Authenticated Key Exchange from Ideal Lattices

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

1. Chinese Acadmy of Sciences; 2. University of Cincinnnati

* Corresponding Authors

Abstract

In this paper, we propose an authenticated key exchange (AKE) protocol from Ideal lattices. The protocol is simple since it does not involve any other cryptographic primitives to achieve authentication (e.g., signatures). This allows us to establish a security proof solely based on the hardness of the well-known ring-LWE problems, thus on some hard lattice problems in the worst-case (e.g., SVP and SIVP). We give the security proof of the proposed AKE protocol in an enhanced variant of the original Bellare-Rogaway (BR) model, which additionally captures weak Perfect Forward Secrecy (wPFS), in the random oracle (RO) model.

1. Introduction

Key Exchange (KE) is a fundamental cryptographic primitive, which allows two parties to generate a common secret key over insecure networks, namely, where all the communications are completely controlled by adversaries. Since a shared-key between parties is essential to protect the transmitted data over public networks by making use of any other (symmetric) cryptographic tools (e.g., AES), KE has become one of the most widely used cryptographic tools in building secure communications (e.g., SSL/TLS, IPSec, SSH, etc). Following the celebrated Diffie-Hellman (DH) KE protocol [9], many cryptographic researchers have proposed plenty of different KE protocols providing various security goals. Among them, Authenticated Key Exchange (AKE) is a class of candidate KE protocols for lots of real applications, which not only allows parties to compute the shared key but also ensures authenticity of the parties. Namely, two parties can compute a shared key only if both participants are the claimed parties known to each other.

In general, each party in AKE usually has a pair of *static public key* and corresponding *static secret key*, where the static public key and the identity of the party are certified together using a public infrastructure such as public key infrastructure (PKI), or ID-based infrastructure. During the execution of the protocol, parties exchange their *ephemeral public keys* that are generated using some *ephemeral secret keys*, compute a *session state* from all the keys and exchanged transcripts, and finally derive a common session key by using some *key derivation function* (KDF). Intuitively, security means that AKE should assure parties that no probabilistic polynomial time (PPT) adversary can obtain *any* useful information about their shared secret key (if there is). Actually, the first indistinguishability-based security notion of AKE was proposed by Bellare and Rogaway [3] (known as the BR model), which captures several basic security goals for AKE such as *known key security* and *impersonation resilience*. In 2001, Canetti and Krawczyk [6] presented a more delicate security model (i.e., CK model) that captures more complicated situations where the adversary may obtain the leakage information of the *static secret key* and *session state* other than the *target session*. However, both models fail to capture more advanced attacks such as the breaking of perfect forward secrecy (PFS). Informally, PFS implies that no PPT adversary can compute a secret key established before the compromise of the static secret keys of the involved parties. However, as shown by [16], no two-pass AKE protocols based on public-key authentication can achieve PFS. Thus, a weak version, called weak PFS (wPFs), is usually considered for such kind of protocols, which says that no PPT adversary can compute a secret key established in an honestly run of the protocol even if it obtains the involved static secret keys later (i.e., after the session has completed).

During the last 30 years, many practical and provably secure AKE protocols have been proposed based on different number-theoretic problems such as factoring, RSA and the computational/decisional Diffie-Hellman problem. However, as we step into the quantum era, those protocols based on classic hard problems become vulnerable to quantum computers [26]. Namely, those protocols may not be able to provide parties with any security guarantees then. In particular, as far as we know, there are only two papers considered constructing AKE protocols based on Code-related and Lattice-based problems that are believed to be hard for quantum algorithms. Recently, Ding et al. [15] made a big step in constructing a post-quantum KE protocol, and proposed a

very simple and efficient protocol based on LWE assumption (thus, establishing the security of the protocol on the hardness of some lattice problems in the worst-case [23, 25]). Ding et al.'s protocol is very similar to the well-known DH-protocol [9], and has only two-pass messages (Unlike the DH-protocol, the two messages must be exchanged in a sequential way). Similar to the DH-protocol, the protocol in [15] can only be proven secure in the passive model, and suffers from the well-known and practical man-in-the-middle attack (since it cannot provide authentication, i.e., it is not an AKE protocol).

1.1 Our Contribution

In this paper, we proposed an efficient AKE from LWE, thus based on some hard lattice problems in the worst case. For entity authentication, each party is associated with a pair of static public key and secret key, which is assumed to be certified with the party's identity. As Ding et al.'s original KE protocol [15], our protocol has two-pass interactions. However, we derive the session key in a totally different way, which is inspired by the celebrated HMQV protocol. The security of our protocol is proven in an enhanced variant of the BR model that simultaneously captures weakly Perfect Forward Secrecy (wPFS).

2. Preliminaries

2.1 Notation

The natural security parameter throughout the paper is n, and all other quantities are implicit functions of 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.). Vectors are in column form and denoted by bold lower-case letters (e.g., $\mathbf{x}$). Denote the l_2 and l_∞ norm by $\|\cdot\|$ and $\|\cdot\|_\infty$ respectively. The ring of polynomial over integers (or $\mathbb{Z}_q$ for some positive integer q, resp.) is denoted by $\mathbb{Z}[x]$ (or $\mathbb{Z}_q[x]$, resp.).

2.2 Security Model for AKE

In this section, we recall the BR model [3] restricted to the two-pass protocols (i.e., the two involved parties only send one-message to each other as decipted in Fig.1).

Sessions. For security parameter k, we fix a positive integer $N = N(k) \in \mathbb{Z}$ to denote the maximum number of honest parties in the AKE protocol. For simplicity, a party is uniquely identified by integers in $[N] := \{1, \ldots, N\}$, and has a pair of static public key and static secret key pairs

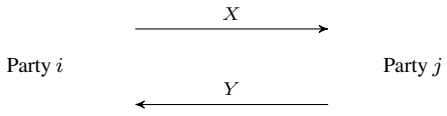

Figure 1: Two-pass Protocol.

(pk_i, sk_i) which are certified with its identity by a Certification Authority (CA). An execution of a protocol is called a session. Session activation is done by an incoming message of the forms (Π, I, i, j) or (Π, R, j, i, X_i), where Π is the protocol identifier, I and R are role identifiers, and $i, j \in [N]$ are party identifiers. If party i is activated with (Π, I, i, j), party i is called the session initiator and will output a message X_i which is intended for party j. If party j is activated with (Π, R, j, i, X_i), j is called the session responder and will output a message Y_j intended for party i. After exchanging two-pass messages, both party i and j compute a session key.

If i is the initiator of a session, the session is identified by $sid = (\Pi, I, i, j)$ or $sid = (\Pi, I, i, j, X_i, Y_j)$. If j is the responder of a session, the session is identified by $sid = (\Pi, R, j, i, X_i, Y_j)$. Fixing a $sid = (\Pi, *, i, j, *[, *])$, the third coordinate i is called the owner of sid, and fourth coordinate j the peer of sid. We say a session is *completed* if its owner computes the session key. The matching session of $sid = (\Pi, I, i, j, X_i, Y_j)$ is session $\widetilde{sid} = (\Pi, R, j, i, X_i, Y_j)$ and vice versa.

Adversarial Capabilities. An adversary $\mathcal{A}$ is modeled as a probabilistic polynomial time (PPT) Turing machine, which controls all communications between parties including session activation. In particular, $\mathcal{A}$ is able to eavesdrop, modify, delete any message sent in the protocol, or inject its own messages. We also suppose $\mathcal{A}$ is able to obtain the leakage information of the session key, and the static secret key of some party. Formally, the above abilities are captured by allowing the adversary to access the following oracles (note that we distinguish the **Send** query in [6] with three notations: $\mathbf{Send}_0$, $\mathbf{Send}_1$ and $\mathbf{Send}_2$, to represent the initial, the first and the second message in the context of two-pass protocols as ours):

- $\mathbf{Send}_0(\Pi, I, i, j)$: $\mathcal{A}$ activates party i as an initiator, and obtains a message X_i intended for party j.
- $\mathbf{Send}_1(\Pi, R, j, i, X_i)$: $\mathcal{A}$ activates party j as a responder by using message X_i on behalf of party i, and obtains a message Y_j intended for party i.
- $\mathbf{Send}_2(\Pi, I, i, j, X_i, Y_j)$: $\mathcal{A}$ sends a message Y_j on behalf of party j to complete a session of i which has responded X_i to a $\mathbf{Send}_0$ query previously.
- $\mathbf{SessionKeyReveal}(sid)$: $\mathcal{A}$ obtains the session key sk for the session sid if the session is completed, else this query is omitted.

- $\mathbf{Corrupt}(i)$: $\mathcal{A}$ obtains the static secret key of party i. If party i is corrupted by the adversary $\mathcal{A}$, we call party i *dishonest*, else we call party i *honest*.
- $\mathbf{Test}(sid^*)$: This query is only restricted to *fresh session* sid (formal definition about freshness is given in Definition 1), and can only be queried once by the adversary $\mathcal{A}$. Upon receiving this query, a random coin $b \leftarrow \{0,1\}$ is chosen. If $b = 0$, the adversary obtains a random key, else it obtains the session key of sid^*.

DEFINITION 1 (Freshness). *Let $sid^* = (\Pi, I, i^*, j^*, X_i, Y_j)$ or $(\Pi, R, j^*, i^*, X_i, Y_j)$ be a completed session between party i^* and party j^*. If the matching session exists, then let $\widetilde{sid}^*$ be the matching session of sid^*. We say session sid^* is fresh if the following conditions holds:*

- *sid^* has not been sent a $\mathbf{SessionKeyReveal}$ query.*
- *$\widetilde{sid}^*$ has not been sent a $\mathbf{SessionKeyReveal}$ query if it exists.*
- *Both party i^* and party j^* have not been sent a $\mathbf{Corrupt}$ query if $\widetilde{sid}^*$ does not exist.*

REMARK 1. *Recall that in the original BR model [3], no corruption query is allowed. In our 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) [16].*

Security Experiment. In security experiment, the adversary $\mathcal{A}$ is given a set of honest parties and makes any sequence of the queries described above, but only makes one $\mathbf{Test}$ query with a fresh session sid^*. The experiment continues until $\mathcal{A}$ outputs a guess b' of b. We say that the adversary $\mathcal{A}$ wins the game if the guess of $\mathcal{A}$ is correct, i.e., $b' = b$. The advantage of $\mathcal{A}$ in the AKE experiment is defined as $\mathrm{Adv}^{\mathrm{AKE}}_{\Pi,\mathcal{A}} = \Pr[\mathcal{A}\ wins] - \frac{1}{2}$.

DEFINITION 2 (Security). *An AKE protocol Π is secure if both the following conditions hold:*

1. *If two honest parties complete matching sessions then, except with negligible probability, they both compute the same session key.*
2. *The advantage $\mathrm{Adv}^{\mathrm{AKE}}_{\Pi,\mathcal{A}}$ is negligible for any PPT adversary $\mathcal{A}$.*

3. Ring-Learning with errors (RLWE)

For any $\alpha \in \mathbb{R}^+$, and $c \in \mathbb{R}$, let $D_{\mathbb{Z},\alpha,c}$ be the discrete Gaussian distribution which can be sampled by first sampling a real $y \in \mathbb{R}$ from the Gaussian of standard deviation α and center c, and then rounding it to the nearest integer x (i.e., $x := \lfloor y \rceil$). For any $c = (c_0, \cdots, c_{n-1}) \in \mathbb{R}^n$, let $D_{\mathbb{Z}^n,\alpha,c}$ be the spherical discrete Gaussian distribution centered at c with standard deviation α, where the i-th dimension is distributed over $D_{\mathbb{Z},\alpha,c_i}$.

Let integer n be a power of 2 (i.e., $n = 2^l$ for some $l \in \mathbb{Z}$), define $f(x) = x^n + 1$, and $R := \mathbb{Z}[x]/\langle f(x)\rangle$ be the ring of all the polynomials in $\mathbb{Z}[x]$ modulo $f(x)$. For any positive integer q, the ring $R_q := \mathbb{Z}_q[x]/\langle f(x)\rangle$ is defined analogously. For any polynomial $y(x) \in R$ (or R_q), we simultaneously treat it as a ring element in R (or R_q) or its coefficient vector in $\mathbb{Z}^n$ (or $\mathbb{Z}_q^n$). The norm of a polynomial $y(x) = \sum_{i=0}^{n-1} y_i x^i \in R$ (or R_q) is the norm of its coefficient vector, e.g., $\|y(x)\|_\infty = \max_i |y_i|$.

LEMMA 1. *Let $f(x) = x^n + 1$ and $R = \mathbb{Z}[x]/\langle f(x)\rangle$ be defined as above. For any $s, t \in R$, we have $\|s \cdot t\| \le \sqrt{n} \cdot \|s\| \cdot \|t\|$ and $\|s \cdot t\|_\infty \le n \cdot \|s\|_\infty \cdot \|t\|\infty$, where the operations are performed in R (i.e., modulo $f(x)$).*

For any $\alpha \in \mathbb{R}^+$, let $\chi_\alpha := D_{\mathbb{Z}^n,\alpha}$. If a bold case $\mathbf{x} \leftarrow \chi_\alpha$ is used, we mean to sample a vector $\mathbf{x}$ from $\mathbb{Z}^n$. Otherwise, we mean to sample a ring element from R_q, whose coefficient vector has distribution χ_α (i.e., $y \leftarrow \chi_\alpha$). We have the following two useful facts:

LEMMA 2 ([13, 20]). *For any real number $\alpha = \omega(\sqrt{\log n})$, we have $\Pr_{\mathbf{x}\leftarrow\chi_\alpha}[\|\mathbf{x}\| > \alpha\sqrt{n}] \le 2^{-n+1}$.*

LEMMA 3 ([5, 14]). *For any real number $\alpha = \omega(\sqrt{\log n})$ and vector $\mathbf{y} \in \mathbb{Z}^n$, 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 10 of [27], which informally says that for appropriate parameter, almost all the ring element sampled from χ_γ are invertible in R_q.

LEMMA 4 ([27]). *Let n be a power of 2, $f(x) = x^n + 1$, and $q = 2^{\omega(\log n)}$ be prime such that $q \mod 2n = 1$. Let $R_q = \mathbb{Z}_q[x]/\langle f(x)\rangle$, and real $\gamma > \sqrt{n \cdot \omega(\log n)} \cdot q^{1/n}$. Then the probability that $\Pr_{d\leftarrow\chi_\gamma}[d \notin R_q^\times]$ is negligible in n, where $R_q^\times$ is the set of invertible elements of R_q. In particular, we have $\Pr_{d_1,d_2\leftarrow\chi_\gamma}[d_1 - d_2 \notin R_q^\times] \ge 1 - negl(n)$, since $d_1 - d_2 \in \chi_{\gamma'}$ for $\gamma' = \sqrt{2}\gamma$ according to the property of Gaussian distribution.*

For simplicity, we describe a special case of the general ring-LWE assumption [18], which we are most interested in this paper, and refer a more details to the original paper [18]. Formally, let $s \leftarrow R_q$ be uniformly random ring element. define $A_{s,\chi_\alpha} \subseteq R_q \times R_q$ as the distribution of variable $(a, as + x)$, where a and x are uniformly chosen from R_q and χ_α respectively, and all operations are performed in R_q.

DEFINITION 3 (Ring-LWE Assumption). *Let R_q, χ_α be defined as in the above paragraphs. The (special case) ring-LWE assumption $\mathrm{RLWE}_{q,\alpha}$ says that, given only polynomial samples, it is hard for any PPT algorithm to distinguish the distribution A_{s,χ_α} from the uniform distribution over $R_q \times R_q$, where s is randomly chosen from R_q.*

The above definition gives the decisional ring-LWE assumption, one can also define the search assumption which asks an algorithm to output s. For some choices of parameter, the search version ring-LWE and the decisional one are polynomially equivalent [18].

PROPOSITION 5 (A special case of [18]). *Let n be a power of 2, real $\alpha = \alpha(n) \in (0,1)$ and prime $q = q(n) \geq 2$ such that $q \mod 2n = 1$, and $\alpha q > \omega(\sqrt{\log n})$. Let $f(x) = x^n + 1$ and $R = \mathbb{Z}[x]/\langle f(x)\rangle$. Then, there exists a polynomial time quantum reduction from $\widetilde{O}(\sqrt{n}/\alpha)$-SIVP (or SVP) in the worst-case to the average-case $\mathrm{RLWE}_{q,\beta}$ given only 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 from the error distribution χ_β [1, 18]. This important variant of the general LWE assumption is known as the "normal form", which is preferable when one tries to control the size of the "error" term, e.g., [4, 5].

Scaling the noise. If t and q are relatively prime, then the ring-LWE assumption still holds if one scale the noise t times, i.e., $(a_i, a_i s + t x_i)$. This variant of ring-LWE has been used to construct several lattice-based cryptographic scheme such as fully homomorphic encryption (FHE) [4, 5] (where $t = 2$ is usually chosen).

4. Authenticated Key Exchange from RLWE

Before presenting our AKE protocol, we first introduce some notations. For odd prime $q > 2$, denote $\mathbb{Z}_q = \{-\frac{q-1}{2}, \ldots, \frac{q-1}{2}\}$. Let $E := \{-\lfloor \frac{q}{4} \rfloor, \ldots, \lfloor \frac{q}{4} \rceil\}$, define its associated characteristic function Cha from $\mathbb{Z}_q$ to $\{0,1\}$:

$$\mathsf{Cha}(v) := \begin{cases} 0 & \text{if } v \in E, \\ 1 & \text{otherwise.} \end{cases}$$

For any $v \in \mathbb{Z}_q$, it is easy to check that $v + \mathsf{Cha}(v) \cdot \frac{q-1}{2} \mod q$ belongs to E. In addition, we define the modular function Mod_2 from $\mathbb{Z}_q \times \{0,1\}$ to $\{0,1\}$:

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

where $v \in \mathbb{Z}_q, w \in \{0,1\}$. For large enough q, we have the following lemma.

LEMMA 6. *Let n be the security parameter, and odd prime $q = 2^{\omega(\log n)}$. For any $b \in \{0,1\}$ and $v' \in \mathbb{Z}_q$, the output distribution of $\mathsf{Mod}_2(v + v', b)$ conditioned on $\mathsf{Cha}(v) \in \{0,1\}$ is statistically close to uniform distribution over $\{0,1\}$, where the probability is taken over the uniform and independent choice of $v \in \mathbb{Z}_q$.*

Proof. We distinguish the proof in two cases:

- If $\mathsf{Cha}(v) = 0$, we have that $v + v' + b \cdot \frac{q-1}{2} \mod q$ is uniformly distributed over $v' + b \cdot \frac{q-1}{2} + E \mod q$. A standard calculation shows that the statistical distance between the output distribution of $\mathsf{Mod}_2(v + v', b)$ (conditioned on $\mathsf{Cha}(v)$) and the uniform distribution over $\{0,1\}$ is at most $\frac{1}{|E|} < \frac{2}{q}$.
- If $\mathsf{Cha}(v) = 1$, we have that $v + v' + b \cdot \frac{q-1}{2} \mod q$ is uniformly distributed over $v' + (b-1) \cdot \frac{q-1}{2} + \widetilde{E} \mod q$, where $\widetilde{E} := E \backslash \{\lfloor \frac{q}{4} \rceil\}$. A standard calculation shows that the statistical distance between the output distribution of $\mathsf{Mod}_2(v, b)$ (conditioned on $\mathsf{Cha}(v)$) and uniform distribution over $\{0,1\}$ is at most $\frac{1}{|E|-1} \leq \frac{3}{q}$.

In all, we have the statistical distance between the output distribution of $\mathsf{Mod}_2(v + v', b)$ conditioned on $\mathsf{Cha}(v)$ and the uniform distribution over $\{0,1\}$ is at most $3/q$, which is negligible in n for $q = 2^{\omega(\log n)}$. □

The next lemma will be used to guarantee the correctness of our scheme. Informally, it says that for sufficiently close $v, w = v + 2e \in \mathbb{Z}_q$, one can compute $\mathsf{Mod}_2(v, \mathsf{Cha}(v))$ by only using the information of w and $\mathsf{Cha}(v)$.

LEMMA 7. *For odd prime $q > 2$, if $w = v + 2e \mod q$ for some $v \in \mathbb{Z}_q$ and $|e| < q/8$, then the value of $\mathsf{Mod}_2(v, \mathsf{Cha}(v))$ is equal to $\mathsf{Mod}_2(w, \mathsf{Cha}(v))$.*

Proof. Note that $w + \mathsf{Cha}(v)\frac{q-1}{2} \mod q = v + \mathsf{Cha}(v)\frac{q-1}{2} + 2e \mod q$. Using the fact that $v + \mathsf{Cha}(v)\frac{q-1}{2} \mod q \in E = \{-\lfloor \frac{q}{4} \rfloor, \ldots, \lfloor \frac{q}{4} \rceil\}$, and $-\frac{q}{8} < e < \frac{q}{8}$, we have $w + \mathsf{Cha}(v)\frac{q-1}{2} \mod q = (v + \mathsf{Cha}(v)\frac{q-1}{2} \mod q) + 2e$ hold over $\mathbb{Z}$. In other words, we have $\mathsf{Mod}_2(w, \mathsf{Cha}(v)) = v + \mathsf{Cha}(v)\frac{q-1}{2} + 2e \mod q \mod 2 = \mathsf{Mod}_2(v, \mathsf{Cha}(v))$. This completes the proof. □

For any element $\mathbf{x} \in \mathbb{Z}_q^n$, one can apply both functions Cha and Mod_2 to $\mathbf{x}$ in an entry-wise way. By a standard hybrid argument, the statistical distance between the output distribution of $\mathsf{Mod}_2(\mathbf{x}, \mathsf{Cha}(\mathbf{x}))$ conditioned on $\mathsf{Cha}(\mathbf{x}) \in \{0,1\}^n$ and the uniform distribution over $\{0,1\}^n$ is at most $\frac{3n}{q}$, where $\mathbf{x}$ is uniformly chosen from $\mathbb{Z}_q^n$. A similar claim holds if one applies the two operations to the coefficient vector of the ring element in R_q.

4.1 The Protocol

In this subsection, we present the full description of our efficient AKE scheme. Let n be a power of 2, and $f(x) = x^n + 1$. Let $q = 2^{\omega(\log n)}$ be an odd prime such that $q \mod 2n = 1$. Let $R = \mathbb{Z}_q/\langle f(x)\rangle$ and $R_q = \mathbb{Z}_q[x]/\langle f(x)\rangle$ defined as in Section 3. For $\gamma \in \mathbb{R}^+$, let $H_1 : \{0,1\}^* \to \chi_\gamma = D_{\mathbb{Z}^n,\gamma}$, which projects a string into a sample in $D_{\mathbb{Z}^n,\gamma}$[1]. Let $H_2 : \{0,1\}^* \to \{0,1\}^k$ for some integer k (i.e., the bit-size of the final shared session key) be the Key derivation function (KDF). Both functions are modeled as random oracles (RO). Let χ_α, χ_β be two Gaussian distributions with parameter $\alpha, \beta \in \mathbb{R}^+$. Let $p_i = a t_i + 2e_i \in R_q$ and t_i be the static public key and secret key of party i, where both s_i, e_i are chosen from the distribution χ_α. Similarly, let $p_j = a t_j + 2e_j \in R_q$ and t_j be the static public key and secret key of party j. Our protocol between party i and party

[1] For instantiation, one can first hash the inputs to a random string by using SHA-2, and then use it as the randomness to sample a vector (or a ring element) from $D_{\mathbb{Z}^n,\gamma}$, which possibly needs the use of a cryptographic pseudorandom generator such as in [24].

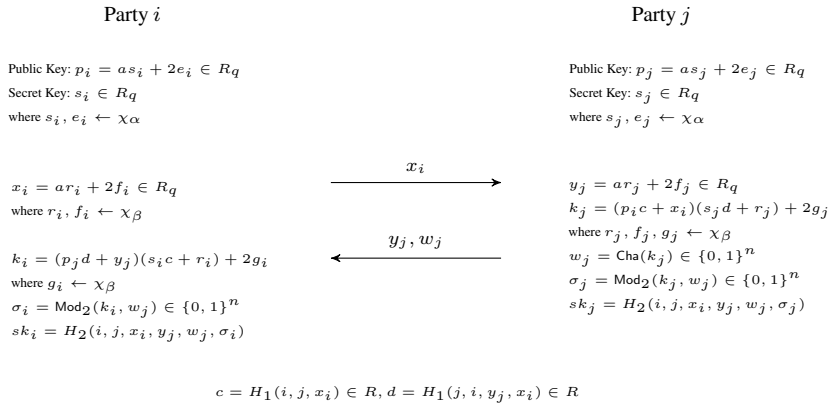

Figure 2: Efficient AKE from Lattices.

j (depicted in Figure 2) works as follows.

Initiation: Party i randomly chooses $r_i, f_i, g_i \leftarrow \chi_\beta$, computes $x_i = ar_i + 2f_i$, and sends x_i to party j.

Response: After receiving x_i from party i, party j randomly chooses $r_j, f_j, g_j \leftarrow \chi_\beta$, and computes $y_j = ar_j + 2f_j$ and $k_j = (p_i c + x_i)(s_j d + r_j) + 2g_j$, where $c = H_1(i, j, x_i), d = H_1(j, i, y_j, x_i) \in \chi_\gamma$. Then, it computes $w_j = \mathsf{Cha}(k_j) \in \{0,1\}^n$, and sends (y_j, w_j) to party i. Finally, it computes $\sigma_j = \mathsf{Mod}_2(k_j, w_j) \in \{0,1\}^n$, and derives $sk_j = H_2(i, j, x_i, y_j, w_j, \sigma_j)$ as the session key.

Finish: After obtaining (y_j, w_j), party i computes $k_i = (p_j d + y_j)(s_i c + r_i) + 2g_i$, where $c = H_1(i, j, x_i)$, and $d = H_1(j, i, y_j, x_i) \in \chi_\gamma$. Finally, it computes $\sigma_i = \mathsf{Mod}_2(k_i, w_j) \in \{0,1\}^n$, and derives the session key $sk_i = H_2(i, j, x_i, y_j, w_j, \sigma_i)$.

4.2 Correctness

To show the correctness of our AKE protocol, it is enough to show that $\sigma_i = \sigma_j$. Note that both σ_i and σ_j are output by Mod_2 with the same second input $\mathsf{Cha}(k_j)$. According to Lemma 7, we only have to show that the entries in k_i and k_j are sufficiently close. Note that both parties will compute k_i and k_j as follows:

$$\begin{aligned} k_i &= (p_j d + y_j)(s_i c + r_i) + 2g_i \\ &= a(s_j d + r_j)(s_i c + r_i) \\ &\quad + (2e_j d + 2f_j)(s_i c + r_i) + 2g_i \\ &= a(s_i c + r_i)(s_j d + r_j) + 2\widetilde{g}_i \end{aligned}$$

$$\begin{aligned} k_j &= (p_i c + x_i)(s_j d + r_j) + 2g_j \\ &= a(s_i c + r_i)(s_j d + r_j) \\ &\quad + (2e_i c + 2f_i)(s_j d + r_j) + 2g_j \\ &= a(s_i c + r_i)(s_j d + r_j) + 2\widetilde{g}_j \end{aligned}$$

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

4.3 Concrete Choices of Parameter

Following [8, 12, 18], we make use of the canonical embedding in the analysis of our scheme. 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

$$\|a\|_\infty^{\text{can}} = \|\sigma(a)\|_\infty.$$

We will use the following useful properties [8, 12, 18] of $\|\cdot\|_\infty^{\text{can}}$:

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

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 $\delta = r^2 n$ (note that $\mathbf{z}_m$ has Euclidean norm exactly n). Following [12], 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 estimate the size of the error term of 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 the coefficient vectors of e_i, e_j, are chosen from χ_α, the coefficient vec-

tors of c, d are chosen from χ_γ, and the coefficient vectors of $f_i, f_j, r_i, r_j, g_i, g_j$ are chosen from χ_γ. Using the bounds in previous paragraph, we have $\|e_j d\|_\infty^{\text{can}}, \|s_i c\|_\infty^{\text{can}} \leq 16\alpha\gamma n$, and $\|f_j\|_\infty^{\text{can}}, \|r_i\|_\infty^{\text{can}}, \|g_i\|_\infty^{\text{can}} \leq 6\beta\sqrt{n}$. Thus, we have $\|\widetilde{g}_i\|_\infty^{\text{can}} \leq (16\alpha\gamma n + 6\beta\sqrt{n})^2 + 6\beta\sqrt{n}$, and the same bound hold for $\|\widetilde{g}_j\|_\infty^{\text{can}}$. Considering $\alpha/\beta = 2^{-\omega(\log n)}$ is required in our security proof (see Section 6), we will set

$$\beta \gg 16\alpha\gamma n \tag{1}$$

to guarantee a small statistical distance according to Lemma 3. Thus, with high probability, we have both $\|\widetilde{g}_j\|_\infty^{\text{can}}$ and $\|\widetilde{g}_j\|_\infty^{\text{can}}$ are smaller than $37\beta^2 n$. Thus, for correctness, it is enough to set q such that

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

Though the ring-LWE problem enjoys a worst-case connection to some hard problems on ideal lattices such as Shortest Vector Problem (SVP) [18], the connection as summarized in Proposition 5 seems less powerful to estimate the actual security for concrete choices of parameter. For this, several works [7, 11, 12, 21, 22] took account experimental results to estimate the hardness of (R)LWE. In our setting, we use the result in [12], which showed 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):

$$n \geq \frac{\log(q/\alpha)(k+110)}{7.2} \tag{3}$$

As recommended in [12, 17], it is enough to set the Gaussian parameter $\alpha \geq 3.2$ so that the discrete Gaussian $D_{\mathbb{Z}^n,\alpha}$ approximates the continuous Gaussian D_α extremely well[2]. In our case, we fix $\alpha = 3.397$ for a better performance when using the Gaussian sampling algorithm [10]. 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 (2) and the security condition (3) for each expected security level. We also take account of equation (3) in the choices of security parameter, and consider $\log \frac{\beta}{\alpha}$ as an "index of confidence" for corresponding security level. Note that n is required to be a power of 2 in our protocol (i.e., it is very sparsely distributed)[3], we can simply tries several possible values of n. In Table 2, we present several candidate choices of parameters, and estimate the sizes of public keys, secret keys, and communication costs.

[2] Note that the Gaussian parameter $s = \alpha\sqrt{2\pi}$ (> 8.0) is used in [17] due to notation difference.

[3] 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 [19] to give a tighter choice of parameter for desired security levels.

5. Implementations and Benchmarks

In this section, we present the details of our proof-of-concept implementations, and give the timings for concrete choices of parameters.

5.1 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 p and $x^n + 1$. An element in R_q can be natural written as a polynomial of degree less than n with coefficients in $\mathbb{Z}_p$, e.g., $g(x) = \sum_{i=0}^{n-1} g_i x^i \in R_q$ (thus it is enough to identity an element by using its coefficient vector $(g_0, \cdots, g_{n-1}) \in R_q$). The addition operation of two elements $g, h \in R_q$ can be done by taking a

5.2 Gaussian Sampling

5.3 Hashing to $D_{\mathbb{Z}^n,\gamma}$

5.4 Timings

6. Security

At first, we would like give some intuitions on the security of our AKE protocol. Note that in the protocol, the public element a and the static public key of each party actually consists of a standard RLWE tuple with Gaussian parameter α. Thus, under the RLWE assumption, the static public key of each party is computationally indistinguishable from a random element in R_q. Similarly, both the interchanged messages x_i and y_j are also computationally indistinguishable from a random element in R_q under the LWE assumption with Gaussian parameter β.

To show the randomness of the session key, we take party j as an example, since the session key of party i should be equal to that of party j by the correctness. Note that if k_j is random over R_q, we have σ_j is statistically close to $\{0,1\}^n$ even conditioned on w_j by Lemma 6. Since H_2 is a random oracle, we have that sk_j is uniformly over $\{0,1\}^k$ as expected. Now, let's check the randomness of $k_j = (p_i c + x_i)(s_j d + r_j) + 2g_j$. As one can imagine, we also want to establish the randomness of k_j on the hardness of the (decisional) RLWE problem, since it is actually a RLWE instance with public element $p_i c + x_i$, the secret $s_j d + r_j$, as well as error g_j. Informally, we will prove that k_j is statistically close to a real RLWE 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 random over R_q whenever p_i or x_i is random (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 $\alpha/\beta = 2^{-\omega(\log n)}$, the distribution of $r_j \leftarrow \chi_\beta$ statistically hides the term $s_j d$ according to Lemma 3 (note that $s_j \leftarrow \chi_\alpha$, and $d \in \chi_\gamma$).

Formally, let N be the maximum number of parties, and m be maximum number of sessions for each party. We separate the security proof for the initiator and responder in the next two subsections, respectively.

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

Table 1: Choices of Parameters (we set $\gamma \approx n^{2/3}$ for the use of Lemma 4), and the bound 6α with $\mathrm{erfc}(6) \approx 2^{-55}$ is used to estimate the secret key size as we've done before.)

n	Security (expt.)	$\log q$	$\log(\beta/\alpha)$	Initiation. (ms)	Response. (ms)	Finish. (ms)
1024	80 bits	40	8	()	()	()
2048	80 bits	78	27	()	()	()
2048	128 bits	63	19	()	()	()
4096	128 bits	125	50	()	()	()
4096	192 bits	97	36	()	()	()
4096	256 bits	81	28	()	()	()

Table 2: Timings of Proof-of-Concept Implementations (The figures in the bracket indicates the timings with pre-computing.)

6.1 Security for the Initiator

In this subsection, we prove the security of our protocol when the initiator is the owner of test session. Let $sid^* = (\Pi, I, i^*, j^*, x_{i^*}, (y_{j^*}, w_{j^*}))$ be the test session, chosen by the adversary $\mathcal{A}$. We distinguish the following two types of adversaries:

Type I: y_{j^*} is output by a session of j^* activated by a $\mathbf{Send}_1(\Pi, R, j^*, i^*, x_{i^*})$.

Type II: y_{j^*} is **not** output by any session of j^* activated by a $\mathbf{Send}_1(\Pi, R, j^*, i^*, x_{i^*})$.

Type I and **Type II** give a complete partition of all the adversaries that choose sid^* as the test session. It is easy to see that if the adversary is a **Type II** one, then the test session has no matching session. However, it does not mean that sid^* matches an existing session if the adversary is of Type I, since the pair (y_{j^*}, w_{j^*}) may not be output by any session of party j^* activated by a $\mathbf{Send}_1(\Pi, R, j^*, i^*, x_{i^*})$. In our security proof, we allow the **Type I** adversary to obtain the static secret keys of both party i^* and j^* by corrupting both parties, to capture the security of weak perfect forward secrecy (wPFS) (but no corruption to either party i^* or party j^* is allowed for a **Type II** adversary).

6.1.1 Type I Adversary

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

LEMMA 8. *If $\alpha/\beta = 2^{-\omega(\log n)}$ and LWE$_{q,n,\alpha}$ 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 $0 \le l \le 4$. We use boxes to highlight the changes of each game with respect to its previous game.

Game $G_{1,0}$**.** $\mathcal{S}$ randomly chooses $i^*, j^* \leftarrow \{1, \ldots, N\}$ and $s_{i^*}, s_{j^*} \leftarrow \{1, \ldots, m\}$, 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^*, and y_{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, $\mathcal{S}$ randomly chooses $a \leftarrow R_q$, 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, randomly chooses $out \in \chi_\gamma$, 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, randomly chooses a vector $out \in \{0,1\}^k$, 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, $\mathcal{S}$ randomly chooses $r_i, f_i \leftarrow \chi_\beta$, returns $x_i = ar_i + 2f_i \in \mathbb{Z}_q^{n \times n}$ to $\mathcal{A}$ on behalf of i.
- $\mathbf{Send}_1(\Pi, R, j, i, x_i)$: $\mathcal{S}$ randomly chooses $r_j, f_j \leftarrow \chi_\beta$, 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, $\mathcal{S}$ randomly chooses $b \leftarrow \{0,1\}$ and $sk_i' \leftarrow \{0,1\}^k$. If $b = 0$, $\mathcal{S}$ returns sk_i', else it returns the real session sk_i of sid.

Game $G_{1,1}$. $\mathcal{S}$ first computes $\boxed{y_j' = ar_j' + 2f_j'}$, where $r_j', f_j' \leftarrow \chi_\beta$. Then, it behaves almost the same as in $G_{1,0}$, 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 $G_{1,0}$. Otherwise, randomly choose $d \leftarrow \chi_\gamma$, and compute $\boxed{y_j = y_j' - p_j d}$. $\mathcal{S}$ aborts if there is a tuple $((j, i, y_j, x_i), *)$ in the L_1 list. Else, the simulator $\mathcal{S}$ 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 $g_j \leftarrow \chi_\beta$. Finally, it honestly computes w_j and sk_j following the protocol, and sends (y_j, w_j) to $\mathcal{A}$.

Game $G_{1,2}$. $\mathcal{S}$ first computes $\boxed{x_i' = ar_i' + 2f_i'}$, where $r_i', f_i' \leftarrow \chi_\beta$. Then, it behaves almost the same as in $G_{1,1}$, 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 $G_{1,1}$. Otherwise, $\mathcal{S}$ randomly chooses $c \leftarrow \chi_\gamma$, and computes $\boxed{x_i = x_i' - p_i c}$. $\mathcal{S}$ 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 $G_{1,1}$. 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 $g_i \leftarrow \chi_\beta$. Finally, it honestly computes sk_i following the protocol.

Game $G_{1,3}$. $\mathcal{S}$ randomly chooses $\boxed{x_i' \leftarrow R_q}$, and behaves almost the same as in $G_{1,2}$ 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^*, or (y_j, w_j) is output by the s_j^*-th session of party j^*, $\mathcal{S}$ behaves the same as in Game $G_{1,2}$. Else, it randomly chooses $\boxed{sk_i \leftarrow \{0,1\}^k}$ as the session key.

Game $G_{1,4}$. $\mathcal{S}$ randomly chooses $\boxed{y_j' \leftarrow R_q}$, and behaves almost the same as in $G_{1,3}$ 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 $G_{1,3}$. Otherwise, randomly choose $d \leftarrow \chi_\gamma$, and compute $y_j = y_j' - p_j d$. $\mathcal{S}$ aborts if there is a tuple $((j, i, y_j, x_i), *)$ in the L_1 list. Else, it adds $((j, i, y_j, x_i), d)$ into L_1. Then, $\mathcal{S}$ randomly chooses $\boxed{k_j \leftarrow R_q}$, and computes w_j, σ_j following the protocol. If $\mathcal{A}$ has made a query $H_2(i, j, x_i, y_j, w_j, \sigma_j)$, $\mathcal{S}$ aborts the simulation. Else, it randomly chooses $sk_j \leftarrow \{0,1\}^k$, and sets $H_2(i, j, x_i, y_j, w_j, \sigma_j) = sk_j$. Finally, it sends (y_j, w_j) to $\mathcal{A}$.

To finish the proof, we prove the following claims.

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 $i^*, j^* \leftarrow \{1, \ldots, N\}$ and $s_{i^*}, s_j^* \leftarrow \{1, \ldots, m\}$ independently from the view of $\mathcal{A}$. □

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 LWE$_{q,n,\alpha}$ is hard, then* $\Pr[F_{1,l}] = \Pr[F_{1,0}] - negl(n)$.

Proof. Note that $(a, y_j' = ar_j' + 2f_j')$ is actually a LWE tuple with $r_j', f_j' \leftarrow \chi_\beta$, we have that y_j' is computationally indistinguishable from uniform distribution over R_q. Thus, the probability that $\mathcal{A}$ guesses the correct $y_j = y_j' - p_j d$ before is negligible. Besides, since $p_j = as_j + 2e_j \in R_q$ with $s_j, e_j \leftarrow \chi_\alpha$, we have $y_j = a(r_j' - s_j d) + 2(f_j' - e_j d)$. As analyzed in Section 4.2, we have that the norm of each entry in both $s_j d$, and $e_j d$ is at most $\tau = \alpha\gamma n\sqrt{n}$, and $|\tau|/\beta = 2^{-\omega(\log n)}$ (since $\alpha/\beta = 2^{-\omega(\log n)}$ and $\gamma = poly(n)$). 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 $G_{1,0}$, which completes the proof. □

CLAIM 3. *If $\alpha/\beta = 2^{-\omega(\log n)}$ and LWE$_{q,n,\alpha}$ 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 scheme, this claim follows. □

Note that we change the real session key sk_i in Game $G_{1,2}$ with a randomly 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 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 technique reasons. Instead, we 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 $l = 2, 3, 4$. Formally, we have the following claim.

CLAIM 4. *If LWE$_{q,n,\alpha}$ 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 LWE 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, $\Pr[F_{1,3}|\neg Q_{1,3}] = \Pr[F_{1,2}|\neg Q_{1,2}] - negl(n)$. □

CLAIM 5. *Under the LWE$_{q,n,\beta}$ 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 LWE 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 LWE 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:

- $\mathbf{Send}_1(\Pi, R, j, i, x_i)$: If $(i, j) \neq (i^*, j^*)$, or it is not the s^*_j-th session of j^*, $\mathcal{D}$ answers the query as in Game $G_{1,3}$. Otherwise, it randomly chooses $d \leftarrow \chi_\gamma$, computes $\boxed{y_j = y'_j - p_j d}$, and 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. Then, $\mathcal{D}$ sets $\boxed{k_j = v_2}$, computes w_j, σ_j following the protocol, and aborts if $\mathcal{A}$ has made a H_2 query $H_2(i, j, x_i, y_j, w_j, \sigma_j)$. Else, it sets $H_2(i, j, x_i, y_j, w_j, \sigma_j) = sk_j$ with a randomly chosen $sk_j \leftarrow \{0,1\}^k$. 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 s', $\mathcal{A}$ is in Game $G_{1,3}$, else it is in Game $G_{1,4}$, which completes the proof. □

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 is randomly chosen from the uniform distribution over R_q, which is independent from both the public keys and transcripts (except w_j). This actually holds even if the adversary obtains sk_j by using a session key reveal query, since sk_j is randomly chosen and H_2 is a random oracle. Let k_i be the element "computed" by $\mathcal{S}$, by the correctness of the protocol k_i and k_j are sufficiently close, namely, $k_i = k_j + \hat{g}$ for some $\hat{g}$ with short element. Since both the public keys and transcripts (except w_j) are randomly and independent from k_j, we have $\hat{g}$ is also independent from k_j in the adversary's view. Note that $\mathsf{Mod}_2(k_i, w'_j) = \mathsf{Mod}_2(k_j + \hat{g}, w'_j)$, we have that $\sigma'_i = \mathsf{Mod}_2(k_i, w'_j)$ conditioned on w_j is also 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 k. 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 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 H_2 is a random oracle. If the adversary doesn't make a query with σ_i exactly, the distribution of sk_i is uniform over $\{0,1\}^k$ in the adversary's view. □

In all, we have $\Pr[F_{1,0}] = \Pr[F_{1,2}] + negl(n)$ by claim 2 and 3. By claim 4, 5 and 6, we have $\Pr[Q_{1,2}] = \Pr[Q_{1,4}] + negl(n) = negl(n)$, and $\Pr[F_{1,2}|\neg Q_{1,2}] = \Pr[F_{1,4}|\neg Q_{1,4}] + negl(n)$. By the law of total probability, we have $\Pr[F_{1,2}] = \Pr[F_{1,2}|\neg Q_{1,2}](1 - \Pr[Q_{1,2}]) + \Pr[F_{1,2}|Q_{1,2}]\Pr[Q_{1,2}]$, thus $\Pr[F_{1,2}] = \Pr[F_{1,2}|\neg Q_{1,2}] - negl(n)$. Combining this with claim 7, we have $\Pr[F_{1,0}] = \Pr[F_{1,2}] + negl(n) = 1/2 + negl(n)$. □

6.1.2 Type II Adversary

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

LEMMA 9. *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. As before, we prove this lemma via a sequence of games $G_{2,l}$ for $0 \le l \le 6$.

Game $G_{2,0}$. $\mathcal{S}$ randomly chooses $i^*, j^* \leftarrow \{1, \ldots, N\}$ and $s_{i^*} \leftarrow \{1, \ldots, m\}$, 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, $\mathcal{S}$ randomly chooses $a \leftarrow R_q$, 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, randomly chooses $out \in \chi_\gamma$, 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, randomly chooses a vector $out \in \{0,1\}^k$, 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, $\mathcal{S}$ randomly chooses $r_i, f_i \leftarrow \chi_\beta$, returns $x_i = ar_i + 2f_i \in \mathbb{Z}_q^{n \times n}$ to $\mathcal{A}$ on behalf of i.
- $\mathbf{Send}_1(\Pi, R, j, i, x_i)$: $\mathcal{S}$ randomly chooses $r_j, f_j \leftarrow \chi_\beta$, 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, $\mathcal{S}$ randomly chooses $b \leftarrow \{0,1\}$ and $sk_i' \leftarrow \{0,1\}^k$. If $b = 0$, $\mathcal{S}$ returns sk_i', else it returns the real session sk_i of sid.

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 $r_i', f_i' \leftarrow \chi_\beta$. Then, it randomly chooses $c \leftarrow \chi_\gamma$, 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 $r_j', f_j' \leftarrow \chi_\beta$. Then, it randomly chooses $d \leftarrow \chi_\gamma$, 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, $\mathcal{S}$ 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 $g_j \leftarrow \chi_\beta$. 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 $g_i \leftarrow \chi_\beta$. Finally, $\mathcal{S}$ computes sk_i following the protocol.

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 $\boxed{p_{j^*} \leftarrow R_q}$.

Game $G_{2,3}$. $\mathcal{S}$ first computes $\boxed{x_i' = ar_i' + 2f_i'}$, where $r_i', f_i' \leftarrow \chi_\beta$. 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, $\mathcal{S}$ randomly chooses $c \leftarrow \chi_\gamma$, 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 $g_i \leftarrow \chi_\beta$. Finally, it computes sk_i following the protocol.

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 $r_i' \leftarrow \chi_\beta$, 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 $f_i' \leftarrow \chi_\beta$. 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 $g_i \leftarrow \chi_\beta$. Finally, it computes sk_i following the protocol.

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, $\mathcal{S}$ randomly chooses $\boxed{k_i \leftarrow R_q}$ and computes sk_i following the protocol.

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

To finish the proof, we prove the following claims.

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 $i^*, j^* \leftarrow \{1, \ldots, N\}$ and $s_{i^*} \leftarrow \{1, \ldots, m\}$ without $\mathcal{A}$ knowing it. □

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. □

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. □

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. □

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 $f'_i, g_i \leftarrow \chi_\beta$. 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. □

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 v_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 random chosen from R_q.

Now, we formally show that $Q_{2,5}$ will happen with negligible probability, which makes heavy 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 $g_i \leftarrow \chi_\beta$ 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 $\tilde{d} \leftarrow \chi_\gamma$, 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 $\Pr[\text{double-}Q_{2,5}] \geq \Pr[Q_{2,5}](\Pr[Q_{2,5}]/q_h - 2^{-n})$.

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 di-

rectly used to solve the $\mathrm{LWE}_{q,n,\alpha}$ 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 randomly chosen from R_q.) 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 $\Pr[\text{double-}Q_{2,6}] \leq 2^{-n} + negl(n)$, which is negligible in n. This completes the proof. □

In summary, by Claim 14 and 15, we have $\Pr[\text{double-}Q_{2,5}] = negl(n)$, which implies that $\Pr[Q_{2,5}] = negl(n)$ by the condition that $\Pr[\text{double-}Q_{2,5}] \geq \Pr[Q_{2,5}](\Pr[Q_{2,5}]/q_h - 1/2^{n^2})$. Combining this with Claim 13, we have $\Pr[F_{2,4}] = 1/2 + negl(n)$. A simple calculation shows that $\Pr[F_{2,0}] = 1/2 + negl(n)$. This completes the proof. □

6.2 Security for the Responder

In this subsection, we prove the security of our protocol, where the responder is the owner of test session. Let $sid^* = (\Pi, R, j^*, i^*, x_{i^*}, (y_{j^*}, w_{j^*}))$ be the test session, as before we distinguish the following three types of adversaries:

Type III: x_{i^*} is **not** output by any session of i^* activated by a $\mathbf{Send}_0(\Pi, I, i^*, j^*)$.

Type IV: x_{i^*} is output by a session of i^* activated by a $\mathbf{Send}_0(\Pi, I, i^*, j^*)$, but i^* never completes the session, or it completes the session with exact y_{j^*}.

Type V: x_{i^*} is output by a session of i^* activated by a $\mathbf{Send}_0(\Pi, I, i^*, j^*)$, but i^* completes the session with another $y'_j \neq y_{j^*}$.

Type III, **Type IV** and **Type V** give a complete partition of all the adversaries that choose sid^* as the test session. It is easy to see that if the adversary is a **Type III** or **Type V** one, then the test session has no matching session. In our security proof, we allow a **Type IV** adversary $\mathcal{A}$ to obtain the static secret keys of both party i^* and j^* by corrupting both parties, to capture the security of weak perfect forward secrecy (wPFS) (but no corruption to either party i^* or party j^* is allowed for a **Type III** or **Type V** adversary).

6.2.1 Type III Adversary

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

LEMMA 10. *If $\alpha/\beta = 2^{-\omega(\log n)}$ and $\mathrm{LWE}_{q,n,\alpha}$ 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 $0 \leq l \leq 7$.

Game $G_{3,0}$**.** $\mathcal{S}$ randomly chooses $i^*, j^* \leftarrow \{1, \ldots, N\}$ and $s_{j^*} \leftarrow \{1, \ldots, m\}$, and hopes that the adversary will choose $sid^* = (\Pi, R, j^*, i^*, x_{i^*}, (y_{j^*}, w_{j^*}))$ as the test session, where (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, $\mathcal{S}$ randomly chooses $a \leftarrow R_q$, 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, randomly chooses $out \in \chi_\gamma$, 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, randomly chooses a vector $out \in \{0,1\}^k$, 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, $\mathcal{S}$ randomly chooses $r_i, f_i \leftarrow \chi_\beta$, returns $x_i = ar_i + 2f_i \in \mathbb{Z}_q^{n \times n}$ to $\mathcal{A}$ on behalf of i.
- $\mathbf{Send}_1(\Pi, R, j, i, x_i)$: $\mathcal{S}$ randomly chooses $r_j, f_j \leftarrow \chi_\beta$, 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, $\mathcal{S}$ randomly chooses $b \leftarrow \{0,1\}$ and $sk'_i \leftarrow \{0,1\}^k$. If $b = 0$, $\mathcal{S}$ returns sk'_i, else it returns the real session sk_i of sid.

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 $i \neq i^*$, $\mathcal{S}$ answers the query as in Game $G_{3,0}$. Else, $\mathcal{S}$ computes $\boxed{x'_i = ar'_i + 2f'_i}$, where $r'_i, f'_i \leftarrow \chi_\beta$. Then, it randomly chooses $c \leftarrow \chi_\gamma$, 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 $j \neq i^*$, $\mathcal{S}$ answers the query as in Game $G_{3,0}$. Else, $\mathcal{S}$ computes $\boxed{y'_j = ar'_j + 2f'_j}$, where $r'_j, f'_j \leftarrow \chi_\beta$. Then, it randomly chooses $d \leftarrow \chi_\gamma$, 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 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 $g_j \leftarrow \chi_\beta$. 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 $i \neq i^*$, $\mathcal{S}$ 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 $g_i \leftarrow \chi_\beta$. Finally, $\mathcal{S}$ computes sk_i following the protocol.

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 $\boxed{p_{i^*} \leftarrow R_q}$.

Game $G_{3,3}$**.** $\mathcal{S}$ first computes $\boxed{y'_j = ar'_j + 2f'_j}$, where $r'_j, f'_j \leftarrow \chi_\beta$. 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, $\mathcal{S}$ randomly chooses $d \leftarrow \chi_\gamma$, 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 $g_j \leftarrow \chi_\beta$. Finally, it computes w_j and sk_j following the protocol, and sends (y_j,w_j) to $\mathcal{A}$.

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 $r'_j \leftarrow \chi_\beta$, 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 $f'_j \leftarrow \chi_\beta$. 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, $\mathcal{S}$ randomly chooses $d \leftarrow \chi_\gamma$, 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 $g_j \leftarrow \chi_\beta$. Finally, it computes w_j and sk_j following the protocol, and sends (y_j,w_j) to $\mathcal{A}$.

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, $\mathcal{S}$ randomly chooses $d \leftarrow \chi_\gamma$, 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 randomly chooses $\boxed{k_j \leftarrow R_q}$, 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 randomly chooses $\boxed{sk_j \leftarrow \{0,1\}^k}$, and sets $H_2(i,j,x_i,y_j,w_j,\sigma_j) = sk_j$. Finally, it sends (y_j,w_j) to $\mathcal{A}$.

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

To finish the proof, we prove the following claims.

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 $i^*, j^* \leftarrow \{1,\ldots,N\}$ and $s^*_j \leftarrow \{1,\ldots,m\}$ independently from the view of $\mathcal{A}$. □

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. □

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. □

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. □

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. □

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 $g_j \leftarrow \chi_\beta$ 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 $\tilde{c} \leftarrow \chi_\gamma$, 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 $\Pr[\text{double-}Q_{3,5}] \geq \Pr[Q_{3,5}](\Pr[Q_{3,5}]/q_h - 2^{-n})$.

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 **??**. 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 $(\tilde{c} - c)$ has full-rank). 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. □

In summary, by Claim 22 and 23, we have $\Pr[\text{double-}Q_{3,5}] = negl(n)$, which implies that $\Pr[Q_{3,5}] = negl(n)$ by the inequality that $\Pr[\text{double-}Q_{3,5}] \geq \Pr[Q_{3,5}](\Pr[Q_{3,5}]/q_h - 2^{-n})$. Combining this with Claim 21, we have $\Pr[F_{3,4}] = 1/2 + negl(n)$. A simple calculation shows that $\Pr[F_{3,0}] = 1/2 + negl(n)$. This completes the proof. □

6.2.2 Type IV Adversary

In this subsection, we prove that our AKE is secure against any PPT **Type IV** 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 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 $G_{4,0}$**.** $\mathcal{S}$ randomly chooses $i^*, j^* \leftarrow \{1, \ldots, N\}$ and $s_{i^*}, s_{j^*} \leftarrow \{1, \ldots, m\}$, 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, $\mathcal{S}$ randomly chooses $a \leftarrow R_q$, 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, randomly chooses $out \in \chi_\gamma$, 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, randomly chooses a vector $out \in \{0,1\}^k$, 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, $\mathcal{S}$ randomly chooses $r_i, f_i \leftarrow \chi_\beta$, returns $x_i = ar_i + 2f_i \in \mathbb{Z}_q^{n \times n}$ to $\mathcal{A}$ on behalf of i.
- $\mathbf{Send}_1(\Pi, R, j, i, x_i)$: $\mathcal{S}$ randomly chooses $r_j, f_j \leftarrow \chi_\beta$, 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, $\mathcal{S}$ randomly chooses $b \leftarrow \{0,1\}$ and $sk_i' \leftarrow \{0,1\}^k$. If $b = 0$, $\mathcal{S}$ returns sk_i', else it returns the real session sk_i of sid.

Game $G_{4,1}$. $\mathcal{S}$ first computes $\boxed{y_j' = ar_j' + 2f_j'}$, where $r_j', f_j' \leftarrow \chi_\beta$. 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, $\mathcal{S}$ randomly chooses $d \leftarrow \chi_\gamma$, 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 list, and computes $\boxed{k_j = (p_i c + x_i) r_j' + 2g_j}$, where $c = H_1(i, j, x_i)$ and $g_j \leftarrow \chi_\beta$. Finally, it computes w_j and sk_j following the protocol, and sends (y_j, w_j) to $\mathcal{A}$.

Game $G_{4,2}$. $\mathcal{S}$ first computes $\boxed{x_i' = ar_i' + 2f_i'}$, where $r_i', f_i' \leftarrow \chi_\beta$. 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, $\mathcal{S}$ randomly chooses $c \leftarrow \chi_\gamma$, and computes $\boxed{x_i = x_i' - p_i c}$. $\mathcal{S}$ 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 $\boxed{k_i = (p_j d + y_j) r_i' + 2g_i}$, where $d = H_1(j, i, y_j, x_i)$ and $g_i \leftarrow \chi_\beta$. Finally, it computes sk_i following the protocol.

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

– $\mathbf{Send}_2(\Pi, I, i, j, x_i, (y_j, w_j))$: 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^*, $\mathcal{S}$ behaves the same as in Game $G_{4,2}$. Else, it randomly chooses $\boxed{sk_i \leftarrow \{0,1\}^k}$ as the session key.

Game $G_{4,4}$. $\mathcal{S}$ randomly chooses $\boxed{y_j' \leftarrow R_q}$, and behaves almost the same as in $G_{4,3}$, 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 $G_{4,3}$. Otherwise, $\mathcal{S}$ randomly chooses $d \leftarrow \chi_\gamma$, 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, $\mathcal{S}$ randomly chooses $\boxed{k_j \leftarrow R_q}$, 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, it randomly chooses $\boxed{sk_j \leftarrow \{0,1\}^k}$, and sets $H_2(i, j, x_i, y_j, w_j, \sigma_j) = sk_j$. Finally, it sends (y_j, w_j) to $\mathcal{A}$.

To finish the proof, we prove the following claims.

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

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

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. □

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. □

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. □

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. □

In all, we have $\Pr[F_{4,0}] = \Pr[F_{4,2}] + negl(n)$ by claim 25 and 26. By claim 28 and 29, we have $\Pr[Q_{4,3}] = \Pr[Q_{4,4}] = negl(n)$, and $\Pr[F_{4,3}] = \Pr[F_{4,4}] + negl(n)$. Since $\Pr[F_{4,3}] = \Pr[F_{4,3}|Q_{4,3}]\Pr[Q_{4,3}] + \Pr[F_{4,3}|\neg Q_{4,3}](1 - \Pr[Q_{4,3}])$, we have $\Pr[F_{4,3}] = \Pr[F_{4,3}|\neg Q_{4,3}] - negl(n)$. Combining this with claim 27 and claim 30, we have $\Pr[F_{4,0}] = \Pr[F_{4,2}] + negl(n) = 1/2 + negl(n)$. □

6.2.3 Type V Adversary

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

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

Game $G_{5,0}$**.** $\mathcal{S}$ randomly chooses $i^*, j^* \leftarrow \{1, \ldots, N\}$ and $s_{i^*}, s_{j^*} \leftarrow \{1, \ldots, m\}$, 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, $\mathcal{S}$ randomly chooses $a \leftarrow R_q$, 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, randomly chooses $out \in \chi_\gamma$, 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, randomly chooses a vector $out \in \{0,1\}^k$, 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, $\mathcal{S}$ randomly chooses $r_i, f_i \leftarrow \chi_\beta$, returns $x_i = ar_i + 2f_i \in \mathbb{Z}_q^{n \times n}$ to $\mathcal{A}$ on behalf of i.
- $\mathbf{Send}_1(\Pi, R, j, i, x_i)$: $\mathcal{S}$ randomly chooses $r_j, f_j \leftarrow \chi_\beta$, 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, $\mathcal{S}$ randomly chooses $b \leftarrow \{0,1\}$ and $sk_i' \leftarrow \{0,1\}^k$. If $b = 0$, $\mathcal{S}$ returns sk_i', else it returns the real session sk_i of sid.

Game $G_{5,1}$**.** $\mathcal{S}$ first computes $\boxed{y_j' = ar_j' + 2f_j'}$, where $r_j', f_j' \leftarrow \chi_\beta$. 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, $\mathcal{S}$ randomly chooses $d \leftarrow \chi_\gamma$, 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 $g_j \leftarrow \chi_\beta$. Finally, it computes w_j and sk_j following the protocol, and sends (y_j, w_j) to $\mathcal{A}$.

Game $G_{5,2}$**.** $\mathcal{S}$ first computes $\boxed{x_i' = ar_i' + 2f_i'}$, where $r_i', f_i' \leftarrow \chi_\beta$. 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, $\mathcal{S}$ randomly chooses $c \leftarrow \chi_\gamma$, 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 $g_i \leftarrow \chi_\beta$. Finally, it computes sk_i following the protocol.

Game $G_{5,3}$**.** $\mathcal{S}$ randomly chooses $\boxed{x_i' \leftarrow R_q}$, 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 randomly chooses $\boxed{sk_i \leftarrow \{0,1\}^k}$.

Game $G_{5,4}$**.** $\mathcal{S}$ randomly chooses $\boxed{y_j' \leftarrow R_q}$, 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, $\mathcal{S}$ randomly chooses $d \leftarrow \chi_\gamma$, 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 randomly chooses $\boxed{k_j \leftarrow R_q}$, 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 randomly chooses $\boxed{sk_j \leftarrow \{0,1\}^k}$, and sets $H_2(i, j, x_i, y_j, w_j, \sigma_j) = sk_j$. Finally, it sends (y_j, w_j) to $\mathcal{A}$.

To finish the proof, we prove the following claims.

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 $i^*, j^* \leftarrow \{1, \ldots, N\}$ and $s_{i^*}, s_j^* \leftarrow \{1, \ldots, m\}$ independently from the view of $\mathcal{A}$. □

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. □

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. □

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. □

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. □

In all, we have $\Pr[F_{5,0}] = \Pr[F_{5,2}] + negl(n)$ by claim 32 and 33. By claim 35, we have $\Pr[F_{5,3}] = \Pr[F_{5,4}] + negl(n)$. Combining this with claim 34 and claim 36, we have $\Pr[F_{5,0}] = \Pr[F_{5,2}] + negl(n) = 1/2 + negl(n)$. □

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. 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.

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

[5] Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In P. Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 505–524. Springer Berlin Heidelberg, 2011.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] X. L. Jintai Ding. A simple provably secure key exchange scheme based on the learning with errors problem. Cryptology ePrint Archive, Report 2012/688, 2012. http://eprint.iacr.org/.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

[23] C. Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, STOC '09, pages 333–342, New York, NY, USA, 2009. ACM.

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

[25] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, STOC '05, pages 84–93, New York, NY, USA, 2005. ACM.

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

[27] 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.