Signal Leakage Attack Meets Depth First Search: an Improved Approach on DXL Key Exchange Protocol

Zhiwei Li\textsuperscript{1,2}, Jun Xu\textsuperscript{1,2}, and Lei Hu\textsuperscript{1,2}

\textsuperscript{1} Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
\textsuperscript{2} School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
{\texttt{lizhiwei,xujun,hulei}@iie.ac.cn}

Abstract. In 2012, Ding, Xie and Lin designed a key exchange protocol based on Ring-LWE problem, called the DXL key exchange protocol, which can be seen as an extended version of the Diffie-Hellman key exchange. In this protocol, Ding et al. achieved key exchange between the communicating parties according to the associativity of matrix multiplications, that is, \((x^T \cdot A) \cdot y = x^T \cdot (A \cdot y)\), where \(x, y\) are column vectors and \(A\) is a square matrix. However, the DXL key exchange protocol cannot resist key reuse attacks. At ESORICS 2022, Qin et al. proposed a method that an adversary can recover the reused private key after forging the public keys for several times. Nevertheless, Qin et al.’s method leads to a lot of redundant operations. In this paper, we improve Qin et al.’s method to a more general case and propose an effective approach to combine signal leakage attacks with depth first search. Compared with state-of-the-art result appeared at ESORICS 2022, the number of reused private key have been decreased from 29 to 10. In other words, if the number of reuses exceeds 10, the private key will be restored. Moreover, we validate the effectiveness of the results through experiments.

Keywords: DXL Key Exchange Protocol · Key Reuse Attack · Signal Leakage Attack · Depth First Search

1 Introduction

1.1 Background

The development of quantum computers is progressing rapidly in recent years, it has become very significant to find public key cryptographic schemes that can resist quantum computer attacks \[23\]. These public key cryptographic schemes can be divided into the following types: lattice-based schemes \[6,8,18,7\], codes-based schemes \[2,4,16\], multivariate-based schemes \[13\], supersingular isogeny schemes \[21\], and hash-based schemes \[5\]. Among these different types of cryptographic schemes, lattice-based schemes are considered to be the most promising schemes. In 2022, NIST \[25\] selected four standardized cryptographic schemes, three of
which are lattice-based schemes, namely, Kyber [6], Dilithium [7], Falcon [18]. Therefore, the study of lattice-based cryptography is very important.

In lattice-based cryptography, there are two important classes of difficulty problems: Small Integer Solution (SIS) problem and Learning With Error (LWE) problem. The LWE problem is briefly defined as follows. Given a uniform random matrix \( \mathbf{A} \in \mathbb{Z}_q^{n \times m} \) and a vector \( \mathbf{b} = \mathbf{A}^T \cdot \mathbf{s} + \mathbf{e} \in \mathbb{Z}_q^m \), where \( \mathbf{s} \in \mathbb{Z}_q^n \) and \( \mathbf{e} \in \mathbb{Z}_q^m \) are unknowns. The adversary cannot recover \( \mathbf{s} \) from \((\mathbf{A}, \mathbf{b})\). The LWE problem and its variants [23,24] are widely used in the design of various cryptographic schemes. Among them, Ding et al. proposed a Ring-LWE-based key exchange protocol for the first time in 2012 [15], called the DXL key exchange (DXL-KE) protocol, which can be regarded as an extended version of Diffie-Hellman key exchange (DHKE) protocol [10]. The DHKE protocol process is described as follows. The communication parties Alice and Bob need to share some public parameters beforehand. These parameters include a prime number \( q \) and a primitive root \( g \). Then, Alice randomly selects \( a \), calculates public key \( \mathbf{A} = g^a \mod q \) and sends the public key \( \mathbf{A} \) to Bob. On the other hand, Bob randomly chooses \( b \), computes public key \( \mathbf{B} = g^b \mod q \) and sends the public key \( \mathbf{B} \) to Alice. Finally, Alice calculates \( \mathbf{sk}_A = \mathbf{B}^a \mod q \) and Bob computes \( \mathbf{sk}_B = \mathbf{A}^b \mod q \). In this way, Alice and Bob can get the shared key \( \mathbf{sk} = g^{ab} \mod q \).

For the case of the LWE problem, Alice calculates public key \( \mathbf{p}_A = \mathbf{A} \cdot \mathbf{s}_A + \mathbf{e}_A \mod q \) and sends \( \mathbf{p}_A \) to Bob; Bob computes public key \( \mathbf{p}_B = \mathbf{A} \cdot \mathbf{s}_B + \mathbf{e}_B \mod q \) and sends \( \mathbf{p}_B \) to Alice. Due to the existence of the error terms \( \mathbf{e}_A, \mathbf{e}_B \), therefore, \( \mathbf{p}_B \cdot \mathbf{s}_A \approx \mathbf{p}_A \cdot \mathbf{s}_B \). In order to negotiate the same shared key, Ding et al. presented an error elimination mechanism in [15]. Roughly speaking, Bob sends an additional signal \( \omega_B \) after sending the public key \( \mathbf{p}_B \). In this way, Alice and Bob can negotiate a shared key.

Key reuse attacks are a common security threat, which refers to the scenario where users use the same private key for a long time. NIST [11] pointed out that cryptographic schemes should be designed to resist key reuse attacks. There are two important types of key reuse attacks: key mismatch attacks and signal leakage attacks. Key mismatch attack on lattice-based schemes is first proposed by Ding, Fluhrer, Saraswathy [29]. This attack has been used to analyze Kyber [29,30], Saber [30] and NewHope [28]. The signal leakage attack is first proposed by Fluhrer [17]. This attack has been used to attack DXL-KE [15] and DBS-KE [12] key exchange protocols [17,11,9,31]. Moreover, there are other types of key reuse attacks appeared in [14,20,36,26,35]. In order to resist existing key reuse attacks on the DXL-KE protocol, several schemes have been proposed [19,34,27].

In [11], Ding et al. first used the signal leakage attack to analyze the DXL-KE protocol. The key of their attack is as follows. Alice and Bob are the communication parties, where Alice is an adversary who wants to recover Bob’s reused private key \( \mathbf{s}_B \). The procedure of recovering the private key can be divided into two parts. The first part is to determine the value of each \( |\mathbf{s}_B[i]| (0 \leq i \leq n - 1) \) (called absolute values recovery), and the second part is to determine the sign of each \( \mathbf{s}_B[i] \) (called relative signs recovery). For the first part, Alice forges public
keys $p_A = k$ for $k = 0, 1, \cdots, q - 1$ and sends them to Bob. The involved modulus $q$ is public. After receiving the public keys $p_A$ from Alice, Bob will honestly execute the DXL-KE protocol and send public key and signal $(p_B, \omega_B)$ to Alice. Therefore, Alice can get $q$ times public keys and signals. As $k$ increases from 0 to $q - 1$, each $\omega_B^{(k)}[i]$ of the signals will change as $0 \rightarrow \cdots \rightarrow 1 \rightarrow \cdots \rightarrow 0 \rightarrow \cdots$. By the number of times the signal $\omega_B[i]$ flips, Alice can determine the value of each $|s_B[i]| (0 \leq i \leq n - 1)$. Subsequently, in order to further determine the sign of each $s_B[i]$, Alice needs to continue forging public keys $p_A = (1 + x) \cdot k$ and sends them to Bob, where the value of $k$ increases from 0 to $q - 1$. Then, Alice recovers the pairs $|s_B[0] + s_B[1]|, |s_B[1] + s_B[2]|, \cdots, |s_B[n - 2] + s_B[n - 1]|, |s_B[n - 1] - s[0]|$. Finally, based on the values of $|s_B[i]|$ and these pairs, the value of the corresponding $s_B[i]$ is determined. In Ding et al.’s work, the number of forged public keys is $q + q = 32770$. Note that the times of forged public keys are equal to the times of key reuses.

In 2021, Bindel et al. \cite{9} improved Ding et al.’s attack and proposed a sparse signal collection method. This method can also be divided into two parts, namely, absolute values recovery part and relative signs recovery part. Based on this method, for the first part, Alice needs forge public keys $p_A = k \cdot 36\alpha$ times to recover $|s_B[i]|$, where $k$ is select from some special values rather than all values from $\{0, 1, \cdots, q - 1\}$, where $\alpha = 3.197$ is the standard deviation in DXL-KE protocol. For the second part, Alice forges $p_A = (1 + x) \cdot k'$ to recover the pairs $|s_B[0] + s_B[1]|, |s_B[1] + s_B[2]|, \cdots, |s_B[n - 2] + s_B[n - 1]|, |s_B[n - 1] - s[0]|$, however, since one or more consecutive 0s in private key $s_B$, which will prevent determining the relative signs, in order to eliminate this case, Bindel et al. recommended Alice should reset $p_A = (1 + x^t) \cdot k', 1 \leq t \leq z$ to determine relative signs, where $k'$ is select from several special values rather than all values from $\{0, 1, \cdots, q - 1\}$, where $z$ (generally, $1 \leq z \leq 4$) is the maximum number of consecutive zeros in the private key. For the second part, Alice should forge $72 \cdot 3\alpha$ times public keys. Therefore, the number of forged public keys is approximately reduced to $36\alpha + 72 \cdot z\alpha = 36(1 + 2z)\alpha$. In this case, $36(1 + 2z)\alpha \approx 824$. It means that the number of forged public keys in the work of Bindel et al. is 824 on average.

At ESORICS 2022, Qin et al. \cite{31} further improved the results in Bindel et al.’s work. The signal leakage attack was converted into a coding problem. In the process of the absolute values recovery, Alice needs to forge her public keys $\lceil \log 16 \rceil (= 4)$ times to recover $|s_B[i]|$ for $i = 0, 1, \cdots, n - 1$. In the process of relative signs recovery, in order to solve the problem that one or more consecutive 0s in private key $s_B$, Alice needs to forge $(1 + z) \cdot \lceil \log 31 \rceil$ public keys to recover the relative signs. It is worth noting that in the work of Qin et al., the values of $k_1, k_2$ selected are different from those of Bindel et al., therefore, the number of forged public keys is reduced to $\lceil \log 16 \rceil + (1 + z) \cdot \lceil \log 31 \rceil = 29$.

From an attacker’s perspective, we expect the attack to still work even with the lowest possible number of reusable private keys, namely, the attacker can recover the reused private key by forging fewer public keys.
1.2 Our Contribution

In this paper, we propose an efficient attack method to recover the reused private key $s_B$ of Bob. In this method, Alice recovers multiple coefficients $\sum_{j=0}^{\lambda-1} s_B[i+j]$, rather than two coefficients $|s_B[i]+s_B[i+1]|$ by forging public keys $\lceil\log(15 \cdot \lambda+1)\rceil$ times, where the parameter $\lambda$ is an integer satisfying $3 \leq \lambda \leq 8$. This is a more general approach to previous work [11,9,31], and will reduce the times of forged public keys in relative signs recovery. To be specific, Alice forges $\lceil\log(15 \cdot \lambda+1)\rceil$ times public keys rather than $(1+z)\cdot\lceil\log 31\rceil$. Therefore, for different $\lambda$, the total number of forged public keys is 10 or 11 instead of 29. On the other hand, we have reduced the running time required to complete key recovery (based on the same platform).

We verify the effectiveness of results through experiments, and give the number of forged public keys and the time needed to complete the recovery of $s_B$. Compared with the previous best results, the number of forged public keys in our work is reduced by 60%. Alice only needs to forge the public keys 10 or 11 times to complete the key recovery, and the time required to complete the key recovery is reduced by about 50%. In this paper, we only need 0.24s to recover the whole private key $s_B$.

1.3 Technical overview

The new algorithm can also be divided into two parts: absolute values recovery and relative signs recovery. For absolute values recovery, the process is the same as that of Qin et al., that is, Alice needs to forge public keys $p_A$ 4 times to recover the absolute values of all coefficients of the private key $s_B$, i.e. the values of $|s_B[i]|$, $i = 0, 1, \cdots, n-1$.

The relative signs recovery is divided into Step A and Step B. In Step A, Alice forges public keys $p_A$ to recover the values of $\sum_{j=0}^{\lambda-1} s_B[i+j]$, $i = 0, 1, \cdots, n-\lambda$.

In Step B, according to the values of $|s_B[i]|$ and $\sum_{j=0}^{\lambda-1} s_B[i+j]$ and the depth first search, Alice recovers the signs of $s_B[i]$, $i = 0, 1, \cdots, n-1$. For ease of understanding, we describe Algorithm 1 at a high level.

The detailed description of Algorithm 1 will be introduced in Section 5.

1.4 Organization

The rest of this paper is organized as follows. We give preliminaries in Section 2, Section 3 presents the DXL-KE protocol, and Section 4 introduces the state-of-the-art attack on DXL-KE protocol. In Section 5, an efficient approach was proposed to reduce the number of forged public keys, Section 6 is experimental results and Section 7 is conclusion.
Algorithm 1 The high level description of the algorithm. Here, $\sum_i = \sum_{j=0}^{\lambda-1} s_B[i+j]$ and $SET = (\sum_0, \sum_1, \cdots, \sum_{n-\lambda})$. The predicate $\tau(s_B^{(t)})$ is used to simplify the check stage, where $\tau(s_B^{(t)})$ returns 1 if $s_B^{(t)}$ is Bob’s reused private key $s_B$, and 0 otherwise.

Input: The forged public keys $p_A$.
Output: The reused private key $s_B$.

1: Recover $|s_B[i]|$ (i.e. absolute values recovery)
2: Recover $SET$ (corresponding to Step A)
3: $t = 0$, $s_B^{(t)} = DFS(|s_B|, SET)$
4: while $\tau(s_B^{(t)}) \neq 1$, do
5: $t \leftarrow t + 1$, $s_B^{(t)} \leftarrow DFS(|s_B|, SET)$ (3-5 corresponds to Step B)
6: end
7: $s_B = s_B^{(t)}$
8: return $s_B$

2 Preliminaries

2.1 Notation

In this paper, we define $\mathbb{Z}_q$ to represent the integer ring of module $q$, $\mathbb{Z}_q[x]$ to represent the polynomial ring of module $q$ and the coefficient range of any polynomial in $\mathbb{Z}_q[x]$ is in the set $\{\frac{-q-1}{2}, \cdots, \frac{q-1}{2}\}$. Polynomials that appear in this paper are elements of the polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$, where polynomials are represented by lowercase letters. For any polynomial $f(x) = f_0 + f_1 x + \cdots + f_{n-1} x^{n-1}$ in $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$, the polynomial can be expressed as the corresponding vector form $f(x) = (f_0, f_1, \ldots, f_{n-1})$. We use bold capital letters to represent matrices such as $A \in \mathbb{R}_q^{n \times n}$. For a fixed matrix $A$, the corresponding transpose is $A^T$. The symbol $\chi_{\alpha}$ is defined as a discrete Gaussian distribution, where the standard deviation is $\alpha$. For a polynomial $f(x)$, if $f(x)$ is sampled from the discrete Gaussian distribution, then each coefficient $f_i(0 \leq i \leq n-1)$ of the polynomial $f(x)$ is selected based on discrete Gaussian distribution. The symbol $\lceil a \rceil$ indicates that the least integer greater than or equal to $a$. The base-2 logarithm is denoted by $\log(\cdot)$.

2.2 Encryption based on the LWE problem

LWE problem is an important kind of difficult problem. We take the Regev’s encryption \cite{32} as an example to show how to encrypt a message using LWE problem: Alice selects $s_A \in \mathbb{Z}_q^n, e_A \in \mathbb{Z}_q^n, A \in \mathbb{Z}_q^{n \times m}$, where $s_A$ is the private key, then Alice computes public key $p_A = A^T \cdot s_A + e_A \mod q$ and sends $p_A, A$ to Bob, Bob selects private key $s_B \in \mathbb{Z}_q^m$, and calculates $c_1 = A \cdot s_B \mod q$ and encrypts a message $c_2 = p_A^T \cdot s_B + m[q/2] \mod q$, then Bob sends ciphertext $(c_1, c_2)$.
to Alice. Finally, Alice decrypts the ciphertext $(c_1, c_2)$ and gets the decrypted message $m' = c_2 - s_A^T c_1 \mod q$.

In the above encryption, Alice needs to send a matrix $A$ to Bob, however, this matrix is very large and will affect the efficiency of the cryptography schemes. Therefore, as a structured LWE problem, the Ring-LWE problem has received widespread attention. In Ring-LWE problem, Alice selects $A, s \in \mathcal{R}_q, e \in \chi_\alpha$, where $A, s, e$ are polynomials and $\chi_\alpha$ is the error/noise distribution. Specifically, $\chi_\alpha$ denotes the discrete Gaussian distribution on $\mathcal{R}_q$ with standard deviation $\alpha$. In Ring-LWE problem, the polynomials $s, e$ are usually sampled from the same error/noise distribution $\chi_\alpha$ [3]. As an important error/noise distribution, the discrete Gaussian distribution is defined as follows:

**Definition 1.** 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 centered at $c$ is defined by the probability function $\rho_{\alpha,c}(x) = \left(\frac{1}{\sqrt{2\pi}\alpha^2}\right)^n \exp\left(-\frac{\|x-c\|^2}{2\alpha^2}\right)$. 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 $\mathcal{D}_{\mathbb{Z}^n, \alpha,c}(x) = \frac{\rho_{\alpha,c}(x)}{\rho_{\alpha,c}(\mathbb{Z}^n)}$, where $x \in \mathbb{Z}^n$. The subscripts $s$ and $c$ are taken to be 1 and 0 (respectively) when omitted.

2.3 Depth First Search

In this section, we recall the depth-first search (DFS) algorithm. Please refer to [22] for more details.

DFS is an algorithm for searching a graph or tree data structure. The algorithm starts at the root (top) node of a tree and goes as far as it can down a given branch (path), then backtracks until it finds an unexplored path, and then explores it. The algorithm does this until the entire graph has been explored. This algorithm uses a stack in order to keep track of visited nodes, as the last node seen is the next one to be visited and the rest are stored to be visited later.

**Algorithm 2** The pseudocode of Depth First Search [22], where $S$ is an empty stack for storing nodes and for each node $u$, define $u$.visited to be false.

1: Push the root (first node to be visited) onto $S$.
2: While $S$ is not empty, do
3: \hspace{1em} $u = S$.pop()
4: \hspace{1em} If $u$.visited = false, then:
5: \hspace{2em} $u$.visited = true
6: \hspace{2em} for each unvisited neighbor $w$ of $u$:
7: \hspace{3em} Push $w$ into $S$.
8: end
9: End process when all nodes have been visited.
For ease of understanding, we present an example in Fig. 1 for the DFS process.

In Section 5, we will propose a new approach by combining with the signal leakage attack and depth first search, which makes the number of forged public keys fewer.

3 The DXL-KE protocol

In this section, we briefly introduce the DXL-KE protocol in [15], which can be divided into three parts: the initialization part, the response part and the finish part.

Initialization part: Alice chooses polynomials $a \in \mathbb{R}_q$, $s_A, e_A \in \chi_\alpha$ and computes $p_A = a \cdot s_A + 2 \cdot e_A \in \mathbb{R}_q$ and sends $(a, p_A)$ to Bob.

Response part: After receiving $p_A$ sent by Alice, Bob selects $s_B, e_B, g_B \in \mathbb{R}_q$. Then, Bob needs to compute $p_B = a \cdot s_B + 2 \cdot e_B \in \mathbb{R}_q$, $k_B = p_A \cdot s_B + 2 \cdot g_A \in \mathbb{R}_q$, $\omega_B = \text{Sig}(k_B)$ and sends $(p_B, \omega_B)$ to Alice. Finally, Bob obtains the shared key $sk_B = \text{Mod}_2(k_B, \omega_B)$.

Finish part: After receiving $(p_B, \omega_B)$ sent by Bob, Alice chooses $g_A \in \chi_\alpha$ and calculates $k_A = p_B \cdot s_A + 2 \cdot g_A \in \mathbb{R}_q$. Finally, Alice gets the shared key $sk_A = \text{Mod}_2(k_A, \omega_B)$.

We depict the details of DXL-KE in Algorithm 3.

In DXL-KE protocol, the parameters are $n = 1024, q = 2^{14} + 1, \alpha = 3.197$ and $E = \{-\left\lfloor \frac{q}{4} \right\rfloor + r, \cdots, \left\lceil \frac{q}{4} \right\rceil + r\}$, where $r$ selects uniformly from $\{0, 1\}$, the signal function is defined as follows:

$$\text{Sig}(x) = \begin{cases} 0 & \text{if } x \in E, \\ 1 & \text{otherwise.} \end{cases} \quad (1)$$

For any polynomial $f(x) = (f_0, f_1, \cdots, f_{n-1})$, the following equation holds $\omega_B = \text{Sig}(f(x)) = (\text{Sig}(f_0), \text{Sig}(f_1), \cdots, \text{Sig}(f_{n-1}))$. In DXL-KE protocol, the
Algorithm 3 The DXL-KE protocol [15]

Alice
Sample \( a \in \mathbb{R}_q \), \( s_A \in \chi_\alpha \), \( e_A \in \chi_\alpha \)
\[ p_A = a \cdot s_A + 2 \cdot e_A \in \mathbb{R}_q \]
Send \( (a, p_A) \) to Bob

Bob
Sample \( s_B \in \chi_\alpha \), \( e_B \in \chi_\alpha \), \( g_B \in \chi_\alpha \)
\[ p_B = a \cdot s_B + 2 \cdot e_B \in \mathbb{R}_q \]
\[ k_B = p_A \cdot s_B + 2 \cdot g_B \in \mathbb{R}_q \]
\[ \omega_B = \text{Sig}(k_B) \]
\[ sk_B = \text{Mod}_2(k_B, \omega_B) \]
Send \( (p_B, \omega_B) \) to Alice.

Sample \( g_A \in \chi_\alpha \)
\[ k_A = p_B \cdot s_A + 2 \cdot g_A \in \mathbb{R}_q \]
\[ sk_A = \text{Mod}_2(k_A, \omega_B) \]

A significant unit is an robust extractor \( \text{Mod}_2(x, \omega) \), which enables Alice and Bob to extract an identical information:
\[
\text{Mod}_2(x, \omega) = (x + \omega \cdot \frac{q-1}{2}) \mod q \mod 2. \tag{2}
\]

In the DXL-KE protocol, the coefficients of the private key \( s_B \) are sampled from the discrete Gaussian distribution \( \chi_\alpha \), and the standard deviation of the discrete Gaussian distribution is \( \alpha = 3.197 \), therefore, for \( i = 0, 1, \cdots, n-1 \), the probability of \( |s_B[i]| < 5 \cdot \alpha = 15.985 \) is 99.9999%, the coefficients of \( e_B, g_B \) are also sampled from the same discrete Gaussian distribution, so we know that the ranges of these coefficients are \( |e_B[i]| \leq 15 \) and \( |g_B[i]| \leq 15 \) respectively.

4 Attacks at ESORICS 2022 against DXL-KE

For the scenario where Bob’s private key \( s_B \) is reused in the DXL-KE protocol, since Bob will send signals \( \omega_B \) to Alice, the adversary impersonates as Alice to recover Bob’s private key according to the signals sent by Bob. The signal leakage attacks can be used to recover Bob’s reused private key \( s_B \). In this section, we review the state-of-the-art attack proposed by Qin et al. in [31]. Please refer to [31] for more details.

Alice forges \( p_A = k_1 = 550 \) and sends it to Bob. After receiving \( p_A \), Bob computes \( p_B = a \cdot s_B + 2 \cdot e_B, k_B = p_A \cdot s_B + 2 \cdot g_A, \omega_B = \text{Sig}(k_B) \), where \( \omega_B = (\omega_0, \omega_1, \cdots, \omega_{n-1}) \) and \( \omega_i = \text{Sig}(k_B[i]), i = 0, 1, \cdots, n-1 \). Alice saves the signals \( \omega_B \) sent by Bob. Similarly, for different \( k_1, k_2, k_3, k_4 \) Alice can save four different signals sent by Bob. Then, for different values of \( k_t, 1 \leq t \leq 4 \), Alice constructs Table 1 which provides a map from the integer in \( \{0, \cdots, 15\} \) to the corresponding binary code-word, namely, the binary code for each column is called a binary code-word.
For the sake of discussion, four different signals $\omega_B$ returned by Bob are respectively defined as $\omega_B^t = (\omega_B^0, \omega_B^1, \cdots, \omega_B^{n-1})$, $t = 1, 2, 3, 4$, where $\omega_B^t$ is the involved signal of the $t$-th time. For example, if the value of $(\omega_B^1, \omega_B^2, \omega_B^3, \omega_B^4)$ equals the binary code-word $(1, 0, 1, 0)$, then Alice can determine the value in Table 1, which can recover the corresponding value of $s_B[i]$, $0 \leq i \leq n - 1$. In other words, Alice checks which of the collected signals matches $\omega$ respectively defined as $\omega = 0 \leq i \leq n - 1$. In this step, the number of forged public keys is $|\log(15 + 1)| = 4$.

In order to recover the values of $s_B[i]$ for $0 \leq i \leq n - 1$, Alice forges the public keys $p_A = k_i \cdot (1 + x)$, where $k_i$ is selected from the set $\{260, 525, 1050, 4000, 8192\}$ in ascending order, Alice sends them to Bob, Bob honestly executes according to the DXL-KE protocol, and sends $(p_B, \omega_B)$ to Alice, for 5 different forged public keys, Alice saved 5 different signals $s_B$. Based on these cases, Alice can build Table 2. It is easy to see that Table 2 presents an encoding of $|s_B[i]| + s_B[i + 1]|$, $0 \leq i \leq n - 1$. Therefore, Alice could obtain the values of $|s_B[i]| + s_B[i + 1]|$, $0 \leq i \leq n - 1$ by checking Table 2. In the step, the number of forged public keys is $|\log(30 + 1)| = 5$.

Note that $(|s_B[i]| + s_B[i + 1]|)^2 = s_B[i]^2 + s_B[i + 1]^2 + 2s_B[i] \cdot s_B[i + 1]$. Hence, after knowing the values of $|s_B[i]| + s_B[i + 1]|$, $|s_B[i]|$ and $|s_B[i + 1]|$, the value of $s_B[i] \cdot s_B[i + 1]$ will be determined. Furthermore, the positive and negative signs of $s_B[i] \cdot s_B[i + 1]$ can be obtained if $s_B[i]$ and $s_B[i + 1]$ are both not equal to 0.

### Table 1: Signals $\omega_B$ for different $k_j$ and $|s_B[i]|$ in DXL-KE with $i \in [0, n - 1]$

| $|s_B[i]|$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|--------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| $k_1$  | 550| 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   |
| $k_2$  | 1050| 0  | 0  | 0  | 0  | 1  | 1  | 1  | 1  | 1  | 1   | 0   | 0   | 0   | 0   | 0   |
| $k_3$  | 4000| 0  | 0  | 1  | 1  | 0  | 0  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   |
| $k_4$  | 8192| 0  | 1  | 0  | 1  | 0  | 1  | 0  | 1  | 0  | 0   | 0   | 0   | 0   | 0   | 0   |

### Table 2: Signals $\omega_B$ for different $k_j$ and $s[i] = |s_B[i]| + s_B[i + 1]|$ in DXL-KE with $i \in [0, n - 1]$

<table>
<thead>
<tr>
<th>$s[i]$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>14</th>
<th>...</th>
<th>29</th>
<th>...</th>
<th>30</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1$</td>
<td>260</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$k_2$</td>
<td>525</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>1</td>
<td>1</td>
<td>...</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$k_3$</td>
<td>1050</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$k_4$</td>
<td>4000</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>...</td>
<td>1</td>
<td>1</td>
<td>...</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$k_5$</td>
<td>8192</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>1</td>
<td>...</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
However, there are one or more consecutive zero cases in the private key $s_B$. In these cases, Alice cannot recover all bits in the reused private key $s_B$. To overcome this problem, Alice needs to forge additional public keys $p_{A,t} = (1 + x^{t+1}) \cdot k_t$ for all $1 \leq t \leq z$, where $z$ is the maximum number of consecutive zeros in private key $s_B$. ($z \leq 4$ is sufficient to successfully launch the attack according to the analysis in [31]). In this step, the number of forged public keys is $z \cdot \lceil \log(30 + 1) \rceil = 5z$.

To sum up, the number of forged public keys is $4 + 5(z + 1)$, which equals 29 by taking $z = 4$.

Note that the number of forged public keys is equal to the number of reusing the private key $s_B$ of Bob. It means that, if the number of reusing $s_B$ will be greater than $4 + 5(z + 1)$, the private key $s_B$ will be revealed. Therefore, from the attacker’s perspective, we expect the attack to still work even with the lowest possible number of reusable private keys.

5 New attack against DXL-KE protocol

In this section, we propose a new algorithm to reduce the number of public key forgeries, that is, the number of times Bob’s private key $s_B$ is reused. This algorithm can also be divided to two parts: absolute values recovery and relative signs recovery. For absolute values recovery, the process is the same as that of Qin et al., therefore, for simplicity, we no longer describe the details of absolute value recovery, but by default Alice has obtained $|s_B[i]|$ by forging the public key 4 times, where $s_B[i]$ ($0 \leq i \leq n - 1$) is the $i$-th coefficient of private key $s_B$. Please refer to Section 4 for more details.

For relative signs recovery, we divide this procedure into the following two steps for the sake of description. In this section, we assume that Alice is an adversary who want to recover Bob’s reused private key $s_B$.

5.1 Step A: Recovering the values of $\sum_{j=0}^{\lambda-1} |s_B[i+j]|$

In this step, Alice recovers the values of $\text{sum}_{i,\lambda} := \sum_{j=0}^{\lambda-1} |s_B[i+j]|$, $i = 0, 1, \cdots, n-\lambda$ rather than $|s_B[i] + s_B[i+1]|$ in the Qin et al.’s attack, where $\lambda > 2$ is an integer chosen by Alice.

In order to recover the values of $\text{sum}_{i,\lambda}$, Alice needs to forge her own public key in the following way:

$$p_{A,t} = (1 - x^{n-1} - \cdots - x^{n-\lambda+1}) \cdot k_t.$$  \hspace{1cm} (3)

Here $p_{A,t}$ is the $t$-th forged public key and $|k_t| \leq \frac{q - 1}{2}$ for $t = 1, \cdots, m$, where $m$ is the number of times public keys of Alice have been forged.
Notice that \( |s_B[i + j]| \leq 15 \) with an overwhelming probability for \( 0 \leq i + j \leq n - 1 \). According to the triangle inequality, we get \( \sum_{j=0}^{\lambda-1} s_B[i + j] \leq 15\lambda \), namely, 
\[ \text{sum}_{i,\lambda} \leq 15\lambda. \]
We can construct a one-to-one mapping from every integer \( I \) in the set \( \{0, \cdots, 15\lambda\} \) to the corresponding binary code-word. It is not hard to see that the least \( m \) equals \( \lceil \log(15 \cdot \lambda + 1) \rceil \). Hence, we can take \( m = \lceil \log(15 \cdot \lambda + 1) \rceil \).
It means that \( m = 6 \) for \( 3 \leq \lambda \leq 4 \), and \( m = 7 \) for \( 5 \leq \lambda \leq 8 \). As \( \lambda \) increases, \( m \) will increase. It implies that the number of forged public keys will increase and it becomes difficult to forge the appropriate public keys. Here, we consider the case of \( 3 \leq \lambda \leq 8 \).

**Remark 1.** For a given \( i \), the range of \( \text{sum}_{i,\lambda} \) is \([0, 45]\) and \([0, 60]\), when \( \lambda = 3 \) and 4, respectively. Note that \([0, 45]\subset [0, 60]\). Hence, the range of \( \text{sum}_{i,4} \) is a part of range of \( \text{sum}_{i,4} \). Similar to the above analysis, we can get that ranges of \( \text{sum}_{i,5}, \text{sum}_{i,6}, \text{sum}_{i,7} \) are all a part of the range of \( \text{sum}_{i,8} \).

According to the remark, we only present two cases for \( \lambda = 4 \) and 8. Next, we focus on the case of \( \lambda = 4 \).

**Finding \( k_t \) in expression (3).** For \( \lambda = 4 \), the involved \( m = 6 \). Bob will calculate
\[ k_B = p_A \cdot s_B + 2 \cdot g_B, \]
where \( p_A = (1 - x^{n-1} - x^{n-2} - x^{n-3}) \cdot k_t \) for \( 1 \leq t \leq 6 \) and \( |k_t| \leq \frac{q-1}{2} \).

Therefore, \( k_B[i] = k_t \cdot (\sum_{j=0}^{3} s_B[i + j]) + 2 \cdot g_B[i] \) for \( i = 0, 1, \cdots, n - 4 \). Note that 
\[ \text{sum}_{i,4} = \sum_{j=0}^{3} s_B[i + j]. \]
Thus we have
\[ k_B[i] = k_t \cdot \text{sum}_{i,4} + 2 \cdot g_B[i] \] for \( i = 0, 1, \cdots, n - 4 \).

For sake of description, we assume that \( k_t > 0 \). According to the triangle inequality, we get
\[ k_t \cdot |\text{sum}_{i,4}| - 2 \cdot |g_B[i]| \leq |k_B[i]| \leq k_t \cdot |\text{sum}_{i,4}| + 2 \cdot |g_B[i]|. \quad (4) \]

Further, there is the following relation
\[ k_t \cdot (\min_i |\text{sum}_{i,4}| - 2 \cdot (\max_i |g_B[i]|)) \leq |k_B[i]| \leq k_t \cdot (\max_i |\text{sum}_{i,4}|) + 2 \cdot (\max_i |g_B[i]|). \quad (5) \]

Next, we review the signal function which is defined as \( \text{Sig}(x) = 0 \), if \( x \in E \), otherwise, \( \text{Sig}(x) = 1 \). Here, the set \( E = \{-\left[\frac{q}{4}\right] + r, \cdots, \left[\frac{q}{4}\right] + r\} \), where \( r \) selects uniformly from \( \{0, 1\} \).
Therefore, for two integers $\alpha_1$ and $\alpha_2$, the following equation holds,

\[
\begin{cases}
\text{if } -\left\lfloor \frac{q}{4} \right\rfloor + 1 + \alpha_1 \cdot q \leq |k_B[i]| \leq \left\lfloor \frac{q}{4} \right\rfloor + \alpha_1 \cdot q, & \text{Sig}(k_B[i]) = 0, \\
\text{if } \left\lceil \frac{q}{4} \right\rceil + 1 + \alpha_2 \cdot q \leq |k_B[i]| \leq \left\lfloor \frac{3}{4} \cdot q \right\rfloor + \alpha_2 \cdot q & \text{Sig}(k_B[i]) = 1.
\end{cases}
\] (6)

We start with the first signal, namely, \([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]\).

1. If $0 \leq |\text{sum}_{i,4}| \leq 15$, then $-\left\lfloor \frac{q}{4} \right\rfloor + 1 \leq |k_B[i]| \leq \left\lfloor \frac{q}{4} \right\rfloor$. It means that $\text{Sig}(k_B[i]) = 0$.
2. If $16 \leq |\text{sum}_{i,4}| \leq 47$, then $\left\lceil \frac{q}{4} \right\rceil + 1 \leq |k_B[i]| \leq \left\lfloor \frac{3}{4} \cdot q \right\rfloor$. It implies that $\text{Sig}(k_B[i]) = 1$.
3. If $48 \leq |\text{sum}_{i,4}| \leq 60$, then $\left\lceil \frac{3}{4} \cdot q \right\rceil + 1 \leq |k_B[i]| \leq \left\lfloor \frac{3}{4} \cdot q \right\rfloor$. It leads to $\text{Sig}(k_B[i]) = 0$.

If conditions 1, 2, and 3 are satisfied, then the first targeted signal can be realized.

For $0 \leq |\text{sum}_{i,4}| \leq 15$, according to the expression (5), if the relation

\[ k_1 \leq \frac{\left\lfloor \frac{q}{4} \right\rfloor - 30}{15} \] (7)

is satisfied, then $-\left\lfloor \frac{q}{4} \right\rfloor + 1 \leq |k_B[i]| \leq \left\lfloor \frac{q}{4} \right\rfloor$. That is, the condition 1 holds.

For $16 \leq |\text{sum}_{i,4}| \leq 47$, from the expression (5), if the relation

\[ k_1 \leq \frac{\left\lceil \frac{3}{4} \cdot q \right\rceil - 30}{47}, k_1 \geq \frac{\left\lceil \frac{q}{4} \right\rceil + 1 + 30}{16} \] (8)

is satisfied, then $\left\lceil \frac{q}{4} \right\rceil + 1 \leq |k_B[i]| \leq \left\lfloor \frac{3}{4} \cdot q \right\rfloor$. Namely, the condition 2 holds.

For $48 \leq |\text{sum}_{i,4}| \leq 60$, based on the expression (5), if the relation

\[ k_1 \leq \frac{\left\lceil \frac{3}{4} \cdot q \right\rceil - 30}{60}, k_1 \geq \frac{\left\lceil \frac{3}{4} \cdot q \right\rceil + 1 + 30}{48} \] (9)

is satisfied, then $\left\lceil \frac{3}{4} \cdot q \right\rceil + 1 \leq |k_B[i]| \leq \left\lfloor \frac{3}{4} \cdot q \right\rfloor$.
is satisfied, then \([\frac{3}{4} \cdot q] + 1 \leq |k_B[i]| \leq \lfloor \frac{5}{4} \cdot q \rfloor\). I.e., the condition 3 holds.

From the expressions (7), (8), (9), the range of \(k_1\) is

\[
\frac{\left[\frac{3}{4}\right] + 1 + 30}{16} \leq k_1 \leq \frac{\lfloor \frac{5}{4} \cdot q \rfloor - 30}{47}.
\]  

(10)

We construct the second targeted signal, namely, \([0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0]\).

1. If \(0 \leq |\text{sum}_{i,4}| \leq 7\), the condition \(\left[\frac{3}{4}\right] + 1 = |k_B[i]| \leq \left[\frac{5}{4}\right]\) holds, which makes \(\text{Sig}(k_B[i]) = 0\).

2. If \(8 \leq |\text{sum}_{i,4}| \leq 23\), the condition \(\left[\frac{3}{4}\right] + 1 \leq |k_B[i]| \leq \lfloor \frac{3}{4} \cdot q \rfloor\) holds, which lets \(\text{Sig}(k_B[i]) = 1\).

3. If \(24 \leq |\text{sum}_{i,4}| \leq 39\), the condition \(\left[\frac{3}{4} \cdot q\right] + 1 \leq |k_B[i]| \leq \left[\frac{5}{4} \cdot q\right]\) holds, which leads to \(\text{Sig}(k_B[i]) = 0\).

4. If \(40 \leq |\text{sum}_{i,4}| \leq 55\), the condition \(\left[\frac{5}{4} \cdot q\right] + 1 \leq |k_B[i]| \leq \left[\frac{7}{4} \cdot q\right]\) holds, which happens \(\text{Sig}(k_B[i]) = 1\).

5. If \(56 \leq |\text{sum}_{i,4}| \leq 60\), the condition \(\left[\frac{5}{4} \cdot q\right] + 1 \leq |k_B[i]| \leq \left[\frac{9}{4} \cdot q\right]\) holds, which occurs \(\text{Sig}(k_B[i]) = 0\).

If the above conditions are satisfied, then the second targeted signal can be realized.

From these conditions, we can get the range of \(k_2\) is

\[
\frac{\left[\frac{3}{4}\right] + 1 + 30}{8} \leq k_2 \leq \frac{\left[\frac{7}{4} \cdot q\right] - 30}{55}.
\]  

(11)

Similarly, we obtain that the ranges of \(k_3, k_4, k_5, k_6\) are as follows:

\[
\frac{\left[\frac{3}{4}\right] + 1 + 30}{4} \leq k_3 \leq \frac{\left[\frac{15}{4} \cdot q\right] - 30}{59}.
\]  

(12)

\[
\frac{\left[\frac{3}{4}\right] + 1 + 30}{2} \leq k_4 \leq \frac{\left[\frac{29}{4} \cdot q\right] - 30}{57}.
\]  

(13)

\[
\frac{\left[\frac{3}{4}\right] + 1 + 30}{1} \leq k_5 \leq \frac{\left[\frac{61}{4} \cdot q\right] - 30}{60}.
\]  

(14)

\[
\frac{\left[\frac{119}{4} \cdot q\right] + 1 + 30}{60} \leq k_6 \leq \frac{\left[\frac{121}{4} \cdot q\right] - 30}{60}.
\]  

(15)

For DXL-KE parameters, this means concretely

\(k_1 \in [258.00, 260.80], k_2 \in [516.00, 520.78], k_3 \in [1032.00, 1040.89],\)

\(k_4 \in [2064.00, 2083.52], k_5 \in [4128.00, 4164.01], k_6 \in [8124.75, 8260.26].\)
Table 3: Signals $\omega_i^{(t)}$ for different $k_t$, $t = 1, 2, \cdots, 6$ and the range of $\text{sum}_{i,4}$ satisfying $0 \leq \text{sum}_{i,4} \leq 60$, where the vector corresponding to each column is called a binary code-word. For example, if the binary code-word is $(\omega_i^{(1)}, \omega_i^{(2)}, \omega_i^{(3)}, \omega_i^{(4)}, \omega_i^{(5)}, \omega_i^{(6)})^T = (0, 1, 0, 0, 0, 1)^T$, it means that $\text{sum}_{i,\lambda} = 15$.

<table>
<thead>
<tr>
<th>$k_1$</th>
<th>258</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_2$</td>
<td>516</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$k_3$</td>
<td>1033</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$k_4$</td>
<td>2067</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$k_5$</td>
<td>4061</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$k_6$</td>
<td>8192</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4: Signals $\omega_i^{(t)}$ for different $k_t$, $t = 1, 2, \cdots, 7$ and the range of $\text{sum}_{i,8}$ satisfying $0 \leq \text{sum}_{i,8} \leq 120$

<table>
<thead>
<tr>
<th>$k_1$</th>
<th>129</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>\cdots</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_2$</td>
<td>258</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\cdots</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$k_3$</td>
<td>516</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\cdots</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$k_4$</td>
<td>1033</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>\cdots</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$k_5$</td>
<td>2065</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>\cdots</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$k_6$</td>
<td>4130</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>\cdots</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$k_7$</td>
<td>8192</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>\cdots</td>
<td>1</td>
<td>0</td>
<td>\cdots</td>
<td>1</td>
</tr>
</tbody>
</table>
Therefore, based on the assumption that $0 < k_t \leq \frac{2^n - 1}{2}$, we can select

$$k_1 = 258, k_2 = 516, k_3 = 1033, k_4 = 2067, k_5 = 4164, k_6 = 8192.$$  

Using the above method, different values of $k_1, k_2, k_3, k_4, k_5, k_6$ will generate different targeted signals. These signals will construct the following Table 3.

To sum up, for the case of $\lambda = 4$, Alice forges public keys 6 times to recover the value of $\text{sum}_{i, \lambda}$ by checking which binary code-word uniquely matches this value.

The analysis for the case of $\lambda = 8$ is similar to that of $\lambda = 4$. We omit the detailed process and only present Table 4.

### 5.2 Step B: Recovering the value of $s_B$

According to the analysis in Step A, Alice can obtain the values of $\text{sum}_{i, \lambda}$ for every $i = 0, 1, \ldots, n - \lambda$. Moreover, in the process of absolute values recovery, Alice has recovered the absolute values $|s_B[i]|, i = 0, 1, \ldots, n - 1$. Next, Alice will show how to reveal the sign of $s_B[i]$. The specific recovery procedure is as follows:

For the sake of simplicity, we ignore the subscript $\lambda$ in the notation $\text{sum}_{i, \lambda}$ and only write $\text{sum}_i$, i.e. $\text{sum}_i = \left| \sum_{j=0}^{\lambda-1} s_B[i + j] \right|$ for $i = 0, 1, \ldots, n - \lambda$. Note that Alice has obtained the absolute values $|s_B[i]|$. In order to determine the first $\lambda - 1$ coefficients $s_B[0], \ldots, s_B[\lambda - 2]$, Alice needs to guess their signs. Therefore, Alice can get the correct signs by guessing at most $2^{\lambda-1}$ times. For each guess, Alice knows the sum $s_B[0] + s_B[1] + \cdots + s_B[\lambda - 2]$. Denote $\text{cnt}_i$ as $s_B[i] + s_B[i + 1] + \cdots + s_B[i + \lambda - 2]$ for $0 \leq i \leq n - \lambda$. Here, $\text{cnt}_0 = s_B[0] + s_B[1] + \cdots + s_B[\lambda - 2]$. Therefore, we can rewrite $\text{sum}_i = |\sum_{j=0}^{\lambda-1} s_B[i + j]|$ as

$$\text{sum}_i = |\text{cnt}_i + s_B[i + \lambda - 1]|$$  

(16) for $0 \leq i \leq n - \lambda$. Alice already knows the values of $\text{sum}_0, \text{cnt}_0, |s_B[\lambda - 1]|$, in order to determine the sign of $s_B[\lambda - 1]$, Alice checks whether

$$\text{sum}_0 = |\text{cnt}_0 + s_B[\lambda - 1]|$$

or

$$\text{sum}_0 = |\text{cnt}_0 - s_B[\lambda - 1]|$$

is correct. The following predicates $P_1(\cdot)$ and $P_2(\cdot)$ simplify this process:

The inputs for predicates $P_1(\cdot)$ and $P_2(\cdot)$ are $\text{sum}_i, \text{cnt}_i$ and $|s_B[i + \lambda - 1]|$. The corresponding outputs are as follows:

$$P_1(\cdot) = \begin{cases} 1 & \text{if } \text{sum}_i = |\text{cnt}_i + s_B[i + \lambda - 1]|, \\ 0 & \text{if } \text{sum}_i \neq |\text{cnt}_i + s_B[i + \lambda - 1]|. \end{cases}$$  

(17)
\[
\mathcal{P}_2(\cdot) = \begin{cases} 
1 & \text{if } \text{sum}_i = |\text{cnt}_i - |s_B[i + \lambda - 1]||, \\
0 & \text{if } \text{sum}_i \neq |\text{cnt}_i - |s_B[i + \lambda - 1]||.
\end{cases}
\] 

(18)

The outputs of predicates \(\mathcal{P}_1(\cdot)\) and \(\mathcal{P}_2(\cdot)\) have four cases:

**Case 1**: The outputs of predicates \(\mathcal{P}_1(\cdot)\) and \(\mathcal{P}_2(\cdot)\) are \((1,0)\).

**Case 2**: The outputs of predicates \(\mathcal{P}_1(\cdot)\) and \(\mathcal{P}_2(\cdot)\) are \((0,1)\).

**Case 3**: The outputs of predicates \(\mathcal{P}_1(\cdot)\) and \(\mathcal{P}_2(\cdot)\) are \((0,0)\).

**Case 4**: The outputs of predicates \(\mathcal{P}_1(\cdot)\) and \(\mathcal{P}_2(\cdot)\) are \((1,1)\).

In the process of determining the sign of \(s_B[\lambda - 1]\), i.e. \(i = 0\), the inputs for predicates \(\mathcal{P}_1(\cdot)\) and \(\mathcal{P}_2(\cdot)\) are \(\text{sum}_0, \text{cnt}_0, |s_B[\lambda - 1]|\). If **Case 1** occurs, the sign of \(s_B[\lambda - 1]\) is positive. If **Case 2** occurs, the sign of \(s_B[\lambda - 1]\) is negative. If **Case 3** occurs, the value of \(\text{cnt}_0\) is incorrect, Alice should guess another value for \(\text{cnt}_0\). If **Case 4** occurs, it means that \(\text{cnt}_0 = 0\) or \(|s_B[\lambda - 1]| = 0\).

For **Case 4**, if \(|s_B[\lambda - 1]| = 0\), then Alice determines \(s_B[\lambda - 1] = 0\), and if \(\text{cnt}_0 = 0\), Alice can not determine \(s_B[\lambda - 1] > 0\) or \(s_B[\lambda - 1] < 0\), but Alice already knows the absolute value \(|s_B[\lambda - 1]|\). Therefore, Alice chooses \(s_B[\lambda - 1] < 0\) (or \(s_B[\lambda - 1] > 0\)) and stores the location of \(s_B[\lambda - 1]\) in memory. After knowing the value of \(s_B[\lambda - 1]\), then Alice obtains the values of \(\text{sum}_1, \text{cnt}_1, |s_B[\lambda]|\). Therefore, Alice could recover the value of \(s_B[\lambda]\). In the process of recovering the remaining signs, the inputs for the predicates \(\mathcal{P}_1(\cdot)\) and \(\mathcal{P}_2(\cdot)\) are \(\text{sum}_i, \text{cnt}_i, |s_B[i + \lambda - 1]|\). Using the same approach, Alice could recover all private key coefficients.

It is worth noting that \(\text{cnt}_i = 0\) may occur several times when Alice recovers the private key coefficient, and the memory stores multiple locations. If the **Case 3** occurs while recovering the sign of \(s_B[t]\) and the memory is not empty, Alice needs to extract the position \(\text{pos}\) that is closest to \(t\) in memory, and reset \(s_B[\text{pos}] > 0\) (or \(s_B[\text{pos}] < 0\)), finally, Alice deletes \(\text{pos}\) from the stored memory. In addition, if the memory is empty, then Alice needs to guess another value of \(\text{cnt}_0\). This approach can be seen as the depth first search (DFS). In following Fig. 3, we show that the depth first search is utilized to recover the reused private key \(s_B\).

**Remark 2.** The **Case 3** will occur several times and it filters out the redundant operations, so we can complete the key recovery in a very short time.
Remark 3. Case 4 will generate two branches and Case 3 will abort the impossible branches. Here, we propose an open problem: are there any methods to predict the number of occurrences of Case 3 and Case 4?

Based on a combination of signal leakage attack and depth first search, several candidate private keys $s_B^{(t)}$ may be recovered. Then it is necessary to determine the real private key $s_B$. In Algorithm 1, we use the predicate $\tau(s_B^{(t)})$ to simplify the check stage. Next we explain how to determine whether a recovered private key $s_B^{(t)}$ is the real private key $s_B$.

Since Alice can obtain the recovered private key $s_B^{(t)}$ and know the polynomials $a \in \mathcal{R}_q$ and $p_B \in \mathcal{R}_q$, Alice computes $2 \cdot e = p_B - a \cdot s_B^{(t)}$ and $||2 \cdot e||_{\infty}$, where $||2 \cdot e||_{\infty} := \max_{i=0}^{n-1} \{|2 \cdot e_i|\}$. In the DXL-KE protocol, the equation $2 \cdot e_B = p_B - a \cdot s_B$ holds, and the ranges of these coefficients satisfy $|e_B[i]| \leq 15$. Hence, $||2 \cdot e_B||_{\infty} \leq 30$.

The specific process of predicate $\tau(s_B^{(t)})$ is that Alice checks whether $||e||_{\infty}$ satisfies $||2 \cdot e||_{\infty} \leq 30$. If $||2 \cdot e||_{\infty} \leq 30$ is satisfied, the predicate $\tau(s_B^{(t)})$ outputs 1, namely, the recovered private key $s_B^{(t)}$ is the real private key $s_B$. Conversely, if $||2 \cdot e||_{\infty} \leq 30$ is not satisfied, the predicate $\tau(s_B^{(t)})$ outputs 0, the recovered private key $s_B^{(t)}$ is not the real private key $s_B$.

It is worth noting that there may be many candidate private keys, and incorrect candidate private keys can be filtered out through the checking stage.

5.3 A simple example

We give a simple example to illustrate the above attack process. In this example, we take $\lambda = 8$. 
In Fig. 4, blue ($-,-,+$) indicates the sign of the first seven coefficients that Alice guesses, and red ($-,-,+$) presents the case where $cnt_i = 0$, Alice needs to guess $s_B[i+7] > 0$ or $s_B[i+7] < 0$, green ($-,-,+$) said Alice determined $s_B[i+7] > 0$ or $s_B[i+7] < 0$ according to the predicates $P_1(\cdot)$ and $P_2(\cdot)$, $\perp$ means an error that occurred during the recovery process.

For the first position, from the value of $|s_B[7]| = 0$, Alice can quickly determine $s_B[7] = 0$.

For the second position, since $cnt_1 = 0, |s_B[8]| = 1$, and $sum_1 = 1$, the outputs of the predicates $P_1(\cdot)$ and $P_2(\cdot)$ are $(1,1)$, Alice cannot get the sign of $s_B[8]$. So Alice randomly select $s_B[8] < 0$, namely, $s_B[8] = -1$, and store the location $8$ in memory $\{8\}$.

For the third position, since $cnt_2 = -3, |s_B[9]| = 1$ and $sum_2 = 0$, the outputs of the predicates $P_1(\cdot)$ and $P_2(\cdot)$ are $(0,0)$ and the memory $\{8\}$ is not empty, it means that Alice selected the sign of $s_B[8]$ is wrong. Thus, Alice should set $s_B[8] > 0$, namely $s_B[8] = 1$, and delete the position $8$ from the stored memory. Then, Alice can determine that $cnt_2 = -1, |s_B[9]| = 1$ and $sum_2 = 0$, the outputs of predicates $P_1(\cdot)$ and $P_2(\cdot)$ are $(1,0)$. Therefore, Alice obtains $s_B[9] = 1$.

For the fourth position, $cnt_3 = 2, |s_B[10]| = 2$ and $sum_3 = 4$, the outputs of the predicates $P_1(\cdot)$ and $P_2(\cdot)$ are $(1,0)$. Hence, Alice knows $s_B[10] = 2$.

For the fifth position, $cnt_4 = 7, |s_B[11]| = 5$ and $sum_4 = 2$, the outputs of the predicates $P_1(\cdot)$ and $P_2(\cdot)$ are $(0,1)$. Therefore, Alice gets $s_B[11] = -5$.

Using the same operation, Alice will get the whole private key $s_B$ one position after another.

**Remark 4.** The above recovery process is an instance of Figure 3.

### 6 Experiments

The experimental environment is running on a personal computer with 2.40GHz Intel(R) Core(TM) i5-10200h CPU and 16GB RAM, and the operating system is Ubuntu 18.04.6LTS. For different $\lambda$ values, we tested DXL-KE protocol for...
1000 times, and obtained the average number of forged public keys and the time required to complete the attack as follows:

Table 5: Comparison of the experimental results on DXL-KE. Ours\(^1\) is the case of \(\lambda = 3\), Ours\(^2\) shows the results of \(\lambda = 4\) and so on. \#Queries presents the number of forged public keys, Time is the time required to complete key recovery.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Attacks</th>
<th>(n)</th>
<th>(\alpha)</th>
<th>(q)</th>
<th>#Queries</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Previous Best [31]</td>
<td></td>
<td></td>
<td></td>
<td>24.23</td>
<td>0.67s</td>
</tr>
<tr>
<td></td>
<td>Ours(^1)</td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td>∞</td>
</tr>
<tr>
<td></td>
<td>Ours(^2)</td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td>0.36s</td>
</tr>
<tr>
<td></td>
<td>Ours(^3)</td>
<td></td>
<td></td>
<td></td>
<td>11</td>
<td>0.24s</td>
</tr>
<tr>
<td></td>
<td>Ours(^4)</td>
<td></td>
<td></td>
<td></td>
<td>11</td>
<td>0.24s</td>
</tr>
<tr>
<td></td>
<td>Ours(^5)</td>
<td></td>
<td></td>
<td></td>
<td>11</td>
<td>0.24s</td>
</tr>
<tr>
<td></td>
<td>Ours(^6)</td>
<td></td>
<td></td>
<td></td>
<td>11</td>
<td>0.24s</td>
</tr>
</tbody>
</table>

7 Conclusion

In this paper, we proposed a more efficient key recovery method for the case of key reuse in DXL-KE protocol, which combines with signal leakage attack and depth first search. This method requires fewer public keys to be forged and the private key can be recovered in a shorter time. Compared with previous attacks, our results show that in terms of the number of forged public keys, our work is about 60% lower, and our work is about 50% lower in the time it takes to complete the whole key recovery. More specifically, Alice needs to forge the public keys 10 or 11 times to recover Bob’s reused private key in 0.36 seconds or 0.24 seconds.

References

A  The choices of $k$ in relative signs recovery

We show how to get the value of $k$ to match the corresponding signals in Table 4.

A.1  The choices of $k$ for $\lambda = 8$

For the case of $\lambda = 8$, the process of obtaining the ranges of $k_1, k_2, \ldots, k_7$ is similar to the case of $\lambda = 4$. We take $k_1$ as an example to analyze the range of $k_1$, and we no longer give specific details but only the ranges of $k_2, \ldots, k_7$.

We start with the first signal in Table 4, namely, $[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$. 

1. If $0 \leq |\text{sum}_{i,8}| \leq 31$, then $\lfloor \frac{q}{4} \rfloor + 1 \leq |k_B[i]| \leq \lfloor \frac{q}{4} \rfloor$. It means that $\text{Sig}(k_B[i]) = 0$.

2. If $32 \leq |\text{sum}_{i,8}| \leq 95$, then $\lfloor \frac{3}{4} * q \rfloor + 1 \leq |k_B[i]| \leq \lfloor \frac{3}{4} * q \rfloor$. It implies that $\text{Sig}(k_B[i]) = 1$.

3. If $96 \leq |\text{sum}_{i,8}| \leq 120$, then $\lfloor \frac{3}{4} * q \rfloor + 1 \leq |k_B[i]| \leq \lfloor \frac{3}{4} * q \rfloor$. It leads to $\text{Sig}(k_B[i]) = 0$.

If conditions 1, 2, and 3 are satisfied, then the first targeted signal can be realized.

For $0 \leq |\text{sum}_{i,8}| \leq 31$, according to the expression (5), if the relation is satisfied, then $\lfloor \frac{q}{4} \rfloor + 1 \leq |k_B[i]| \leq \lfloor \frac{q}{4} \rfloor$. That is, the condition 1 holds.

For $32 \leq |\text{sum}_{i,8}| \leq 95$, from the expression (5), if the relation is satisfied, then $\lfloor \frac{3}{4} * q \rfloor + 1 \leq |k_B[i]| \leq \lfloor \frac{3}{4} * q \rfloor$. Namely, the condition 2 holds.

For $96 \leq |\text{sum}_{i,8}| \leq 120$, based on the expression (5), if the relation is satisfied, then $\lfloor \frac{3}{4} * q \rfloor + 1 \leq |k_B[i]| \leq \lfloor \frac{3}{4} * q \rfloor$. I.e., the condition 3 holds.

From the expressions (19), (20), (21), the range of $k_1$ is

$$\frac{q}{4} + 1 + 30 \leq k_1 \leq \frac{3}{4} * q - 30.$$
Similarly, the ranges of \( k_2, k_3, k_4, k_5, k_6, k_7 \) are as follows:

\[
\frac{q^4 + 1 + 30}{16} \leq k_2 \leq \frac{q^7 * q - 30}{111} \quad (23)
\]

\[
\frac{q^4 + 1 + 30}{8} \leq k_3 \leq \frac{q^7 * q - 30}{119} \quad (24)
\]

\[
\frac{q^4 + 1 + 30}{4} \leq k_4 \leq \frac{q^7 * q - 30}{115} \quad (25)
\]

\[
\frac{q^4 + 1 + 30}{2} \leq k_5 \leq \frac{q^7 * q - 30}{117} \quad (26)
\]

\[
\frac{q^4 + 1 + 30}{1} \leq k_6 \leq \frac{q^7 * q - 30}{120} \quad (27)
\]

\[
\frac{239 * q + 1 + 30}{120} \leq k_7 \leq \frac{241 * q - 30}{120} \quad (28)
\]

For DXL-KE parameters, this means concretely:

\( k_1 \in [129.00, 129.03], \ k_2 \in [258.00, 258.04], \ k_3 \in [516.00, 516.07], \ k_4 \in [1032.00, 1032.70], \ k_5 \in [2064.00, 2065.36], \ k_6 \in [4128.00, 4130.13], \ k_7 \in [8158.62, 8226.38]. \)

Therefore, we can select \( k_1 = 129, k_2 = 258, k_3 = 516, k_4 = 1032, k_5 = 2065, k_6 = 4130, k_7 = 8192. \)