Complete Attack on RLWE Key Exchange with reused keys, without Signal Leakage

Jintai Ding[1], Scott Fluhrer[2], and Saraswathy RV[1]

[1] University of Cincinnati
[2] Cisco Systems

Abstract. Key Exchange (KE) from RLWE (Ring-Learning with Errors) is a potential alternative to Diffie-Hellman (DH) in a post quantum setting. Key leakage with RLWE key exchange protocols in the context of key reuse has already been pointed out in previous work. The Signal leakage attack relies on changes in the signal sent by the responder reusing his key, in a sequence of key exchange sessions initiated by an attacker with a malformed key. A possible defense against this attack would be by requiring the initiator of the key exchange to send the signal, which is the one pass case of the KE protocol. The initial attack described bu Fluhrer is designed in such a way that it only works on Peikert's KE protocol and its variants that derives the shared secret from the most significant bits of the approximately equal keys computed by both parties. It does not work on the Ding's key exchange that uses the least significant bits to derive a shared key. In this work, we describe a new attack on Ding's one pass case without relying on the signal function output but using only the information of whether the final key of both parties agree. We also use LLL reduction to make the adversary's keys random looking to the party being compromised. This completes the series of attacks on RLWE key exchange with key reuse for all variants in both cases of the initiator and responder sending the signal. This work shows that when a party fixes their public key for a long term, the protocol can always be broken by a malicious user. Moreover, we show that the previous Signal leakage attack can be made more efficient with fewer queries and how it can be extended to Peikert's key exchange, which was used in the BCNS implementation and integrated with TLS and a variant used in the New Hope implementation.

Keywords: RLWE, key exchange, post quantum, key reuse, active attacks.

1 Introduction

Post-quantum cryptography refers to cryptographic algorithms (usually public key algorithms) that are thought to be secure against an attack by a quantum computer. According to studies, a sufficiently large quantum computer can efficiently break most widely used public-key algorithms such as RSA and ECDSA. In 1994, Shor devised a quantum algorithm [26] that can be used to solve the discrete log problem (the hardness of which the security of different variants of Diffie-Hellman (DH) key exchange algorithms are based on) in polynomial time by quantum computers [26]. This led to the search for quantum resistant cryptographic protocols. Cryptographic primitives that are believed to be resistant to quantum computer attacks include Multivariate, Hash based, Code based and Lattice based, that have their security based on mathematical problems that are hard to solve with currently known efficient quantum algorithms. In the recent years, lattice based cryptographic primitives have proven to have versatile applications in Key Exchange, Signature, FHE (Fully Homomorphic Encryption) and more. Key exchange protocols allow two or more participants to derive a shared cryptographic key, often used for authenticated encryption. RLWE (Ring-Learning With Errors) key exchange is a lattice based variant of DH type protocol that also has properties like quantum resistance, forward and provable security that makes it a desirable replacement for currently used DH protocols. In RLWE key exchange, the two parties in a key exchange initially compute approximately equal values, after which one of the parties sends information about the interval in which its computation of the key value lies, to the other party. Then, both the parties use this information to derive a final shared secret. This additional information, referred to as the signal was exploited by active adversaries to retrieve the secret of a reused key as shown in [11] and applies to the RLWE based key exchange protocol in [16] and all its variants [24],[7],[5]. The signal function attack works under the set up of the responder (party that reuses its key) sending the signal and can be defended against by requiring the initiator to send the signal. In this work, we explore a new and more sophisticated attack to recover

the secret without using the signal function output by querying the party with reused key for mismatch of the final shared key. The attack is set up for the one pass case of the protocol, when the initiator (instead of the responder) performing the key exchange sends the signal to the other party. The other details of the KE protocol remains the same as the two pass case. This work is an attack description on the KE protocol in [16] which uses the least significant bits of the computed keys to derive a shared secret key. The work in [12] focuses on an attack on KE protocol in [24] and its variants [7],[5] that uses the most significant bits to derive the shared key. With this attack description, we show that all RLWE based KE protocols are vulnerable to attacks when keys are reused.

1.1 Previous Work

Key leakage in RLWE based key exchange with key reuse was pointed out in [17] but without any concrete description of an attack to exploit the leakage. An attack was described by Fluhrer in [12] with the attack strategy that tries to use the agreement of final shared key to derive information about the secret but does not work in the case of [16] where the final shared key is derived from the least significant bits.

Another attack presented in [15] is executed on the one pass case of the Authenticated key exchange protocol from RLWE in [27] and exploits properties of the CRT (Chinese Remainder Theorem) basis of R_q. It recovers every CRT coefficient of the secret s of a key $p = as + 2e$ in order to recover s, with an attack complexity of $\frac{q-1}{2}(\delta.q^{\delta} + q - \frac{\delta.q}{n})$, where δ is a moderately large constant.

The signal function attack is used to recover information about the secret of a reused key in an attack description in [11]. It works by looking at the number of times the signal value of the key computation k_B changes when varying k across all values in $\mathbb{Z}_q$ in the adversary's public key of $p_A = as_A + ke_A$. The number of signal changes is expected to be exactly 2 times the secret value by the choice of s_A, e_A and the definition of the signal region. The secret is recovered with $2q$ queries to the party with the reused key.

1.2 Our Contributions

We present a new attack on RLWE based key exchange in the context of key reuse. We focus on the one pass case of the KE protocol since the other case can already be attacked with previous work. Thus, having the initiator send the signal is not a possible defense against attacks with key reuse and unsuccessful key exchange sessions can be used to reveal information about the secret. We carefully work through the details of the adversary's queries and perform an attack with query complexity $O(n^2\alpha)$. The query complexity is independent of q, making it more efficient than the signal function attack. Here, α is the standard deviation of the error distribution. The goal of the work is to show that RLWE keys when reused in key exchange can always be exploited and broken. The success of such attacks comes from the hardness of distinguishing RLWE samples from uniform. Section 3 reviews definitions and results that are relevant to indistinguishability of RLWE samples. We have verified the success of our attacks with experiments.

This attack does not rely on the leakage of the signal and can still be applied to protocols in the case that the initiator is required to send the signal to avoid the signal function attack. Although the attack approach is similar to [12] in using key mismatch to compromise the secret, we use a different strategy for the attack. In [12], the attack focuses on key exchange protocols that derive the most significant bits of the approximately equal key computed. The approach is to query for the boundary between 0 and 1, corresponding to the signal quadrants defined in the protocol. But this does not work in the case of key exchange protocols that use the least significant bit to derive the final shared key. In our attack, the attacker forces the other party to reveal information about the secret from the final key mismatch. A mismatch in the final shared key is linked with a change in sign of a particular coefficient of the intermediate (approximately equal) key computed. We choose a secret $s_{\mathcal{A}}$ for the adversary such that the $n-1$-th coordinate of the key computation is small, by solving linear equations involving the reused public key $p_B = as_B + 2e_B$. Here, p_B is an RLWE public key with secrets s_B, e_B sampled from an error distribution and a uniform randomly sampled from the ring. To recover useful information using success or failure of a session, the attacker's secret needs to be small. This is because the attacker only checks for match or mismatch of final key in one coordinate to recover the secret s_B but with a key exchange session failure, the attacker cannot know which coordinates of the key did not match. So, keeping $s_{\mathcal{A}}$ small ensures that the other coordinates are computed following the protocol and matches for both parties, implying that a key exchange session success or failure relies on match or mismatch of the specific coordinate of the final key. To ensure that

$s_{\mathcal{A}}$ is small, we apply the LLL reduction algorithm on the solution space of the system of linear equations solved. We refer to this work as a complete attack since it fills the gap on available attacks for all variants of RLWE based KE protocols and both cases where the initiator and responder sends the signal. We also discuss the signal function attack to make it more efficient in terms of the query complexity. Later, we discuss about extending the signal attack to the key exchange in [24] which follows the same approach as in [16] and uses a slightly different signal function, referred to as the cross rounding function. The BCNS implementation uses the key exchange in [24]. The New Hope implementation uses a modified version with a different error distribution and error reconciliation, and was tested in Google Chrome Canary browser for its post quantum experiment[2].

2 Organization

In Section 3, We discuss some background on RLWE and the functions used in the key exchange protocol. Section 4 describes the relevance of this work, in terms of practical implementation. The protocol being attacked is reviewed in Section 5. The attack is described in Section 6, which is divided into two parts - simplified and improved. The simplified attack aims at providing a basic understanding of the attack assuming that the attacker's secret is 0. The improved attack further builds on the simplified case to describe the actual attack strategy. Other subsections of this attack section discusses query complexity and experiments we performed to verify the attack. Section 7 reviews the signal function attack and describes how it can be applied to the KE protocol in [24]. Section 8 discusses about reducing the query complexity of the signal function attack.

3 Preliminaries

3.1 Notation

Let n be an integer and a power of 2. Define $f(x) = x^n + 1$ and consider the ring $R := \mathbb{Z}[x]/\langle f(x)\rangle$. For any positive integer q, we define the ring $R_q = \mathbb{Z}_q[x]/\langle f(x)\rangle$ analogously, where the ring of polynomials over $\mathbb{Z}$ (respectively $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$) we denote by $\mathbb{Z}[x]$ (respectively $\mathbb{Z}_q[x]$). Let χ_α denote the discrete Gaussian distribution on R_q, naturally induced by that over $\mathbb{Z}^n$ with standard deviation α. Let the norm $\|p\|$ of a polynomial $p \in R$ (or R_q) be defined as the norm of the corresponding coefficient vector in $\mathbb{Z}$ (or $\mathbb{Z}_q$). For a vector $v = (v_0, \ldots, v_{n-1})$ in $\mathbb{R}^n$ or $\mathbb{C}^n$ and $p \in [1, \infty)$, we define the ℓ_p norm as $\|v\|_p = (\sum_{i=0}^{n-1} |v_i|^p)^{1/p}$ and the ℓ_∞ norm as $\|v\|_\infty = max_{i\in[n]}|v_i|$. The ℓ_2 norm corresponds to the ℓ_p norm with $p = 2$ and is denoted as $\|.\|$ in this paper. In applying the norms, we assume the coefficient embedding of elements from R to $\mathbb{R}^n$. For any element $s = \sum_{i=0}^{n-1} s_i x^i$ of R, we can embed this element into $\mathbb{R}^n$ as the vector $(s_0, \ldots s_{n-1})$.

3.2 Learning with Errors and RLWE

A Lattice $L(b_1, \ldots, b_n) = \{\sum_{i=1}^{n} x_i b_i | x_i \in \mathbb{Z}\}$ is formed by integer linear combinations of n linearly independent vectors $b_1, \ldots, b_n \in \mathbb{R}^n$ called the Lattice Basis. In 1996, Ajtai's seminal result [4] heralded the use of lattices for constructing cryptographic systems, with the security based on hardness of problems such as the Shortest Vector Problem (SVP) and Closest Vector Problem (CVP). The Learning with Errors (LWE) problem introduced by Oded Regev in 2005 [25] is a generalization of the parity-learning problem. The reduction from solving hard problems in lattices in the worst case to solving LWE in the average case provides strong security guarantees for LWE based cryptosystems, yet it is not efficient enough for practical applications due to its large key sizes of $O(n^2)$. Ring-Learning with Errors(RLWE) is the version of LWE in the ring setting, that overcomes the efficiency disadvantages of LWE. Similar to LWE, there is a quantum reduction from solving worst case lattice problems in ideal lattices to solving the RLWE problem in average case. The search version of RLWE is to find a secret s in R_q given $(a, as + e)$ for polynomial number of samples, where a is sampled uniform from R_q and e is sampled according to the error distribution χ_α. An equivalent problem of the search version is the decision version which is commonly used for security proof of cryptographic algorithms based on RLWE. Let A_{s,χ_α} denote the distribution of the pair $(a, as + e)$, where a, s is sampled uniformly from R_q and e is sampled according to the error distribution χ_α. The

decision version of the RLWE problem is to distinguish A_{s,χ_α} from the uniform distribution on $R_q \times R_q$ with polynomial number of samples. We provide the definition of the Discrete Gaussian distribution (error distribution) here:

Discrete Gaussian Distribution

Definition 1. *[27] For any positive real $\alpha \in \mathbb{R}$, and vectors $c \in \mathbb{R}^n$, the continuous Gaussian distribution over $\mathbb{R}^n$ with standard deviation α centered at c is defined by the probability function $\rho_{\alpha,c}(x) = (\frac{1}{\sqrt{2\pi}\alpha})^n exp(\frac{-\|x-c\|^2}{2\alpha^2})$. For integer vectors $c \in \mathbb{R}^n$, let $\rho_{\alpha,c}(\mathbb{Z}^n) = \sum_{x\in\mathbb{Z}^n} \rho_{\alpha,c}(x)$. Then, we define the discrete Gaussian distribution over $\mathbb{Z}^n$ as $D_{\mathbb{Z}^n,\alpha,c}(x) = \frac{\rho_{\alpha,c}(x)}{\rho_{\alpha,c}(\mathbb{Z}^n)}$, where $x \in \mathbb{Z}^n$. The subscripts s and c are taken to be 1 and 0 (respectively) when omitted.*

In practice, we use a spherical gaussian distribution where each coordinate is sampled independently from a one dimensional discrete gaussian distribution $D_{\mathbb{Z},\alpha}$. In Regev's paper introducing the LWE problem [25], the discretization of a continuous distribution ϕ is defined as follows:

$$\bar{\phi} : \mathbb{Z}_q \to \mathbb{R}^+ : \bar{\phi}(x) = \int_{(x-1/2)/q}^{(x+1/2)/q} \phi(x)dx$$

The distribution $\bar{\phi}$ is referred to as the Rounded Gaussian distribution when ϕ denotes the continuous one dimensional gaussian distribution. Sampling from the distribution $\bar{\phi}$ can be done by sampling from the continuous distribution ϕ and rounding to the nearest integer mod q. In the definition of LWE problem in Regev's paper, the Rounded gaussian function is used as the error distribution. This has been replaced with a discrete gaussian for better analysis in later work and it can be verified that the two distributions are statistically close.

We recall two useful lemmas here:

Lemma 1 ([27]). *Let $f(x)$ and R be defined as above. Then, for any $s, t \in R$, we have $\|s \cdot t\| \leq \sqrt{n} \cdot \|s\| \cdot \|t\|$ and $\|s \cdot t\|_\infty \leq n \cdot \|s\|_\infty \cdot \|t\|_\infty$.*

Lemma 2 ([14, 21]). *For any real number $\alpha = \omega(\sqrt{\log n})$, we have $\Pr_{\boldsymbol{x}\leftarrow\chi_\alpha}[\|\boldsymbol{x}\| > \alpha\sqrt{n}] \leq 2^{-n+1}$.*

The *normal form* [8, 9] of the RLWE problem is by modifying the above definition by choosing s from the error distribution χ_α rather than uniformly. It has been proven that the ring-LWE assumption still holds even with this variant [6, 20].

Proposition 1 ([20]). *Let n be a power of 2, let α be a real number in $(0, 1)$, and q a prime such that $q \bmod 2n = 1$ and $\alpha q > \omega(\sqrt{\log n})$. Define $R = \mathbb{Z}[x]/\langle x^n + 1\rangle$ as above. Then there exists a polynomial time quantum reduction from $\tilde{O}(\sqrt{n}/\alpha)$-SIVP (Short Independent Vectors Problem) in the worst case to average-case RLWE$_{q,\beta}$ with ℓ samples, where $\beta = \alpha q \cdot (n\ell/\log(n\ell))^{1/4}$.*

For the Key Exchange from RLWE presented in [16], the signal function is required for the two parties in the key exchange to derive a final shared key. The signal function is usually sent by the responding party to the initiator of the key exchange, which gives additional information about whether the responder's key computed lies in a specific region. The case when the initiator sends the signal is the One pass protocol. It is formally defined as follows:

Definition 2. *Signal function: Given $\mathbb{Z}_q = \{-\frac{q-1}{2}, \ldots, \frac{q-1}{2}\}$ and the middle subset $E \coloneqq \{-\lfloor \frac{q}{4} \rfloor, \ldots, \lfloor \frac{q}{4} \rceil\}$, we define Sig as the characteristic function of the* complement *of E: $Sig(v) = 0$ if $v \in E$ and 1 otherwise.*

Definition 3. *The final key is derived using the Mod_2 function (Reconciliation) defined as below: $\mathsf{Mod}_2 : \mathbb{Z}_q \times \{0, 1\} \to \{0, 1\}$:*

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

To discuss the key exchange in [24], we recall the following definitions: Let $I_0 \coloneqq \{0, 1, \ldots \lfloor \frac{q}{4} \rceil - 1\}$, $I_1 \coloneqq \{-\lfloor \frac{q}{4} \rfloor, \ldots - 1\}$ and $E' \coloneqq [-\frac{q}{8}, \frac{q}{8}) \cap \mathbb{Z}$. Let $I'_0 = \frac{q}{2} + I_0$ and $I'_1 = \frac{q}{2} + I_1$.

Definition 4. *The cross rounding function,* $<.>_2 : \mathbb{Z}_q \to \mathbb{Z}_2$ *is defined as*

$$<v>_2 \coloneqq \lfloor \frac{4}{q}.v \rfloor \bmod 2$$

Definition 5. *The randomization function* $dbl : \mathbb{Z}_q \to \mathbb{Z}_{2q}$*, which is used in the case of an odd modulus* q *is defined as* $dbl(v) = 2v - \bar{e}$*, where* $\bar{e}$ *is uniform over* $\{0, 1\}$*. In practice,* $\bar{e}$ *is chosen such that* $Pr(\bar{e} = 0) = \frac{1}{2}$ *and* $Pr(\bar{e} = \pm 1) = \frac{1}{4}$

Definition 6. *The final key derivation of the initiator of the key exchange uses the reconciliation function,* $rec : \mathbb{Z}_q \times \mathbb{Z}_2 \to \mathbb{Z}_2$ *which is defined as*

$$rec(w, b) = \begin{cases} 0 & w \in I_b + E(\mod q), \\ 1 & \textit{otherwise.} \end{cases}$$

Definition 7. *The Modular rounding function* $\lfloor . \rceil_2 : \mathbb{Z}_q \to \mathbb{Z}_2$*, is defined as*

$$\lfloor x \rceil_2 = \lfloor \frac{2}{q}.x \rceil \bmod 2.$$

4 Key Reuse in Practical Applications

The motivation for this work is the adaptability of post quantum RLWE based key exchange in real applications, where key reuse is widely adopted for efficiency reasons. Key exchange is a fundamental cryptographic primitive in secure protocols such as SSH (Secure Shell), the Transport Layer Security (TLS) and IKE (Internet Key Exchange). The Transport Layer security and SSL (Secure Sockets Layer, predecessor of TLS) are cryptographic protocols in the application layer of TCP/IP reference model to provide security in a communication network and ensures privacy and data integrity between communicating applications. A key exchange is executed in the handshake protocol before any application data is transmitted and enables the client and server to establish an algorithm and shared key for encrypted communication.

TLS handshake roughly performs the following exchange: The client starts with a `Client_Hello` message sent to the server with the message containing data such as a random nonce, protocol version, session ID, ciphersuites. It also sends its share of the necessary parameters and key share (public key) for the key exchange through the "`key_share`" extension. The server responds with a `Server_hello` and server "`key_share`" and sends the certificate if it is to be authenticated. The Server computes the shared secret and then responds with a `Finished` message. The client, on receiving the server "`key_share`" also computes the shared secret and sends the `Finished` message. After the client and server exchange the `Finished` messages, the encrypted application data can be exchanged between the client and the server. This is a typical 2-RTT (Round Trip Time) handshake. In practice, session resumption is an important aspect of TLS that allows the client to use information from a previous exchange for estabilishing faster connections. In TLS 1.2 and below, this is achieved using session tickets and session IDs to establish 1 RTT connections. TLS 1.3 draft 7 [1] allows reuse of keys in a 0 RTT (Round Trip Time) that maintains a long term public key which is sent through a `ServerConfiguration` message to the client in the initial handshake. The 0-RTT mode has been changed in the later draft 13 of TLS 1.3 [3] to use a PSK(preshared key) identity that is sent to the client on the initial handshake to be used for encrypting early data on future handshakes.

Another such instance of key reuse appears in the Internet Key Exchange (IKE). The first pair of messages in an IKE session negotiate cryptographic algorithms, exchange nonces, and do a Diffie-Hellman exchange. Currently with classical DH, some implementations of IKE do reuse the keys for improved computational efficiency and latency. IPSec is a cryptographic protocol suite in the internet layer of the TCP/IP reference model and is used to secure communications over Internet Protocol (IP) networks. It provides authentication, data integrity and confidentiality to IP datagrams. The IKE (Internet Key Exchange) is often used in IPSec to establish a security Association (SA) between nodes in the network to agree on the cryptographic algorithms and keys to use for the algorithms. Possibilities of such key reuse in important internet protocols calls for an analysis of its effect when transitioning to quantum safe primitives.

5 The Protocol

Let the notations be as defined in Section 3. We recall the key exchange protocol in [16]. Generate the parameters q, n, α for the protocol and choose public $a \leftarrow R_q$ uniformly.

Init : Party A chooses a secret $s_A \leftarrow \chi_\alpha$ and computes $p_A = as_A + 2e_A$, where $e_A \leftarrow \chi_\alpha$. Party A then sends p_A to party B.

Response : On receiving p_A, party B chooses a secret element s_B and $e_B, g_B \leftarrow \chi_\alpha$. Party B then computes $p_B = as_B + 2e_B$, $k_B = p_A s_B + 2g_B$ and $w_B = Sig(k_B)$, sends p_B, w_B. party B obtains a shared key $sk_B = \mathsf{Mod}_2(k_B, w_B)$.

Finish : On receiving p_B, w_B from party B, party A computes $k_A = s_A p_B + 2g_A$, where $g_A \leftarrow \chi_\alpha$ and obtains the shared key $sk_A = \mathsf{Mod}_2(k_A, w_B)$.

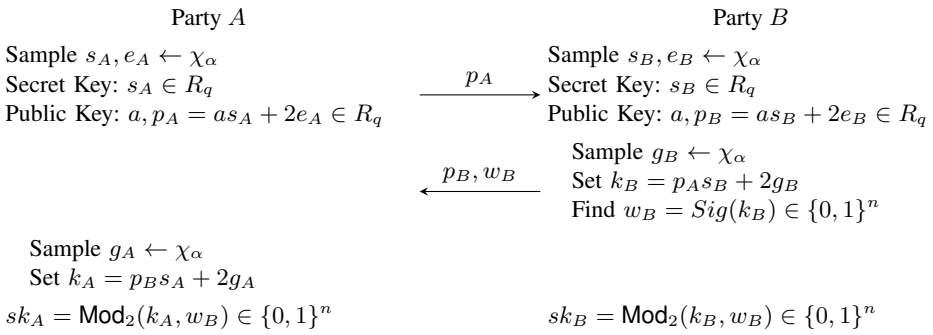


Fig. 1: Protocol from [16]

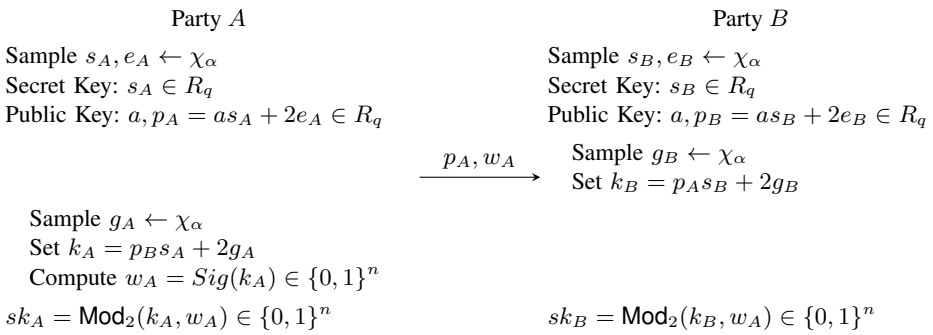


Fig. 2: Protocol from [16] - One pass case

6 New Attack Using Key Mismatch - one pass case

Suppose that party B reuses its public key p_B and $\mathcal{A}$ is an active adversary with the knowledge of p_B and with the ability to initiate multiple key exchange sessions to query party B. We present an attack in the one

pass case of the KE protocol, in which the adversary can initiate multiple key exchange sessions with party B and use key mismatch in each session to retrieve the secret s_B. We use the notation $p_{\mathcal{A}}$ for the public key of the adversary and $s_{\mathcal{A}}, e_{\mathcal{A}}$ for the corresponding secret and error respectively.

6.1 Simplified Attack

We first consider the simpler case when party B does not add the error term g_B to its key computation k_B, to explain the attack strategy and then extend to the case of adding the noise.

Choice of $s_{\mathcal{A}}$ and $e_{\mathcal{A}}$: The attacker chooses $s_{\mathcal{A}}$ to be 0 in R_q (This is later improved by choosing $s_{\mathcal{A}}$ to be non-zero and hence $p_{\mathcal{A}}$ to be random looking). For recovering the i-th coefficient $s_B[i]$, the attacker $\mathcal{A}$ chooses an $e_{\mathcal{A}}$ with coefficient vector that consists of all zeros, except for the coordinate $n-1-i$, for which it is 1, and coordinate $n-1-j$, which is a small integer k. So, we have $e_{\mathcal{A}}[i] = 0$ for all $i = 0, \ldots n-1$ except $i = n-1-i, n-1-j$ and $e_{\mathcal{A}}[n-1-i] = 1$, $e_{\mathcal{A}}[n-1-j] = k$. He then performs the protocol honestly, except that he deliberately flips bit $n-1$ of the signal vector $w_{\mathcal{A}}$ that he sends. The index j is chosen such that $s_B[j] = \pm 1$. Thus, the attacker first needs to identify such a j. This is explained in Section 6.4.

Remark 1. The attacker can actually flip any bit of the signal $w_{\mathcal{A}}$ and use the corresponding index of the final shared key to look for mismatch to recover the secret; we use the bit $n-1$ because that allows us to ignore the complications with signs during polynomial multiplication in the ring, simplifying the attack. For example, if we want to use the 0-th coefficient of the final shared key to recover value of $s_B[i]$, we can choose the $(n-i)$-th coordinate of the coefficient vector of $e_{\mathcal{A}}$ to be -1 and $(n-j)$-th coordinate to be $-k$ and flip the 0-th bit of the signal $w_{\mathcal{A}}$ that he sends.

If we look at party B's computation of the key k_B, we have $k_B = s_B p_{\mathcal{A}}$ which results in $k_B[n-1] = 2s_B[i] + 2ks_B[j] = 2s_B[i] + 2k$ by the choice of $s_{\mathcal{A}}, e_{\mathcal{A}}$ of the attacker. Since the $(n-1)$-th coordinate of the signal $w_{\mathcal{A}}$ received from the attacker is flipped to be 1, we have $sk_B[n-1] = k_B[n-1] + \frac{q-1}{2} \bmod q \bmod 2$. Also, the attacker's final shared key is $sk_{\mathcal{A}} = 0$ since $s_{\mathcal{A}} = 0$.

Constructing Oracle $\mathcal{B}$: We build an oracle $\mathcal{B}$ that performs the action of party B and the adversary $\mathcal{A}$ has access to this oracle to make multiple queries. $\mathcal{B}$ takes $(p_{\mathcal{A}}, w_{\mathcal{A}}, sk_{\mathcal{A}})$ as input where $p_{\mathcal{A}}, sk_{\mathcal{A}}$ corresponds to the public key and the final shared key respectively of $\mathcal{A}$. $w_{\mathcal{A}}$ corresponds to the signal sent by $\mathcal{A}$ with the $n-1$ bit flipped to 1. The oracle computes $k_B = p_{\mathcal{A}} s_B$ and $sk_B = \mathsf{Mod}_2(k_B, w_{\mathcal{A}})$. $\mathcal{B}$ then outputs 1 if $sk_B = sk_{\mathcal{A}}$ and 0 otherwise.

From the construction of the oracle, it is clear that the oracle indicates if a key exchange session is successful or not. Then the attacker can invoke the oracle $\mathcal{B}$ with $p_{\mathcal{A}}$ corresponding to different values of k to check for key mismatch. Because the attacker performs the protocol mostly honestly (and both $s_{\mathcal{A}}$ and $e_{\mathcal{A}}$ qualify as small vectors until k remains small), the attacker can compute the value sk_B, except for index $n-1$, for which he flips the signal bit. The attacker can then determine the value of that bit by guessing a sk_B that has a 0 in that position and the computed values elsewhere (In the case of $s_{\mathcal{A}} = 0$, all other index values are also 0 but this is not the case when $s_{\mathcal{A}}$ is not 0), and checking with the oracle $\mathcal{B}$ to see if his guess was correct.

Flipping the signal bit allows the attacker to force party B to change the parity of the final $sk_B[n-1]$ before the mod 2 operation, in every instantiation of a session with the attacker. This is useful in associating a change in output of $\mathcal{B}$ with a change from positive to negative values of $k_B[n-1]$ or vice versa as explained here:

Notice that the terms $2s_B[i] + 2k$ of $sk_B[n-1]$ are even and also from the usual choice of parameters for RLWE (following from Lemma 1) such that $q = 1 \bmod 2n$, we have $\frac{q-1}{2}$ to be even. Thus, if $s_B[i]$ is negative, we have $sk_B[n-1] = 0$ as long as $2s_B[i]+2k$ is negative and there is no change in the parity. So, a query to $\mathcal{B}$ with these values of $(p_{\mathcal{A}}, w_{\mathcal{A}}, sk_{\mathcal{A}})$ results in an output of 1. As k increases in value, we can see that $k_B[n-1]$ changes from negative to positive values. As this happens, we have $sk_B[n-1] = 1$ since the addition of $\frac{q-1}{2}$ to a positive value changes its parity by the representation of $\mathbb{Z}_q$ to be $\{-\frac{q-1}{2} \ldots \frac{q-1}{2}\}$

and the output of $\mathcal{B}$ becomes 0. So, a change from negative to positive values of $k_B[n-1]$ results in a change of output from 1 to 0 of $\mathcal{B}$.

Also, if $s_B[i]$ is negative, then as k varies, $k_B[n-1]$ changes from negative to positive values at the point when $2k$ is greater than the absolute value of $2s_B[i]$ i.e, $k>|s_B[i]|$. Thus, the k value when there is a change in output of $\mathcal{B}$ reveals the value of $s_B[i]$.

But if $s_B[i]$ is positive, sk_B does not change parity until k takes on larger values (change only occurs when $2s_B[i]+2k>q$ by the representation of $\mathbb{Z}_q$). As k becomes large, the output of $\mathcal{B}$ is no longer reliable to indicate the difference in the $n-1$-th index since the errors amplify and other indexes of sk_B are not guaranteed to match with that of $sk_{\mathcal{A}}$. To handle this, the query that the attacker sends modifies the $e_{\mathcal{A}}$ chosen above so that $e_{\mathcal{A}}[n-1-j]=-k$, when $s_B[i]$ is positive.

So if $s_B[i]$ is positive, then we have $k_B[n-1]=2s_B[i]-2k$ and this value changes from positive to negative as k increases when $k>s_B[i]$. Also, $sk_B[n-1]=1$ as long as $k_B[n-1]$ is positive because of the change in parity of $sk_B[n-1]$ caused by adding $\frac{q-1}{2}$ and results in the output of $\mathcal{B}$ to be 0. As the value changes to negative, the output of $\mathcal{B}$ changes to 1.

The attack can be summarized with the following steps for every coefficient i of the secret s_B, i from 0 to $n-1$:

Step 1: The first step is to create an $e_{\mathcal{A}}$ as described above and thus involves identifying a j such that $s_B[j]=\pm 1$. This is discussed in detail in Section 6.4. The consequent steps here assume that the attacker succeeds in finding such a j.

Step 2: Now, the attacker needs to resolve the sign of $s_B[i]$ to create queries accordingly. The attacker queries $\mathcal{B}$ with $p_{\mathcal{A}}, w_{\mathcal{A}}, sk_{\mathcal{A}}$. Here, $p_{\mathcal{A}}=2e_{\mathcal{A}}$ corresponds to $e_{\mathcal{A}}[n-1-j]=k=0$ and will result in $k_B[n-1]=2s_B[i]$. $w_{\mathcal{A}}$ and $sk_{\mathcal{A}}$ correspond to the signal with the last bit flipped to 1 and final shared key of the attacker with the guess for the $n-1$ coefficient to be 0, respectively. This can be used by the attacker to determine the sign of $s_B[i]$ since the sign of $k_B[n-1]$ and $s_B[i]$ are the same. If the output is 1 (i.e, the final keys match), the attacker concludes that the sign of $s_B[i]$ is negative and if the output is 0, then the sign is positive. One problem here is that if the coefficient value is 0, the output of $\mathcal{B}$ would still be 1. So, to identify 0 values, the attacker can query again corresponding to $k=0$ but with $e_{\mathcal{A}}[n-1-i]=-1$ which results in $k_B[n-1]=-2s_B[i]$. If the output of $\mathcal{B}$ remains the same for both queries for a coefficient, then the coefficient value has to be 0.

Step 3: If $s_B[i]$ is negative, as inferred from the previous step, the attacker creates e_B with $e_{\mathcal{A}}[n-1-j]=k$ and varies k over values from 0 until there is a change in the output of $\mathcal{B}$. If s_B is positive, e_B is created with $e_{\mathcal{A}}[n-1-j]=-k$.

Step 4: Looking for the k value when the output of $\mathcal{B}$ changes from 0 to 1 reveals the exact value of a negative $s_B[i]$ and a change from 1 to 0 reveals the value of a positive $s_B[i]$.
Note here that the output of $\mathcal{B}$ only gives information about whether the final shared key of both parties agree or not. It is not possible for the attacker to know which coordinates of the final key match and which ones don't. But the attack works since a change in the output bit of $\mathcal{B}$ for smaller values of k would mean that it is caused by the $n-1$-th index by the bit flip in the signal as $s_{\mathcal{A}}$ and $e_{\mathcal{A}}$ remain small. As k becomes larger, there is no assurance for the keys to match in the other indexes.

Step 5 : The recovered secret s_B can be verified by checking the distribution of p_B-as_B.

Remark 2. Consider the case $s_{\mathcal{A}}[j]=-1$; by following the above logic, the attacker can flip the sign of k in $e_{\mathcal{A}}$ to recover $s_B[i]$.

As we can repeat the above process for all i, this means we can read party Bs secret key directly. The attack for one query is shown in Figure 3 to recover a negative $s_B[i]$. Here, the adversary computes $sk_{\mathcal{A}}=0$ and B computes $sk_B=0$ until $k_B[n-1]$ is negative.

6.2 Extending the attack when adding the error g_B

In this case, the number of queries required to recover $s_B[i]$ increases compared to the steps above, due to the complexity involved in eliminating the effect of the noise $g_B[n-1]$. The strategy here is to look at the distribution of k values when there is a change in the output of $\mathcal{B}$, while running the attack on the same

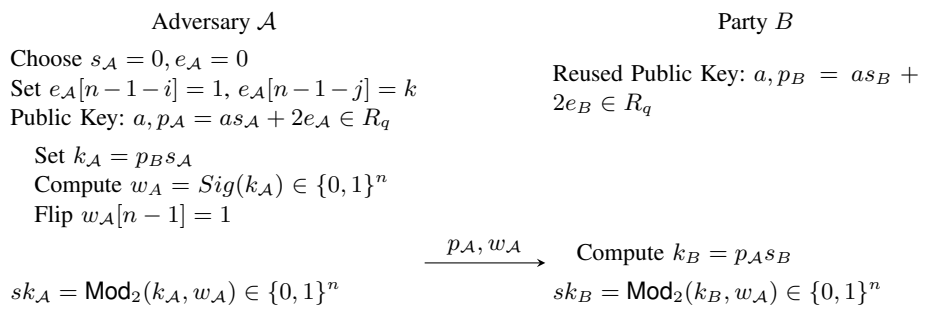


Fig. 3: One instance of the attack in the simplified case choosing Adversary's secret $s_{\mathcal{A}} = 0$, when error g_B is not added to the key computation k_B

coefficient of s_B multiple times. The error $g_B[n-1]$ fluctuates k_B but the k value when k_B changes from positive to negative or vice versa is centered around the actual value of $s_B[i]$ since $g_B[n-1]$ values are sampled from an error distribution(Discrete Gaussian) centered at 0. The oracle $\mathcal{B}$ can be modified to be contructed as follows:
Constructing Oracle $\mathcal{B}$: $\mathcal{B}$ takes p, w, sk as input where p, sk corresponds to the public key and the final shared key respectively of $\mathcal{A}$. The oracle computes $k_B = ps_B + 2g_B$, where $g_B \leftarrow \chi_\alpha$ and $sk_B = \mathsf{Mod}_2(k_B, w)$. $\mathcal{B}$ then outputs 1 if $sk_B = sk$ and 0 otherwise.

Thus, the steps for the attack in this case are the same as above except that step 3 is repeated a constant number of times and the distribution of k values reveal the exact value of $s_B[i]$ for every coefficient i. For step 2, the attacker queries by modifying $(n-1-i)$-th coordinate to be 2 so that $k_B[n-1] = 4s_B[i] \pm 2k + 2g_B[n-1]$ to override the effect of $g_B[n-1]$ on the sign.

In our experiments, we queried for the same $s_B[i]$ coefficient 1000 times and derived the value from the distribution of k values corresponding to a change in output of $\mathcal{B}$, obtained from each run. The number of runs 1000 is chosen to derive a reasonable number of samples for analyzing the distribution of k with a certain confidence level and is independent of the choice of parameters n, q, α for the protocol. For a confidence level of 95%, we estimated the number of samples to be ≈ 1000 with margin of error 3%. From the description of the attack, the distribution of k obtained for a coefficient value of 7 (on the left) and -3 (on the right) are shown in Figure 4.

The attacker can generate the distribution of k corresponding to different values of a coefficient by running an initial attack, choosing a p_B himself and then perform the actual attack on party B.

6.3 Improved attack

Finally, a simple randomness check at party B's end could protect B from this attack, as the public key is just all 0s and non-zero in 2 coordinates. To avoid this, we perform the attack when the attacker's public key is random looking and indistinguishable from uniform. We choose $s_{\mathcal{A}}$ to be such that $p_B s_{\mathcal{A}}[n-1] = 0$. This way we can obtain an $s_{\mathcal{A}}$ such that for the index $n-1$, the value of $as_B s_{\mathcal{A}}$ is small since $p_B s_{\mathcal{A}} = as_B s_{\mathcal{A}} + 2e_B s_{\mathcal{A}}$, where $e_B s_{\mathcal{A}}$ is small. We require such a $s_{\mathcal{A}}$ so that the $as_B s_{\mathcal{A}}[n-1]$ term in $k_B[n-1]$ cannot override $2s_B[i] + 2k$ and the attack strategy can still be used. Since p_B is known to the adversary, he can solve the polynomial equation to find $s_{\mathcal{A}}$ such that $p_B s_{\mathcal{A}}[n-1] = 0$. However, such an $s_{\mathcal{A}}$ is not necessarily small. If $s_{\mathcal{A}}$ is not small, the errors amplify in the final key computed by the adversary and party B and the two final keys need not necessarily match. Thus, the adversary can no longer guess the final key computation of party B. To handle this, we use LLL reduction on the solution space of the equation $p_B s_{\mathcal{A}}[n-1] = 0$ to derive a small $s_{\mathcal{A}}$ that satisfies the equation. We achieved this in our implementation using Magma.

$\mathcal{B}$'s computation of k_B yields $k_B[n-1] = as_B s_{\mathcal{A}}[n-1] + 2(s_B[i] + k + g_B[n-1])$. Then, the attacker can perform the following process to recover the secret $s_B[i]$:

Step 1 : To determine the sign of $s_B[i]$, the attacker queries with $e_{\mathcal{A}}$ such that $e_{\mathcal{A}}[n-1-i] = 4$ and $e_{\mathcal{A}}[n-1-j] = k = 0$, so that the key k_B can override the effect of $as_B s_{\mathcal{A}}[n-1]$ and $g_B[n-1]$

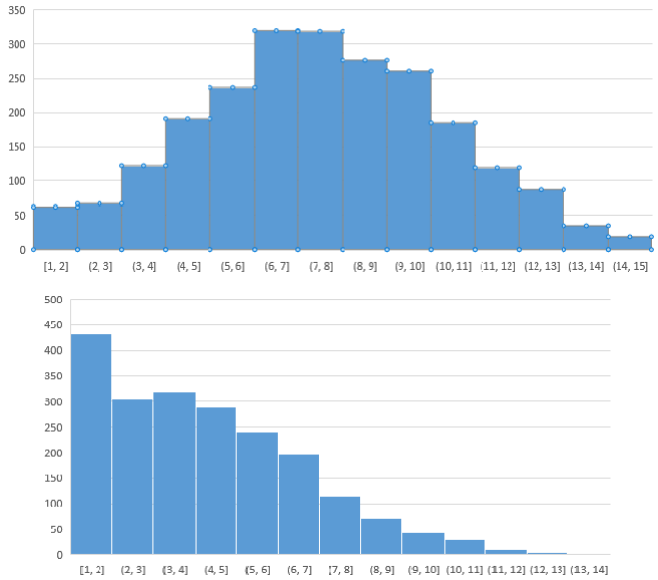


Fig. 4: Comparison of distribution of k while recovering coefficients 7 and -3 respectively

on the sign of $s_B[i]$. This is possible since we know that $as_Bs_{\mathcal{A}}[n-1]$ is small, by the choice of $s_{\mathcal{A}}$. Querying $\mathcal{B}$ a constant number of times, further counters the effect of $g_B[n-1]$ and reveals the sign of $s_B[i]$. If the output of $\mathcal{B}$ is 1, then $s_B[i]$ is negative and if $\mathcal{B}$ output is 0, then $s_B[i]$ is positive. Querying again with $e_{\mathcal{A}}[n-1-i] = -4$ resolves the 0 value coordinates of s_B.

Step 2 : Run the attack to obtain k value, denote k_1 that recovers the value of $as_Bs_{\mathcal{A}}[n-1] + s_B[i]$.

Step 3 : Repeat the attack by modifying $e_{\mathcal{A}}$ such that $e_{\mathcal{A}}[n-1-i] = 2$, which results in party B's computation of $k_B[n-1] = as_Bs_{\mathcal{A}}[n-1] + 2(2s_B^{\lceil}i] + k + g_B[n-1])$. Recover k (denote k_2) value corresponding to change in output of $\mathcal{B}$, hence recovering $as_Bs_{\mathcal{A}}[n-1] + 2s_B[i]$.

Step 4 : Compute $k_2 - k_1$ to recover the value $s_B[i]$.

There is one possibility in the above attack that $as_{\mathcal{A}}s_B[n-1] = -2s_B[i]$ in which case $k_B[n-1] = 2k + 2g_B[n-1]$. In this case, as we increase k, the mismatch of final keys does not reveal the value of $s_B[i]$. This case can be identified by querying with $k = -1$ and checking if the output of $\mathcal{B}$ is different from the output corresponding to $k = 0$. Recovering every coefficient of s_B by running the attack recovers the secret.

6.4 Determining index j such that $s_B[j] = 1$

If $s_B[j] = 0$, then modifying k doesnt affect index $n-1$ at all and thus can be easily identified. Also, this case is already identified while determining the sign of the coefficients.

We repeat for each coefficient j of s_B, the following procedure until a j such that $s_B[j]$ is identified, starting with the first positive coefficient, denote j_1. Since we can already determine the signs of every coefficient, it is enough to check through only the positive coefficients for value 1.

Step 1 : Start with $j = j_1$. Assuming $s_B[j_1] = 1$, perform the attack on other coefficients of s_B.

If $s_B[j_1] = 1$, then running the attack would yield the correct secret s_B. We can verify that this value of s_B recovered is actually the secret by verifying the distribution of $p_B - as_B$. This is possible since a, p_B are known and as_B can be computed using the recovered s_B. Now, suppose $s_B[j_1]>1$, the key k_B of party B recovers very small values since $k_B = as_{\mathcal{A}}s_B[n-1] + 2s_B[i] + 2ks_B[j_1] + 2g_B[n-1]$ changes from negative to positive faster when $s_B[j_1]$ is greater than 1 and $s_B[i]$ is negative. The same logic applies for $s_B[i]$ positive. Thus, all the coefficients recovered are very small and $p_B - as_B$ computed with this recovered s_B does not follow the error distribution.

Step 2 : Repeat Step 1 through all positive coefficients until a j such that $s_B[j] = 1$ is found.

If none of the positive coefficients are 1, then we can follow the same process with a different $e_{\mathcal{A}}$ (sign of k flipped) to check through the negative coefficients to find a j such that $s_B[j] = -1$.

Remark 3. There exists an index j such that $s[j] = \pm 1$ with high probability when $s \leftarrow \chi_\alpha$.

Since the error distribution χ_α used is the discrete gaussian distribution and we use the polynomial representation with two power cyclotomics, sampling an element $s \in R_q$ is equivalent to sampling each coordinate of its coefficient vector as a one dimensional discrete gaussian. The probability density function of the continuous one dimensional gaussian distribution with mean 0 and standard deviation α is given by $\phi_\alpha(x) = \frac{1}{\sqrt{2\pi}\alpha} e^{-x^2/2\alpha^2}$.

We use the discretized distribution $\bar{\phi}$ (defined in Section 3) of ϕ_α to do the analysis. For the parameter choice used in the experiments with $q = 12289, \alpha = 2.828, n = 1024$, we have the probability of a coefficient $s[i]$ of $s \leftarrow \chi_\alpha$ to be ± 1 given by

$$Pr(s[i] = 1) + Pr(s[i] = -1) = \int_{-1.5}^{-0.5} \frac{1}{\sqrt{2\pi}\alpha} e^{-\frac{x^2}{2\alpha^2}} dx + \int_{0.5}^{1.5} \frac{1}{\sqrt{2\pi}\alpha} e^{-\frac{x^2}{2\alpha^2}} dx$$

$$= 2 * \int_{-1.5}^{-0.5} \frac{1}{\sqrt{2\pi}\alpha} e^{-\frac{x^2}{2\alpha^2}} dx = 2 * \int_{-1.5}^{-0.5} \frac{1}{\sqrt{2\pi}(2.828)} e^{-\frac{x^2}{2(2.828)^2}} dx = 0.263836$$

So, the failure probability of the vector s sampled from the error distribution not having a coefficient ± 1 is given by $(1 - 0.263836)^{1024} \approxeq 0$. This can also be verified in general when n is large and α is small.

In the extreme case that there does not exist an index j for which $s[j] = 1$, the attack can still be performed by choosing 2 indexes j_1, j_2 such that $s_B[j_1] + s_B[j_2] = 1$. In this case, the public key of the attacker would be $p_{\mathcal{A}} = as_{\mathcal{A}} + e_{\mathcal{A}}$ where the vector $e_{\mathcal{A}}$ has k in the $n-1-j_1$ and $n-1-j_2$ coordinates and 1 in the $(n-1-i)$ coordinate. Thus, we have $p_A s_B[n-1] = as_{\mathcal{A}} s_B[n-1] + 2s_B[i] + 2k(s_B[j_1] + s_B[j_2]) = 2s_B[i] + 2k$.

6.5 Adversary query complexity

To compute the query complexity of the attack, we compute the query complexity of each phase of the attack: 1) Determining the sign of each coefficient, 2) Determining index j such that $s_B[j] = \pm 1$, 3) Determining a coefficient value $s_B[i]$ when the error term g_B is added to the key k_B of party B when the attacker's secret $s_{\mathcal{A}} = 0$, 4) Recovering (using query complexity of 1, 2 and 3) the secret s_B with $s_{\mathcal{A}}$ non-zero to make the public key of the attacker random looking.

1) The sign is determined by querying with $p_{\mathcal{A}}$ corresponding to $k = 0$ a small constant number of times (in our experiments, 10 queries were sufficient). Thus, the query complexity here is constant for each coefficient, so the query complexity to recover the signs of all the coefficients of s_B is $2c'n \approx 20n$, where c' is a constant.

2) s_B is sampled from the error distribution that has standard deviation α. So, to determine each coefficient, we need atmost $t\alpha$ queries, where t is a constant. Thus, to recover complete s_B, we need $nt\alpha$ queries. Since the error distribution we consider is the discrete gaussian and 99% of the values lie within 3 standard deviations of the mean, in our experiments with $\alpha = 2.828$, we run 16 queries for each coefficient, allowing for fluctuations when error g_B is added. Also, this is run atleast 1000 times to get the distribution of k, as described in Section 6.1 in attack extension. Thus, the attack complexity in this case would be $1000nt\alpha = Cn\alpha$, where C is the constant $= 1000t$.

3) Recovering the secret with $s_{\mathcal{A}}$ of attacker non-zero: This is the actual attack performed. In this case, the complete attack is run twice with different $e_{\mathcal{A}}$. So, the number of queries required is $2Cn\alpha$.

4) Determining index j such that $s_B[j] = 1$: This requires running the attack for every coefficient i assuming that $s_B[j] = 1$ starting with the first positive coefficient until such a j is found. So, the best case query complexity is $2Cn\alpha$, when the first positive coefficient turns out to be the required index with $s_B[j] = 1$ The same applies for searching -1. The worst case query complexity is $2Cn^2\alpha$.

Thus the query complexity of the complete attack would be $2c'n + 2Cn^2\alpha \approx O(n^2\alpha)$ in the worst case and $2c'n + 2Cn\alpha \approx O(n\alpha)$ in the best case.

6.6 Experiments

We have run experiments to verify the attack strategy. We use parameters $n = 1024, q = 12289, \alpha = 2.828$, used in [5] implementation. We used C++ with NTL and p_B value hardcoded to be fixed for the experiments on a Windows 10, 64 bit system equipped with a 2.40 GHz Intel(R) Core(TM) i7-4700MQ CPU and 8 GB RAM. The LLL reduction to find an appropriate short secret $s_{\mathcal{A}}$ of the attacker was executed using Magma. In our preliminary experiments, with the attacker's key of the form $p = as_{\mathcal{A}} + 2e_{\mathcal{A}}$, $s_{\mathcal{A}}$ non-zero chosen as described above, the time taken for running 1000 queries for one coefficient value to get the distribution of k is 35.1 mins with FFT for polynomial multiplication without any optimization. This time taken is to run 16000 queries to party B with queries varying k from 0 to 15 are run 1000 times to get the distribution of k for one coefficient.

7 Extending Signal Function Attack

Protocol Review: We note that the signal function attack can also be extended to the key exchange by Peikert [24] that was implemented in [7]. We review the key exchange protocol in [24] that uses the cross rounding function (Signal) for sending the additional information to compute the final shared key. Please refer to Section 3 for notations and definitions of the functions used in the protocol. The key exchange is as described below:

Party A : Set $p_A = as_A + e_A$, where $s_A, e_A \leftarrow \chi_\alpha$ and publish p_A.
Party B : On receiving p_A, choose $s_B, e_B, g_B \leftarrow \chi_\alpha$ and compute $p_B = as_B + e_B$. Then to obtain the shared key, compute $k_B = p_A s_B + g_B$. Let $\bar{k}_B = dbl(k_B)$, $w_B =< \bar{k}_B >_2$ and output p_B, w_B to party A. The final shared key is $sk_B = \lfloor \bar{k}_B \rceil_2$.
Party A : To finish the key exchange, compute $k_A = p_B s_A$ and the final shared key $sk_A = rec(2k_A, w_B)$.

The protocol and the signal functions are summarized in Figures 5 and 6.

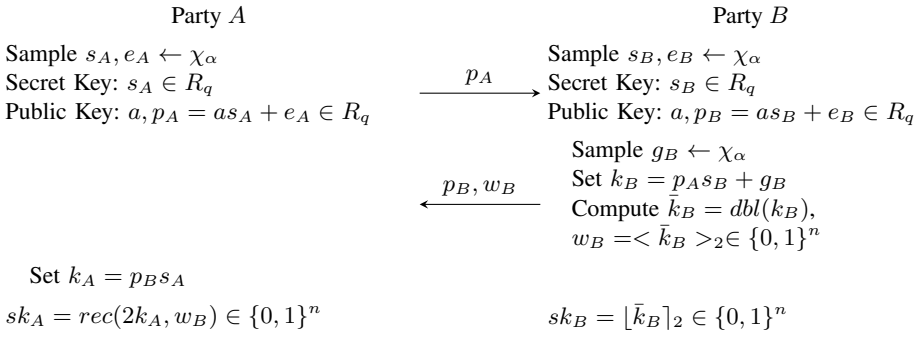


Fig. 5: KE Protocol from [24]

The Attack: The attack here is very similar to the attack using the signal function in [11]. In this case, the additional information required for agreeing on the final shared key sk_A, sk_B is achieved by party B sending the value of the cross rounding function $< . >_2$. By definition, $<v>_2$ returns 0 when $v \in I_0, I_0'$ and 1 when $v \in I_1, I_1'$, where the sets I_0, I_0', I_1 and I_1' are as defined in Section 3. Thus, we refer to the output w_B of the cross rounding function $<\bar{k}_B>_2$ as the signal. The variation here, compared to the signal function in [16] is that the signal regions are defined as quadrants as opposed to E, E^c in Definition 2 and the signal function is applied on $dbl(k_B)$. The dbl function is applied on k_B in the protocol to remove bias when q is odd, which is usually the case in RLWE instantiations.

The strategy behind the initial signal function attack is that when the attacker's key is chosen in such a way that party B's computation of $k_B = ks_B$ for k values ranging over all values in $\mathbb{Z}_q$, $k_B[i]$ value

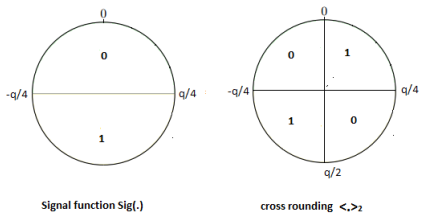


Fig. 6: Comparison of Signal in the two RLWE based key exchange protocols in [16] and [24].

varies in multiples of $s_B[i]$ and the number of signal changes is exactly $2s_B[i]$ for every coefficient i. This is because there are 2 boundary points (from the way the signal regions E, E^c are defined) where the signal bit flips.

In the key exchange described in Figure 5, the cross rounding function divides $\mathbb{Z}_q$ into quadrants resulting in 4 boundary points where the value of the signal flips. Thus, following the same approach as the signal function attack in [11], the number of signal changes while using the cross rounding function is exactly $4s_B[i]$, for every coefficient i. So, the secret can be compromised in $2q$ queries to the honest party reusing the key. Essentially we get the signal values of party B's secret with the error $2g_B - \bar{e}$ causing fluctuations in the signal changes with this protocol as well. This can be handled by not counting the fluctuations as signal changes. The fluctuations are easier to identify since the changes are within a smaller interval.

8 Signal Function Attack with Reduced Query Complexity

The signal function attack works by counting the number of times the signal bit $Sig(k_B)$ changes for each coefficient of k_B, for k across all values of $\mathbb{Z}_q$ in the public key $p_{\mathcal{A}} = as_{\mathcal{A}} + ke_{\mathcal{A}}$ of the adversary. The adversary specifically chooses $s_{\mathcal{A}}$ to be 0 and $e_{\mathcal{A}}$ to be 1 in R_q in the simplified form of the attack. $\mathcal{A}$ then queries with his public key as $(1+x)p_{\mathcal{A}}$ to eliminate the ambiguity of the $\pm$ sign of the coefficients recovered from previous queries and determine the exact values. The attack is also then extended to the case when $s_{\mathcal{A}}$ is sampled from the error distribution χ_α so that the adversary's public key $p_{\mathcal{A}}$ is an RLWE sample indistinguishable from uniform. This attack requires $2q$ queries to party B to extract the exact value of the secret s_B. For a detailed description of the attack, refer to [11].

We now show that the attack can be more efficient with fewer queries. This comes from the observation that it is not necessary to vary k through all the values of $\mathbb{Z}_q$ to determine the value of s_B accurately. For each coefficient of the secret $s_B[i]$, as k varies from 0 to $q-1$, the key value $k_B[i]$ changes in multiples of $s_B[i]$. Thus, depending on the value of $s_B[i]$, the period of signal change varies and this can be used to perform the attack more efficiently. We consider the different cases of the protocol here to see how fewer queries can still successfully recover the secret.

Case 1 : First, we consider the simplified case when the error term g_B is not added to the key computation k_B and the secret of the adversary $s_{\mathcal{A}}$ is 0, with public key $p_{\mathcal{A}} = k$. It is then clear that determining the first k value when the signal changes gives the value of $s_B[i]$ upto $\pm$ sign since the first flip of the signal bit happens when k changes from $\lfloor \frac{q}{4s_B[i]} \rceil$ to $\lfloor \frac{q}{4s_B[i]} \rceil + 1$ by the definition of the signal region E, E^c. Also, instead of querying for each coefficient separately, we can query for all coefficients at once varying k from 0 to $q/4+2$. This is because the smaller $s_B[i]$ values need more number of queries for counting the first signal change. For example, $s_B[i] = \pm 1$ needs $q/4+2$ queries, $s_B[i] = \pm 2$ needs $q/4+1$ queries and so on. Again using $q/4+2$ queries to party B with public key of adversary $p_{\mathcal{A}} = (1+x)k$, the ambiguity of $\pm$ sign is resolved. Thus, the adversary can recover s_B with $2(q/4+2) = q/2+4$ queries thus reducing the query complexity by a factor of $1/4$ compared to previous complexity of $2q$ described above.

Case 2 : This is the case of the original protocol where the adversary only slightly deviates from the protocol by choosing $e_{\mathcal{A}} = 1$, $s_{\mathcal{A}}$ is chosen according to the error distribution χ_α and $p_{\mathcal{A}} = as_{\mathcal{A}} + ke_{\mathcal{A}} = as_{\mathcal{A}} + k$ so that an attacker's public key cannot be distinguished from uniform. In this case, we

cannot use the first k where the signal flips to determine the value of s_B since $k_B = as_{\mathcal{A}}s_B + ks_B + 2g_B$; For every coefficient i, we have $as_{\mathcal{A}}s_B[i]$ as a constant value that is unknown to the adversary along with the noise g_B, added to $ks_B[i]$. In order to count the number of signal changes here, the attacker varies k starting with $k = 0$ and records the first signal change at $k = k_1$. Then he can vary k for negative values and record the first signal change in this direction at $k = k_2$. Now, $k_1 - k_2$ is the span of the region E or E^c in multiples of $s_B[i]$. Thus, $\lfloor \frac{q}{2(k_1-k_2)} \rceil$ reveals the value of $s_B[i]$ upto $\pm$ sign since the period of the signal change is $k_1 - k_2$. When the error g_B is added to the key computation k_B, the signal change does not happen in specific intervals due to the fluctuations. Here, we can query a small constant number of times more than $q/2$ until the signal stabilizes after a change. Thus, with $\frac{q}{2} + c$ queries where c is a small constant, we can recover $s_B[i]$ upto sign. c is small since the values stabilize when k increases and $ks_B[i]$ is away from the boundary points. So, to recover the exact value of the secret requires $q + c$ queries.

This is further illustrated with the help of an example in Appendix A.

9 Conclusion

In this work, we have presented a new attack on the RLWE key exchange showing that even an unsuccesful key exchange session, when the final computed keys of both parties do not match can be used to recover the secret of a fixed public key. We also extend a previous attack based on the signal function to the KE protocol described in [24]. This shows that reuse of keys should always be avoided while replacing a key exchange protocol based on RLWE as a potential post-quantum alternative. We also note that in the New Hope implementation, the public a is chosen at random for every new key exchange session. However, the active attacks on the KE protocols rely on the fact that the public key is reused in certain internet protocols. So, even if the New Hope implementation is integrated into such protocols, a new a might not be chosen for every key exchange session as suggested in the work and hence is vulnerable to such attacks.

References

[1] The transport layer security (tls) protocol version 1.3 (December 2015), https://tools.ietf.org/html/draft-ietf-tls-tls13-07

[2] Experimenting with post-quantum cryptography (July 2016), https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html

[3] The transport layer security (tls) protocol version 1.3 (May 2016), https://tools.ietf.org/html/draft-ietf-tls-tls13-13

[4] Ajtai, M.: Generating hard instances of lattice problems (extended abstract). In: Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing. pp. 99–108. STOC '96, ACM, New York, NY, USA (1996), http://doi.acm.org/10.1145/237814.237838

[5] Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange—a new hope. In: 25th USENIX Security Symposium (USENIX Security 16). pp. 327–343. USENIX Association, Austin, TX (2016), https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/alkim

[6] Applebaum, B., Cash, D., Peikert, C., Sahai, A.: Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In: Halevi, S. (ed.) Advances in Cryptology – CRYPTO 2009, Lecture Notes in Computer Science, vol. 5677, pp. 595–618. Springer Berlin Heidelberg (2009)

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

[8] Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. pp. 309–325. ACM (2012)

[9] Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-lwe and security for key dependent messages. In: Rogaway, P. (ed.) Advances in Cryptology – CRYPTO 2011, Lecture Notes in Computer Science, vol. 6841, pp. 505–524. Springer Berlin Heidelberg (2011)

[10] Diffie, W., Hellman, M.: New directions in cryptography. IEEE Trans. Inf. Theor. 22(6), 644–654 (Sep 2006), http://dx.doi.org/10.1109/TIT.1976.1055638
[11] Ding, J., Alsayigh, S., Saraswathy, R.V., Fluhrer, S., Lin, X.: Leakage of signal function with reused keys in rlwe key exchange. In: 2017 IEEE International Conference on Communications (ICC). pp. 1–6 (May 2017)
[12] Fluhrer, S.: Cryptanalysis of ring-lwe based key exchange with key share reuse. Cryptology ePrint Archive, Report 2016/085 (2016), http://eprint.iacr.org/2016/085
[13] Fujioka, A., Suzuki, K., Xagawa, K., Yoneyama, K.: Strongly Secure Authenticated Key Exchange from Factoring, Codes, and Lattices, pp. 467–484. Springer Berlin Heidelberg, Berlin, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-30057-8_28
[14] Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Proceedings of the 40th annual ACM symposium on Theory of computing. pp. 197–206. STOC '08, ACM, New York, NY, USA (2008)
[15] Gong, B., Zhao, Y.: Cryptanalysis of RLWE-Based One-Pass Authenticated Key Exchange, pp. 163–183. Springer International Publishing, Cham (2017), https://doi.org/10.1007/978-3-319-59879-6_10
[16] Jintai Ding, Xiang Xie, X.L.: 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/
[17] Kirkwood, D., Lackey, B.C., McVey, J., Motley, M., Solinas, J.A., Tuller, D.: Failure is not an option: Standardization issues for post-quantum key agreement (2016), http://csrc.nist.gov/groups/ST/post-quantum-2015/presentations/session7-motley-mark.pdf
[18] Lyubashevsky, V.: Lattice-Based Identification Schemes Secure Under Active Attacks, pp. 162–179. Springer Berlin Heidelberg, Berlin, Heidelberg (2008), http://dx.doi.org/10.1007/978-3-540-78440-1_10
[19] Lyubashevsky, V.: Fiat-Shamir with Aborts: Applications to Lattice and Factoring-Based Signatures, pp. 598–616. Springer Berlin Heidelberg, Berlin, Heidelberg (2009), http://dx.doi.org/10.1007/978-3-642-10366-7_35
[20] Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) Advances in Cryptology – EUROCRYPT 2010, LNCS, vol. 6110, pp. 1–23. Springer Berlin / Heidelberg (2010)
[21] Micciancio, D., Regev, O.: Worst-case to average-case reductions based on gaussian measures. SIAM J. Comput. 37, 267–302 (April 2007)
[22] NIST: Post quantum cryptography: Nist's plan for the future (2016), http://csrc.nist.gov/groups/ST/post-quantum-crypto/documents/pqcrypto-2016-presentation.pdf
[23] Peikert, C.: Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In: Proceedings of the 41st annual ACM symposium on Theory of computing. pp. 333–342. STOC '09, ACM, New York, NY, USA (2009)
[24] Peikert, C.: Lattice Cryptography for the Internet, pp. 197–219. Springer International Publishing, Cham (2014), http://dx.doi.org/10.1007/978-3-319-11659-4_12
[25] Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing. pp. 84–93. STOC '05, ACM, New York, NY, USA (2005)
[26] Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. 26(5), 1484–1509 (Oct 1997), http://dx.doi.org/10.1137/S0097539795293172
[27] Zhang, J., Zhang, Z., Ding, J., Snook, M., Dagdelen, Ö.: Authenticated key exchange from ideal lattices. In: Advances in Cryptology-EUROCRYPT 2015, pp. 719–751. Springer (2015)

A Toy Example

Here, we illustrate the attacks described above with the help of a toy example. First, we show the attack for the simplified case when g_B is not added to party B's key. The examples use the coefficient vector

representation of the elements of R_q.

Attack using final key mismatch detailed in Section 6:

Let $n = 8$, $q = 257$, $\alpha = 3.192$. Sampling a uniformly random from R_q and s_B, e_B from χ_α, let $a = (-15, 69, 33, -57, -3, 87, -105, 7)$, $s_B = (2, 3, 0, 0, -5, 2, 3, 1)$ and $e_B = (-1, -5, 0, 2, -3, 3, 0, 3)$. Then $p_B = as_B + 2e_B = (-122, -109, -103, -24, 106, -120, -28, -9)$. Here, $s_B[7] = 1$ and the attacker uses this information in creating his $e_\mathcal{A}$.

Step 1: Determining the signs of the coefficients: To determine the sign of each coefficient, query $\mathcal{B}$ with $p_\mathcal{A}$ correponding to $k = 0$

$e_\mathcal{A}$	$k_B[n-1]$	$\mathcal{B}$ output
(0,0,0,0,0,0,0,1)	$2s_B[0]$	0
(0,0,0,0,0,0,1,0)	$2s_B[1]$	0
(0,0,0,0,0,1,0,0)	$2s_B[2]$	1
(0,0,0,0,1,0,0,0)	$2s_B[3]$	1
(0,0,0,1,0,0,0,0)	$2s_B[4]$	1
(0,0,1,0,0,0,0,0)	$2s_B[5]$	0
(0,1,0,0,0,0,0,0)	$2s_B[6]$	0
(1,0,0,0,0,0,0,0)	$2s_B[7]$	0

Table 1: Queries with $k = 0$

Table 1 indicates that the coefficients for $i = 0, 1, 5, 6, 7$ are positive and the coefficients $i = 2, 3, 4$ are negative. Now, we can run the queries again with $e_\mathcal{A}$ that has -1 in the $n - 1 - i$ for coefficients i that are identified to be negative.

e_B	$k_B[n-1]$	$\mathcal{B}$ output
(0,0,0,0,0,-1,0,0)	$-2s_B[2]$	1
(0,0,0,0,-1,0,0,0)	$-2s_B[3]$	1
(0,0,0,-1,0,0,0,0)	$-2s_B[4]$	0

Table 2: Queries to confirm the sign of coefficients

From the output of $\mathcal{B}$ in Table 2, the attacker can conclude that $s_B[4]$ is negative and $s_B[2], s_B[3]$ are 0 since the output of $\mathcal{B}$ remains the same in both cases of $e_\mathcal{A}$.

Step 2:To determine $s_B[0]$, choose $e_\mathcal{A} = (-k, 0, 0, 0, 0, 0, 0, 1)$ and $s_\mathcal{A} = 0$. Then $p_\mathcal{A} = 2e_\mathcal{A}$ with coefficient vector $(-2k, 0, 0, 0, 0, 0, 0, 2)$, $k_\mathcal{A} = (0, 0, 0, 0, 0, 0, 0, 0)$ and final key $sk_\mathcal{A} = (0, 0, 0, 0, 0, 0, 0, 0)$. The signal computed by $\mathcal{A}$ results in $w_\mathcal{A} = (0, 0, 0, 0, 0, 0, 0, 0)$ in which the last bit $n-1 = 3$ is flipped to be 1, so modified $w_\mathcal{A} = (0, 0, 0, 0, 0, 0, 0, 1)$. The attacker now queries $\mathcal{B}$ with $p_\mathcal{A} = (-2k, 0, 0, 0, 0, 0, 0, 2)$, $w_\mathcal{A} = (0, 0, 0, 0, 0, 0, 0, 1)$, $sk_\mathcal{A} = (0, 0, 0, 0, 0, 0, 0, 0)$.

$\mathcal{B}$ computes $k_B = (-4k - 6, -6k, 0, 10, -4 + 10k, -4k - 6, -6k - 2, -2k + 4)$ and the output for each iteration of k is as follows:

k	k_B	sk_B	$\mathcal{B}$ output
0	(-6,0,0,10,-4,-6,-2,4)	$(0, 0, 0, 0, 0, 0, 0, 1)$	0
1	(-10,-6,0,10,6,-10,-8,2)	$(0, 0, 0, 0, 0, 0, 0, 1)$	0
2	(-14,-12,0,10,16,-14,-14,0)	$(0, 0, 0, 0, 0, 0, 0, 0)$	1

Table 3: Queries to determine the coefficient $s_B[0]$

Change in output of $\mathcal{B}$ at $k = 2$ reveals that $s_B[0] = 2$, since the sign of the coefficient value is already known from step 1.

Determining j: From step 1, we have identified that $i = 0, 1, 5, 6, 7$ are the positive coefficients, we run the attack on all the coefficients assuming $s_B[0] = 1$.

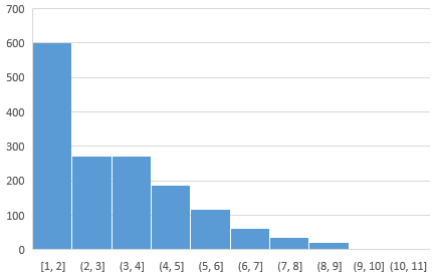
Run the attack on $s_B[1]$: Then the attacker chooses $e_\mathcal{A} = (0, 0, 0, 0, 0, 0, 1, -k)$ and w_A, sk_A as mentioned above. Querying with these values yield the following results:

k	k_B	sk_B	$\mathcal{B}$ output
0	(0,0,10,-4,-6,-2,4,6)	$(0,0,0,0,0,0,0,1)$	0
1	(6,0,10,-14,-2,4,6,2)	$(0,0,0,0,0,0,0,1)$	0
2	(12,0,10,-24,2,10,8,-2)	$(0,0,0,0,0,0,0,0)$	1

Table 4: Queries to determine $s_B[1]$ assuming $s_B[0] = 1$

This reveals that $s_B[1] = 2$. Repeating the attack on coefficients $s_B[i]$ for $i = 4, 5, 6, 7$ yields values $-2, 1, 2, 1$ respectively. Thus, the recovered secret is $s_B = (1, 2, 0, 0, -2, 1, 2, 1)$ assuming $s_B[0] = 1$. Now, compute and verify the distribution of $p_B - as_B = (-9, 108, -40, 19, 77, 118, -63, -76)$. In practical implementations, the dimension n is larger, usually 1024 that gives enough samples to verify the distribution. With this, we can conclude that the assumption $s_B[0] = 1$ is false. Continuing the same process recovering the secret assuming one of the coefficient value is 1 finally gives the index $j = 7$ for which $s_B[7] = 1$.

Case 2: When error g_B is added to k_B: In this case, we have the key computation of $k_B = 2e_{\mathcal{A}}s_B + 2g_B$. Thus, the attacker queries as in step 2 above multiple times and looks at the distribution of k to recover $s_B[0]$. The distribution obtained for this example is shown in Figure **??**.



Case 3: When $s_{\mathcal{A}}$ of the attacker is non-zero: Now, we solve the equation $p_B s_{\mathcal{A}}[7] = 0$ and apply LLL reduction on the solution space to find such a small $s_{\mathcal{A}}$. This is found to be $(0, -1, -1, 0, 0, 0, -1, 0)$ using Magma. Now, the public key of the attacker is $p_{\mathcal{A}} = as_{\mathcal{A}} + 2e_{\mathcal{A}}$, where $e_{\mathcal{A}}$ is as above and $s_{\mathcal{A}}$ is the computed value that satisfies the equation $p_B s_{\mathcal{A}}[7] = 0$.

So, $p_{\mathcal{A}} = (192, 222, 200, 242, 176, 67, 188, 206) + 2e_{\mathcal{A}}$. To recover $s_B[0]$, we choose $e_{\mathcal{A}} = (-k, 0, 0, 0, 0, 0, 0, 1)$ as above. So, $p_{\mathcal{A}} = (192 - 2k, 222, 200, 242, 176, 67, 188, 208)$.

Compute the signal $w_{\mathcal{A}} = Sig(k_{\mathcal{A}})$. we have $k_{\mathcal{A}} = p_B s_{\mathcal{A}} = (117, 89, 80, 92, 99, 166, 136, 0)$. So, $w_{\mathcal{A}} = (1, 1, 1, 1, 1, 1, 1, 0)$ and final key computation of attacker $sk_{\mathcal{A}} = (0, 0, 1, 1, 0, 1, 1, 0)$.

$\mathcal{B}$'s key computation will be $k_B = p_{\mathcal{A}}s_B = as_{\mathcal{A}}s_B + 2e_{\mathcal{A}}s_B = (111, 77, 74, 76, 103, 158, 134, 253) + 2e_{\mathcal{A}}s_B$. Notice that $as_{\mathcal{A}}s_B[7] = -4$ is small since we choose such an $s_{\mathcal{A}}$. Also, $p_{\mathcal{A}}s_B[7] = as_{\mathcal{A}}s_B[7] + 2s_B[7] - 2k$.

1. Determining the signs:

$e_{\mathcal{A}}$	$k_B[n-1]$	$\mathcal{B}$ output
(0,0,0,0,0,0,0,4)	$as_As_B[7] + 8s_B[0]$	0
(0,0,0,0,0,0,4,0)	$as_As_B[7] + 8s_B[1]$	0
(0,0,0,0,0,4,0,0)	$as_As_B[7] + 8s_B[2]$	1
(0,0,0,0,4,0,0,0)	$as_As_B[7] + 8s_B[3]$	1
(0,0,0,4,0,0,0,0)	$as_As_B[7] + 8s_B[4]$	1
(0,0,4,0,0,0,0,0)	$as_As_B[7] + 8s_B[5]$	0
(0,4,0,0,0,0,0,0)	$as_As_B[7] + 8s_B[6]$	0
(4,0,0,0,0,0,0,0)	$as_As_B[7] + 8s_B[7]$	0

Table 5: Queries to determine the signs of s_B coefficients in the improved attack when $s_A \neq 0$

The coefficients with 0 value can again be identified the same way as in the previous case.
2. Run the attack now for $s_B[0]$ with input $p_{\mathcal{A}} = ((192 - 2k, 222, 200, 242, 176, 67, 188, 208)), w_{\mathcal{A}} = (1, 1, 1, 1, 1, 1, 1, 1)$ (with the last bit flipped to 1), $sk_{\mathcal{A}} = (0, 0, 1, 1, 0, 1, 1, 0)$ to $\mathcal{B}$ for different values of k. Now, $\mathcal{B}$ computes k_B

k	k_B	sk_B	$\mathcal{B}$ output
-1	(109,83,74,86,89,156,138,2)	$(0, 0, 1, 1, 0, 1, 1, 1)$	0
0	(105,77,74,86,99,152,132,0)	$(0, 0, 1, 1, 0, 1, 1, 0)$	1

Table 6: Queries to determine $s_B[0]$ in the improved attack

Now, run the same attack as above with $e_{\mathcal{A}} = (-k, 0, 0, 0, 0, 0, 0, 2)$. Then $\mathcal{B}$ computes k_B.

k	k_B	sk_B	$\mathcal{B}$ output
0	(99 77 74 96 95 146 130 4)	$(0, 0, 1, 1, 0, 1, 1, 1)$	0
1	(95 71 74 96 105 142 124 2)	$(0, 0, 1, 1, 0, 1, 1, 1)$	0
2	(91 65 74 96 115 138 118 0)	$(0, 0, 1, 1, 0, 1, 1, 0)$	1

Table 7: Queries to determine $s_B[0]$ in the improved attack with $e_{\mathcal{A}}[n-1] = 2$

Here, running the attack with $n - 1 - 0$ coordinate of $e_{\mathcal{A}}$ as 1 yields $k_1 = 0$ to be the value when there is a change in output of $\mathcal{B}$. Similarly, we have $k_2 = 2$ to be the value where the output changes when the $n - 1 - 0$ coordinate is 2. So, the recovered value of $s_B[0]$ here would be $k_2 - k_1 = 2 - 0 = 2$. When the error g_B is added to k_B, these values of k_1, k_2 would be derived from the distributions of k values obtained by running the above two iterations a constant number of times as in case 2 (1000 in our experiments).
For illustrating the other two attacks with the help of an example, we use a smaller q since this involves querying q times. The value of q is chosen to be small only for illustration purposes. The attack still works for q values used in real applications.

Signal function attack with lesser queries: Let $n = 4$, $q = 17$, $\alpha = 1.6$ (Choosing such a value of α for the sake of the example to obtain reasonable sample values). Sampling a uniformly random from R_q and s_B, e_B from χ_α, let $a = (9, 4, 9, 3)$, $s_B = (-1, 0, 0, 2)$ and $e_B = (1, -1, -1, 0)$. Then $p_B = as_B + 2e_B = (2, -7, 0, -2)$.
The signal region $E := \{-\lfloor \frac{q}{4} \rfloor, \ldots, \lfloor \frac{q}{4} \rceil\} = \{-4, -3, -2, -1, 0, 1, 2, 3, 4\}$
Choose $e_{\mathcal{A}}$ of the adversary to be 1. So $p_{\mathcal{A}} = k$.
Oracle $\mathcal{B}$: On input of $p_{\mathcal{A}} = k$ from the adversary, $\mathcal{B}$ computes $k_B = p_{\mathcal{A}} s_B = k s_B = (-k, 0, 0, 2k)$ and $w_B = Sig(k s_B)$, outputs w_B.

k	$w_B[3]$	$k_B[3]$
0	0	0
1	0	2
2	0	4
3	1	6
4	1	8

Table 8: Lesser queries in the simplified case

Since the change in signal happens when k changes from $\lfloor \frac{q}{4 s_B[i]} \rceil$ to $\lfloor \frac{q}{4 s_B[i]} \rceil + 1$ and here the signal changes from $k = 2$ to $k = 3$, we have $2 = \lfloor \frac{q}{4 s_B[3]} \rceil$ which gives $s_B[3] = 2$.
Case 2: When $s_A \neq 0$: sample s_A according to the error distribution χ_α gives $s_A = (1, 0, 1, 1)$. Then, the attacker's public key is $p_A = as_A + ke_A = (-4 + k, -8, -2, -1)$.
Oracle $\mathcal{B}$: On input of $p_{\mathcal{A}} =$ from the adversary, $\mathcal{B}$ computes $k_B = p_{\mathcal{A}} s_B = (3 - k, -5, 4, -7 + 2k)$ and $w_B = Sig(k_B)$, outputs w_B.

k	$w_B[3]$	$k_B[3]$
-4	0	2
-3	0	4
-2	1	6
-1	1	8
0	1	-7
1	1	-5
2	0	-3
3	0	-1

Table 9: Lesser queries in the improved case

Here $k = 2$ is the value when the signal changes varying k over positive values and $k = -3$ is the value when the signal changes while varying k in the opposite direction. So, $\lfloor \frac{q}{2(k_1-k_2)} \rceil$ reveals $s_B[3] = \lfloor \frac{17}{2*5} \rceil = 2$
With the same example as above, when the error g_B (sampled values 2,0,2,-1,-1,1,1,1,0 for $g_B[3]$ for the iterations below) is added to k_B,

k	$w_B[3]$	$k_B[3]$
-5	0	4
-4	0	2
-3	1	8
-2	0	4
-1	1	6
0	1	-5
1	0	-3
2	0	-1
3	0	-1

Table 10: Queries with the error g_B

Here, we can see that the $g_B[3]$ term in $k_B[3]$ causes fluctuations between $k = -1$ and $k = -4$. So, we run more queries than in the previous case and see that the signal changes happen at $k = -4$ and $k = 1$. So the recovered value is $\lfloor \frac{q}{2(1-(-4))} \rceil$ reveals $s_B[3] = \lfloor \frac{17}{2*5} \rceil = 2$.

Signal function attack on the public key p_B if the KE protocol used is from [24]:
Now, for the same example values of a, p_B, we demonstrate the attack on the KE protocol in [24] to recover $s_B[0]$.
Choose $s_{\mathcal{A}}, e_{\mathcal{A}}$ of the adversary to be $0, 1$ respectively. So $p_{\mathcal{A}} = k$.
Oracle $\mathcal{S}$: On input of $p_{\mathcal{A}} = k$ from the adversary, $\mathcal{S}$ computes $k_B = p_{\mathcal{A}} s_B + 2g_B = k s_B + 2g_B$ and $\bar{k}_B = dbl(k_B)$ and $w_B = Sig(\bar{k}_B)$, outputs w_B.

The signal region I_0, I_0', I_1, I_1' are as follows: $I_0 = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$,
$I_0' = \{17, -16, -15, -14, -13, -12, -11, -10, -9\}$,
$I_1 = \{-8, -7, -6, -5, -4, -3, -2, -1\}$ and $I_1' = \{9, 10, 11, 12, 13, 14, 15, 16\}$.

k	$k_B[0]$	$k_B[0]$	$w_B[0]$
0	0	0	0
1	-1	-2	1
2	-2	-5	1
3	-3	-5	1
4	-4	-8	1
5	-5	-9	0
6	-6	-12	0
7	-7	-15	0
8	-8	17	0
9	8	16	1
10	7	14	1
11	6	13	1
12	5	10	1
13	4	7	0
14	3	6	0
15	2	5	0
16	1	2	0

Table 11: Queries with Signal response

Now, if the error g_B is added to the key computation k_B, with g_B values sampled according to the error distribution χ_α (suppose $g_B[0]$ values are $0, 1, 1, 0, -1, 1, 0, 0, -1, -1, -2, 1, 0, 0, 2, 0, 0$), we have the following results:

k	$k_B[0]$	$k_B[0]$	$w_B[0]$
0	0	0	0
1	0	0	0
2	-1	-3	1
3	-3	-5	1
4	-5	-10	0
5	-4	-7	1
6	-6	-12	0
7	-7	-15	0
8	8	15	1
9	7	14	1
10	5	10	1
11	7	15	1
12	5	10	1
13	4	7	0
14	5	10	1
15	2	5	0
16	1	2	0

Table 12: Queries with fluctuations in signal

The fluctuations in signal change caused in the above example between $k = 4$ to $k = 5$ and between $k = 13$ to $k = 14$ are due to the error g_B and can be interpreted the same way as in [11] to count the number of signals to be 4, thus revealing $s_B[0] = \pm 1$.