a simple KE protocol based on (Ring-)LWE by exploring a similar commutativity property as for the DH protocol (i.e., $(g^a)^b = (g^b)^a$ in cyclic groups). Like the standard DH protocol, the protocol in [14] cannot provide authentication—i.e., it is not an AKE protocol—and is thus weak to man-in-the-middle attacks. Besides, no papers above provide suggestions on practical implementations, e.g., estimation of actual security levels (e.g., 80 bits), choices of concrete parameters etc.

1.2 Our Contribution

In this paper, we propose an efficient AKE protocol based on the RLWE problem, which in turn is as hard as some hard problems (e.g., SIVP) in the worst-case on ideal lattices. Our method avoids introducing extra cryptographic primitives, thus simplifying the design and reducing overhead. In particular, the communicating parties are not required to either encrypt any messages with the other's public key, nor sign any of their own messages during key exchange. Furthermore, by having the key exchange as a self-contained system, we reduce the security assumptions needed, and are able to rely directly on the hardness of RLWE.

Our protocol has only two-pass messages like Ding et al.'s KE protocol [14]. We employ some useful properties of RLWE and discrete Gaussian distributions (on ideal lattices), and establish an approach to combine both the static and ephemeral public/secret keys, in a manner similar to HMQV [30]. Moreover, as captured by the enhanced BR model, our protocol achieves weak PFS property, which is known as the best PFS notion for two-pass protocols [30].

Taking careful considerations on both the correctness of the protocol and the state of art in solving (Ring-)LWE problems, we select concrete choices of parameters (ranging from 80 bits to 256 bits security) together with estimations on the key sizes and the communication overheads. We also construct a proof-of-concept implementation to examine the efficiency of our protocol with the selected parameters. The results show that our protocol is a practical candidate post-quantum key exchange protocol. Besides, our implementation has not undergone any real optimization, and it can be much improved.

2. Preliminaries

2.1 Notation

Throughout the paper, let n be the natural security parameter, and all quantities are implicitly dependent on n. Let $poly(n)$ denote an unspecified function $f(n) = O(n^c)$ for some constant c. The function $\log$ denotes the natural logarithm. We use standard notation O, ω to classify the growth of functions. If $f(n) = O(g(n) \cdot \log^c n)$, we denote $f(n) = \tilde{O}(g(n))$. We say a function $f(n)$ is negligible if for every $c > 0$, there exists a N such that $f(n) < 1/n^c$ for all $n > N$. We use $negl(n)$ to denote a negligible function of n, and we say a probability is overwhelming if it is $1 - negl(n)$.

The set of real numbers (integers) is denoted by $\mathbb{R}$ ($\mathbb{Z}$, resp.). We use $\leftarrow_r$ to denote randomly choosing an element from some distribution (or the uniform distribution over some finite set). Vectors are in column form and denoted by bold lower-case letters (e.g., $\mathbf{x}$). The ℓ_2 and ℓ_∞ norms we designate by $\|\cdot\|$ and $\|\cdot\|_\infty$. The ring of polynomials over $\mathbb{Z}$ ($\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$, resp.) we denote by $\mathbb{Z}[x]$ ($\mathbb{Z}_q[x]$, resp.).

2.2 Security Model for AKE

We now recall the BR security model [4], restricted to the case where each party sends the other only a single message.

Sessions We fix a positive integer N to be the maximum number of honest parties that use the AKE protocol. Each party is uniquely identified by an integer i in $\{1, 2, \ldots, N\}$, and has a static key pair consisting of a static secret key sk_i and static public key pk_i, which is signed by a Certificate Authority (CA). A single run of the protocol is called a *session*. A session is activated at a party by an incoming message of the form (Π, I, i, j) or the form (Π, R, j, i, X_i), where Π is a protocol identifier; I and R are role identifiers; i and j are party identifiers. If party i receives a message of the form (Π, I, i, j), we say that i is the session initiator. Party i then outputs the response X_i intended for party j. If party j receives a message of the form (Π, R, j, i, X_i), we say that j is the session responder; party j then outputs a response Y_j to send back to party i. After exchanging these messages, both parties compute a session key.

If a session activated at party i and has i as the initiator, we associate with it a *session identifier* $\mathrm{sid} = (\Pi, I, i, j, X_i)$ or $\mathrm{sid} = (\Pi, I, i, j, X_i, Y_j)$. Similarly, if a session activated at party j and has j as the responder, the session identifier has the form $\mathrm{sid} = (\Pi, R, j, i, X_i, Y_j)$. For a session identifier $\mathrm{sid} = (\Pi, *, i, j, *[, *])$, the third coordinate—that is, the first party identifier—is called the owner of the session; the other party is called the peer of the session. A session is said to be *completed* when its owner computes a session key. The matching session of $\mathrm{sid} = (\Pi, I, i, j, X_i, Y_j)$ is the session with identifier $\widetilde{\mathrm{sid}} = (\Pi, R, j, i, X_i, Y_j)$ and vice versa.

Adversarial Capabilities We model the adversary $\mathcal{A}$ as a probabilistic polynomial time (PPT) Turing machine with full control over all communications channels between parties, including control over session activations. In particular, $\mathcal{A}$ can intercept all messages, reading them all, and can delete or modify any desired messages as well as inject its own messages. We also suppose $\mathcal{A}$ is capable of obtaining hidden information about the parties, including static secret keys and session keys. We formalize these abilities by giving $\mathcal{A}$ access to the following oracles (we split the Send query in [7] into Send_0, Send_1, and Send_2 to distinguish the initial message for activating an initiator, the first and the second exchanged messages for two-pass protocols):

- $\mathsf{Send}_0(\Pi, I, i, j)$: $\mathcal{A}$ activates party i as an initiator. The oracle returns a message X_i intended for party j.
- $\mathsf{Send}_1(\Pi, R, j, i, X_i)$: $\mathcal{A}$ activates party j as a responder using message X_i. The oracle returns a message Y_j intended for party i.
- $\mathsf{Send}_2(\Pi, R, i, j, X_i, Y_j)$: $\mathcal{A}$ sends party i the message Y_j to complete a session previously activated with a $\mathsf{Send}_0(\Pi, I, i, j)$ query that returned X_i.
- $\mathsf{SessionKeyReveal}(\mathrm{sid})$: The oracle returns the session key associated with the session sid. This query can only be made if sid identifies a completed session, since no session key exists to be returned otherwise.
- $\mathsf{Corrupt}(i)$: The oracle returns the static secret key belonging to party i. A party whose key is given to $\mathcal{A}$ in this way is called *dishonest*; a party not compromised in this way is called *honest*.
- $\mathsf{Test}(\mathrm{sid}^*)$: The oracle chooses a bit $b \leftarrow_r \{0, 1\}$. If $b = 0$, it returns a key chosen uniformly at random; if $b = 1$, it returns the session key associated with sid^*. Note that we impose some restrictions on this query. We only allow $\mathcal{A}$ to query this oracle once, and only on a fresh (see Definition 1) session sid^*.

Definition 1 (Freshness)**.** *Let* $\mathrm{sid}^* = (\Pi, I, i^*, j^*, X_i, Y_j)$ *or* $(\Pi, R, j^*, i^*, X_i, Y_j)$ *be a completed session with initiator party* i^* *and responder party* j^**. If the matching session exists, denote it* $\widetilde{\mathrm{sid}}^*$*. We say that* sid^* *is fresh if the following conditions all hold:*

- *$\mathcal{A}$ has not made a $\mathsf{SessionKeyReveal}$ query on* sid^*.
- *$\mathcal{A}$ has not made a $\mathsf{SessionKeyReveal}$ query on* $\widetilde{\mathrm{sid}}^*$ *(if it exists).*
- *If* $\widetilde{\mathrm{sid}}^*$ *does not exist, neither party i^* nor j^* is dishonest. I.e., $\mathcal{A}$ has not made a $\mathsf{Corrupt}$ query on either of them.*

Remark 1. *Recall that in the original BR model [4], no corruption query is allowed. In the above freshness definition, we allow the adversary to corrupt both parties of* sid^* *if the matching session exists in order to capture the weak Perfect Forward Security (wPFS) [30].*

Security Game The security of a two-pass AKE protocol is defined in terms of the following game. The adversary $\mathcal{A}$ makes any sequence of queries to the oracles above, so long as only one Test query is made on a fresh session, as mentioned above. The game ends when $\mathcal{A}$ outputs a guess b' for b. We say $\mathcal{A}$ wins the game if its guess is correct, so that $b' = b$. The advantage of $\mathcal{A}$, $\mathrm{Adv}_{\Pi,\mathcal{A}}$, is defined as the probability that $\mathcal{A}$ wins minus $1/2$.

Definition 2 (Security)**.** *We say that an AKE protocol* Π *is secure if the following conditions hold:*

- *If two honest parties complete matching sessions then they compute the same session key with overwhelming probability.*
- *For any PPT adversary $\mathcal{A}$, the advantage* $\mathrm{Adv}_{\Pi,\mathcal{A}}$ *is negligible.*

2.3 Ring Learning with Errors

Let the integer n be a power of 2, and consider the ring $R = \mathbb{Z}[x]/(x^n + 1)$. For any positive integer q, we define the ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ analogously. For any polynomial $y(x)$ in R (or R_q), we identify y with its coefficient vector in $\mathbb{Z}^n$ (or $\mathbb{Z}_q^n$). In doing so we also define the norm of a polynomial to be the norm of its coefficient vector.

Lemma 1. *Let R be defined as above. Then, for any $s, t \in R$, we have* $\|s \cdot t\| \leq \sqrt{n} \cdot \|s\| \cdot \|t\|$ *and* $\|s \cdot t\|_\infty \leq n \cdot \|s\|_\infty \cdot \|t\|_\infty$.

For any $\alpha \in \mathbb{R}^+$ and $c \in \mathbb{R}$, denote $D_{\mathbb{Z},\alpha,c}$ as the 1-dimensional discrete Gaussian distribution centered at c with standard deviation α. To sample an element from $D_{\mathbb{Z},\alpha,c}$, one can first sample an element Y according to a Gaussian distribution with standard deviation α centered on c and set $X = \lfloor Y \rceil$, where $\lfloor \cdot \rceil$ represents rounding to the nearest integer [6]. For $\mathbf{c} = (c_0, \ldots, c_{n-1}) \in \mathbb{R}^n$, we also define the spherical discrete Gaussian distribution $D_{\mathbb{Z}^n,\alpha,\mathbf{c}}$ over $\mathbb{Z}^n$ as the distribution where the ith coordinate is distributed according to $D_{\mathbb{Z},\alpha,c_i}$. If the center $\mathbf{c}$ is zero, we write the distribution $D_{\mathbb{Z}^n,\alpha,\mathbf{0}}$ as χ_α. We now adopt the following notational convention: since bold-face variables denote vectors, $\mathbf{x} \leftarrow_r \chi_\alpha$ means we sample the vector $\mathbf{x}$ from the distribution χ_α; for normal weight variables (e.g. $y \leftarrow_r \chi_\alpha$) we sample an element of R whose coefficient vector is distributed according to χ_α. We have the following useful facts:

Lemma 2 ([22, 38])**.** *For any real number* $\alpha = \omega(\sqrt{\log n})$*, we have* $\Pr_{\mathbf{x} \leftarrow_r \chi_\alpha}[\|\mathbf{x}\| > \alpha\sqrt{n}] \leq 2^{-n+1}$.

Lemma 3 ([6, 24]). *For any real number $\alpha = \omega(\sqrt{\log n})$ and any $\mathbf{y} \in \mathbb{Z}^n$, the statistical distance between the distributions χ_α and $\chi_\alpha + \mathbf{y}$ is at most $\|\mathbf{y}\|/\alpha$.*

The following lemma is implicit in Lemma 2.10 of [45], which states that for an appropriate γ, elements sampled from χ_γ are invertible except with negligible probability.

Lemma 4 ([45]). *Let $q = 2^{\omega(\log n)}$ be a prime such that $q \bmod 2n = 1$, let R_q be defined as above, and let real $\gamma > \sqrt{n \cdot \omega(\log n)} \cdot q^{1/n}$. Then, for element $d \leftarrow_r \chi_\gamma$, d is invertible in R_q with overwhelming probability. In particular, if $d_1, d_2 \leftarrow_r \chi_\gamma$, then $d = d_1 - d_2$ is invertible with overwhelming probability since d follows the distribution $\chi_{\sqrt{2}\gamma}$.*

Now we come to the statement of the Ring-LWE assumption; we will use a special case detailed in [34]. Let R_q as defined above, and $s \leftarrow_r R_q$. We define A_{s,χ_α} to be the distribution of the pair $(a, as + x) \in R_q \times R_q$, where $a \leftarrow_r R_q$ is uniformly chosen and $x \leftarrow_r \chi_\alpha$ is independent of a.

Definition 3 (Ring-LWE Assumption). *Let R_q, χ_α be defined as above, and let $s \leftarrow_r R_q$. The Ring-LWE assumption $RLWE_{q,\alpha}$ states that it is hard for any PPT algorithm to distinguish A_{s,χ_α} from the uniform distribution on $R_q \times R_q$ with only polynomial samples.*

One can also modify the above definition by requiring the PPT algorithm to find s rather than distinguish the two distributions, giving a search problem rather than a decision problem. For certain parameter choices, the two forms are polynomially equivalent [34].

Proposition 5 (A special case of [34]). *Let n be a power of 2, let α be a real number in $(0, 1)$, and q a prime such that $q \bmod 2n = 1$ and $\alpha q > \omega(\sqrt{\log n})$. Define $R = \mathbb{Z}[x]/\langle x^n + 1\rangle$ as above. Then there exists a polynomial time quantum reduction from $\tilde{O}(\sqrt{n}/\alpha)$-SIVP (Short Independent Vectors Problem) in the worst case to average-case $RLWE_{q,\beta}$ with ℓ samples, where $\beta = \alpha q \cdot (n\ell/\log(n\ell))^{1/4}$.*

It has been proven that the Ring-LWE assumption still holds even if the secret s is chosen according to the error distribution χ_β rather than uniformly [1, 34]. This variant is known as the *normal form*, and is preferable for controlling the size of the error term [5, 6]. The underlying Ring-LWE assumption also holds when scaling the error by a constant t relatively prime to q, i.e., using the pair $(a_i, a_i s + tx_i)$ rather than $(a_i, a_i s + x_i)$. Several lattice-based cryptographic schemes have been constructed based on this variant [5, 6].

3. Authenticated Key Exchange from RLWE

We now introduce some notation before presenting our protocol. For odd prime $q > 2$, denote $\mathbb{Z}_q = \{-\frac{q-1}{2}, \dots, \frac{q-1}{2}\}$ and define the subset $E := \{-\lfloor \frac{q}{4} \rfloor, \dots, \lfloor \frac{q}{4} \rceil\}$ as the middle half of $\mathbb{Z}_q$. We also define Cha to be the characteristic function of *the complement of* E, so $\mathsf{Cha}(v) = 0$ if $v \in E$ and 1 otherwise. It is easy to verify that for any v in $\mathbb{Z}_q$, $v + \mathsf{Cha}(v) \cdot \frac{q-1}{2} \bmod q$ belongs to E. We define an auxiliary modular function, $\mathsf{Mod}_2 \colon \mathbb{Z}_q \times \{0, 1\} \to \{0, 1\}$:

$$\mathsf{Mod}_2(v, w) = (v + w \cdot \frac{q-1}{2}) \bmod q \bmod 2.$$

The following two useful lemmas on Mod_2 are used for the security and correctness of our AKE protocol.

Lemma 6. *Let n be the security parameter, and let $q = 2^{\omega(\log n)}$ be an odd prime, and $v \leftarrow_r \mathbb{Z}_q$. For any $b \in \{0, 1\}$ and any $v' \in \mathbb{Z}_q$, the output distribution of $\mathsf{Mod}_2(v + v', b)$ given $\mathsf{Cha}(v)$ is statistically close to uniform on $\{0, 1\}$.*

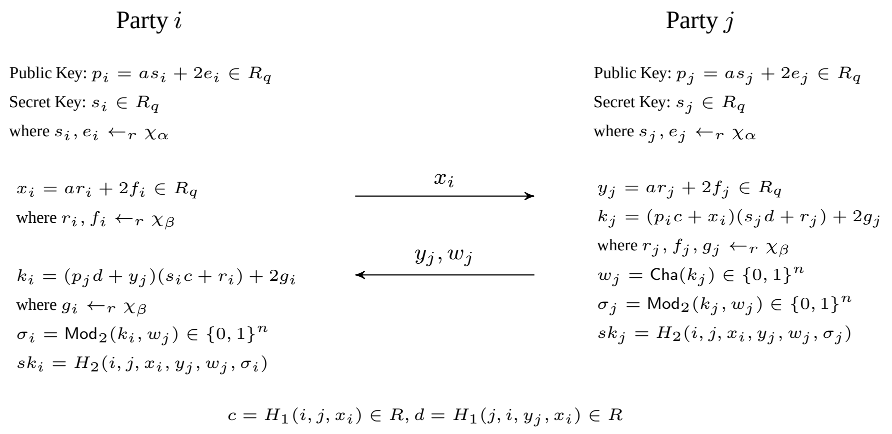
Proof. We condition on $\mathsf{Cha}(v)$:

- If $\mathsf{Cha}(v) = 0$, then $v + v' + b \cdot \frac{q-1}{2}$ is uniformly sampled from $v' + b \cdot \frac{q-1}{2} + E \bmod q$. This shifted set has $(q + 1)/2$ elements, which are either consecutive integers—if the shift is small enough—or two sets of consecutive integers—if the shift is large enough to cause wrap-around. Thus, we must distinguish a few cases:
  - If $|E|$ is even and no wrap-around occurs, then the result of $\mathsf{Mod}_2(v + v', b)$ is clearly uniform on $\{0, 1\}$.
  - If $|E|$ is odd and no wrap-around occurs, then we can assume without loss of generality that there are $t_0 = (|E| + 1)/2$ even integers and $t_1 = (|E| - 1)/2$ odd integers that we sample uniformly from. Then the statistical distance between the distribution of Mod_2 and uniform is

$$d = \left(\frac{t_0}{|E|} - \frac{1}{2}\right) + \left(\frac{1}{2} - \frac{t_1}{|E|}\right) = \frac{2}{2|E|} = \frac{1}{|E|}.$$

  - If $|E|$ is odd and wrap-around does occur, then we have split our sample space into two parts, one with an even number of elements, and one with an odd number of elements. This situation leads to the same calculations as with no wrap-around.
  - If $|E|$ is even and wrap-around occurs, then our sample space is split into either two even-sized sets, or two odd sized sets. If both are even, then once again our distribution is uniform. If both are odd, then we have $t_0 = (|E| - 3)/2$ and $t_1 = (|E| + 3)/2$ as the number of even and odd integers, respectively, in our sample space. Then the same calculation as before gives a statistical distance of $3/|E|$.

  Thus, the statistical distance between these two distributions is at most $\frac{3}{|E|} < \frac{6}{q}$. With $q = 2^{\omega(\log n)}$, this is negligible in n.
- If $\mathsf{Cha}(v) = 1$, then our sample set is instead $\tilde{E} = \mathbb{Z}_q \setminus E$, shifted as appropriate. Now $|\tilde{E}| = |E| - 1$, so by splitting into the same cases as $\mathsf{Cha}(v) = 0$, the statistical distance is at most $\frac{3}{|E|-1}$, which is still negligible for the given q. □



Party i
Party j
Public Key: $p_i = as_i + 2e_i \in R_q$
Secret Key: $s_i \in R_q$
where $s_i, e_i \leftarrow_r \chi_\alpha$
Public Key: $p_j = as_j + 2e_j \in R_q$
Secret Key: $s_j \in R_q$
where $s_j, e_j \leftarrow_r \chi_\alpha$
$x_i = ar_i + 2f_i \in R_q$
where $r_i, f_i \leftarrow_r \chi_\beta$
x_i
$y_j = ar_j + 2f_j \in R_q$
$k_j = (p_i c + x_i)(s_j d + r_j) + 2g_j$
where $r_j, f_j, g_j \leftarrow_r \chi_\beta$
$w_j = \mathsf{Cha}(k_j) \in \{0,1\}^n$
$\sigma_j = \mathsf{Mod}_2(k_j, w_j) \in \{0,1\}^n$
$sk_j = H_2(i, j, x_i, y_j, w_j, \sigma_j)$
y_j, w_j
$k_i = (p_j d + y_j)(s_i c + r_i) + 2g_i$
where $g_i \leftarrow_r \chi_\beta$
$\sigma_i = \mathsf{Mod}_2(k_i, w_j) \in \{0,1\}^n$
$sk_i = H_2(i, j, x_i, y_j, w_j, \sigma_i)$
$c = H_1(i, j, x_i) \in R, d = H_1(j, i, y_j, x_i) \in R$

Figure 1: Efficient AKE based on RLWE.

Lemma 7. *Let q be an odd prime, $v \in \mathbb{Z}_q$ and $e \in \mathbb{Z}_q$ such that $|e| < q/8$. Then, for $w = v + 2e$, we have $\mathsf{Mod}_2(v, \mathsf{Cha}(v)) = \mathsf{Mod}_2(w, \mathsf{Cha}(v))$.*

Proof. Note that $w+\mathsf{Cha}(v)\frac{q-1}{2} \bmod q = v+\mathsf{Cha}(v)\frac{q-1}{2}+2e \bmod q$. Now, $v + \mathsf{Cha}(v)\frac{q-1}{2} \bmod q$ is in E as we stated above; that is, $-\lfloor\frac{q}{4}\rfloor \le v + \mathsf{Cha}(v)\frac{q-1}{2} \bmod q \le \lfloor\frac{q}{4}\rceil$. Thus, since $-q/8 < e < q/8$, we have $-\lfloor\frac{q}{2}\rfloor \le v + \mathsf{Cha}(v)\frac{q-1}{2} \bmod q + 2e \le \lfloor\frac{q}{2}\rceil$. Therefore, we have $v + \mathsf{Cha}(v)\frac{q-1}{2} \bmod q + 2e = v + \mathsf{Cha}(v)\frac{q-1}{2} + 2e \bmod q = w + \mathsf{Cha}(v)\frac{q-1}{2} \bmod q$. Thus, $\mathsf{Mod}_2(w, \mathsf{Cha}(v)) = \mathsf{Mod}_2(v, \mathsf{Cha}(v))$. □

We further extend the definitions of Cha and Mod_2 to $\mathbb{Z}_q^n$ by applying them entry-wise to vectors. By a standard hybrid argument, the statistical distance between the uniform distribution on $\{0,1\}^n$ and the distribution of $\mathsf{Mod}_2(\mathbf{x}, \mathsf{Cha}(\mathbf{x}))$ given $\mathsf{Cha}(\mathbf{x})$ is also negligible, assuming $\mathbf{x}$ is uniformly chosen from $\mathbb{Z}_q^n$. A similar claim holds for coefficient vectors of ring elements in R_q.

3.1 The Protocol

We now describe our protocol in detail. Let n be a power of 2, and $q = 2^{\omega(\log n)}$ be an odd prime such that $q \bmod 2n = 1$. Take $R = \mathbb{Z}[x]/(x^n+1)$ and $R_q = \mathbb{Z}_q[x]/(x^n+1)$ as above. For $\gamma \in \mathbb{R}^+$, let $H_1 \colon \{0,1\}^* \to \chi_\gamma = D_{\mathbb{Z}^n,\gamma}$ be a hash function with output distribution χ_γ (In practice, one can take a function such as SHA-2 to obtain a uniformly random string, and then use that to sample from $D_{\mathbb{Z}^n,\gamma}$). Let $H_2 \colon \{0,1\}^* \to \{0,1\}^\kappa$ be the key derivation function, where κ is the bit-length of the final shared key. We model both functions as random oracles [3]. Let χ_α, χ_β be two discrete Gaussian distributions with parameters $\alpha, \beta \in \mathbb{R}^+$. Let $a \in R_q$ be the global public parameter uniformly chosen from R_q at random. Let $p_i = as_i + 2e_i \in R_q$ be party i's static public key, where s_i is the corresponding static secret key; both s_i and e_i are taken from the distribution χ_α. Similarly, party j has static public key $p_j = as_j + 2e_j$ and static secret key s_j. Our protocol consists of the following steps, illustrated in Figure 1:

Initiation Party i randomly samples $r_i, f_i, g_i \leftarrow_r \chi_\beta$ and computes $x_i = ar_i + 2f_i$, which he sends to party j.

Response Party j receives x_i from party i, randomly samples $r_j, f_j, g_j \leftarrow_r \chi_\beta$ and computes $y_j = ar_j + 2f_j$, similar to x_i. Party j also computes $c = H_1(i, j, x_i)$, $d = H_1(j, i, y_j, x_i)$, and $k_j = (p_i c + x_i)(s_j d + r_j) + 2g_j$ using x_i. Note c and d are both distributed according to χ_γ. Next, party j computes $w_j = \mathsf{Cha}(k_j) \in \{0,1\}^n$ and sends the pair (k_j, w_j) to party i. Lastly, party j computes $\sigma_j = \mathsf{Mod}_2(k_j, w_j)$ and derives the session key $sk_j = H_2(i, j, x_i, y_j, w_j, \sigma_j)$.

Finish Party i receives the pair (y_j, w_j), and uses it compute $c = H_1(i, j, x_i)$, $d = H_1(j, i, y_j, x_i)$, and $k_i = (p_j d + y_j)(s_i c + r_i) + 2g_i$. The quantities c and d are the same as computed by party j. Finally, party i computes $\sigma_i = \mathsf{Mod}_2(k_i, w_j)$ and derives the session key $sk_i = H_2(i, j, x_i, y_j, w_j, \sigma_i)$.

3.2 Correctness

To show the correctness of our AKE protocol, i.e. that both parties compute the same session key $sk_i = sk_j$, it suffices to show that $\sigma_i = \sigma_j$. Now, σ_i and σ_j are both the output of Mod_2 with $\mathsf{Cha}(k_j)$ as the second argument. By Lemma 7, we need only to show that k_i and k_j are sufficiently close. Now, the two parties will compute k_i and k_j as follows:

$$
\begin{aligned}
k_i = \;& (p_j d + y_j)(s_i c + r_i) + 2g_i \\
= \;& a(s_j d + r_j)(s_i c + r_i) \\
& +(2e_j d + 2f_j)(s_i c + r_i) + 2g_i \\
= \;& a(s_i c + r_i)(s_j d + r_j) + 2\widetilde{g}_i \\
k_j = \;& (p_i c + x_i)(s_j d + r_j) + 2g_j \\
= \;& a(s_i c + r_i)(s_j d + r_j) \\
& +(2e_i c + 2f_i)(s_j d + r_j) + 2g_j \\
= \;& a(s_i c + r_i)(s_j d + r_j) + 2\widetilde{g}_j
\end{aligned}
$$

Choice of Parameters	n	Security (expt.)	α	γ	$\log \frac{\beta}{\alpha}$	$\log q$ (bits)	Size (KB)			
							pk	sk (expt.)	init. msg	resp. msg
I*	1024	80 bits	3.397	101.919	8.5	40	5 KB	0.75 KB	5 KB	5.125 KB
II	2048	80 bits	3.397	161.371	27	78	19.5 KB	1.5 KB	19.5 KB	19.75 KB
III	2048	128 bits	3.397	161.371	19	63	15.75 KB	1.5 KB	15.75 KB	16 KB
IV	4096	128 bits	3.397	256.495	50	125	62.5 KB	3 KB	62.5 KB	63 KB
V	4096	192 bits	3.397	256.495	36	97	48.5 KB	3 KB	48.5 KB	49 KB
VI	4096	256 bits	3.397	256.495	28	81	40.5 KB	3 KB	40.5 KB	41 KB

Table 1: Choices of Parameters (The bound 6α with $\mathrm{erfc}(6) \approx 2^{-55}$ is used to estimate the size of secret keys.)

where $\widetilde{g}_i = (e_j d + f_j)(s_i c + r_i) + g_i$, and $\widetilde{g}_j = (e_i c + f_i)(s_j d + r_j) + g_j$. Then $k_i = k_j + 2(\widetilde{g}_i - \widetilde{g}_j)$. Thus, the correctness follows if $\|\widetilde{g}_i - \widetilde{g}_j\|_\infty < q/8$ (by Lemma 7).

3.3 Concrete Choices of Parameters

Following [11, 21, 34], we make use of the canonical embedding in the analysis of our protocol. Formally, for our choices of n (i.e., a power of 2) and R, the canonical embedding of $a \in R$ into $\mathbb{C}^n$ is the n-vector of complex numbers $\sigma(a) = (a(\zeta_m^i))$, where $m = 2n$, ζ_m is a complex primitive m-th root of unity and the indexes i range over all of $\mathbb{Z}_m^*$. We call the norm of $\sigma(a)$ the canonical embedding norm of a, and denote it by $\|a\|_\infty^{\mathrm{can}} = \|\sigma(a)\|_\infty$. There are two useful properties [11, 21, 34] of $\|\cdot\|_\infty^{\mathrm{can}}$:

– For all $a, b \in R$, $\|a \cdot b\|_\infty^{\mathrm{can}} \le \|a\|_\infty^{\mathrm{can}} \cdot \|b\|_\infty^{\mathrm{can}}$.

– For all $a \in R$, $\|a\|_\infty \le \|a\|_\infty^{\mathrm{can}}$.

Note that the evaluation $a(\zeta_m)$ is the inner product between the coefficient vector of a and the vector $\mathbf{z}_m = (1, \zeta_m, \zeta_m^2, \ldots, \zeta_m^{n-1})$. Thus, if the coefficient vector of a is chosen from Gaussian distribution $D_{\mathbb{Z}^n,\alpha}$ with standard deviation α, the random variable $a(\zeta_m)$ is distributed to a complex Gaussian random variable with variance $\delta^2 = \alpha^2 n$ (note that $\mathbf{z}_m$ has Euclidean norm exactly n). Following [21], we use 6δ as a high-probability bound on the size of $a(\zeta_m)$ (note that the complementary error function $\mathrm{erfc}(6) \approx 2^{-55}$). For a product of two random variables with variance δ_1^2 and δ_2^2, respectively, we use $16\delta_1\delta_2$ as our high probability bound. Since $\mathrm{efrc}(4) \approx 2^{-25}$, the probability that both variables exceeds four times of their standard deviation is about 2^{-50}.

We now turn to bounding the size of the error term in our protocol, i.e., $\|\widetilde{g}_i - \widetilde{g}_j\|_\infty$. Note that $\widetilde{g}_i = (e_j d + f_j)(s_i c + r_i) + g_i$, and $\widetilde{g}_j = (e_i c + f_i)(s_j d + r_j) + g_j$, where $e_i, e_j \leftarrow_r \chi_\alpha$, $c, d \leftarrow_r \chi_\gamma$, and $f_i, f_j, r_i, r_j, g_i, g_j \leftarrow_r \chi_\beta$. Using the bounds in last paragraph, namely, $\|e_j d\|_\infty^{\mathrm{can}}, \|s_i c\|_\infty^{\mathrm{can}} \le 16\alpha\gamma n$, and $\|f_j\|_\infty^{\mathrm{can}}, \|r_i\|_\infty^{\mathrm{can}}, \|g_i\|_\infty^{\mathrm{can}} \le 6\beta\sqrt{n}$, we have that $\|\widetilde{g}_i\|_\infty^{\mathrm{can}} \le (16\alpha\gamma n + 6\beta\sqrt{n})^2 + 6\beta\sqrt{n}$ holds with overwhelming probability, (the same bound holds for $\|\widetilde{g}_j\|_\infty^{\mathrm{can}}$). Considering that $\alpha/\beta = 2^{-\omega(\log n)}$ is required in our security proof (see Claim 2), we therefore assume that the inequation $6\beta \gg 16\alpha\gamma\sqrt{n}$ holds[2]. In this case, both $\|\widetilde{g}_i\|_\infty^{\mathrm{can}}$ and $\|\widetilde{g}_j\|_\infty^{\mathrm{can}}$ are smaller than $37\beta^2 n$ with overwhelming probability. For correctness, it is enough to set q satisfying

$$16 * 37\beta^2 n < q. \tag{1}$$

Though the Ring-LWE problem enjoys a worst-case connection to some hard problems (e.g., SIVP [34]) on ideal lattices, the connection as summarized in Proposition 5 seems less powerful to estimate the actual security for concrete choices of parameters. For this, several works [9, 19, 21, 39, 40] took account experimental results to estimate the difficulties of solving (R)LWE. Especially, we use the result in [21], which shows how to set integer n (i.e., the rank of the underlying lattice), given the modulus q, the Gaussian parameter α (i.e., the error distribution is $D_{\mathbb{Z}^n,\alpha}$), and the concrete security parameter k (i.e., the time/advanatage ratio is of at least 2^k):

$$n \ge \frac{\log(q/\alpha)(k + 110)}{7.2}. \tag{2}$$

As recommended in [21, 32], it is enough to set the Gaussian parameter $\alpha \ge 3.2$ so that the discrete Gaussian $D_{\mathbb{Z}^n,\alpha}$ approximates the continuous Gaussian D_α extremely well[3]. In our experiment, we fix $\alpha = 3.397$ for a better performance of the Gaussian sampling algorithm in [15]. As for the choices of γ, we set $\gamma \approx n^{2/3}$ for the use of Lemma 4 in our security proof. In Table 2, we set all other parameters β, n, q to satisfies the correctness condition (1) and the security condition (2) for each expected security level. We also take account of $\log(\beta/\alpha)$ as an "index of confidence" for corresponding security levels in the choices of parameters. Note that n is required to be a power of 2 in our protocol (i.e., it is very sparsely distributed[4]), we can simply tries several possible values of n. In Table 1, we present several candidate choices of parameters, and estimate the sizes of public keys, secret keys, and communication overheads.

[2] We clarify that the choice of parameters I in Table 1 does not obey this assumption, but this does not violate the correctness property of our protocol for thousands of experimental runs using the parameters.

[3] Only α is considered here simply because $\gamma, \beta \gg \alpha$, and the (R-)LWE problem becomes harder as α grows bigger (for a fixed modulus q).

[4] We remark such a choice of n is not necessary, but it gives a simple analysis and implementation. In practice, one might use the techniques for Ring-LWE cryptography in [35] to give a tighter choice of parameters for desired security levels.

4. Implementation and Timings

In this section, we present the details of our proof-of-concept implementation, and show the timings of each operation.

4.1 Ring Representations and Operations

Recall that we are working on a ring $R_q := \mathbb{Z}_q[x]/(x^n+1)$, i.e., the ring of polynomials in x with integer coefficients, modulo q and x^n+1, where n is a power of 2. An element in R_q can be natural written as a polynomial of degree less than n with coefficients in $\mathbb{Z}_q$, e.g., $g(x) = \sum_{i=0}^{n-1} g_i x^i \in R_q$, and its coefficient vector $(g_0, \cdots, g_{n-1}) \in \mathbb{Z}_q^n$.

The addition operation of two ring elements $g, h \in R_q$ can be done by taking n coefficient-wise additions over $\mathbb{Z}_q$. But a naive multiplication operation of two ring elements $g, h \in R_q$ needs $O(n^2)$ multiplication operations over $\mathbb{Z}_q$, and then reduce the resulting polynomial modulo x^n+1. However, it is well-known that one can do the multiplication in $O(n \log n)$ time by using Fast Fourier Transform (FFT) according to the convolution theorem [10]. Concretely, the multiplication algorithm first computes the $2n$ Fourier coefficients G (H, resp.) of g (h, resp.), i.e., the values on all the $2n$-th roots of unity over the complex field $\mathbb{C}$. Then, it multiplies the $2n$ Fourier coefficients in a coefficient-wise way. Finally, it interpolates back to a polynomial $d(x) = g(x)h(x)$ with degree $< 2n$ by applying an inverse FFT, and reduces $d(x)$ by the polynomial x^n+1 to R_q.

Since we set prime q such that $q \mod 2n = 1$, the field $\mathbb{Z}_q$ actually contains a multiplicative subgroup of order $2n$ whose elements are all the $2n$-th roots of unity in $\mathbb{Z}_q$. In particular, if $w \in \mathbb{Z}_q$ is an element of order $2n$, then the odd powers $w, w^3, \ldots, w^{2n-1}$ are exactly the *primitive* $2n$-th roots of unity. As in [33], it suffices to compute only the n *primitive* Fourier coefficients of $g(x)$, i.e., the values of $g(x)$ at the points $w, w^3, \ldots, w^{2n-1}$. Note that

$$g(w^{2k+1}) = \sum_{i=0}^{i=n-1} g_i (w^{2k+1})^i = \sum_{i=0}^{n-1} (g_i \cdot w^i) \cdot (w^2)^{ik},$$

one can compute all the n coefficients by first taking n multiplications to compute $\hat{g}_i = g_i \cdot w^i$, and then applying an n-dimensional FFT on the polynomial $\hat{g}(x) = \sum_{i=0}^{i=n-1} \hat{g}_i x^i$ with n-th primitive root of unity w^2. In this setting, the Fourier coefficients of an element $g(x) \in R_q$ is still a vector of length n in $\mathbb{Z}_q$, which can be stored by using the same size space for storing the coefficient vector of $g(x)$. Besides, since $\{w^{2k+1}\}_{k=0,\ldots,n-1}$ are actually the roots of x^n+1, the reduction by the polynomial x^n+1 after taking an inverse FFT operation is saved.

According to the above observations, it is computationally efficient to keep the public parameters a, and the public keys in their Fourier representations (thus, one does not need to compute the Fourier representations of those elements online). Actually, in our implementation, we always use the Fourier representations of ring elements during the execution of the protocol, and only take one inverse FFT operation until we have to compute σ_i (σ_j, resp.), where the coefficient representations of k_i (k_j, resp.) are needed. This also means that the exchanged ring elements x_i, y_i are sent in their Fourier representations (note that this does not increase the communication overheads). Table 3 indicates the actual representations of ring elements, and the online/offline FFT operations in our implementation.

Ring element	Representation	FFT operation
a, p_i, p_j, s_i, s_j	Fourier	offline FFT
r_i, f_j, r_j, f_j, c, d	Fourier	online FFT
x_i, y_j	Fourier	-
k_i, k_j	Fourier and Coefficient	online inverse FFT
g_i, g_j	Coefficient	-

Table 3: Ring Elements and FFT Operations

4.2 Gaussian Sampling and Hashing to $D_{\mathbb{Z}^n,\gamma}$

For any $\alpha \in \mathbb{R}^+$, there are several algorithms in the literatures, e.g., [15, 22, 41], to sample an element from the discrete Gaussian distribution $D_{\mathbb{Z}^n,\alpha}$. Gentry et al. [22] gave a general sampling algorithm by using rejections. Roughly, the algorithm chooses $x \leftarrow_r [-\tau\alpha, \tau\alpha]$, and only outputs x with probability $\exp(-x^2/2\alpha^2)$, where τ is a Gaussian "tail-cut" factor (e.g., $\tau = 6$). This GPV algorithm [22] is general to handle any standard deviation $\alpha > 0$, but it needs a large averaged online time to successfully sample an element.

The other two sampling algorithms in [15, 41] involve storing of pre-computed data. Peikert [41] suggested that one can efficiently sample elements from $D_{\mathbb{Z}^n,\alpha}$ by using a pre-computed cumulative distribution table (CDT). This algorithm is very efficient, but it needs to store $\lambda\tau\alpha$ bits for the CDT table with λ bits of precision. When the standard deviation α is very large, this algorithm might be infeasible (such as $D_{\mathbb{Z}^n,\beta}$ for our choices of β in Table 1). In CRYPTO 2013, Ducas et al. [15] gave a new sampling algorithm, D-DLL, providing a better tradeoff between time and memory, which is slightly slower than the CDT algorithm, but only needs a memory about $\lambda \log(2.4\tau\alpha^2)$. For a better online performance, we will use both the CDT and DDLL algorithms. Concretely, the CDT algorithm is used for sampling from $D_{\mathbb{Z}^n,\alpha}$ and $D_{\mathbb{Z}^n,\gamma}$, and the DDLL algorithm is used for sampling from $D_{\mathbb{Z}^n,\beta}$. The total memory used for storing the pre-computed tables for both algorithms is less than 51 KB for the choice of parameters IV in Table 1.

As for the hashing to $D_{\mathbb{Z}^n,\gamma}$ operation, we first use the hash algorithm, SHA-256, to compress the inputs to obtain a 256-bit randomness seed, and then take it as input to the NTL pseudorandomness generator to generate the random coins used for sampling $D_{\mathbb{Z}^n,\gamma}$ (by using the CDT algorithm).

4.3 Timings

We implement our AKE protocol by using the NTL library compiled with the option NTL_GMP_LIP=on (i.e., building

Parameters	FFT operation	Sampling $D_{\mathbb{Z}^n,\beta}$	Hashing to $D_{\mathbb{Z}^n,\gamma}$	Initiation	Response	Finish
I	0.92 ms	0.56 ms	0.54 ms	3.22 ms (0.02 ms)	8.50 ms (4.69 ms)	5.23 ms (4.73 ms)
II	2.76 ms	2.90 ms	1.68 ms	12.00 ms (0.04 ms)	29.33 ms (14.64 ms)	17.28 ms (14.61 ms)
III	2.67 ms	2.19 ms	1.42 ms	10.33 ms (0.04 ms)	25.83 ms (13.46 ms)	15.58 ms (13.40 ms)
IV	7.33 ms	33.53 ms	4.85 ms	83.61 ms (0.08 ms)	156.58 ms (39.86 ms)	73.11 ms (39.73 ms)
V	6.02 ms	23.99 ms	3.86 ms	61.74 ms (0.08 ms)	117.81 ms (32.58 ms)	55.64 ms (32.20 ms)
VI	6.01 ms	6.29 ms	3.57 ms	25.42 ms (0.08 ms)	62.31 ms (31.32 ms)	36.80 ms (31.29 ms)

Table 2: Timings of Proof-of-Concept Implementations in ms (The figures in the parentheses indicate the timings with pre-computing. For comparison, by simply using the "speed" command in openssl on the same machine, the timing for dsa1024 signing algorithm is about 0.7 ms, and for dsa2048 is about 2.3 ms).

NTL using the GNU Multi-Precision package). The implementations are written in C++ without any parallel computations or multi-threads programming techniques. The program is run on a Dell Optiplex 780 computer with Ubuntu 12.04 TLS 64-bit system, equipped with a 2.83GHz Intel Core 2 Quad CPU and 3.8GB RAM. We use a n-dimensional FFT as discussed in 4.1 with a pre-computed tables for w^i and w^{2i}, which needs about 94KB memory (i.e., for storing $1.5n$ elements of $\mathbb{Z}_q$) for choice of parameters IV, where w is the $2n$-th primitive roots of unity, and $i = \{0, \ldots, n-1\}$. We use the DDLL algorithm [15] for sampling from $D_{\mathbb{Z}^n,\beta}$, and use the CDT algorithm [41] as a tool for hashing to $D_{\mathbb{Z}^n,\gamma}$.

In Table 2, we present the timings of each operation, and the figures represent the averaged timing (in millisecond, ms) for 1000 executions. Note that the exchanged ring elements $x_i = ar_i + f_i, y_j = ar_j + f_j$ are independent of other information in a particular execution of the AKE protocol. Each party can actually pre-compute those values in advance, e.g., sampling r_i, f_i, g_i (r_j, f_j, g_j, resp.) and computing $x_i = ar_i + f_i$ ($y_j = ar_j + f_j$ resp.) immediately after the protocol is started (e.g., each party can do it simultaneously during exchanging the necessary messages before executing the AKE protocol, e.g., at the time of sending the "Hello" message to the other party). As shown in Table 2, pre-computing those values will save a lot of online time especially for choices of parameters IV and V.

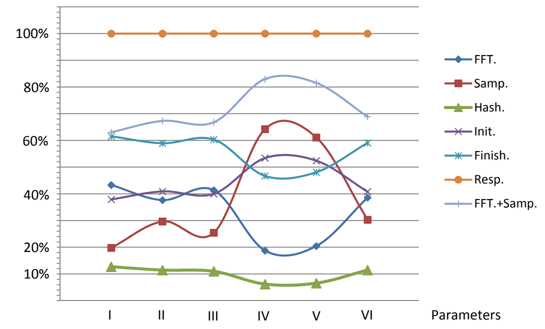


Figure 2: Comparison between different operations

In a complete execution of our AKE protocol (without pre-computation), one need to do 3 FFT operations and 1 inverse FFT operation (which almost takes about the same time as an FFT operation), 3 sampling from $D_{\mathbb{Z}^n,\beta}$ operations, and 2 hashing to $D_{\mathbb{Z}^n,\gamma}$ operations. In Figure 2, we treat the Response time as the total time needed for a party to execute the protocol, and check the cost of other operations. Since γ is small and the CDT algorithm is employed, the hashing to $D_{\mathbb{Z}^n,\gamma}$ operation takes about 10% of the total time. As β grows bigger, it becomes significantly slower for sampling from $D_{\mathbb{Z}^n,\beta}$, which takes $> 60\%$ of the total time for parameters choices IV and V. Another heavy operation is FFT, which takes about 40% of the total time. Note that this figure for parameters choices IV and V is about 20%, because sampling from $D_{\mathbb{Z}^n,\beta}$ takes most of the total time due to big integer operations involved in the DDLL algorithms. However, the time used for both FFT and sampling from $D_{\mathbb{Z}^n,\beta}$ always takes $> 60\%$ of the total time, which might be the bottleneck of our AKE protocol.

Finally, we have not yet really optimized our implementation, and the performance of our protocol can be greatly improved by using better FFT algorithms (e.g., parallel FFT [10, 43]) and sampling algorithms, or any other optimizations. For example, when we switch from the case $\log(\beta/\alpha) = 28$ (i.e., parameters choice VI) to the case $\log(\beta/\alpha) = 36$ (i.e., parameters choice V), we have to use the "ZZ" data type in NTL instead of the 64-bit single precision integer, which makes the DDLL algorithm (i.e., sampling from $D_{\mathbb{Z}^n,\beta}$) significantly slower. In addition, parallelization can substantially improve our implementation.

5. Security Proof

Theorem 8. *Let n be a power of 2, prime $q = 2^{\omega(\log n)}$ satisfying $q = 1 \mod 2n$, $\alpha/\beta = 2^{-\omega(\log n)}$, $\gamma \approx n^{2/3}$. Then, if RLWE$_{q,\alpha}$ is hard, the proposed AKE is secure with respect to Definition 2 in the random oracle model.*

The intuition behind our proof is quite simple. Since the public element a and the public key of each party (e.g., $p_i = as_i + 2e_i$) actually consist of a RLWE$_{q,\alpha}$ tuple with Gaussian parameter α (scaled by 2), the parties' static public keys are computationally indistinguishable from uniformly distributed elements in R_q under the RLWE assumption. Similarly, both the exchanged elements x_i and y_j are also computationally indistinguishable from uniformly distributed elements in R_q under the RLWE$_{q,\beta}$ assumption.

Without loss of generality, we take party j as an example to check the distribution of the session key. Note that if k_j is uniformly distributed over R_q, we have σ_j is statistically close to uniform over $\{0,1\}^n$ even conditioned on w_j by Lemma 6. Since H_2 is a random oracle, we have that sk_j is uniformly distributed over $\{0,1\}^\kappa$ as expected. Now, let's check the distribution of $k_j = (p_i c + x_i)(s_j d + r_j) + 2g_j$. As one can imagine, we want to establish the randomness of k_j based on pseudorandomness of "RLWE samples" with public element $\hat{a} = p_i c + x_i$, the secret $\hat{s} = s_j d + r_j$, as well as the error term $2g_j$. Informally, we will prove that k_j is statistically close to a real $\text{RLWE}_{q,\beta}$ instance with both the secret and the error chosen from χ_β by using the following two facts: 1) $p_i c + x_i$ is uniformly distributed over R_q whenever p_i or x_i is uniform (if c is invertible in R_q, which is guaranteed by Lemma 4); 2) $s_j d + r_j$ has distribution statistically close to χ_β, since when $\alpha/\beta = 2^{-\omega(\log n)}$, the distribution of $r_j \leftarrow_r \chi_\beta$ statistically hides the term $s_j d$ according to Lemma 3 (recall that $s_j \leftarrow_r \chi_\alpha$, and $d \leftarrow_r \chi_\gamma$).

Formally, let N be the maximum number of parties, and m be maximum number of sessions for each party. We distinguish the following five types of adversaries:

Type I: $\text{sid}^* = (\Pi, I, i^*, j^*, x_{i^*}, (y_{j^*}, w_{j^*}))$ is the test session, and y_{j^*} is output by a session activated at party j by a $\mathsf{Send}_1(\Pi, R, j^*, i^*, x_{i^*})$ query.

Type II: $\text{sid}^* = (\Pi, I, i^*, j^*, x_{i^*}, (y_{j^*}, w_{j^*}))$ is the test session, and y_{j^*} is **not** output by a session activated at party j^* by a $\mathsf{Send}_1(\Pi, R, j^*, i^*, x_{i^*})$ query.

Type III: $\text{sid}^* = (\Pi, R, j^*, i^*, x_{i^*}, (y_{j^*}, w_{j^*}))$ is the test session, and x_{i^*} is **not** output by a session activated at party i^* by a $\mathbf{Send}_0(\Pi, I, i^*, j^*)$ query.

Type IV: $\text{sid}^* = (\Pi, R, j^*, i^*, x_{i^*}, (y_{j^*}, w_{j^*}))$ is the test session, and x_{i^*} is output by a session activated at party i^* by a $\mathbf{Send}_0(\Pi, I, i^*, j^*)$ query, but i^* either never completes the session, or i^* completes it with exact y_{j^*}.

Type V: $\text{sid}^* = (\Pi, R, j^*, i^*, x_{i^*}, (y_{j^*}, w_{j^*}))$ is the test session, and x_{i^*} is output by a session activated at party i^* by a $\mathbf{Send}_0(\Pi, I, i^*, j^*)$ query, but i^* completes the session with another $y'_j \neq y_{j^*}$.

The partition of the five adversaries are very similar to that in [30], and give a complete partition of all the adversaries. The weak perfect forward secrecy (wPFS) is captured by allowing **Type I** and **Type IV** adversaries to obtain the static secret keys of both party i^* and j^* by using $\mathsf{Corrupt}$ queries. Since sid^* definitely has no matching session for **Type II**, **Type III**, and **Type IV** adversaries, no corruption to either party i^* or party j^* is allowed by Definition 1. The security proofs for the five types of adversaries are similar, except the forking lemma [2] is involved for **Type II**, **Type III**, and **Type IV** adversaries by using the assumption that H_1 is a random oracle. Informally, the adversary must first "commit" x_i (or y_j) before seeing c (or d), thus it cannot determine the value $p_i c + x_i$ (or $p_j d + y_i$) in advance (but the simulator can determine the values by programming H_1 when it tries to embed RLWR instances with respect to either $p_i c + x_i$ or $p_j d + y_i$ as discussed before).

5.1 Type I Adversary

In this subsection, we prove that our AKE is secure against any PPT **Type I** adversary $\mathcal{A}$.

Lemma 9. *If $\alpha/\beta = 2^{-\omega(\log n)}$ and $\text{RLWE}_{q,\alpha}$ is hard, the proposed AKE is secure against any PPT* ***Type I*** *adversary $\mathcal{A}$ in the random oracle model.*

Proof. We prove this lemma via a sequence of games $G_{1,l}$ for $0 \leq l \leq 4$. Boxes are used to highlight the changes of each game with respect to its previous game.

Game $G_{1,0}$ $\mathcal{S}$ chooses $i^*, j^* \leftarrow_r \{1, \ldots, N\}$, $s_{i^*}, s_{j^*} \leftarrow_r \{1, \ldots, m\}$, and hopes that the adversary will use $\text{sid}^* = (\Pi, I, i^*, j^*, x_{i^*}, (y_{j^*}, w_{j^*}))$ as the test session, where x_{i^*} is output by the s_{i^*}-th session of party i^*, and y_{j^*} is output by the s^*_j-th session of party j^* activated by a $\mathsf{Send}_1(\Pi, R, j^*, i^*, x_{i^*})$ query. Then, $\mathcal{S}$ chooses $a \leftarrow_r R_q$, generates static public keys for all parities (by choosing $s_i, e_i \leftarrow_r \chi_\alpha$), and simulates the security game for $\mathcal{A}$. Specifically, $\mathcal{S}$ maintains two tables L_1, L_2 for the random oracles H_1, H_2 respectively, and answers the queries from $\mathcal{A}$ as follows:

- $H_1(in)$: If there doesn't exist a tuple (in, out) in L_1, choose an element $out \leftarrow_r \chi_\gamma$, and add (in, out) into L_1. Then, return out to $\mathcal{A}$.
- $H_2(in)$ queries: If there doesn't exist a tuple (in, out) in L_2, choose a vector $out \leftarrow_r \{0,1\}^\kappa$, and add (in, out) into L_2. Then, return out to $\mathcal{A}$.
- $\mathsf{Send}_0(\Pi, I, i, j)$: $\mathcal{A}$ activates a new session of i with intended party j, $\mathcal{S}$ chooses $r_i, f_i \leftarrow_r \chi_\beta$, and returns $x_i = ar_i + 2f_i \in \mathbb{Z}_q^{n \times n}$ to $\mathcal{A}$.
- $\mathsf{Send}_1(\Pi, R, j, i, x_i)$: $\mathcal{S}$ chooses $r_j, f_j \leftarrow_r \chi_\beta$, and honestly computes $y_j = ar_j + 2f_j \in R_q$, k_j, w_j, and sk_j following the protocol. Finally, return (y_j, w_j) to $\mathcal{A}$.
- $\mathsf{Send}_2(\Pi, I, i, j, x_i, (y_j, w_j))$: $\mathcal{S}$ computes k_i and sk_i by using r_i and s_i following the protocol.
- $\mathsf{SessionKeyReveal}(sid)$: Let $sid = (\Pi, *, i, *, *, *, *)$, $\mathcal{S}$ returns sk_i if the session key of sid has been generated.
- $\mathsf{Corrupt}(i)$: Return the static secret key s_i of i to $\mathcal{A}$.
- $\mathsf{Test}(sid)$: Let $sid = (\Pi, I, i, j, x_i, (y_j, w_j))$, $\mathcal{S}$ aborts if $(i, j) \neq (i^*, j^*)$, or x_i and y_j are not output by the s_{i^*}-th session of i^* and the s^*_j-th session of j^* respectively. Else, $\mathcal{S}$ chooses $b \leftarrow_r \{0,1\}$, returns $sk'_i \leftarrow_r \{0,1\}^\kappa$ if $b = 0$. Otherwise, return the session key sk_i of sid.

Claim 1. *The probability that $\mathcal{S}$ will not abort in $G_{1,0}$ is at least $\frac{1}{m^2 N^2}$.*

Proof. This claim directly follows from the fact that $\mathcal{S}$ randomly chooses $i^*, j^* \leftarrow_r \{1, \dots, N\}$ and $s_{i^*}, s_j^* \leftarrow_r \{1, \dots, m\}$ independently from the view of $\mathcal{A}$. $\square$

Game $G_{1,1}$ $\mathcal{S}$ first computes $\boxed{y_j' = ar_j' + 2f_j'}$, where $r_j', f_j' \leftarrow_r \chi_\beta$. Then, it behaves almost the same as in $G_{1,0}$, except in the following case:

- $\mathsf{Send}_1(\Pi, R, j, i, x_i)$: If $(i, j) \neq (i^*, j^*)$, or it is not the s_j^*-th session of j^*, $\mathcal{S}$ answers the query as in Game $G_{1,0}$. Otherwise, choose $d \leftarrow_r \chi_\gamma$, and compute $\boxed{y_j = y_j' - p_j d}$. $\mathcal{S}$ aborts the simulation if there is a tuple $((j, i, y_j, x_i), *)$ in L_1. Else, it adds $((j, i, y_j, x_i), d)$ into L_1, and computes $\boxed{k_j = (p_i c + x_i) r_j' + 2g_j}$, where $c = H_1(i, j, x_i)$ and $g_j \leftarrow_r \chi_\beta$. Finally, it derives w_j and sk_j following the protocol, and sends (y_j, w_j) to $\mathcal{A}$.

In the following, we denote $F_{1,l}$ as the event that $\mathcal{A}$ outputs a guess b' that equals to b in Game $G_{1,l}$.

Claim 2. *If* $\alpha/\beta = 2^{-\omega(\log n)}$ *and RLWE*$_{q,\alpha}$ *is hard, then* $\Pr[F_{1,l}] = \Pr[F_{1,0}] - negl(n)$.

Proof. First, we show that $\mathcal{S}$ aborts with negligible probability in Game $G_{1,1}$ for the reason that $\mathcal{A}$ has made a H_1 query with (j, i, y_j, x_i) before seeing y_j. Actually, if $\mathcal{A}$ can make the query before seeing y_j with non-negligible probability, we can construct an algorithm $\mathcal{B}$ that breaks the RLWE$_{q,\alpha}$ assumption. Formally, after giving a challenge RLWE (u_1, v_1) tuple with error distribution χ_α (scaled by multiplying $t = 2$), $\mathcal{B}$ sets $a = u_1$ and $y_j' = v_1$, simulates the attack environment for $\mathcal{A}$ as in Game $G_{1,1}$ until it computes $y_j = y_j' - p_j d$. Then, if checks if there is a tuple $((j, i, y_j, x_i), *)$ in L_1. If yes, it returns 1 and aborts, else it returns 0 and aborts. Note that if (u_1, v_1) is a LWE tuple, we have $\mathcal{A}$ has the same view as in $G_{1,1}$ until the point that $\mathcal{B}$ computes y_j, thus that probability $\mathcal{A}$ will make the H_1 query with (j, i, y_j, x_i) is non-negligible. While if (u_1, v_1) is a uniformly random tuple in $R_q \times R_q$, we have y_j is uniformly random over R_q, the probability that $\mathcal{A}$ will make the H_1 query with (j, i, y_j, x_i) is non-negligible.

Second, conditioned on that $\mathcal{A}$ will not make a H_1 query with (j, i, y_j, x_i) before seeing y_j (i.e., $\mathcal{B}$ will not abort for this reason), we show that $G_{1,1}$ is statistically close to $G_{1,0}$. Since $p_j = as_j + 2e_j \in R_q$ with $s_j, e_j \leftarrow_r \chi_\alpha$, we have $y_j = a(r_j' - s_j d) + 2(f_j' - e_j d)$. By Lemma 1 and Lemma 2, each entry in both $s_j d$ and $e_j d$ has size at most $\tau = \alpha\gamma n\sqrt{n}$, and $\tau/\beta = negl(n)$ (since $\alpha/\beta = 2^{-\omega(\log n)}$ and $\gamma \approx n^{2/3}$). Thus, both $r_j' - s_j d$ and $f_j' - e_j d$ have distribution negligibly close to χ_β by Lemma 3. This implies that the distribution of y_j in Game $G_{1,1}$ is statistically close to that in Game $G_{1,0}$. This shows that $G_{1,1}$ is statistically close to $G_{1,0}$.

In all, we have $G_{1,1}$ is computationally indistinguishable from $G_{1,0}$, which completes the proof. $\square$

Game $G_{1,2}$ $\mathcal{S}$ first computes $\boxed{x_i' = ar_i' + 2f_i'}$, where $r_i', f_i' \leftarrow_r \chi_\beta$. Then, it behaves almost the same as in $G_{1,1}$, except for the following cases:

- $\mathsf{Send}_0(\Pi, I, i, j)$: If $(i, j) \neq (i^*, j^*)$, or it is not the s_{i^*}-th session of i^*, $\mathcal{S}$ answers as in Game $G_{1,1}$. Otherwise, choose $c \leftarrow_r \chi_\gamma$, and compute $\boxed{x_i = x_i' - p_i c}$. $\mathcal{S}$ aborts if there is a tuple $((i, j, x_i), *)$ in L_1, else it adds $((i, j, x_i), c)$ into L_1. Finally, it returns x_i to $\mathcal{A}$.
- $\mathsf{Send}_2(\Pi, I, i, j, x_i, (y_j, w_j))$: If $(i, j) \neq (i^*, j^*)$, or it is not the s_{i^*}-th session of i^*, $\mathcal{S}$ behaves as in Game $G_{1,1}$. Otherwise, if (y_j, w_j) is output by the s_j^*-th session of party j^*, $\mathcal{S}$ sets $\boxed{sk_i = sk_j}$, where sk_j is the session key of $sid = (\Pi, R, j, i, x_i, (y_j, w_j))$. Else, $\mathcal{S}$ computes $\boxed{k_i = (p_j d + y_j) r_i' + 2g_i}$, and derives sk_i following the protocol, where $d = H_1(j, i, y_j, x_i)$ and $g_i \leftarrow_r \chi_\beta$.

Claim 3. *If* $\alpha/\beta = 2^{-\omega(\log n)}$ *and RLWE*$_{q,\alpha}$ *is hard, then* $\Pr[F_{1,2}] = \Pr[F_{1,1}] - negl(n)$.

Proof. The proof that the distribution of x_i is statistically close to that in Game $G_{1,1}$ is the same as the proof of Claim 2, and $\mathcal{A}$ will make a H_1 query with x_i with negligible probability, so the probability that $\mathcal{S}$ aborts in $G_{1,2}$ is negligibly close to that of $G_{1,1}$. Combining this with the correctness of our AKE protocol, this claim follows. $\square$

Game $G_{1,3}$ $\mathcal{S}$ chooses $\boxed{x_i' \leftarrow_r R_q}$, and behaves almost the same as in $G_{1,2}$ except in the following case:

- $\mathsf{Send}_2(\Pi, I, i, j, x_i, (y_j, w_j))$: If $(i, j) \neq (i^*, j^*)$, or it is not the s_{i^*}-th session of i^*, or (y_j, w_j) is output by the s_j^*-th session of party j^*, $\mathcal{S}$ behaves as in Game $G_{1,2}$. Else, choose $\boxed{sk_i \leftarrow_r \{0,1\}^\kappa}$ as the session key.

Note that we change the real session key sk_i in Game $G_{1,2}$ with a uniformly chosen one in Game $G_{1,3}$, when (y_j, w_j') is output by the s_j^*-th session of party j^* but $w_j \neq w_j'$. Ideally, the adversary will not be aware of such a difference if it does not make a query to H_2 with the exact σ_i derived from k_i (since H_2 is a random oracle). However, we cannot prove this claim immediately for technical reasons. Instead, we have to employ the "deferred analysis" proof technique in [20], which informally allows us to proceed the security games by patiently postponing some tough probability analysis to a later game, "where it will be much easier" [20]. Specially, denote $Q_{1,l}$ as the event that in Game $G_{1,l}$ $\mathcal{A}$ makes a query to H_2 with σ_i for the s_{i^*}-th session of party i^*, when (y_j, w_j') is output by the s_j^*-th session of party j^* but $w_j \neq w_j'$, where $l = 2, 3, 4$. For our purpose, we will show that 1) $\Pr[Q_{1,2}] \approx \Pr[Q_{1,3}] \approx \Pr[Q_{1,4}]$, and 2) $\Pr[Q_{1,4}]$ is negligible in n.

Claim 4. *If RLWE*$_{q,\alpha}$ *is hard,* $\Pr[Q_{1,3}] = \Pr[Q_{1,2}] - negl(n)$*, and* $\Pr[F_{1,3}|\neg Q_{1,3}] = \Pr[F_{1,2}|\neg Q_{1,2}] - negl(n)$.

Proof. Note that H_2 is a random oracle, the event $Q_{1,2}$ is independent from the distribution of the corresponding sk_i. Namely, no matter whether or not $\mathcal{A}$ obtains sk_i, $\Pr[Q_{1,2}]$ is the same, which also holds for $\Pr[Q_{1,3}]$. In particular, under the RLWE$_{q,\alpha}$ assumption we have that the public information (i.e., static public keys and public transcripts) in $G_{1,2}$ and $G_{1,3}$ is computationally indistinguishable, and that $\Pr[Q_{1,3}] = \Pr[Q_{1,2}] - negl(n)$. Besides, if $\Pr[Q_{1,l}]$ for $l = 2, 3$ does not happen, the distribution of sk_i is the same in both games. In other words, $\Pr[F_{1,3}|\neg Q_{1,3}] = \Pr[F_{1,2}|\neg Q_{1,2}] - negl(n)$. □

Game $G_{1,4}$ $\mathcal{S}$ chooses $\boxed{y'_j \leftarrow_r R_q}$, and behaves almost the same as in $G_{1,3}$ except in the following case:

- $\mathsf{Send}_1(\Pi, R, j, i, x_i)$: If $(i, j) \neq (i^*, j^*)$, or it is not the s^*_j-th session of j^*, $\mathcal{S}$ answers as in Game $G_{1,3}$. Otherwise, choose $d \leftarrow_r \chi_\gamma$, and compute $y_j = y'_j - p_j d$. $\mathcal{S}$ aborts if there is a tuple $((j, i, y_j, x_i), *)$ in L_1. Else, it adds $((j, i, y_j, x_i), d)$ into L_1. Then, $\mathcal{S}$ chooses $\boxed{k_j \leftarrow_r R_q}$, and derives w_j, sk_j following the protocol. Finally, it returns (y_j, w_j) to $\mathcal{A}$.

Claim 5. *Under the RLWE$_{q,\beta}$ assumption, Game $G_{1,3}$ and $G_{1,4}$ are computationally indistinguishable. In particular, we have* $\Pr[Q_{1,4}] = \Pr[Q_{1,3}]$, *and* $\Pr[F_{1,4}|\neg Q_{1,4}] = \Pr[F_{1,3}|\neg Q_{1,3}] - negl(n)$.

Proof. Let $(u_1, v_1), (u_2, v_2)$ be two challenge RLWE tuples with error distribution χ_β (scaled by multiplying $t = 2$). Assume there is an adversary that distinguishes Game $G_{1,3}$ and $G_{1,4}$, we now construct a distinguisher $\mathcal{D}$ that solves the RLWE problem. Specifically, $\mathcal{D}$ first sets public parameter $a = u_1$, and $x'_i = u_2$ and $y'_j = v_1$. Then, it behaves the same as $\mathcal{S}$ in Game $G_{1,3}$, except for the following:

- $\mathsf{Send}_1(\Pi, R, j, i, x_i)$: If $(i, j) \neq (i^*, j^*)$, or it is not the s^*_j-th session of j^*, $\mathcal{D}$ answers as in Game $G_{1,3}$. Else, choose $d \leftarrow_r \chi_\gamma$, compute $\boxed{y_j = y'_j - p_j d}$, and aborts if there is a tuple $((j, i, y_j, x_i), *)$ in L_1. Otherwise, it adds $((j, i, y_j, x_i), d)$ into L_1. Then, $\mathcal{D}$ sets $\boxed{k_j = v_2}$, and derives w_j, sk_j following the protocol. Finally, it sends (y_j, w_j) to $\mathcal{A}$.

Note that if $(u_1, v_1), (u_2, v_2)$ are RLWE tuples for some secret r'_j, i.e., $v_1 = u_1 r'_j + 2f'_j = a r'_j + 2f'_j$ and $v_2 = u_2 r'_j + 2g_j$, we have that $k_j = v_2 = x'_i r'_j + 2g_j = (p_i c + x_i) r'_j + 2g_j$, and the view of $\mathcal{A}$ is the same as in $G_{1,3}$. Otherwise, the view of $\mathcal{A}$ is the same as in Game $G_{1,4}$. □

Claim 6. $\Pr[Q_{1,4}] = negl(n)$

Proof. Let (y_j, w_j) be output by the s^*_j-th session of party $j = j^*$, (y_j, w'_j) be the message that is used to complete the test session (i.e., the s_{i^*}-th session of party $i = i^*$). Note that in $G_{1,4}$, $k_j \leftarrow_r R_q$ is chosen independently of both the public keys and transcripts (except w_j). In particular, it is also independent of sk_j (which the adversary might obtain via a $\mathsf{SessionKeyReveal}$ query), since H_2 is a random oracle and sk_j is randomly chosen. Let k_i be the element "computed" by $\mathcal{S}$. By the correctness of the protocol, we have $k_i = k_j + \hat{g}$ for some $\hat{g}$ with small coefficients in Game $G_{1,3}$. Since the adversary cannot distinguish the ways how $\mathcal{S}$ "computes" k_j in Game $G_{1,3}$ and Game $G_{1,4}$ by Claim 5, we can assume that the equation $k_i = k_j + \hat{g}$ still holds in Game $G_{1,4}$ in the adversary's view. Note that k_j is randomly chosen from R_q, and the adversary can only obtain the information of k_j from the public w_j, the dependence of $\hat{g}$ on k_j should be totally determined by the information of w_j. Thus, we have that $\sigma'_i = \mathsf{Mod}_2(k_i, w'_j) = \mathsf{Mod}_2(k_j + \hat{g}, w'_j)$ conditioned on w_j is statistically close to $\{0,1\}^n$ according to Lemma 6. In other words, the probability that the adversary makes a query $H_2(i, j, x_i, y_j, w'_j, \sigma'_i)$ is at most $2^{-n} + negl(n)$, which is negligible in n. This completes the proof. □

Claim 7. $\Pr[F_{1,4}|\neg Q_{1,4}] = 1/2 + negl(n)$

Proof. Let (y_j, w_j) be output by the s^*_j-th session of party $j = j^*$, (y_j, w'_j) be the message that is used to complete the test session (i.e., the s_{i^*}-th session of party $i = i^*$). We distinguish the following two cases:

- $w_j = w'_j$: In this case, we have $sk_i = sk_j = H_2(i, j, x_i, y_j, w_j, \sigma_j)$, where $\sigma_j = \mathsf{Mod}_2(k_j, w_j)$. Note that in $G_{1,4}$, k_j is randomly chosen from the uniform distribution over R_q, we have σ_j (conditioned on w_j) is statistically close to uniform distribution over $\{0,1\}^n$ in the adversary's view according to Lemma 6. Thus, the probability that $\mathcal{A}$ has made a H_2 query with σ_i is less than $2^{-n} + negl(n)$.
- $w_j \neq w'_j$: By assumption that $Q_{1,4}$ does not happen, we have $\mathcal{A}$ will never make a H_2 query with σ_i.

In all, the probability that $\mathcal{A}$ has made a H_2 query with σ_i is negligible. This claim follows from the fact that if the adversary doesn't make a query with σ_i exactly, the distribution of sk_i is uniform over $\{0,1\}^k$ due to the random oracle property of H_2. Combining the result in Claim 1∼7, we have that Lemma 9 follows. □

5.2 Type II Adversary

In this subsection, we prove that our AKE is secure against any PPT **Type II** adversary $\mathcal{A}$.

Lemma 10. *If $\alpha/\beta = 2^{-\omega(\log n)}$ and LWE$_{q,n,\alpha}$ is hard, the proposed AKE is secure against any PPT **Type II** adversary $\mathcal{A}$ in the random oracle model.*

Proof. We prove this lemma via a sequence of games $G_{2,l}$ for $0 \leq l \leq 6$.

Game $G_{2,0}$. $\mathcal{S}$ chooses $i^*, j^* \leftarrow_r \{1, \ldots, N\}$ and $s_{i^*} \leftarrow_r \{1, \ldots, m\}$, and hopes that the adversary will choose $sid^* = (\Pi, I, i^*, j^*, x_{i^*}, (y_{j^*}, w_{j^*}))$ as the test session, where x_{i^*} is output by the s_{i^*}-th session of party i^* with intended party j^* (note that sid^* has no matching session for **Type II** adversary). Then, $\mathcal{S}$ chooses $a \leftarrow_r R_q$, honestly generates static public keys for all parities (by randomly choosing s_i and e_i from χ_α), and simulates the attack environment for $\mathcal{A}$. Specifically, $\mathcal{S}$ maintains two tables L_1, L_2 for the random oracles H_1, H_2 respectively, and answers the queries from $\mathcal{A}$ as follows:

– $H_1(in)$: If there doesn't exist a tuple (in, out) in the L_1 list, choose an element $out \leftarrow_r \chi_\gamma$, and add (in, out) to the L_1 list. Then, return out to $\mathcal{A}$.

– $H_2(in)$ queries: If there doesn't exist a tuple (in, out) in the L_2 list, choose an element $out \leftarrow_r \{0,1\}^k$, and add (in, out) to the L_2 list. Then, return out to $\mathcal{A}$.

– $\mathbf{Send}_0(\Pi, I, i, j)$: $\mathcal{A}$ initiates a new session of i with intended partner j, $\mathcal{S}$ chooses $r_i, f_i \leftarrow_r \chi_\beta$, returns $x_i = ar_i + 2f_i \in \mathbb{Z}_q^{n\times n}$ to $\mathcal{A}$ on behalf of i.

– $\mathbf{Send}_1(\Pi, R, j, i, x_i)$: $\mathcal{S}$ chooses $r_j,\ f_j \leftarrow_r \chi_\beta$, and honestly computes $y_j = ar_j + 2f_j \in R_q$, k_j, w_j, and sk_j following the protocol. Finally, return (y_j, w_j) to $\mathcal{A}$.

– $\mathbf{Send}_2(\Pi, I, i, j, x_i, (y_j, w_j))$: $\mathcal{S}$ computes k_i and sk_i by using r_i and s_i following the protocol.

– $\mathbf{SessionKeyReveal}(sid)$: Let $sid = (\Pi, *, i, *, *, *, *)$, $\mathcal{S}$ returns sk_i if the session key of sid has been generated.

– $\mathbf{Corrupt}(i)$: Return the static secret key s_i of i to $\mathcal{A}$.

– $\mathbf{Test}(sid)$: Let $sid = (\Pi, I, i, j, x_i, (y_j, w_j))$, if $(i,j) \neq (i^*, j^*)$, or x_i and y_j are not output by the s_{i^*}-th session of i^* and the s_j^*-th session of j^* respectively, $\mathcal{S}$ aborts. Otherwise, $\mathcal{S}$ chooses $b \leftarrow_r \{0,1\}$ and $sk_i' \leftarrow_r \{0,1\}^k$. If $b = 0$, $\mathcal{S}$ returns sk_i', else it returns the real session sk_i of sid.

Claim 8. *The probability that $\mathcal{S}$ will not abort in $G_{2,0}$ is at least $\frac{1}{mN^2}$.*

Proof. This claim directly follows from the fact that $\mathcal{S}$ randomly chooses $i^*, j^* \leftarrow_r \{1, \dots, N\}$ and $s_{i^*} \leftarrow_r \{1, \dots, m\}$ without $\mathcal{A}$ knowing it. □

Game $G_{2,1}$. $\mathcal{S}$ behaves almost the same as in $G_{2,0}$, except in the following cases:

– $\mathbf{Send}_0(\Pi, I, i, j)$: If $i \neq j^*$, $\mathcal{S}$ answers the query as in Game $G_{2,1}$. Else, $\mathcal{S}$ computes $\boxed{x_i' = ar_i' + 2f_i'}$, where $r_i', f_i' \leftarrow_r \chi_\beta$. Then, it chooses $c \leftarrow_r \chi_\gamma$, and computes $\boxed{x_i = x_i' - p_i c}$. If there is a tuple $((i, j, x_i), *)$ in L_1 list, $\mathcal{S}$ aborts the simulation. Else, it adds $((i, j, x_i), c)$ into L_1, and returns x_i to $\mathcal{A}$.

– $\mathbf{Send}_1(\Pi, R, j, i, x_i)$: If $j \neq j^*$, $\mathcal{S}$ answers the query as in Game $G_{2,0}$. Else, $\mathcal{S}$ computes $\boxed{y_j' = ar_j' + 2f_j'}$, where $r_j', f_j' \leftarrow_r \chi_\beta$. Then, choose $d \leftarrow_r \chi_\gamma$, and compute $\boxed{y_j = y_j' - p_j d}$. If there is a tuple $((j, i, y_j, x_i), *)$ in the L_1 list, $\mathcal{S}$ aborts. Else, $\mathcal{S}$ adds $((j, i, y_j, x_i), d)$ into the L_1 list, and computes $\boxed{k_j = (p_i c + x_i) r_j' + 2g_j}$, where $c = H_1(i, j,\ x_i)$ and $g_j \leftarrow_r \chi_\beta$. Finally, it computes w_j and sk_j following the protocol, and sends (y_j, w_j) to $\mathcal{A}$.

– $\mathbf{Send}_2(\Pi, I, i, j, x_i, (y_j, w_j))$: If $i \neq j^*$, $\mathcal{S}$ answers the query as in Game $G_{2,1}$. Otherwise, let $x_i = x_i' - p_i c$ for $x_i' = ar_i' + 2f_i'$, the simulator $\mathcal{S}$ computes $\boxed{k_i = (p_j d + y_j) r_i' + 2g_i}$, where $g_i \leftarrow_r \chi_\beta$. Finally, $\mathcal{S}$ computes sk_i following the protocol.

In the following, let $F_{2,l}$ denote the event that $\mathcal{A}$ outputs a guess b' that equals to b in Game $G_{2,l}$.

Claim 9. *If $\alpha/\beta = 2^{-\omega(\log n)}$ and LWE$_{q,n,\alpha}$ is hard, then* $\Pr[F_{2,1}] = \Pr[F_{2,0}] - negl(n)$.

Proof. The proof is similar to Claim 2, we omit the details. □

Game $G_{2,2}$. $\mathcal{S}$ behaves almost the same as in $G_{2,1}$, except it replaces the public key for party j^* with a uniformly chosen $\boxed{p_{j^*} \leftarrow_r R_q}$.

Claim 10. *If LWE$_{q,n,\alpha}$ is hard, then* $\Pr[F_{2,2}] = \Pr[F_{2,1}] - negl(n)$.

Proof. Since the only difference between $G_{2,1}$ and $G_{2,2}$ is that $\mathcal{S}$ replaces $p_{j^*} = as_{j^*} + 2e_{j^*}$ in $G_{2,1}$ with a randomly chosen over R_q in $G_{2,2}$, an adversary that can distinguish the difference between $G_{2,1}$ and $G_{2,2}$ could be directly used to solve the LWE$_{q,n,\alpha}$ problem. □

Game $G_{2,3}$. $\mathcal{S}$ first computes $\boxed{x_i' = ar_i' + 2f_i'}$, where $r_i', f_i' \leftarrow_r \chi_\beta$. Then, it behaves almost the same as in $G_{2,2}$, except in the following cases:

– $\mathbf{Send}_0(\Pi, I, i, j)$: If $(i,j) \neq (i^*, j^*)$, or it is not the s_{i^*}-th session of i^*, $\mathcal{S}$ answers the query as in Game $G_{2,2}$. Otherwise, $\mathcal{S}$ chooses $c \leftarrow_r \chi_\gamma$, and computes $\boxed{x_i = x_i' - p_i c}$. If there is a tuple $((i, j, x_i), *)$ in L_1 list, $\mathcal{S}$ aborts the simulation. Else, it adds $((i, j, x_i), c)$ into L_1, and returns x_i to $\mathcal{A}$.

– $\mathbf{Send}_2(\Pi, I, i, j, x_i, (y_j, w_j))$: If $(i,j) \neq (i^*, j^*)$, or it is not the s_{i^*}-th session of i^*, $\mathcal{S}$ answers the query as in Game $G_{2,2}$. Otherwise, the simulator $\mathcal{S}$ computes $\boxed{k_i = (p_j d + y_j) r_i' + 2g_i}$, where $d = H_1(j, i, y_j, x_i)$ and $g_i \leftarrow_r \chi_\beta$. Finally, it computes sk_i following the protocol.

Claim 11. *If $\alpha/\beta = 2^{-\omega(\log n)}$ and LWE$_{q,n,\alpha}$ is hard, then* $\Pr[F_{2,3}] = \Pr[F_{2,2}] - negl(n)$.

Proof. The proof is similar to Claim 2, we omit the details. □

Game $G_{2,4}$. $\mathcal{S}$ first computes $v_1 = ar_i' + 2\tilde{f}_i'$, $v_2 = p_j r_i' + t\tilde{e}_i'$ where $r_i' \leftarrow_r \chi_\beta$, and $\tilde{f}_i', \tilde{e}_i' \leftarrow \chi_\alpha$. Then, it computes $\boxed{x_i' = v_1 + 2f_i' = ar_i' + 2(\tilde{f}_i' + f_i')}$ where $f_i' \leftarrow_r \chi_\beta$.

Finally, it behaves almost the same as in $G_{2,3}$ except in the following case:

- $\mathbf{Send}_2(\Pi, I, i, j, x_i, (y_j, w_j))$: If $(i, j) \neq (i^*, j^*)$, or it is not the s_{i^*}-th session of i^*, $\mathcal{S}$ answers the query as in Game $G_{2,3}$. Otherwise, the simulator $\mathcal{S}$ computes $\boxed{k_i = dv_2 + y_j r_i' + 2g_i} = (p_j d + y_j)r_i' + 2(d\tilde{e}_i' + g_i)$, where $d = H_1(j, i, y_j, x_i)$ and $g_i \leftarrow_r \chi_\beta$. Finally, it computes sk_i following the protocol.

Claim 12. *If $\alpha/\beta = 2^{-\omega(\log n)}$ and $LWE_{q,n,\alpha}$ is hard, then* $\Pr[F_{2,4}] = \Pr[F_{2,3}] - negl(n)$.

Proof. In Game $G_{2,4}$, we have $x_i' = ar_i' + 2(\tilde{f}_i' + f_i')$ and $k_i = (p_j d + y_j)r_i' + 2(d\tilde{e}_i' + g_i)$, where $\tilde{e}_i', \tilde{f}_i' \leftarrow \chi_\alpha$ and $f_i', g_i \leftarrow_r \chi_\beta$. By Lemma 3, the distributions of both $\tilde{f}_i' + f_i'$ and $d\tilde{e}_i' + g_i$ are statistically close to χ_β. This claim follows. □

Game $G_{2,5}$. $\mathcal{S}$ behaves almost the same as in $G_{2,4}$ except in the following case:

- $\mathbf{Send}_2(\Pi, I, i, j, x_i, (y_j, w_j))$: If $(i, j) \neq (i^*, j^*)$, or it is not the s_{i^*}-th session of i^*, $\mathcal{S}$ answers the query as in Game $G_{2,4}$. Else, $\mathcal{S}$ chooses $\boxed{k_i \leftarrow_r R_q}$ and computes sk_i following the protocol.

Note that the only difference between $G_{2,4}$ and $G_{2,5}$ is that $\mathcal{S}$ replaces the real $k_i = dv_2 + y_j r_i' + 2g_i$ in Game $G_{2,4}$ with a randomly chosen $k_i \in R_q$ in Game $G_{2,5}$. Considering H_2 is a random oracle, such a difference will not affect the view of $\mathcal{A}$ until it makes a H_2 query with σ_i derived from k_i. Formally, denote $Q_{2,l}$ for $l = 4, 5, 6$ as the event that $\mathcal{A}$ makes a H_2 query with σ_i derived from k_i.

Claim 13. $\Pr[Q_{2,5}] = \Pr[Q_{2,4}]$*, and* $\Pr[F_{2,4}|\neg Q_{2,4}] = \Pr[F_{2,5}|\neg Q_{2,5}] = 1/2 + negl(n)$.

Proof. Since H_2 is a random oracle, the event $Q_{2,4}$ is independent from the distribution of the corresponding sk_i. Namely, no matter whether or not $\mathcal{A}$ obtains sk_i, $\Pr[Q_{2,4}]$ is the same, which also holds for $\Pr[Q_{2,5}]$. Besides, if $Q_{2,l}$ for $l = 4, 5$ does not happen, $G_{2,5}$ is actually the same as $G_{2,4}$ in the adversary's view. Especially, the distribution of sk_i is random and uniform over $\{0, 1\}^k$, which means that the advantage of $\mathcal{A}$ in guessing b is negligible, given that the event $Q_{2,5}$ does not happen. □

Note that if $\Pr[Q_{2,5}] \leq negl(n)$, we have already completed the proof. However, it is highly non-trivial to prove such a claim. Actually, though v_2 is pseudorandom in the adversary's view (under the RLWE assumption), we cannot immediately obtain that $k_i = dv_2 + y_j r_i' + 2g_i$ is pseudorandom since $y_j r_i'$ is correlated with dv_2. Fortunately, such a correlation can somehow be removed by the use of the random oracle H_1, which guarantees that the adversary must first commit y_j before seeing the random element d (i.e., by making a corresponding random oracle query). In particular, if we program the corresponding H_1 query with another randomly chosen $\tilde{d}$ and obtain $k_i' = \tilde{d}v_2 + y_j r_i' + 2g_i$, we have $k_i' = k_i + (\tilde{d} - d)v_2$. In other words, we have $(\tilde{d} - d)v_2 = (k_i' - k_i)$. Intuitively, if the adversary can distinguish k_i (and k_i') from a uniformly chosen one, it can distinguish v_2 (which is computationally hidden under the RLWE assumption) from a randomly chosen element in R_q.

Now, we formally show that $Q_{2,5}$ will happen with negligible probability, which makes use of the Forking Lemma [2]. Let $sid^* = (\Pi, I, i^*, j^*, x_i, (y_j, w_i))$ be the test session. By our assumption that $\mathcal{A}$ is a **Type II** adversary, namely, y_j is not output by party j^* in response to a $\mathbf{Send}_1(\Pi, R, j^*, i^*, x_i)$ query. In other words, $\mathcal{S}$ does not make a H_1 hash query $H_1(j^*, i^*, y_j, x_i)$ by itself in producing y_j. Given $v_1 = ar_i' + 2\tilde{f}_i'$, $v_2 = p_j r_i' + t\tilde{e}_i'$, and $g_i \leftarrow_r \chi_\beta$ in Game $G_{2,5}$, denote $k_i = dv_2 + y_j r_i' + 2g_i$ (which is the same as that in Game $G_{2,4}$), where $H_1(j^*, i^*, y_j, x_i) = d$. By our assumption, $\mathcal{A}$ will make a H_2 query with σ_i derived from k_i with probability at least $\Pr[Q_{2,5}]$.

Now, fixing v_1, v_2, r_i' and g_i (note that all those values are chosen by $\mathcal{S}$, and are independent from the adversary's behaviors), $\mathcal{S}$ reprograms the hash query $H_1(j^*, i^*, y_j, x_i) = \tilde{d} \neq d$ by using another randomly chosen $\tilde{d} \leftarrow_r \chi_\gamma$, and sets $k_i' = \tilde{d}v_2 + y_j r_i' + 2g_i = k_i + (\tilde{d} - d)v_2$. According to the forking lemma [2], the adversary $\mathcal{A}$ will use the same y_j to complete the test session, and makes a H_2 query with σ_i' derived from k_i' with probability at least $\Pr[Q_{2,5}](\Pr[Q_{2,5}]/q_h - 2^{-n})$, where q_h is maximum number of H_1 queries. Denote by double-$Q_{2,l}$ such an event that, for $l = 5, 6$, $\mathcal{A}$ in Game $G_{2,l}$ will make both σ_i and σ_i' in two runs of $\mathcal{A}$, where σ_i is derived from k_i in the first run of $\mathcal{A}$, and σ_i' is derived from $k_i' = k_i + (\tilde{d} - d)v_2$ in the second run of $\mathcal{A}$. In particular, we have $\Pr[\text{double-}Q_{2,5}] \geq \Pr[Q_{2,5}](\Pr[Q_{2,5}]/q_h - 2^{-n})$. In the following, we will again employ the "deferred analysis" technique [20] and show that $\Pr[\text{double-}Q_{2,5}]$ is negligible.

Game $G_{2,6}$. $\mathcal{S}$ chooses $\boxed{v_1, v_2 \leftarrow_r R_q}$, and behaves almost the same as in $G_{2,5}$.

Claim 14. *Under the $LWE_{q,n,\alpha}$ assumption, Game $G_{2,6}$ is computationally indistinguishable from $G_{2,5}$. In particular,* $\Pr[\textit{double-}Q_{2,6}] = \Pr[\textit{double-}Q_{2,5}] - negl(n)$

Proof. Since the only difference between $G_{2,5}$ and $G_{2,6}$ is that $\mathcal{S}$ replaces $v_1 = ar_i' + 2\tilde{f}_i'$ and $v_2 = p_j r_i' + t\tilde{e}_i'$ with randomly chosen elements in R_q, an adversary that can distinguish the difference between $G_{2,5}$ and $G_{2,6}$ could be directly used to solve the $\text{LWE}_{q,n,\alpha}$ problem. □

Claim 15. $\Pr[\textit{double-}Q_{2,6}] = negl(n)$

Proof. Note that in Game $G_{2,6}$, $\mathcal{S}$ does not really compute k_i and k_i'. (Actually, it cannot compute the values since v_1 and v_2 are uniformly chosen from R_q at random.) Here, we denote k_i and k_i' (i.e., the values determined before and after $\mathcal{S}$ reprograms the H_1 query) as the target values in the $\mathcal{A}$'s view. In particular, the condition $k_i' = k_i + (\tilde{d} - d)v_2$ holds, since $\mathcal{A}$ cannot efficiently distinguish Game $G_{2,6}$ from $G_{2,5}$ by Claim 14. However, since v_2 is uniformly distributed over

R_q and is independent from $\mathcal{A}$'s view (thus is independent from both k_i and k_i'), we have $\sigma_i' = \mathbf{Mod}_2(k_i', w_i')$ is statistically close to uniform over $\{0,1\}^n$ even conditioned on $\sigma_i = \mathbf{Mod}_2(k_i, w_i)$ by Lemma 6. (Note that $(\tilde{d} - d)$ is invertible with overwhelming probability by Lemma 4). Thus, the probability that $\mathcal{A}$ will make a H_2 query with σ_i' is at most $2^{-n} + negl(n)$. In other words, the probability $\Pr[\text{double-}Q_{2,6}] \le 2^{-n} + negl(n)$, which is negligible in n. This completes the proof. □

Combining the result in Claim 8∼15, we have that Lemma 10 follows. □

5.3 Type III Adversary

In this subsection, we prove that our AKE is secure against any PPT **Type III** adversary $\mathcal{A}$.

Lemma 11. *If $\alpha/\beta = 2^{-\omega(\log n)}$ and $LWE_{q,n,\alpha}$ is hard, the proposed AKE is secure against any PPT **Type III** adversary $\mathcal{A}$ in the random oracle model.*

Proof. We prove this lemma via a sequence of games $G_{3,l}$ for $0 \le l \le 6$.

Game $G_{3,0}$. $\mathcal{S}$ chooses $i^*, j^* \leftarrow_r \{1,\dots,N\}$ and $s_{j^*} \leftarrow_r \{1,\dots,m\}$, and hopes that the adversary will choose $sid^* = (\Pi, R, j^*, i^*, x_{i^*}, (y_{j^*}, w_{j^*}))$ as the test session, where (y_{j^*}, w_{j^*}) is output by the s_j^*-th session of party j^* activated by a $\mathbf{Send}_0(\Pi, R, j^*, i^*, x_{i^*})$ for some x_{i^*}. Then, $\mathcal{S}$ chooses $a \leftarrow_r R_q$, honestly generates static public keys for all parities (by randomly choosing s_i and e_i from χ_α), and simulates the attack environment for $\mathcal{A}$. Specifically, $\mathcal{S}$ maintains two tables L_1, L_2 for the random oracles H_1, H_2 respectively, and answers the queries from $\mathcal{A}$ as follows:

- $H_1(in)$: If there doesn't exist a tuple (in, out) in the L_1 list, choose an element $out \leftarrow_r \chi_\gamma$, and add (in, out) to the L_1 list. Then, return out to $\mathcal{A}$.
- $H_2(in)$ queries: If there doesn't exist a tuple (in, out) in the L_2 list, choose an element $out \leftarrow_r \{0,1\}^k$, and add (in, out) to the L_2 list. Then, return out to $\mathcal{A}$.
- $\mathbf{Send}_0(\Pi, I, i, j)$: $\mathcal{A}$ initiates a new session of i with intended partner j, $\mathcal{S}$ chooses $r_i, f_i \leftarrow_r \chi_\beta$, returns $x_i = ar_i + 2f_i \in \mathbb{Z}_q^{n\times n}$ to $\mathcal{A}$ on behalf of i.
- $\mathbf{Send}_1(\Pi, R, j, i, x_i)$: $\mathcal{S}$ chooses r_j, $f_j \leftarrow_r \chi_\beta$, and honestly computes $y_j = ar_j + 2f_j \in R_q$, k_j, w_j, and sk_j following the protocol. Finally, return (y_j, w_j) to $\mathcal{A}$.
- $\mathbf{Send}_2(\Pi, I, i, j, x_i, (y_j, w_j))$: $\mathcal{S}$ computes k_i and sk_i by using r_i and s_i following the protocol.
- $\mathbf{SessionKeyReveal}(sid)$: Let $sid = (\Pi, *, i, *, *, *, *)$, $\mathcal{S}$ returns sk_i if the session key of sid has been generated.
- $\mathbf{Corrupt}(i)$: Return the static secret key s_i of i to $\mathcal{A}$.
- $\mathbf{Test}(sid)$: Let $sid = (\Pi, I, i, j, x_i, (y_j, w_j))$, if $(i,j) \ne (i^*, j^*)$, or x_i and y_j are not output by the s_{i^*}-th session of i^* and the s_j^*-th session of j^* respectively, $\mathcal{S}$ aborts. Otherwise, $\mathcal{S}$ chooses $b \leftarrow_r \{0,1\}$ and $sk_i' \leftarrow_r \{0,1\}^k$. If $b = 0$, $\mathcal{S}$ returns sk_i', else it returns the real session sk_i of sid.

Claim 16. *The probability that $\mathcal{S}$ will not abort in $G_{3,0}$ with probability at least $\frac{1}{mN^2}$.*

Proof. This claim directly follows from the fact that $\mathcal{S}$ randomly chooses $i^*, j^* \leftarrow_r \{1,\dots,N\}$ and $s_j^* \leftarrow_r \{1,\dots,m\}$ independently from the view of $\mathcal{A}$. □

Game $G_{3,1}$. $\mathcal{S}$ behaves almost the same as in $G_{3,0}$, except in the following cases:

- $\mathbf{Send}_0(\Pi, I, i, j)$: If $i \ne i^*$, $\mathcal{S}$ answers the query as in Game $G_{3,0}$. Else, $\mathcal{S}$ computes $\boxed{x_i' = ar_i' + 2f_i'}$, where $r_i', f_i' \leftarrow_r \chi_\beta$. Then, it chooses $c \leftarrow_r \chi_\gamma$, and computes $\boxed{x_i = x_i' - p_i c}$. If there is a tuple $((i,j,x_i), *)$ in L_1 list, $\mathcal{S}$ aborts, else it adds $((i,j,x_i), c)$ into L_1, and returns x_i to $\mathcal{A}$.
- $\mathbf{Send}_1(\Pi, R, j, i, x_i)$: If $j \ne i^*$, $\mathcal{S}$ answers the query as in Game $G_{3,0}$. Else, $\mathcal{S}$ computes $\boxed{y_j' = ar_j' + 2f_j'}$, where $r_j', f_j' \leftarrow_r \chi_\beta$. Then, choose $d \leftarrow_r \chi_\gamma$, and compute $\boxed{y_j = y_j' - p_j d}$. If there is a tuple $((j,i,y_j,x_i), *)$ in the L_1 list, $\mathcal{S}$ aborts. Else, it adds $((j,i,y_j,x_i), d)$ into the L_1 list, and computes $\boxed{k_j = (p_i c + x_i) r_j' + 2g_j}$, where $c = H_1(i,j,x_i)$ and $g_j \leftarrow_r \chi_\beta$. Finally, it computes w_j and sk_j following the protocol, and sends (y_j, w_j) to $\mathcal{A}$.
- $\mathbf{Send}_2(\Pi, I, i, j, x_i, (y_j, w_j))$: If $i \ne i^*$, $\mathcal{S}$ answers the query as in Game $G_{3,0}$. Otherwise, let $x_i = x_i' - p_i c$ for $x_i' = ar_i' + 2f_i'$, the simulator $\mathcal{S}$ computes $\boxed{k_i = (p_j d + y_j) r_i' + 2g_i}$, where $g_i \leftarrow_r \chi_\beta$. Finally, $\mathcal{S}$ computes sk_i following the protocol.

In the following, we use $F_{3,l}$ to denote the event that $\mathcal{A}$ outputs a guess b' that equals to b in Game $G_{3,l}$.

Claim 17. *If $\alpha/\beta = 2^{-\omega(\log n)}$ and $LWE_{q,n,\alpha}$ is hard, then* $\Pr[F_{3,1}] = \Pr[F_{3,0}] - negl(n)$.

Proof. The proof is similar to Claim 2, we omit the details. □

Game $G_{3,2}$. $\mathcal{S}$ behaves almost the same as in $G_{3,1}$, except it replaces the public key for party i^* with a randomly chosen $\boxed{p_{i^*} \leftarrow_r R_q}$.

Claim 18. *If $LWE_{q,n,\alpha}$ is hard, then* $\Pr[F_{3,2}] = \Pr[F_{3,1}] - negl(n)$.

Proof. The proof is similar to Claim 11, we omit the details. □

Game $G_{3,3}$. $\mathcal{S}$ first computes $\boxed{y_j' = ar_j' + 2f_j'}$, where $r_j', f_j' \leftarrow_r \chi_\beta$. Then, it behaves almost the same as in $G_{3,2}$, except in the following cases:

– $\mathbf{Send}_1(\Pi, R, j, i, x_i)$: If $(i,j) \neq (i^*, j^*)$, or it is not the s_j^*-th session of j^*, $\mathcal{S}$ answers the query as in Game $G_{3,2}$. Otherwise, $\mathcal{S}$ chooses $d \leftarrow_r \chi_\gamma$, and computes $\boxed{y_j = y_j' - p_j d}$. $\mathcal{S}$ aborts if there is a tuple $((j, i, y_j, x_i), *)$ in the L_1 list. Else, it adds $((j, i, y_j, x_i),\ d)$ into L_1 list, and computes $\boxed{k_j = (p_i c + x_i) r_j' + 2g_j}$, where $c = H_1(i, j,\ x_i)$ and $g_j \leftarrow_r \chi_\beta$. Finally, it computes w_j and sk_j following the protocol, and sends (y_j, w_j) to $\mathcal{A}$.

Claim 19. *If $\alpha/\beta = 2^{-\omega(\log n)}$ and LWE$_{q,n,\alpha}$ is hard, then* $\Pr[F_{3,3}] = \Pr[F_{3,2}] - negl(n)$.

Proof. The proof is similar to Claim 2, we omit the details. □

Game $G_{3,4}$. $\mathcal{S}$ first computes $v_1 = ar_j' + 2\tilde{f}_j'$, $v_2 = p_i r_j' + t\tilde{e}_j'$ where $r_j' \leftarrow_r \chi_\beta$, and $\tilde{f}_j', \tilde{e}_j' \leftarrow \chi_\alpha$. Then, it computes $\boxed{y_j' = v_1 + 2f_j'} = ar_j' + 2(\tilde{f}_j' + f_j')$ where $f_j' \leftarrow_r \chi_\beta$. Finally, it behaves almost the same as in $G_{3,3}$ except in the following case:

– $\mathbf{Send}_1(\Pi, R, j, i, x_i)$: If $(i,j) \neq (i^*, j^*)$, or it is not the s_j^*-th session of j^*, $\mathcal{S}$ answers the query as in Game $G_{3,3}$. Otherwise, $\mathcal{S}$ chooses $d \leftarrow_r \chi_\gamma$, and computes $\boxed{y_j = y_j' - p_j d}$. If there is a tuple $((j, i, y_j, x_i), *)$ in the L_1 list, $\mathcal{S}$ aborts. Otherwise, it adds $((j, i, y_j, x_i), d)$ into L_1 list, and computes $\boxed{k_j = cv_2 + x_i r_j' + 2g_j} = (p_i c + x_i) r_j' + 2(c\tilde{e}_j' + g_j)$, where $c = H_1(i, j, x_i)$ and $g_j \leftarrow_r \chi_\beta$. Finally, it computes w_j and sk_j following the protocol, and sends (y_j, w_j) to $\mathcal{A}$.

Claim 20. *If $\alpha/\beta = 2^{-\omega(\log n)}$ and LWE$_{q,n,\alpha}$ is hard, then* $\Pr[F_{3,4}] = \Pr[F_{3,3}] - negl(n)$.

Proof. The proof is similar to Claim 2, we omit the details. □

Game $G_{3,5}$. $\mathcal{S}$ behaves almost the same as in $G_{3,4}$ except in the following case:

– $\mathbf{Send}_1(\Pi, R, j, i, x_i)$: If $(i,j) \neq (i^*, j^*)$, or it is not the s_j^*-th session of j^*, $\mathcal{S}$ answers the query as in Game $G_{3,4}$. Otherwise, $\mathcal{S}$ chooses $d \leftarrow_r \chi_\gamma$, and computes $y_j = y_j' - p_j d$. If there is a tuple $((j, i, y_j, x_i), *)$ in the L_1 list, $\mathcal{S}$ aborts, else it adds $((j, i, y_j, x_i), d)$ into L_1. Then, it chooses $\boxed{k_j \leftarrow_r R_q}$, computes w_j and σ_j as described in the protocol. If $\mathcal{A}$ has made a H_2 query $H_2(i, j, x_i, y_j, w_j, \sigma_j)$, $\mathcal{S}$ aborts. Else, it chooses $\boxed{sk_j \leftarrow_r \{0,1\}^k}$, and sets $H_2(i, j, x_i, y_j, w_j,\ \sigma_j) = sk_j$. Finally, it sends (y_j, w_j) to $\mathcal{A}$.

Note that the only difference between $G_{3,4}$ and $G_{3,5}$ is that $\mathcal{S}$ replaces the real $k_j = cv_2 + x_i r_j' + 2g_j$ in Game $G_{3,4}$ with a randomly chosen $k_j \in R_q$ in Game $G_{3,5}$. Considering H_2 is a random oracle, such a difference will not affect the view of $\mathcal{A}$ until it makes a H_2 query with σ_j derived from k_j. Formally, denote $Q_{3,l}$ for $l = 4, 5, 6$ as the event that $\mathcal{A}$ makes a H_2 query with σ_j derived from k_j.

Claim 21. $\Pr[Q_{3,4}] = \Pr[Q_{3,5}]$ *and* $\Pr[F_{3,4} | \neg Q_{3,4}] = \Pr[F_{3,5} | \neg Q_{3,5}] = 1/2 + negl(n)$.

Proof. Since H_2 is a random oracle, the event $Q_{3,4}$ is independent from the distribution of the corresponding sk_i. Namely, no matter whether or not $\mathcal{A}$ obtains sk_i, $\Pr[Q_{2,5}]$ is the same, which also holds for $\Pr[Q_{3,5}]$. Besides, if $Q_{3,l}$ for $l = 4, 5$ does not happen, $G_{3,5}$ is actually the same as $G_{3,4}$ in the adversary's view. Especially, the distribution of sk_j is random and uniform over $\{0,1\}^k$, which means that the advantage of $\mathcal{A}$ in guessing b is negligible, given that the event $Q_{3,5}$ does not happen. □

Similarly, let $sid = (\Pi, R, j^*, i^*, x_i, (y_j, w_i))$ be the test session. By our assumption that $\mathcal{A}$ is a **Type III** adversary, x_i is not output by party i^*. In other words, $\mathcal{S}$ itself does not make a H_1 hash query $H_1(i^*, j^*, x_i)$ in producing x_i. Given $v_1 = ar_j' + 2\tilde{f}_j'$, $v_2 = p_i r_j' + t\tilde{e}_j'$, and $g_j \leftarrow_r \chi_\beta$ in Game $G_{3,5}$, we denote $k_j = cv_2 + x_i r_j' + 2g_j$ as the target key in adversary $\mathcal{A}$'s view (which is the same as in Game $G_{3,4}$), where $H_1(i^*, j^*, x_i) = c$. By our assumption, $\mathcal{A}$ will make a H_2 query with σ_j derived from k_j with probability at least $\Pr[Q_{3,5}]$.

Now, fixing v_1, v_2, r_j' and g_j (note that all those values are determined by $\mathcal{S}$, and are independent from the adversary's behaviors), $\mathcal{S}$ reprograms the hash query $H_1(i^*, j^*,\ x_i) = \tilde{c} \neq c$ by using another randomly chosen $\tilde{c} \leftarrow_r \chi_\gamma$, and sets $k_j' = \tilde{c} v_2 + x_i r_j' + 2g_j = k_j + (\tilde{c} - c) v_2$. According to the forking lemma [2], the adversary $\mathcal{A}$ will use the same x_i in the test session, and and makes a H_2 query with σ_j' derived from k_j' with probability at least $\Pr[Q_{3,5}](\Pr[Q_{3,5}]/q_h - 2^{-n})$, where q_h is maximum number of H_1 queries. Denote by double-$Q_{3,l}$ such an event that, for $l = 5, 6$, $\mathcal{A}$ in Game $G_{3,l}$ will make both σ_i and σ_i' in two runs of $\mathcal{A}$, where σ_j is derived from k_j in the first run of $\mathcal{A}$, and σ_j' is derived from $k_j' = k_j + (\tilde{c} - c) v_2$ in the second run of $\mathcal{A}$. In particular, we have $\Pr[\text{double-}Q_{3,5}] \geq \Pr[Q_{3,5}](\Pr[Q_{3,5}]/q_h - 2^{-n})$.

Game $G_{3,6}$. $\mathcal{S}$ chooses $\boxed{v_1, v_2 \leftarrow_r R_q}$, and behaves almost the same as in $G_{3,5}$.

Claim 22. *Under the LWE$_{q,n,\alpha}$ assumption, Game $G_{3,6}$ is computationally indistinguishable from $G_{3,5}$. In particular,* $\Pr[\textit{double-}Q_{3,6}] = \Pr[\textit{double-}Q_{3,5}] - negl(n)$.

Proof. Since the only difference between $G_{3,5}$ and $G_{3,6}$ is that $\mathcal{S}$ replaces $v_1 = ar_i' + 2\tilde{f}_i'$ and $v_2 = p_j r_i' + t\tilde{e}_i'$ with randomly chosen elements in R_q, and an adversary that can distinguish the difference between $G_{3,5}$ and $G_{3,6}$ could be used to solve the LWE$_{q,n,\alpha}$ problem. □

Claim 23. $\Pr[\textit{double-}F_{3,6}] = negl(n)$.

Proof. Note that in Game $G_{3,6}$, $\mathcal{S}$ does not really compute k_j and k_j'. (Actually, it cannot compute the values since v_1 and v_2 are randomly chosen from R_q.) Here, we denote

k_j and k'_j (i.e., the values determined before and after $\mathcal{S}$ reprograms the H_1 query) as the target values in the $\mathcal{A}$'s view. In particular, the condition $k'_j = k_j + (\tilde{d} - d)v_2$ holds, since $\mathcal{A}$ cannot efficiently distinguish Game $G_{3,6}$ from $G_{3,5}$ by Claim 22. However, since v_2 is uniformly distributed over R_q and is independent from the $\mathcal{A}$'s view (thus is independent from both k_j and k'_j), we have $\sigma'_j = \mathbf{Mod}_2(k'_j, w'_j)$ is statistically close to uniform over $\{0,1\}^n$ even conditioned on $\sigma_j = \mathbf{Mod}_2(k_j, w_j)$ by Lemma 6. (Note that $(\tilde{c} - c)$ is invertible with overwhelming probability by Lemma 4). Thus, the probability that $\mathcal{A}$ will make a H_2 query with σ'_i is at most $2^{-n} + negl(n)$. In other words, the probability $\Pr[\text{double-}F_{3,6}] \leq 2^{-n} + negl(n)$, which is negligible in n. This completes the proof. □

Combining the result in Claim 16∼23, we have that Lemma 11 follows. □

5.4 Type IV Adversary

In this subsection, we prove that our AKE is secure against any PPT **Type IV** adversary $\mathcal{A}$.

Lemma 12. *If $\alpha/\beta = 2^{-\omega(\log n)}$ and LWE$_{q,n,\alpha}$ is hard, the proposed AKE is secure against any PPT* ***Type IV*** *adversary $\mathcal{A}$ in the random oracle model.*

Proof. We prove this lemma via a sequence of games $G_{4,l}$ for $0 \leq l \leq 4$.

Game $G_{4,0}$. $\mathcal{S}$ first chooses $i^*, j^* \leftarrow_r \{1, \ldots, N\}$ and $s_{i^*}, s_{j^*} \leftarrow_r \{1, \ldots, m\}$, and hopes that the adversary will choose $sid^* = (\Pi, R, j^*, i^*, x_{i^*}, (y_{j^*}, w_{j^*}))$ as the test session, where x_{i^*} is output by the s_{i^*}-th session of party i^*, and (y_j*, w_{j^*}) is output by the s^*_j-th session of party j^* activated by a $\mathbf{Send}_1(\Pi, R, j^*, i^*, x_{i^*})$. Then, $\mathcal{S}$ chooses $a \leftarrow_r R_q$, honestly generates static public keys for all parities (by randomly choosing s_i and e_i from χ_α), and simulates the attack environment for $\mathcal{A}$. Specifically, $\mathcal{S}$ maintains two tables L_1, L_2 for the random oracles H_1, H_2 respectively, and answers the queries from $\mathcal{A}$ as follows:

– $H_1(in)$: If there doesn't exist a tuple (in, out) in the L_1 list, choose an element $out \leftarrow_r \chi_\gamma$, and add (in, out) to the L_1 list. Then, return out to $\mathcal{A}$.

– $H_2(in)$ queries: If there doesn't exist a tuple (in, out) in the L_2 list, choose an element $out \leftarrow_r \{0,1\}^k$, and add (in, out) to the L_2 list. Then, return out to $\mathcal{A}$.

– $\mathbf{Send}_0(\Pi, I, i, j)$: $\mathcal{A}$ initiates a new session of i with intended partner j, $\mathcal{S}$ chooses $r_i, f_i \leftarrow_r \chi_\beta$, returns $x_i = ar_i + 2f_i \in \mathbb{Z}_q^{n\times n}$ to $\mathcal{A}$ on behalf of i.

– $\mathbf{Send}_1(\Pi, R, j, i, x_i)$: $\mathcal{S}$ chooses r_j, $f_j \leftarrow_r \chi_\beta$, and honestly computes $y_j = ar_j + 2f_j \in R_q$, k_j, w_j, and sk_j following the protocol. Finally, return (y_j, w_j) to $\mathcal{A}$.

– $\mathbf{Send}_2(\Pi, I, i, j, x_i, (y_j, w_j))$: $\mathcal{S}$ computes k_i and sk_i by using r_i and s_i following the protocol.

– $\mathbf{SessionKeyReveal}(sid)$: Let $sid = (\Pi, *, i, *, *, *, *)$, $\mathcal{S}$ returns sk_i if the session key of sid has been generated.

– $\mathbf{Corrupt}(i)$: Return the static secret key s_i of i to $\mathcal{A}$.

– $\mathbf{Test}(sid)$: Let $sid = (\Pi, I, i, j, x_i, (y_j, w_j))$, if $(i,j) \neq (i^*, j^*)$, or x_i and y_j are not output by the s_{i^*}-th session of i^* and the s^*_j-th session of j^* respectively, $\mathcal{S}$ aborts. Otherwise, $\mathcal{S}$ chooses $b \leftarrow_r \{0,1\}$ and $sk'_i \leftarrow_r \{0,1\}^k$. If $b = 0$, $\mathcal{S}$ returns sk'_i, else it returns the real session sk_i of sid.

Claim 24. *The probability that $\mathcal{S}$ will not abort in $G_{4,0}$ is at least $\frac{1}{m^2N^2}$.*

Proof. This claim directly follows from the fact that $\mathcal{S}$ randomly chooses $i^*, j^* \leftarrow_r \{1, \ldots, N\}$ and $s_{i^*}, s^*_j \leftarrow_r \{1, \ldots, m\}$ independently from the view of $\mathcal{A}$. □

Game $G_{4,1}$. $\mathcal{S}$ first computes $\boxed{y'_j = ar'_j + 2f'_j}$, where $r'_j, f'_j \leftarrow_r \chi_\beta$. Then, it behaves almost the same as in $G_{4,0}$, except in the following case:

– $\mathbf{Send}_1(\Pi, R, j, i, x_i)$: If $(i,j) \neq (i^*, j^*)$, or it is not the s^*_j-th session of j^*, $\mathcal{S}$ answers the query as in Game $G_{4,0}$. Otherwise, $\mathcal{S}$ chooses $d \leftarrow_r \chi_\gamma$, and computes $\boxed{y_j = y'_j - p_jd}$. If there is a tuple $((j, i, y_j, x_i), *)$ in the L_1 list, $\mathcal{S}$ aborts. Else, it adds $((j, i, y_j, x_i), d)$ into L_1 list, and computes $\boxed{k_j = (p_ic + x_i)r'_j + 2g_j}$, where $c = H_1(i, j, x_i)$ and $g_j \leftarrow_r \chi_\beta$. Finally, it computes w_j and sk_j following the protocol, and sends (y_j, w_j) to $\mathcal{A}$.

In the following, we define $F_{4,l}$ as the event that $\mathcal{A}$ outputs a guess b' that equals to b in Game $G_{4,l}$.

Claim 25. *If $\alpha/\beta = 2^{-\omega(\log n)}$ and LWE$_{q,n,\alpha}$ is hard, then* $\Pr[F_{4,l}] = \Pr[F_{4,0}] - negl(n)$.

Proof. The proof is similar to Claim 2, we omit the details. □

Game $G_{4,2}$. $\mathcal{S}$ first computes $\boxed{x'_i = ar'_i + 2f'_i}$, where $r'_i, f'_i \leftarrow_r \chi_\beta$. Then, it behaves almost the same as in $G_{4,1}$, except for the following cases:

– $\mathbf{Send}_0(\Pi, I, i, j)$: If $(i,j) \neq (i^*, j^*)$, or it is not the s_{i^*}-th session of i^*, $\mathcal{S}$ answers the query as in Game $G_{4,1}$. Otherwise, $\mathcal{S}$ chooses $c \leftarrow_r \chi_\gamma$, and computes $\boxed{x_i = x'_i - p_ic}$. $\mathcal{S}$ aborts if there is a tuple $((i, j, x_i), *)$ in L_1 list, else it adds $((i, j, x_i), c)$ into L_1. Finally, it returns x_i to $\mathcal{A}$.

– $\mathbf{Send}_2(\Pi, I, i, j, x_i, (y_j, w_j))$: If $(i,j) \neq (i^*, j^*)$, or it is not the s_{i^*}-th session of i^*, $\mathcal{S}$ answers the query as in Game $G_{4,1}$. Otherwise, if (y_j, w_j) is output by the s^*_j-th session of party j^*, let sk_j be the session key of session $sid = (\Pi, R, j, i, x_i, (y_j, w_j))$, $\mathcal{S}$ sets $\boxed{sk_i = sk_j}$. Otherwise, it computes $\boxed{k_i = (p_jd + y_j)r'_i + 2g_i}$, where

$d = H_1(j, i, y_j, x_i)$ and $g_i \leftarrow_r \chi_\beta$. Finally, it computes sk_i following the protocol.

Claim 26. *If $\alpha/\beta = 2^{-\omega(\log n)}$ and LWE$_{q,n,\alpha}$ is hard, then* $\Pr[F_{4,2}] = \Pr[F_{4,l}] - negl(n)$.

Proof. The proof is similar to Claim 3, we omit the details. □

Game $G_{4,3}$. $\mathcal{S}$ first chooses $\boxed{x_i' \leftarrow_r R_q}$. Then, it behaves almost the same as in $G_{4,2}$, except for the following cases:

- **Send**$_2(\Pi, I, i, j, x_i, (y_j, w_j))$: If $(i, j) \neq (i^*, j^*)$, or it is not the s_{i^*}-th session of i^*, or (y_j, w_j) is output by the s_j^*-th session of party j^*, $\mathcal{S}$ behaves the same as in Game $G_{4,2}$. Else, choose $\boxed{sk_i \leftarrow_r \{0,1\}^k}$ as the session key.

Denote $Q_{4,l}$ be event that in Game $G_{4,l}$ for $l = 2, 3, 4$, $\mathcal{A}$ makes a H_2 query with σ_i for the s_{i^*}-th session of party i^*, when (y_j, w_j') is output by the s_j^*-th session of party j^* but $w_j \neq w_j'$.

Claim 27. *If LWE$_{q,n,\alpha}$ is hard,* $\Pr[Q_{4,3}] = \Pr[Q_{4,2}] - negl(n)$*, and* $\Pr[F_{4,3}|\neg Q_{4,3}] = \Pr[F_{4,2}|\neg Q_{4,2}] - negl(n)$.

Proof. The proof is similar to Claim 4, we omit the details. □

Game $G_{4,4}$. $\mathcal{S}$ chooses $\boxed{y_j' \leftarrow_r R_q}$, and behaves almost the same as in $G_{4,3}$, except in the following case:

- **Send**$_1(\Pi, R, j, i, x_i)$: If $(i, j) \neq (i^*, j^*)$, or it is not the s_j^*-th session of j^*, $\mathcal{S}$ answers the query as in Game $G_{4,3}$. Otherwise, $\mathcal{S}$ chooses $d \leftarrow_r \chi_\gamma$, and computes $\boxed{y_j = y_j' - p_j d}$. If there is a tuple $((j, i, y_j, x_i), *)$ in the L_1 list, $\mathcal{S}$ aborts, else it adds $((j, i, y_j, x_i), d)$ into L_1. Then, $\mathcal{S}$ chooses $\boxed{k_j \leftarrow_r R_q}$, and computes w_j, σ_j following the protocol. If $\mathcal{A}$ has made a H_2 query $H_2(i, j, x_i, y_j, w_j, \sigma_j)$, $\mathcal{S}$ aborts. Else, choose $\boxed{sk_j \leftarrow_r \{0,1\}^k}$, and set $H_2(i, j, x_i, y_j, w_j, \sigma_j) = sk_j$. Finally, it sends (y_j, w_j) to $\mathcal{A}$.

Claim 28. *Under the LWE$_{q,n,\beta}$ assumption, Game $G_{4,3}$ and $G_{4,4}$ is computationally indistinguishable. In particular, we have* $\Pr[Q_{4,4}] = \Pr[Q_{4,3}]$*, and* $\Pr[F_{4,4}|\neg Q_{4,4}] = \Pr[F_{4,3}|\neg Q_{4,3}] - negl(n)$.

Proof. The proof is similar to Claim 5, we omit the details. □

Claim 29. $\Pr[Q_{4,4}] = negl(n)$.

Proof. The proof is similar to Claim 6, we omit the details. □

Claim 30. $\Pr[F_{4,4}|\neg Q_{4,4}] = 1/2 + negl(n)$.

Proof. The proof is similar to Claim 7, we omit the details. □

Combining the result in Claim 24∼30, we have that Lemma 12 follows.

□

5.5 Type V Adversary

In this subsection, we prove that our AKE is secure against any PPT **Type V** adversary $\mathcal{A}$.

Lemma 13. *If $\alpha/\beta = 2^{-\omega(\log n)}$ and LWE$_{q,n,\alpha}$ is hard, the proposed AKE is secure against any PPT **Type V** adversary $\mathcal{A}$ in the random oracle model.*

Proof. We prove this lemma via a sequence of games $G_{5,l}$ for $0 \leq l \leq 4$.

Game $G_{5,0}$. $\mathcal{S}$ chooses $i^*, j^* \leftarrow_r \{1, \dots, N\}$ and $s_{i^*}, s_{j^*} \leftarrow_r \{1, \dots, m\}$, and hopes that the adversary will choose $sid^* = (\Pi, R, j^*, i^*, x_{i^*}, (y_{j^*}, w_{j^*}))$ as the test session, where x_{i^*} is output by the s_{i^*}-th session of party i^*, and (y_j*, w_{j^*}) is output by the s_j^*-th session of party j^* activated by a **Send**$_1(\Pi, R, j^*, i^*, x_{i^*})$. Then, $\mathcal{S}$ chooses $a \leftarrow_r R_q$, honestly generates static public keys for all parities (by randomly choosing s_i and e_i from χ_α), and simulates the attack environment for $\mathcal{A}$. Specifically, $\mathcal{S}$ maintains two tables L_1, L_2 for the random oracles H_1, H_2 respectively, and answers the queries from $\mathcal{A}$ as follows:

- $H_1(in)$: If there doesn't exist a tuple (in, out) in the L_1 list, choose an element $out \leftarrow_r \chi_\gamma$, and add (in, out) to the L_1 list. Then, return out to $\mathcal{A}$.
- $H_2(in)$ queries: If there doesn't exist a tuple (in, out) in the L_2 list, choose an element $out \leftarrow_r \{0,1\}^k$, and add (in, out) to the L_2 list. Then, return out to $\mathcal{A}$.
- **Send**$_0(\Pi, I, i, j)$: $\mathcal{A}$ initiates a new session of i with intended partner j, $\mathcal{S}$ chooses $r_i, f_i \leftarrow_r \chi_\beta$, returns $x_i = ar_i + 2f_i \in \mathbb{Z}_q^{n\times n}$ to $\mathcal{A}$ on behalf of i.
- **Send**$_1(\Pi, R, j, i, x_i)$: $\mathcal{S}$ chooses r_j, $f_j \leftarrow_r \chi_\beta$, and honestly computes $y_j = ar_j + 2f_j \in R_q$, k_j, w_j, and sk_j following the protocol. Finally, return (y_j, w_j) to $\mathcal{A}$.
- **Send**$_2(\Pi, I, i, j, x_i, (y_j, w_j))$: $\mathcal{S}$ computes k_i and sk_i by using r_i and s_i following the protocol.
- **SessionKeyReveal**(sid): Let $sid = (\Pi, *, i, *, *, *, *)$, $\mathcal{S}$ returns sk_i if the session key of sid has been generated.
- **Corrupt**(i): Return the static secret key s_i of i to $\mathcal{A}$.
- **Test**(sid): Let $sid = (\Pi, I, i, j, x_i, (y_j, w_j))$, if $(i, j) \neq (i^*, j^*)$, or x_i and y_j are not output by the s_{i^*}-th session of i^* and the s_j^*-th session of j^* respectively, $\mathcal{S}$ aborts. Otherwise, $\mathcal{S}$ chooses $b \leftarrow_r \{0,1\}$ and $sk_i' \leftarrow_r \{0,1\}^k$. If $b = 0$, $\mathcal{S}$ returns sk_i', else it returns the real session sk_i of sid.

Claim 31. *The probability that $\mathcal{S}$ will not abort in $G_{5,0}$ with probability at least $\frac{1}{m^2N^2}$.*

Proof. This claim directly follows from the fact that $\mathcal{S}$ randomly chooses $i^*, j^* \leftarrow_r \{1, \dots, N\}$ and $s_{i^*}, s_j^* \leftarrow_r \{1, \dots, m\}$ independently from the view of $\mathcal{A}$. □

Game $G_{5,1}$. $\mathcal{S}$ first computes $\boxed{y'_j = ar'_j + 2f'_j}$, where $r'_j, f'_j \leftarrow_r \chi_\beta$. Then, it behaves almost the same as in $G_{5,0}$, except in the following case:

- $\mathbf{Send}_1(\Pi, R, j, i, x_i)$: If $(i,j) \neq (i^*, j^*)$, or it is not the s^*_j-th session of j^*, $\mathcal{S}$ answers the query as in Game $G_{5,0}$. Otherwise, $\mathcal{S}$ chooses $d \leftarrow_r \chi_\gamma$, and computes $\boxed{y_j = y'_j - p_j d}$. If there is a tuple $((j,i,y_j,x_i),*)$ in the L_1 list, $\mathcal{S}$ aborts. Else, it adds $((j,i,y_j,x_i),d)$ into L_1. Then, $\mathcal{S}$ computes $\boxed{k_j = (p_i c + x_i) r'_j + 2g_j}$, where $c = H_1(i,j,x_i)$ and $g_j \leftarrow_r \chi_\beta$. Finally, it computes w_j and sk_j following the protocol, and sends (y_j, w_j) to $\mathcal{A}$.

In the following, let $F_{5,l}$ denote the event that $\mathcal{A}$ outputs a guess b' that equals to b in Game $G_{5,l}$.

Claim 32. *If* $\alpha/\beta = 2^{-\omega(\log n)}$ *and* $LWE_{q,n,\alpha}$ *is hard, then* $\Pr[F_{5,l}] = \Pr[F_{5,0}] - negl(n)$.

Proof. The proof is similar to Claim 2, we omit the details. □

Game $G_{5,2}$. $\mathcal{S}$ first computes $\boxed{x'_i = ar'_i + 2f'_i}$, where $r'_i, f'_i \leftarrow_r \chi_\beta$. Then, it behaves almost the same as in $G_{5,1}$, except for the following cases:

- $\mathbf{Send}_0(\Pi, I, i, j)$: If $(i,j) \neq (i^*, j^*)$, or it is not the s_{i^*}-th session of i^*, $\mathcal{S}$ answers the query as in Game $G_{5,1}$. Otherwise, $\mathcal{S}$ chooses $c \leftarrow_r \chi_\gamma$, and computes $\boxed{x_i = x'_i - p_i c}$. If there is a tuple $((i,j,x_i),*)$ in L_1 list, $\mathcal{S}$ aborts, else it adds $((i,j,x_i),c)$ into L_1. Finally, it returns x_i to $\mathcal{A}$.
- $\mathbf{Send}_2(\Pi, I, i, j, x_i, (y_j, w_j))$: If $(i,j) \neq (i^*, j^*)$, or it is not the s_{i^*}-th session of i^*, $\mathcal{S}$ answers the query as in Game $G_{5,1}$. Otherwise, if (y_j, w_j) is output by the s^*_j-th session of party j^*, let sk_j be the session key of session $sid = (\Pi, R, j, i, x_i, (y_j, w_j))$, $\mathcal{S}$ sets $\boxed{sk_i = sk_j}$. Else, $\mathcal{S}$ computes $\boxed{k_i = (p_j d + y_j) r'_i + 2g_i}$, where $d = H_1(j,i,y_j,x_i)$ and $g_i \leftarrow_r \chi_\beta$. Finally, it computes sk_i following the protocol.

Claim 33. *If* $\alpha/\beta = 2^{-\omega(\log n)}$ *and* $LWE_{q,n,\alpha}$ *is hard, then* $\Pr[F_{5,2}] = \Pr[F_{5,l}] - negl(n)$.

Proof. The proof is similar to Claim 3, we omit the details. □

Game $G_{5,3}$. $\mathcal{S}$ chooses $\boxed{x'_i \leftarrow_r R_q}$, and behaves almost the same as in $G_{5,2}$, with the following exception:

- $\mathbf{Send}_2(\Pi, I, i, j, x_i, (y_j, w_j))$: If $(i,j) \neq (i^*, j^*)$, or it is not the s_{i^*}-th session of i^*, or (y_j, w_j) is output by the s^*_j-th session of party j^*, $\mathcal{S}$ answers the query as in Game $G_{5,2}$. Else, it chooses $\boxed{sk_i \leftarrow_r \{0,1\}^k}$.

Claim 34. $\Pr[F_{5,3}] = \Pr[F_{5,2}] - negl(n)$.

Proof. The claim can be proved via a sequence of Games as we have done from Game $G_{2,4}$ to $G_{2,6}$. We omit the details here. □

Game $G_{5,4}$. $\mathcal{S}$ chooses $\boxed{y'_j \leftarrow_r R_q}$, and behaves almost the same as in $G_{5,3}$, except in the following case:

- $\mathbf{Send}_1(\Pi, R, j, i, x_i)$: If $(i,j) \neq (i^*, j^*)$, or it is not the s^*_j-th session of j^*, $\mathcal{S}$ answers the query as in Game $G_{5,3}$. Otherwise, $\mathcal{S}$ chooses $d \leftarrow_r \chi_\gamma$, and computes $\boxed{y_j = y'_j - p_j d}$. If there is a tuple $((j,i,y_j,x_i),*)$ in the L_1 list, $\mathcal{S}$ aborts. Else, it adds $((j,i,y_j,x_i),d)$ into L_1, and chooses $\boxed{k_j \leftarrow_r R_q}$, and computes w_j, σ_j following the protocol. $\mathcal{S}$ aborts the simulation if $\mathcal{A}$ has made a H_2 query $H_2(i,j,x_i,y_j,w_j,\sigma_j)$. Otherwise, it chooses $\boxed{sk_j \leftarrow_r \{0,1\}^k}$, and sets $H_2(i,j,x_i,y_j,w_j,\sigma_j) = sk_j$. Finally, it sends (y_j, w_j) to $\mathcal{A}$.

Claim 35. *Under the* $LWE_{q,n,\beta}$ *assumption, we have that* $\Pr[F_{5,4}] = \Pr[F_{5,3}] - negl(n)$.

Proof. The proof is similar to Claim 5, we omit the details. □

Claim 36. $\Pr[F_{5,4}] = 1/2 + negl(n)$.

Proof. The proof is similar to Claim 7, we omit the details. □

Combining the result in Claim 31∼36, we have that Lemma 13 follows. □

References

[1] B. Applebaum, D. Cash, C. Peikert, and A. Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In S. Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 595–618. Springer Berlin Heidelberg, 2009.

[2] M. Bellare and G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, pages 390–399, New York, NY, USA, 2006. ACM.

[3] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and Communications Security*, pages 62–73. ACM Press, 1993.

[4] M. Bellare and P. Rogaway. Entity authentication and key distribution. In D. Stinson, editor, *Advances in Cryptology – CRYPTO 93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer Berlin Heidelberg, 1994.

[5] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. Fully homomorphic encryption without bootstrapping. *Innovations in Theoretical Computer Science, ITCS*, pages 309–325, 2012.

[6] Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In P. Rogaway, editor, *Advances in Cryptology – CRYP-*

TO 2011, volume 6841 of *Lecture Notes in Computer Science*, pages 505–524. Springer Berlin Heidelberg, 2011.

[7] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In B. Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer Berlin Heidelberg, 2001.

[8] L. Chen. Practical impacts on qutumn computing. Quantum-Safe-Crypto Workshop at the European Telecommunications Standards Institute, 2013. http://docbox.etsi.org/Workshop/2013/201309_CRYPTO/S05_DEPLOYMENT/NIST_CHEN.pdf.

[9] Y. Chen and P. Nguyen. Bkz 2.0: Better lattice security estimates. In D. Lee and X. Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 1–20. Springer Berlin Heidelberg, 2011.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, et al. *Introduction to algorithms*, volume 2. MIT press Cambridge, 2001.

[11] I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In R. Safavi-Naini and R. Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer Berlin Heidelberg, 2012.

[12] T. Dierks. The transport layer security (TLS) protocol version 1.2. 2008.

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

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

[15] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky. Lattice signatures and bimodal gaussians. In R. Canetti and J. Garay, editors, *Advances in Cryptology - CRYPTO 2013*, volume 8042 of *Lecture Notes in Computer Science*, pages 40–56. Springer Berlin Heidelberg, 2013.

[16] A. Freier. The SSL protocol version 3.0. *http://wp. netscape. com/eng/ssl3/draft302. txt*, 1996.

[17] A. Fujioka, K. Suzuki, K. Xagawa, and K. Yoneyama. Strongly secure authenticated key exchange from factoring, codes, and lattices. In *PKC*, pages 467–484. 2012.

[18] A. Fujioka, K. Suzuki, K. Xagawa, and K. Yoneyama. Practical and post-quantum authenticated key exchange from one-way secure key encapsulation mechanism. In *ASIACCS*, pages 83–94, 2013.

[19] N. Gama and P. Nguyen. Predicting lattice reduction. In N. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 31–51. Springer Berlin Heidelberg, 2008.

[20] R. Gennaro and V. Shoup. A note on an encryption scheme of kurosawa and desmedt. Cryptology ePrint Archive, Report 2004/194, 2004. http://eprint.iacr.org/.

[21] C. Gentry, S. Halevi, and N. Smart. Homomorphic evaluation of the AES circuit. In R. Safavi-Naini and R. Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867. Springer Berlin Heidelberg, 2012.

[22] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, STOC '08, pages 197–206, New York, NY, USA, 2008. ACM.

[23] F. Giesen, F. Kohlar, and D. Stebila. On the security of TLS renegotiation. In *ACM Conference on Computer and Communications Security – CCS '13*, pages 387–398, 2013.

[24] S. Goldwasser, Y. Kalai, C. Peikert, and V. Vaikuntanathan. Robustness of the learning with errors assumption. In *Proceedings of the Innovations in Computer Science 2010*. Tsinghua University Press, 2010.

[25] D. Harkins, D. Carrel, et al. The internet key exchange (IKE). Technical report, RFC 2409, november, 1998.

[26] ISO/IEC. 11770-3:2008 information technology – security techniques – key management – part 3: Mechanisms using asymmetric techniques.

[27] J. Katz and V. Vaikuntanathan. Smooth projective hashing and password-based authenticated key exchange from lattices. In M. Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 636–652. Springer Berlin / Heidelberg, 2009.

[28] C. Kaufman, P. Hoffman, Y. Nir, and P. Eronen. Internet key exchange protocol version 2 (IKEv2). Technical report, RFC 5996, September, 2010.

[29] H. Krawczyk. SIGMA: The SIGn-and-MAc approach to authenticated Diffie-Hellman and its use in the IKE protocols. In *CRYPTO*, pages 400–425. 2003.

[30] H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 546–566. Springer Berlin Heidelberg, 2005.

[31] H. Krawczyk, K. Paterson, and H. Wee. On the security of the TLS protocol: A systematic analysis. In *CRYPTO*, pages 429–448. 2013.

[32] R. Lindner and C. Peikert. Better key sizes (and attacks) for lwe-based encryption. In A. Kiayias, editor, *Topics in Cryptology – CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*, pages 319–339. Springer Berlin Heidelberg, 2011.

[33] V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen. SWIFFT: A modest proposal for FFT hashing. In K. Nyberg, editor, *Fast Software Encryption – FSE 2008*, volume 5086 of *LNCS*, pages 54–72. Springer Berlin / Heidelberg, 2008.

[34] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In H. Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer Berlin / Heidelberg, 2010.

[35] V. Lyubashevsky, C. Peikert, and O. Regev. A toolkit for ring-lwe cryptography. In T. Johansson and P. Nguyen, editors,

Advances in Cryptology C EUROCRYPT 2013, volume 7881 of *Lecture Notes in Computer Science*, pages 35–54. Springer Berlin Heidelberg, 2013.

[36] N. Mavrogiannopoulos, F. Vercauteren, V. Velichkov, and B. Preneel. A cross-protocol attack on the TLS protocol. In *ACM Conference on Computer and Communications Security – CCS '12*, pages 62–72, 2012.

[37] A. Menezes, M. Qu, and S. Vanstone. Some new key agreement protocols providing mutual implicit authentication. In *Selected Areas in Cryptography*, 1995.

[38] D. Micciancio and O. Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM J. Comput.*, 37:267–302, April 2007.

[39] D. Micciancio and O. Regev. Lattice-based cryptography. In D. Bernstein, J. Buchmann, and E. Dahmen, editors, *Post-Quantum Cryptography*, pages 147–191. Springer Berlin Heidelberg, 2009.

[40] M. Naehrig, K. Lauter, and V. Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, pages 113–124, New York, NY, USA, 2011. ACM.

[41] C. Peikert. An efficient and parallel gaussian sampler for lattices. In *CRYPTO*, pages 80–97. 2010.

[42] C. Peikert. Lattice cryptography for the internet. Cryptology ePrint Archive, Report 2014/070, 2014. `http://eprint.iacr.org/`.

[43] T. Pöppelmann and T. Güneysu. Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware. In *LATINCRYPT*, pages 139–158. 2012.

[44] P. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[45] D. Stehlé and R. Steinfeld. Making NTRU as secure as worst-case problems over ideal lattices. In K. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 27–47. Springer Berlin / Heidelberg, 2011.

[46] A. C.-C. Yao and Y. Zhao. OAKE: A new family of implicitly authenticated Diffie-Hellman protocols. In *ACM Conference on Computer and Communications Security – CCS '13*, pages 1113–1128, 2013.