spKEX: An optimized lattice-based key-exchange

Sauvik Bhattacharya, Oscar Garcia-Morchon,
Ronald Rietman, Ludo Tolhuizen
Philips Research, Eindhoven, The Netherlands
{sauvik.bhattacharya,oscar.garcia-morchon,ronald.rietman,ludo.tolhuizen}@philips.com

Abstract

The advent of large-scale quantum computers has resulted in significant interest in quantum-safe cryptographic primitives. Lattice-based cryptography is one of the most attractive post-quantum cryptographic families due to its well-understood security, efficient operation and versatility. However, LWE-based schemes are still relatively bulky and slow.

In this work, we present spKEX, a forward-secret, post-quantum, unauthenticated lattice-based key-exchange scheme that combines four techniques to optimize performance. spKEX relies on Learning with Rounding (LWR) to reduce bandwidth; it uses sparse and ternary secrets to speed up computations and reduce failure probability; it applies an improved key reconciliation scheme to reduce bandwidth and failure probability; and computes the public matrix A by means of a permutation to improve performance while allowing for a fresh A in each key exchange.

For a quantum security level of 128 bits, our scheme requires 32% lesser bandwidth than the LWE-based key-exchange proposal Frodo [8] and allows for a fast implementation of the key exchange.

Contents

1 Introduction

The exchange of sensitive information, e.g. financial, military or private data, over communication systems requires solutions to ensure the confidentiality of these information transactions. Confidentiality can be enabled by means of cryptographic primitives such as public-key encryption [21] and key-establishment [14]. However, the approach of quantum computing, in combination with Shor's [23] and Grover's [16] algorithms has resulted in a need for quantum-resistant algorithms since a majority of the world's communication today depend on cryptography that is vulnerable to cryptanalysis by quantum computers. This need is recognized by organizations such as NIST and ETSI that are currently standardizing such solutions.

Key-exchange is one of the most sensitive and urgent application areas where post-quantum cryptography is required. The reason is that an attacker who collects encrypted data of interest to him (even assuming that forward-secret cryptography is used) will be able to decrypt this data when a quantum computer becomes available. Thus, this paper focuses on this primitive.

Lattice-based cryptography is a promising candidate for quantum-resistant cryptography due to its (relatively) good performance, versatility in different cryptographic schemes and resistance against all known quantum algorithms. In particular, the Learning with Errors (LWE) problem [20] is a hard mathematical problem with quantum reductions to the worst-case hard lattice problems GapSVP and SIVP [20] and classical reductions to GapSVP [18, 9]. In one of the flavors of LWE, the attacker is given many pairs $(a_i, \{b_i = a_i s + e_i \pmod q\})$ and his task is to recover s, where a_i and s are randomly chosen vectors from a uniform distribution and e_i are randomly taken from a Gaussian distribution. The Learning with Rounding (LWR) problem [5] is a deterministic variant of the LWE problem that replaces standard Gaussian errors with errors introduced by rounding to a smaller modulus. The motivation is to achieve higher efficiency due to the difficulty of sampling from a Gaussian distribution and to reduce the ciphertext length [10]. For both LWE and LWR, *small-secret* and *sparse* variants are also possible, i.e., ones in which the secret is sampled from a binary or ternary distribution, secrets that are sparse, and secrets that are both sparse and small [12].

The schemes of Ding *et al* [15] and the Frodo scheme due to Bos *et al* [8] are post-quantum, unauthenticated key-exchange schemes based on the Learning with Errors (LWE) problem. They use public keys derived using secret vectors and Gaussian noise to establish a shared secret between two entities in an ephemeral manner. The sampling from a Gaussian distribution while creating the LWE error leads to a significant overhead. The Lizard scheme of Cheon *et al* [13] is a public-key encryption (PKE) scheme that uses the reduction of the LWE problem to the Learning with Rounding (LWR) problem [5, 7, 4] in order to replace the slower Gaussian sampling with more efficient rounding for incorporating noise. Furthermore, Lizard proposes the use of sparse, ternary secrets in order to increase efficiency, and is based on both LWE and the LWR problem with sparse-ternary secrets (sparse-ternary LWR or sp-terLWR). However,

these schemes are still relatively bulky and the open question is if they can be further optimized for real-world usage scenarios.

This document describes an unauthenticated, ephemeral lattice-based key-exchange – spKEX, based on the LWR problem with sparse-ternary secrets. This key-exchange proposal builds on the lattice-based KEX proposed by [15] and [8]. Performance improvements are achieved by a combination of four design techniques:

- Introduction of noise by using rounding [5], which is computationally more efficient than sampling noise from a discrete Gaussian distribution [2, 8, 13] and allows reducing bandwidth requirements.
- Use of the generalized reconciliation mechanism from [24] to achieve a (sufficiently low) target failure probability with smaller configuration parameters.
- The application of sparse ternary secrets to optimize computations.
- Introduction of a computationally inexpensive approach to refresh the public parameter $\boldsymbol{A}$ by permuting a fixed, master public parameter. This aims at preventing pre-computation or backdoor-like attacks [19] while being more efficient than purely random approaches [8, 2].

Although some of the techniques above are known, they have not been used in actual key exchanges yet. For instance, Frodo mentions the potential usage of LWR but it does not describe the design of such a key exchange nor does it quantify its security level. Thus, the present work aims at showing how efficient a lattice-based key exchange can be made if all known techniques that improve performance are applied together, while providing a thorough analysis of the corresponding security and operational results. The remainder of this paper is organized as follows: Section 2 describes our notation. Section 3 describes spKEX. Section 4, 5, and 6 respectively analyze the failure probability, security, and performance of spKEX. Section 7 concludes this paper.

2 Notation

A vector is denoted in small letters, bold-face, as $\boldsymbol{v}$, and is always considered as a column vector. Matrices are denoted in capital letters, bold-face, as $\boldsymbol{M}$. We denote the inner product of two vector $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ of equal length as $(\boldsymbol{v}_1, \boldsymbol{v}_2)$; that is, $(\boldsymbol{v}_1, \boldsymbol{v}_2) = \boldsymbol{v}_1^\intercal \cdot \boldsymbol{v}_2$. The norm of a vector is the Euclidean or ℓ_2 norm unless mentioned otherwise. Logarithms are assumed to be base 2 unless otherwise specified. We denote by $||\boldsymbol{M}||_\infty$ the maximum of the absolute value of the entries of $\boldsymbol{M}$. That is, for any $n \times m$ matrix $\boldsymbol{M}$,

$$||\boldsymbol{M}||_\infty = \max\{|M_{i,j}| \mid 1 \leq i \leq n, 1 \leq j \leq m\}.$$

For each real number x, we write $\lfloor x \rfloor$ for x rounded downwards to the closest integer, so

$$\lfloor x \rfloor = \max\{n \in \mathbb{Z} \mid n \leq x\}.$$

Moreover, $\lfloor x \rceil$ denotes the integer closest to x (rounded upwards in case of a tie), so $\lfloor x \rceil = \lfloor x + \frac{1}{2} \rfloor$. For each integer x and each integer positive integer p, we define $\langle x \rangle_p$ and $\{x\}_p$ as the integer in $(0, p-1]$ and in $(-p/2, p/2]$, respectively, that is equivalent to x modulo p. Note that for each integer x and each positive integer p, $\{x\}_p = \langle x + \lfloor \frac{p}{2} \rfloor \rangle_p - \lfloor \frac{p}{2} \rfloor$. We extend these notations to matrices by component wise application.

If χ is a probability distribution over a set S, then $x \xleftarrow{\$} \chi$ denotes sampling $x \in S$ according to χ. If S is a set, then $U(S)$ denotes the uniform distribution on S, and we denote sampling of x uniformly and randomly from S either by $x \xleftarrow{\$} U(S)$ or by $x \xleftarrow{\$} S$. For a matrix $\boldsymbol{M}$ or vector $\boldsymbol{v}$, the notation $\boldsymbol{M} \xleftarrow{\$} \chi$ and $\boldsymbol{v} \xleftarrow{\$} \chi$ signifies that the entries of $\boldsymbol{M}$ and $\boldsymbol{v}$ are drawn independently according to distribution χ. For a real $\sigma > 0$, we use D_σ to denote the discrete Gaussian distribution, which has support $\mathbb{Z}$ and assigns a probability proportional to $\exp(-\pi x^2/\sigma^2)$ to each $x \in \mathbb{Z}$. For $n, h \in \mathbb{Z}$ and $0 \leq h \leq n$, $\mathcal{HWT}_n(h)$ represents the set of vectors from $\{-1, 0, 1\}^n$ of Hamming weight h, i.e., each such vector contains exactly h non-zero entries.

3 The spKEX Key-Exchange Protocol

3.1 Underlying Problem

The security of our key-exchange scheme is based on the hardness of the sparse-ternary LWR problem:

Definition 1 (Sparse-Ternary LWR (sp-terLWR)) *Let n, p, q, h be positive integers.*
*The **search-sp-terLWR** is to find $s \in \{-1, 0, 1\}^n$ of Hamming weight h from arbitrarily many independent samples of the form $(\boldsymbol{a}_i, b_i = \lfloor \frac{p}{q} \cdot \langle \boldsymbol{a}, \boldsymbol{s} \rangle_q \rceil)$.*
*The **decision-sp-terLWR** problem is to distinguish $(\boldsymbol{a}, b = \lfloor \frac{p}{q} \cdot \langle \boldsymbol{a}, \boldsymbol{s} \rangle_q \rceil)$ from the uniform distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_p$ with non-negligible advantage, for a fixed $\boldsymbol{s} \in \{-1, 0, 1\}^n$ of Hamming weight h.*

According to [13], the hardness of this problem can be obtained from that of LWE with similar secret distributions since the reduction from LWE to LWR is independent of the secret's distribution [7]. Even if a formal reduction is not presented in [13], this statement is sound. Consider LWE with uniform noise on $V = (-\frac{q}{2p}, \frac{q}{2p}] \cap \mathbb{Z}$. For a given secret $\boldsymbol{s}$, the attacker observes $(\boldsymbol{a}, b) \in \mathbb{Z}_q^n \times Z_q$ where $b = \langle (\boldsymbol{a}, \boldsymbol{s}) \rangle_q + e \mod q$, and $e \xleftarrow{\$} V$. We thus have that $\frac{p}{q} \cdot b = \frac{p}{q} \cdot \langle (\boldsymbol{a}, \boldsymbol{s}) \rangle_q + \frac{p}{q} \cdot e \pmod p$. As $\frac{p}{q} \cdot e \in (-1/2, 1/2]$, we infer that if b is a multiple of $\frac{q}{p}$, then $\frac{p}{q} \cdot b = \lfloor \frac{p}{q} \cdot \langle (\boldsymbol{a}, \boldsymbol{s}) \rangle_q \rceil$. Hence, if $\boldsymbol{s}$ can be obtained from m LWR samples, then it can be obtained from $\frac{q}{p}m$ samples from LWE with uniform noise on $(-\frac{q}{2p}, \frac{q}{2p}]$. This is a similar reasoning as in [7, Thm.3]. Note that this reduction places no restrictions on the distribution from which the components of $\boldsymbol{s}$ can be chosen.

Results on the hardness of LWR are relevant for spKEX. It is shown in [7] that the search and decision LWR problems are at least as hard as the corresponding LWE problems when the number of problem instances m is bounded. This however requires an increase of the LWR problem dimension by a factor of $\log q$. In [4] this reduction is tightened, showing that an LWR problem is at least as hard as an LWE problem with uniform, bounded noise that supplies a greater number of samples but has the *same* dimension n and modulus q. This reduction is notable since it is the first to preserve the dimension n and modulus q between LWE (albeit with uniform, bounded noise) and LWR without resorting to noise-flooding [5] to re-normalize the rounding errors. In spKEX, an attacker has a bounded number of samples since it is a (ephemeral) key exchange in which parameters are refreshed. Thus, LWR and its variants are suitable choices for the construction of an ephemeral key exchange protocol, considering the bound on m.

3.2 Protocol Description

This section describes our key-exchange protocol between Alice and Bob that is sketched in Table 1. The protocol begins with the choice of the following public parameters:

- q: The large modulus of the LWR problem.
- p: The rounding modulus of the LWR problem, an integer multiple of 2^{B+b_h+1} and satisfying $p|q$.
- n: Dimension of the LWR problem, also representing the dimension of the public matrix $\boldsymbol{A} \in \mathbb{Z}_q^{n \times n}$.
- h: The Hamming weight of secrets.
- $\overline{n}, \overline{m}$: Number of LWR problem instances created by Alice and Bob, respectively.
- B: Number of bits extracted per coefficient of each party's raw key.
- b_h: The number of reconciliation bits per coefficient of each party's raw key.
- $\boldsymbol{A}_{\text{master}}$: The LWR master public matrix from which a fresh instance is generated for each key-exchange session by means of a random fresh permutation Π.
- A hash function `hash`.

Based on these public system parameters, Alice and Bob use the spKEX protocol detailed in Table 1 to agree on a shared secret $\boldsymbol{K}$, and its hash – the final symmetric key μ. This protocol is similar to [8] with a number of key differences.

Table 1: spKEX Protocol

Alice		**Bob**
1. Choose a random permutation Π.		
2. $\boldsymbol{A} = \Pi(\boldsymbol{A}_{\text{master}}) \in \mathbb{Z}_q^{n \times n}$.		
3. $\boldsymbol{S}_a \in \{0, \pm 1\}^{n \times \overline{n}} \leftarrow \mathcal{HWT}_n(h)^{\overline{n}}$.		
4. $\boldsymbol{P}_a = \langle \lfloor (\frac{p}{q}) \langle \boldsymbol{A} \cdot \boldsymbol{S}_a \rangle_q \rceil \rangle_p \in \mathbb{Z}_p^{n \times \overline{n}}$.		
	$\xrightarrow{5.\ \Pi,\ \boldsymbol{P}_a}$	
		6. $\boldsymbol{A} = \Pi(\boldsymbol{A}_{\text{master}}) \in \mathbb{Z}_q^{n \times n}$.
		7. $\boldsymbol{S}_b \in \{0, \pm 1\}^{n \times \overline{m}} \leftarrow \mathcal{HWT}_n(h)^{\overline{m}}$.
		8. $\boldsymbol{P}_b = \langle \lfloor (\frac{p}{q}) \langle \boldsymbol{S}_b^\intercal \cdot \boldsymbol{A} \rangle_q \rceil \rangle_p \in \mathbb{Z}_p^{\overline{m} \times n}$.
		9. $\boldsymbol{K}_b = \langle \boldsymbol{S}_b^\intercal \cdot \boldsymbol{P}_a \rangle_p \in \mathbb{Z}_p^{\overline{m} \times \overline{n}}$.
		10. $\boldsymbol{H} = \lfloor \frac{2^{B+b_h}}{p} \langle \boldsymbol{K}_b \rangle_{p/2^B} \rfloor \in \mathbb{Z}_{2^{b_h}}^{\overline{m} \times \overline{n}}$.
	$\xleftarrow{11.\ \boldsymbol{P}_b,\ \boldsymbol{H}}$	
12. $\boldsymbol{K}_a = \langle \boldsymbol{P}_b \cdot \boldsymbol{S}_a \rangle_p \in \mathbb{Z}_p^{\overline{m} \times \overline{n}}$.		
13. $\boldsymbol{K} = \langle \lfloor \frac{2^B}{p} \boldsymbol{K}_a - \frac{1}{2^{b_h}} \boldsymbol{H} - \frac{1}{2^{b_h+1}} + \frac{1}{2} \rfloor \rangle_{2^B}$		13. $\boldsymbol{K} = \lfloor \frac{2^B}{p} \boldsymbol{K}_b \rfloor$
14. $\mu = \texttt{hash}(\boldsymbol{K})$		14. $\mu = \texttt{hash}(\boldsymbol{K})$

First, the public matrix $\boldsymbol{A} \in \mathbb{Z}_q^{n \times n}$ is generated from a fixed, master public matrix $\boldsymbol{A}_{\text{master}}$ and a random permutation Π as $\boldsymbol{A} = \Pi(\boldsymbol{A}_{\text{master}})$. This allows efficient computation every time of a fresh $\boldsymbol{A}$, i.e., this approach is a trade-off between the performance advantages of a fixed matrix $\boldsymbol{A}$ [19] and the security considerations of having a fresh random $\boldsymbol{A}$ to prevent pre-computation or backdoor-like attacks [8], e.g., lattice-reduction.

Second, spKEX uses sparse ternary secrets $\boldsymbol{S}_a$, $\boldsymbol{S}_b$ of which each column has Hamming weight h so that operations are more efficient. Because $\boldsymbol{S}_a$ and $\boldsymbol{S}_b$ are fresh for each session, spKEX achieves forward secrecy. *Third*, the public-keys are computed by applying rounding. This avoids expensive sampling of Gaussian noise, and reduces bandwidth requirements to exchange public-keys. Furthermore, the "raw keys" $\boldsymbol{K}_a$ and $\boldsymbol{K}_b$ can be obtained by performing computations modulo the (smaller) modulus p.

Fourth, to achieve an *exact* key agreement we use the reconciliation mechanism from [24] that allows for larger differences between the raw keys than the mechanism from [19].

4 Analysis of Operational Correctness

In this section, we analyze the operational correctness of spKEX. That is, we show that for a proper choice of parameters, Alice and Bob agree on a common key with a high probability.

4.1 Difference between Raw Keys

We define the $n \times \overline{n}$ matrix $\boldsymbol{E}_a$ and the $\overline{m} \times n$ matrix $\boldsymbol{E}_b$ as

$$\boldsymbol{E}_a = \boldsymbol{P}_a - \frac{p}{q}\langle \boldsymbol{A} \cdot \boldsymbol{S}_a \rangle_q = \lfloor \frac{p}{q}\langle \boldsymbol{A} \cdot \boldsymbol{S}_a \rangle_q \rceil - \frac{p}{q}\langle \boldsymbol{A} \cdot \boldsymbol{S}_a \rangle_q \tag{1}$$

$$\boldsymbol{E}_b = \boldsymbol{P}_b - \frac{p}{q}\langle \boldsymbol{S}_b \cdot \boldsymbol{A} \rangle_q = \lfloor \frac{p}{q}\langle \boldsymbol{S}_b \cdot \boldsymbol{A} \rangle_q \rceil - \frac{p}{q}\langle \boldsymbol{S}_b \cdot \boldsymbol{A} \rangle_q \tag{2}$$

The definition of the $\lfloor \cdot \rceil$ function implies that

$$||\boldsymbol{E}_a||_\infty \leq \frac{1}{2} \text{ and } ||\boldsymbol{E}_b||_\infty \leq \frac{1}{2}. \tag{3}$$

In fact, all entries of $\boldsymbol{E}_a$ and $\boldsymbol{E}_b$ are integer multiples of $\frac{p}{q}$.
From the definitions of $\boldsymbol{K}_a, \boldsymbol{K}_b, \boldsymbol{E}_a$ and $\boldsymbol{E}_b$, it follows that

$$\begin{aligned} \boldsymbol{K}_a &\equiv \boldsymbol{P}_b \cdot \boldsymbol{S}_a \\ &= \{\frac{p}{q} \cdot (\boldsymbol{S}_b^\intercal \cdot \boldsymbol{A} + q \cdot \boldsymbol{U}_1) + \boldsymbol{E}_b\} \cdot \boldsymbol{S}_a \qquad \text{where } \boldsymbol{U}_1 \in \mathbb{Z}^{\overline{m} \times n} \\ &\equiv \frac{p}{q}(\boldsymbol{S}_b^\intercal \cdot \boldsymbol{A} \cdot \boldsymbol{S}_a) + \boldsymbol{E}_b \cdot \boldsymbol{S}_a \pmod p, \end{aligned} \tag{4}$$

and similarly

$$\boldsymbol{K}_b \equiv \frac{p}{q}(\boldsymbol{S}_b^\intercal \cdot \boldsymbol{A} \cdot \boldsymbol{S}_a) + \boldsymbol{S}_b^\intercal \cdot \boldsymbol{E}_a \pmod p. \tag{5}$$

As a consequence,

$$\boldsymbol{K}_a - \boldsymbol{K}_b \equiv \boldsymbol{E}_b \cdot \boldsymbol{S}_a - \boldsymbol{S}_b^\intercal \cdot \boldsymbol{E}_a \pmod p. \tag{6}$$

4.2 Reconciliation Condition

As shown in [24], Alice and Bob agree on entry i, j of K if

$$\{(\boldsymbol{K}_a - \boldsymbol{K}_b)_{i,j}\}_p \leq \Delta := \frac{p}{2^{B+1}} - \frac{p}{2^{B+b_h+1}}. \tag{7}$$

We invoke (6) to analyze $\{\boldsymbol{K}_a - \boldsymbol{K}_b\}_p$. Each column of $\boldsymbol{S}_a$ and each row of $\boldsymbol{S}_b^\intercal$ has h non-zero entries, and each non-zero entry equals 1 or -1. Moreover, all entries of $\boldsymbol{E}_a$ and $\boldsymbol{E}_b$ are in the interval $(-1/2, 1/2]$. As a consequence, all entries of $\boldsymbol{E}_b \cdot \boldsymbol{S}_a$ and $\boldsymbol{S}_b^\intercal \cdot \boldsymbol{E}_a$ are in $(-\frac{1}{2}h, \frac{1}{2}h]$, and so $||\boldsymbol{E}_b \cdot \boldsymbol{S}_a - \boldsymbol{S}_b \cdot \boldsymbol{E}_a||_\infty \leq h$. Hence, if $h \leq \Delta$, then Alice and Bob always agree on a common key.

We wish to have a more relaxed condition that ensures that Alice and Bob agree on a common key *with high probability*. In Appendix A we show the following result. If we model the entries of each row of $\boldsymbol{E}_a$ and each row of $\boldsymbol{E}_b$ as being independently and uniformly distributed on $(-1/2, 1/2]$, then

$$\Pr(\text{Alice and Bob do not agree on a common key}) \leq 4 \cdot \overline{m} \cdot \overline{n} \cdot 2^{h \cdot f((\Delta+1)/h)}, \tag{8}$$

$$\text{where } f(a) = \frac{1}{\ln 2} \min\{\ln(\frac{e^t - e^{-t}}{2t}) - at \mid t > 0\}. \tag{9}$$

We have not been able to derive a closed formula for f, but the minimization can easily be performed numerically. We defer details to Appendix A.

5 Security Analysis

Our security analysis considers four aspects, namely, *a*) attacks using lattice reduction and parameter estimations, *b*) specialized attacks exploiting sparse ternary secrets, *c*) biases in the computed keys, and *d*) security considerations regarding the computation of the public parameter $\boldsymbol{A}$.

5.1 Attacks based on Lattice Reduction

The use of ephemeral secrets and refreshed public parameters in the key-exchange strictly limits the number of samples available to an attacker, and restricts relevant attacks to those based on lattice reduction – the *primal* or decoding attack [3] and the *dual* or distinguishing attack [1].

The attacker can use the public keys $\boldsymbol{P_a} = \langle \lfloor \frac{p}{q} \langle \boldsymbol{A}\boldsymbol{S}_a \rangle_q \rceil \rangle_p$ and $\boldsymbol{P_b} = \langle \lfloor \frac{p}{q} \langle \boldsymbol{S}_b^\intercal \boldsymbol{A} \rangle_q \rceil \rangle_p$ to obtain information on $\boldsymbol{S}_a$ and $\boldsymbol{S}_b$, respectively. We work out how this is done for $\boldsymbol{S}_a$, the case for $\boldsymbol{S}_b$ being similar. Let $1 \leq i \leq n$ and $1 \leq j \leq \overline{n}$. If we denote the i-th row of $\boldsymbol{A}$ by $\boldsymbol{a}_i^\intercal$ and the j-th column of $\boldsymbol{S}_a$ by $\boldsymbol{s}_j$, then

$$(\boldsymbol{P}_a)_{i,j} = \langle \lfloor \frac{p}{q} \langle (\boldsymbol{a}_i, \boldsymbol{s}_j) \rangle_q \rceil \rangle_p.$$

By the definition of the rounding function $\lfloor \cdot \rceil$, we have that

$$(\boldsymbol{P}_a)_{i,j} \equiv \frac{p}{q} \langle (\boldsymbol{a}_i, \boldsymbol{s}_j) \rangle_q + e_{i,j} \pmod{p} \text{ with } e_{i,j} \in (-1/2, 1/2].$$

As $\langle (\boldsymbol{a}_i, \boldsymbol{s}_j) \rangle_q = (\boldsymbol{a}_i, \boldsymbol{s}_j) + \lambda q$ for some integer λ, we infer that

$$\frac{q}{p}(\boldsymbol{P}_a)_{i,j} \equiv (\boldsymbol{a}_i, \boldsymbol{s}_j) + \frac{q}{p} e_{i,j} \pmod{q}. \tag{10}$$

So we have n equations involving $\boldsymbol{s}_j$. Unlike conventional LWE, the errors $\frac{q}{p} e_{i,j}$ reside in a bounded interval, namely $(-\frac{q}{2p}, \frac{q}{2p}]$. In what follows, we will only consider the case that p divides q.

Primal Attack: In (10), we write $\boldsymbol{s}$ for $\boldsymbol{s}_j$, denote by $\boldsymbol{b}$ the vector of length m with j-th component $\frac{q}{p}(\boldsymbol{P}_a)_{i,j}$, and with $\boldsymbol{A}_m$ the matrix consisting of the m top rows of $\boldsymbol{A}$. We then have, for $\boldsymbol{e} \in (-\frac{q}{2p}, \frac{q}{2p}]^m$

$$\boldsymbol{b} \equiv \boldsymbol{A}_m \boldsymbol{s} + \boldsymbol{e} \pmod{q} \tag{11}$$

so that $\boldsymbol{v} = (\boldsymbol{s}^\intercal, \boldsymbol{e}^\intercal, 1)^\intercal$ is in the lattice Λ defined as

$$\Lambda = \{ \boldsymbol{x} \in \mathbb{Z}^{n+m+1} : (\boldsymbol{A}_m | \boldsymbol{I}_m | -\boldsymbol{b}) \boldsymbol{x} = \boldsymbol{0} \pmod{q} \} \tag{12}$$

of dimension $d = n + m + 1$ and volume q^m [8, 2]. The attacker then searches for a short vector in Λ which hopefully equals $\boldsymbol{v}$, thus enabling him to recover the secret $\boldsymbol{s}$.

Lattice Rescaling: The vector $\boldsymbol{v} = (\boldsymbol{s}, \boldsymbol{e}, 1)$ is unbalanced due to $\|\boldsymbol{s}\| \ll \|\boldsymbol{e}\|$. For exploiting this fact, a rescaling technique originally due to [3], and analyzed further in [13] and [1] is applied in order to balance the expected norms of the components of $\boldsymbol{s}$ and of $\boldsymbol{e}$. As $\boldsymbol{s}$ is ternary and has h non-zero entries, the expected norm of each entry equals $\frac{h}{n}$. Approximating each entry of $\boldsymbol{e}$ as being drawn uniformly from $[-\frac{q}{2p}, \frac{q}{2p})$, its expected norm equals $\frac{1}{12}\frac{q^2}{p^2}$. We therefore choose the scaling factor ω satisfying

$$\omega = \sigma' \cdot \sqrt{\frac{n}{h}}, \text{ where } \sigma'^2 = \frac{q^2}{12p^2} \tag{13}$$

Multiplying the first n columns of Λ's basis (see Eq. 12) with ω yields the following weighted or rescaled lattice,

$$\Lambda_\omega = \{\boldsymbol{x} \in \mathbb{Z}^{n+m+1} : \big((\omega \cdot \boldsymbol{A}^\intercal)|\boldsymbol{I}_m|-\boldsymbol{b}\big)\boldsymbol{x} = \boldsymbol{0} \pmod{\mathrm{q}}\} \tag{14}$$

in which the attacker then searches for the shortest vector, that he hopes to be equal to $\boldsymbol{v}_\omega = (\omega \cdot \boldsymbol{s}^\intercal, \boldsymbol{e}^\intercal, 1)^\intercal$. This search is typically done by using a lattice reduction algorithm to obtain a reduced basis of the lattice, the first vector of which will be the shortest of that basis due to a common heuristic.

If the quality of the lattice reduction is good enough, the reduced basis will contain $\boldsymbol{v}_\omega$. The attack success condition is as in [2] assuming that BKZ [11, 22] with block-size b is used as the lattice reduction algorithm. The vector $\boldsymbol{v}_\omega$ will be detected if its projection $\widetilde{\boldsymbol{v}}_b$ onto the vector space of the last b Gram-Schmidt vectors of Λ is shorter than the $(d-b)^{th}$ Gram-Schmidt vector $\widetilde{\boldsymbol{b}}_{d-b}$, where d is the dimension of Λ [2, Sec. 6.3],[8]. The condition that must be satisfied for the primal attack to succeed is therefore:

$$\begin{aligned} & \|\widetilde{\boldsymbol{v}}_b\| < \|\widetilde{\boldsymbol{b}}_{d-b}\| \\ \text{i.e.,} \quad & \|\widetilde{\boldsymbol{v}}_b\| < \delta^{2b-d-1} \cdot (\mathrm{Vol}(\Lambda))^{\frac{1}{d}} \\ \text{where,} \quad & \delta = ((\pi b)^{\frac{1}{b}} \cdot \frac{b}{2\pi\mathbf{e}})^{\frac{1}{2(b-1)}} \end{aligned} \tag{15}$$

The attack success condition (15) yields the following *security* condition that must be satisfied by the key-exchange parameters in order for spKEX to remain secure against the primal attack:

$$\begin{aligned} & \sqrt{(\omega^2 \cdot h + \sigma'^2 m) \cdot \frac{b}{n+m}} \geq \delta^{2b-d-1} \cdot (q^{d-n-1}\omega^n)^{\frac{1}{d}} \\ \text{where,} \quad & \delta = ((\pi b)^{\frac{1}{b}} \cdot \frac{b}{2\pi\mathbf{e}})^{\frac{1}{2(b-1)}}, \\ & \omega = \sigma' \cdot \sqrt{n/h}, \\ & \sigma' = (q/2\sqrt{3}p), \\ \text{and} \quad & d = m + n + 1. \end{aligned} \tag{16}$$

The running time of the BKZ lattice reduction is $b \cdot 2^{cb}$ CPU clock cycles [8], where c is an experimental constant, and depends on the underlying sieving

algorithm used. In our later analysis, we will use three values for c, like in [8, Sec 4.1]. Classically, the best known constant is $c_C = \log_2 \sqrt{3/2} \approx 0.292$, which is provided by the sieving algorithm from [6]. In the quantum context, the best constant equals $c_Q = \log_2 \sqrt{13/9} \approx 0.265$, [17, Sec. 14.2.10]. Finally, we use the "paranoid" value $c_P = \log_2 \sqrt{4/3} \approx 0.2075$, see [8, Sec 4.1].

Dual Attack: The dual attack against LWE attempts to find a short vector $(\boldsymbol{v}, \boldsymbol{w}) \in \mathbb{Z}^m \times \mathbb{Z}^n$ in the dual lattice

$$\Lambda^* = \{(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{Z}^m \times \mathbb{Z}^n : \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y} \pmod q\}. \tag{17}$$

This vector is used to construct the distinguisher $z = \{(\boldsymbol{v}, \boldsymbol{b})\}_q$. If $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e} \pmod q$, then $z = \{(\boldsymbol{v}, \boldsymbol{b})\}_q \equiv \{(\boldsymbol{w}, \boldsymbol{s}) + (\boldsymbol{v}, \boldsymbol{e})\}_q$, and z is therefore small and distributed as a Gaussian.

For an LWR distribution with uniform rounding error $\boldsymbol{e}'$ and corresponding variance $\sigma'^2 = q^2/12p^2$, the distinguisher checks whether $z = \{(\boldsymbol{v}, \boldsymbol{b})\}_q$ is small. For a non-LWR distribution, z is distributed uniformly modulo q. For an LWR distribution, z's distribution approaches a Gaussian distribution of zero mean and variance $\|\boldsymbol{v}\|^2 \cdot \sigma'^2$ as the lengths of the vectors $\boldsymbol{v}$ and $\boldsymbol{e}'$ increase, due to the Central limit theorem. The maximal statistical distance between this Gaussian distribution and the uniform distribution modulo q is bounded by $\epsilon \approx (1/\sqrt{2}) \exp(-2\pi^2(\|\boldsymbol{v}\| \cdot \sigma'/q)^2)$, a more detailed derivation of this result can be found in Appendix B. The attacker uses the BKZ algorithm with block-size b that outputs a short vector of length $\delta^{d-1} \cdot \mathrm{Vol}(\Lambda^*)^{1/d}$, where $d = m + n$ is the dimension of the dual lattice Λ^*, and its volume is $\mathrm{Vol}(\Lambda^*) = q^n$.

As the key is hashed, a small advantage ϵ is not sufficient. As explained in [2], assuming BKZ with block size b, the attack must be repeated at least $R = \max(1, 1/2^{0.2075b} \cdot \epsilon^2)$ times. The cost of using the dual attack to distinguish an LWR distribution from uniform, employing BKZ with block size b and root-Hermite factor δ, using m samples, where the LWR distribution is generated from an LWR problem instance of dimension n, large modulus q, rounding modulus p is:

$$\begin{aligned}
\mathrm{Cost}_{\text{Dual attack}} &= (b \cdot 2^{cb}) \cdot \max(1, 1/(\epsilon^2 \cdot 2^{0.2075 \cdot b})), \\
\text{where,} \qquad \epsilon &= \frac{1}{\sqrt{2}} \cdot \mathbf{e}^{-2\pi^2 \left(\frac{\|\boldsymbol{v}\| \cdot \sigma'}{q^2}\right)^2}, \\
\|\boldsymbol{v}\| &= \delta^{m+n-1} \cdot \left(\frac{q}{\sigma' \cdot \sqrt{n/h}}\right)^{1/(m+n)}, \\
\delta &= ((\pi b)^{\frac{1}{b}} \cdot \frac{b}{2\pi\mathbf{e}})^{\frac{1}{2(b-1)}} \\
\text{and} \qquad \sigma' &= (q/2\sqrt{3}p).
\end{aligned} \tag{18}$$

The first term in the cost above, i.e., $(b \cdot 2^{cb})$ is the cost of running BKZ lattice reduction with block-size b, where the constant c is the BKZ sieving exponent. Finally, note that the cost in Eq. 18 also accounts for the fact that the dual

attack can be adapted against the sparse-ternary LWR problem by rescaling the dual lattice using an appropriate scaling factor ω [1] (similar to lattice rescaling in the primal attack), resulting in the following rescaled dual lattice:

$$\Lambda^*_\omega = \{(\boldsymbol{x}, \boldsymbol{y}/\omega) \in \mathbb{Z}^m \times \left(\frac{1}{\omega} \cdot \mathbb{Z}^n\right) : \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y} \pmod q\} \tag{19}$$

This scales the volume of the dual lattice from q^n to $(q/\omega)^n$, which correspondingly scales the norm $\|\boldsymbol{v}\|$ of the short vector $\boldsymbol{v}$ and hence the distinguishing advantage.

5.2 Choice of Hamming Weight

Albrecht *et al* [1] describe an attack towards LWE and LWR with sparse secrets $\boldsymbol{s}$. Since most rows of the public matrix $\boldsymbol{A}$ become irrelevant while calculating the product $\boldsymbol{A} \cdot \boldsymbol{s}$ for such secrets, Albrecht's attack ignores a random $k \leq n$ number of components of $\boldsymbol{s}$ and brings down the lattice dimension (and hence attack cost) during lattice reduction. Non-zero components in $\boldsymbol{s}$ may be incorrectly ignored, and the attack must therefore be repeated $1/(1-\frac{h}{n})^k$ times so that with high likelihood, at least one of the times indeed all k ignored components of $\boldsymbol{s}$ indeed equal zero. [1] also proposes an extension of this attack that attempts to reduce the above inaccuracy, however this extension involves significantly more repetitions, hence we do not consider it. For the standard Albrecht attack, we estimate the cost (in bits) for a given Hamming weight $h \leq n$, as the number of repetitions each low cost attack is performed times the cost of the low-cost attack on a lattice of smaller dimension $n-k$:

$$\frac{\binom{n}{k}}{\binom{n-h}{k}} \times \text{Cost}_{\text{Lattice Reduction}}(n, k, h)$$

Here $\text{Cost}_{\text{Lattice Reduction}}(n, k, h)$ is defined as

$$\min\{b \cdot 2^{cb} \mid b \in \mathbb{N}, \text{ there exists } m \in \mathbb{N} \text{ such that } m \leq n \text{ and (20) is satisfied.}\}$$

$$\begin{aligned} \sqrt{(\omega^2 \cdot h + \sigma'^2 m) \cdot \frac{b}{n+m}} &< \delta^{2b-d-1} \cdot (q^{d-(n-k)-1}\omega^{n-k})^{\frac{1}{d}} \\ \text{where,} \quad \delta &= ((\pi b)^{\frac{1}{b}} \cdot \frac{b}{2\pi\mathbf{e}})^{\frac{1}{2(b-1)}}, \\ \omega &= \sigma' \cdot \sqrt{(n-k)/h}, \\ \sigma' &= (q/2\sqrt{3}p), \\ \text{and} \quad d &= m+n-k+1. \end{aligned} \tag{20}$$

The term $b \cdot 2^{cb}$ represents the cost of running BKZ lattice reduction with blocksize b. The attack runs on a LWE problem of dimension $n-k$ with $m \leq n$ samples. Condition 20, which is essentially Condition 15, ensures that such an attack has chances of succeeding. Note that although the above applies to the primal attack, a similar analysis is possible for the dual attack, in which

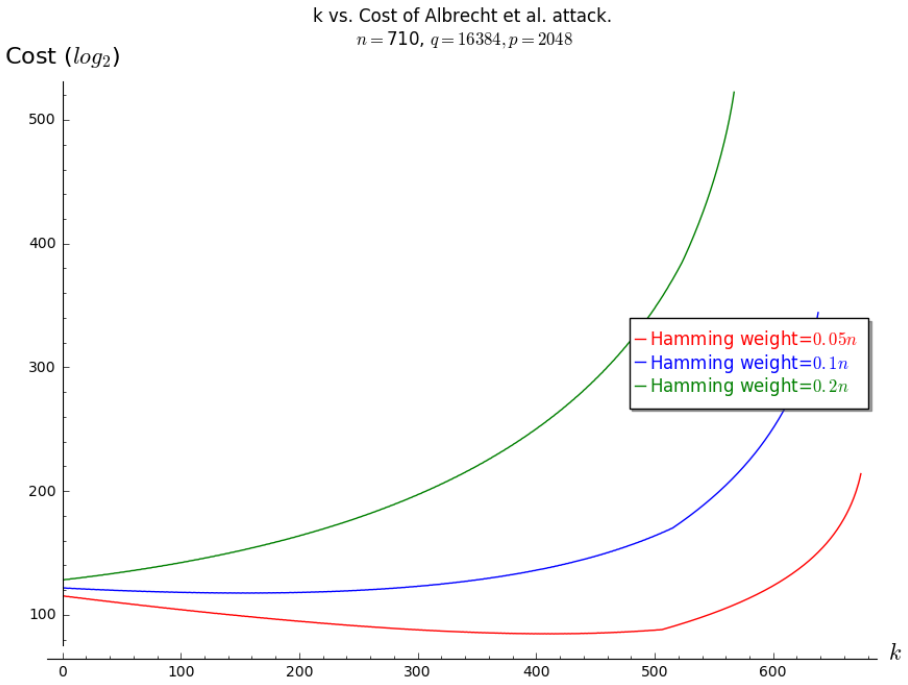

Figure 1: Cost of Albrecht *et al*'s [1] specialized (dual) attack against sparse-ternary LWR with Hamming weight $0.05n$, $0.1n$ and $0.2n$. Lowest cost for the two smaller Hamming weights is 2^{85} and 2^{118}, for $k = 412$ and $k = 151$ ignored secret components respectively. Lowest cost for Hamming weight $0.2n$ is 2^{128}, the same as non-sparse ternary LWR. The attack thus has no advantage against this Hamming weight.

case $\text{Cost}_{\text{Lattice Reduction}}(n, k, h)$ is calculated as in Eq. 18, with the parameter n replaced by $n - k$.

This specialized attack only gives an advantage if an attacker is able to choose a k for which the total attack cost is less than a standard lattice-reduction attack on a lattice of dimension n. Figure 1 depicts our estimations of how this specialized attack (cost) scales with k for different Hamming weights, considering the primal attack and assuming the usage of BKZ. Our experiments considered values of n in the range $510 \leq n \leq 910$. The attack cost for sparse-ternary LWR is estimated to be equivalent to the cost of attacking ternary LWR, for a Hamming weight of $h = 0.2n$, beyond which this specialized attack's cost increases with h, quickly becoming unfeasible.

Furthermore, to ensure that an exhaustive, brute-force search of each secret-key vector in the secret-key using Grover's quantum search algorithm [16] has

a cost of at least λ (in bits), the chosen Hamming weight should satisfy:

$$\sqrt{\binom{n}{h} \cdot 2^h} > 2^\lambda \tag{21}$$

Note that for a typical security level of $\lambda = 128$, a dimension of at least $n = 512$ would be secure against Grover's quantum search, for any Hamming weight h that is at least $0.1n$.

5.3 Unbiased Keys

spKEX's keys are unbiased. First, p is a divisor of q so that the rounding does not introduce any bias in the public-keys, as indicated in [5]. Moreover, as 2^B divides p, there is no bias in $\boldsymbol{K}$ either. As p is an integer multiple of 2^{B+b_h+1}, the helper data does not leak information on $\boldsymbol{K}$.

5.4 Choice and Computation of the Public Parameter $\boldsymbol{A}$

A fresh $\boldsymbol{A}$ is computed in each protocol interaction (see Section 3) by permuting a fixed, public $\boldsymbol{A}_{\text{master}}$ as $\boldsymbol{A} = \Pi(\boldsymbol{A}_{\text{master}})$. This is a trade-off between the two approaches from literature. Using a fixed $\boldsymbol{A}$ (e.g., [19]) increases efficiency, but opens up the possibility of pre-computation or backdoor-like attacks. Thus, [8] uses a fresh, random $\boldsymbol{A}$ for *every* key-exchange session computed from a fresh random seed and a pseudo-random function. However, this causes a significant performance overhead [8], in particular, if a hardware accelerator for the pseudo-random function is not available. Such an approach is especially problematic in client-server scenarios where a single server engages in multiple key-exchange handshakes with multiple clients, and must generate $\boldsymbol{A}$ for each client.

Security-wise, our approach ensures a sufficiently high randomness and freshness of $\boldsymbol{A}$ preventing pre-computation attacks: a pre-computation attack based on lattice reduction is not feasible since the permutation would obtain a completely new lattice so that an attacker would be forced to perform lattice reduction on it. Performance-wise, our approach has a low overhead since permuting $\boldsymbol{A}_{\text{master}}$ is computationally inexpensive and the random permutation can be efficiently encoded in a short random seed.

An example of a suitable permutation $\Pi()$ randomly selects $n_r \leq n$ rows in $\boldsymbol{A}_{\text{master}}$ and shifts each of the selected rows by a random amount. This leads to a total of:

$$\binom{n}{n_r} \cdot n^{n_r} \tag{22}$$

possible permutations of $\boldsymbol{A}_{\text{master}}$. Considering a typical cryptographically large value of n, e.g., 512, and $n_r = n$, this leads to at least 2^{256} possible random permutations $\boldsymbol{A}$.

5.5 Parameter Selection

Similar to [13] and [8], three sets of parameters are provided, namely classical, quantum and paranoid parameters. The parameters are chosen in such a way that the computational costs are at at least 2^{128} CPU cycles when running BKZ. The cost of running BKZ with block size b is estimated as $b \cdot 2^{cb}$, where for the classical, quantum and paranoid parameters we take $c = c_C = 0.292$, $c = c_Q = 0.265$ and $c_P = 0.2075$, respectively, see [8, Sec. 4.1]. For given parameters p, q, n, we minimize the attack cost over the block size b and the number of samples m that an attacker can use, such that the condition for successful attack is satisfied, for the primal, dual and specialized attack.

In all cases, we took the Hamming weight of the secret equal to $0.2n$. The chosen parameters are further optimized by scanning the values of b_h, p, and q that allow achieving the target security level for a failure probability lesser than 2^{-40} and minimize bandwidth requirements. Finally, it is required that the length of the generated key, which is $\overline{n} \cdot \overline{m} \cdot B$, is at least equal to the desired key length, that is, at least 128 for the classical parameters, and at least 256 for the quantum and paranoid parameters. In order to avoid brute-force search over keys, we made sure that (21) is satisfied, with $\lambda = 128$ for the classical parameters, and $\lambda = 256$ for the quantum and paranoid parameters.

Table 2 shows the three sets of configuration parameters. We further compare them with related key-exchange and public-key encryption schemes based on the LWE and/or LWR problems, namely Frodo [8] and Lizard [13]. Note that the authors of [8] calculate the security-levels of Frodo [8] considering that a discretized Gaussian error distribution is used, which might have a lower security than using an ideal Gaussian distribution. Lizard [13] does not disclose the optimal values of the BKZ block-size b and LWE or LWR samples m considered for calculating the security-levels mentioned in Table 2.

6 Performance Analysis

6.1 Bandwidth performance

Table 3 summarizes the proposed parameter sets introduced in Section 5.5. As mentioned before, we use a Hamming weight of $0.2n$, based on the reasoning in Section 5.2. The variance of noise introduced by LWR is computed as $\sigma^2 = \frac{q^2}{12p^2}$. The bandwidth requirements (in bits) are calculated as:

$$n \cdot (\overline{n} + \overline{m}) \cdot \lceil \log p \rceil + \overline{n} \cdot \overline{m} \cdot b_h \tag{23}$$

Table 4 compares spKEX with existing key-exchange, key-encapsulation and public-key encryption schemes based on the LWE and LWR problems. The use of rounding has the highest impact on the public-key sizes and hence *reduces the bandwidth* requirements. The improved reconciliation method from [24] allows us a win-win-win scenario of *tolerating higher error* (as compared to Frodo), which results in (slightly) *higher security level*, and yet achieving a

Table 2: spKEX parameters and security level compared with Frodo [8] and Lizard [13]. Secrets have Hamming weight $0.2n$.

Case	b	m	n	q	p	σ	**Security Level** (bits)
spKEX (P=Primal attack, D=Dual attack)							
Classical	417 (P), 408 (D)	385 (P), 395 (D)	570 (LWR)	2^{12}	2^{9}	2.309 (LWR)	131 (P), 128 (D) (LWR)
Quantum	458 (P), 450 (D)	506 (P), 512 (D)	710 (LWR)	2^{14}	2^{11}	2.309 (LWR)	130 (P), 128 (D) (LWR)
Paranoid	584 (P), 575 (D)	603 (P), 616 (D)	867 (LWR)	2^{14}	2^{11}	2.309 (LWR)	130 (P), 128 (D) (LWR)
Frodo (P=Primal attack, D=Dual attack)							
Classical	442 (P), 438 (D)	549 (P), 544 (D)	592 (LWE)	2^{12}	-	1.0 (LWE)	132 (P), 130 (D) (LWE)
Quantum	489 (P), 485 (D)	716 (P), 737 (D)	752 (LWE)	2^{15}	-	1.323 (LWE)	132 (P), 130 (D) (LWE)
Paranoid	581 (P), 576 (D)	793 (P), 833 (D)	864 (LWE)	2^{15}	-	1.323 (LWE)	129 (P), 128 (D) (LWE)
Lizard (Dual)							
Classical	?	?	544 (LWE), 840 (LWR)	2^{10}	2^{8}	5.988 (LWE), 1.155 (LWR)	128 (LWE), 132 (LWR)
Quantum	?	?	608 (LWE), 960 (LWR)	2^{10}	2^{8}	5.626 (LWE), 1.155 (LWR)	128 (LWE), 130 (LWR)
Paranoid	?	?	736 (LWE), 1450 (LWR)	2^{10}	2^{8}	6.4 (LWE), 1.155 (LWR)	128 (LWE), 129 (LWR)

lower failure probability than Frodo (see Table 3). The improved error tolerance allowed us to further decrease p, thus dropping more bits from the public-keys, further *reducing their size and bandwidth* requirements. The usage of sparse ternary secrets leads to a higher value of n and thus increases bandwidth needs; however, as shown in the next section it has a positive bearing on the error, and on implementation.

6.2 Computational Performance

Three features of our key exchange improve the computational performance compared with related work.

Table 3: Proposed parameter sets for spKEX along with performance

Security level	**Key size**	B, b_h	**Failure**	$\log_2 q$	n	$\log_2 p$	σ	**Bandwidth**
Classical ($c = 0.292$)	128 bits	2, 2	2^{-40}	12	570	9	2.309	10.04 KB
Quantum ($c = 0.265$)	256 bits	4, 3	2^{-44}	14	710	11	2.309	15.3 KB
Paranoid ($c = 0.2075$)	256 bits	4, 4	2^{-40}	14	867	11	2.309	18.66 KB

Table 4: Comparison of spKEX with other schemes for Quantum security parameters

Scheme	n, q, p, σ	Bandwidth	Failure
Frodo [8]	$752, 2^{15}, -, 1.323$	22.06 KB	2^{-39}
spLWE-KEM [12]	$565, 520, -, 5.0$	158 KB	2^{-7}
Lizard-PKE [13] (128-bit classical IND-CPA, 32-bit plaintext)	960 (LWR), 608 (LWE), $2^{10}, 2^{8}, 5.62$	32.88 KB	?
spKEX	$710, 2^{14}, 2^{11}, 2.309$	15.3 KB	2^{-44}

First, the usage of a permutation to compute $\boldsymbol{A}$ involves lower computational requirements. In particular, it requires the computation of $n \lceil \log n \rceil$ bits compared with $n^2 \lceil \log q \rceil /2$ bits when all elements in A are computed randomly resulting in an improvement of up to $\frac{n \lceil \log q \rceil}{2 \lceil \log n \rceil}$ times. To illustrate this, we implement spKEX using the row-wise permutation detailed in Section 5.4 and two variants of it, one in which $\boldsymbol{A}$ is static and one in which $\boldsymbol{A}$ is generated from a seed by means of a SHA3-based pseudo-random function. The row-wise permutation is computed from a 256-bit random seed by applying the same SHA3-based pseudo-random function on this seed to compute the shifts of each of the rows. The total key exchange for spKEX is then performed in 3.657 msec. (fixed $\boldsymbol{A}$), 7.774 msec. (permuted $\boldsymbol{A}$), and 77.571 msec. (randomly generated $\boldsymbol{A}$) respectively.

Table 5 details the performance of spKEX on a 64-bit Intel Core i5-4210U CPU @ 1.70 GHz. Performance numbers illustrate the advantages of the remaining design features of spKEX: small rounding modulus p and sparse ternary secrets. The usage of small rounding modulus implies smaller operations and avoids the sampling of Gaussian noise, this speeds up both the computation of the public-keys and the final shared secret. The usage of sparse ternary secrets has implementation advantages, simplifying multiplications of matrix components.

Table 5: CPU performance of spKEX for quantum-security level

Key exchange phase	Time (msec.)
Alice's secret-key	0.162
Alice public-key	1.871
Bob's secret-key	0.161
Bob's public-key	1.359
Bob's shared key	0.017
Alice's shared key	0.017
Total handshake	3.587

7 Conclusions & Future Work

We have presented spKEX, a key-exchange scheme based on the sparse ternary secrets and Learning with Rounding. This combination seems to provide the best performance when bandwidth and implementation is considered as a whole. We further use the improved reconciliation method in [24] to achieve smaller public-key sizes, smaller bandwidth requirements, tolerance of higher error and a better failure probability. Finally, our approach to refresh the public matrix $\boldsymbol{A}$ improves computational performance of existing approaches [8, 2] while preventing pre-computation attacks.

The usage of all these optimizations in spKEX together allows reducing the bandwidth requirements of Frodo around 30% while having significant implementation advantages.

An interesting direction of future work is the design of other cryptographic primitives such as key-encapsulation (with ephemeral serets) based on similar design principles. Further research directions are the security analysis of the scheme and optimized implementations resistant to side-channel attacks.

References

[1] Martin R. Albrecht. On dual lattice attacks against small-secret LWE and parameter choices in HElib and SEAL. Cryptology ePrint Archive, Report 2017/047, 2017. http://eprint.iacr.org/2017/047.

[2] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - a new hope. Cryptology ePrint Archive, Report 2015/1092, 2015. http://eprint.iacr.org/2015/1092.

[3] Shi Bai and Steven D. Galbraith. Lattice Decoding Attacks on Binary LWE. Cryptology ePrint Archive, Report 2013/839, 2013. http://eprint.iacr.org/2013/839.

[4] Shi Bai, Adeline Langlois, Tancrède Lepoint, Amin Sakzad, Damien Stehle, and Ron Steinfeld. Improved security proofs in lattice-based cryptography: using the rényi divergence rather than the statistical distance. Cryptology ePrint Archive, Report 2015/483, 2015. http://eprint.iacr.org/2015/483.

[5] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. Cryptology ePrint Archive, Report 2011/401, 2011. http://eprint.iacr.org/2011/401.

[6] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. Cryptology ePrint Archive, Report 2015/1128, 2015. http://eprint.iacr.org/2015/1128.

[7] Andrej Bogdanov, Siyao Guo, Daniel Masny, Silas Richelson, and Alon Rosen. On the hardness of learning with rounding over small modulus. Cryptology ePrint Archive, Report 2015/769, 2015. http://eprint.iacr.org/2015/769.

[8] Joppe Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! Practical, Quantum-Secure Key Exchange from LWE. Cryptology ePrint Archive, Report 2016/659, 2016. http://eprint.iacr.org/2016/659.

[9] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 575–584, New York, NY, USA, 2013. ACM.

[10] Zvika Brakerski and Vinod Vaikuntanathan. Efficient Fully Homomorphic Encryption from (Standard) LWE. Cryptology ePrint Archive, Report 2011/344, 2011. http://eprint.iacr.org/2011/344.

[11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better Lattice Security Estimates. In *Proceedings of the 17th International Conference on The Theory and Application of Cryptology and Information Security*, ASIACRYPT'11, pages 1–20, Berlin, Heidelberg, 2011. Springer-Verlag.

[12] Jung Hee Cheon, Kyoo Hyung Han, Jinsu Kim, Changmin Lee, and Yongha Son. A Practical Post-Quantum Public-Key Cryptosystem Based on spLWE. Cryptology ePrint Archive, Report 2016/1055, 2016. http://eprint.iacr.org/2016/1055.

[13] Jung Hee Cheon, Duhyeong Kim, Joohee Lee, and Yongsoo Song. Lizard: Cut off the Tail! Practical Post-Quantum Public-Key Encryption from LWE and LWR. Cryptology ePrint Archive, Report 2016/1126, 2016. http://eprint.iacr.org/2016/1126.

[14] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654, September 1976.

[15] Jintai Ding, Xiang Xie, and Xiaodong Lin. 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/2012/688.

[16] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 212–219, New York, NY, USA, 1996. ACM.

[17] Thijs Laarhoven. *Search problems in cryptography.* PhD thesis, Eindhoven University of Technology, 2015.

[18] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: Extended abstract. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 333–342, New York, NY, USA, 2009. ACM.

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

[20] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 84–93, New York, NY, USA, 2005. ACM.

[21] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[22] C. P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66(2):181–199, September 1994.

[23] Peter W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.*, 26(5):1484–1509, October 1997.

[24] Ludo Tolhuizen, Ronald Rietman, and Oscar Garcia-Morchon. Improved key-reconciliation method. Cryptology ePrint Archive, Report 2017/295, 2017. `http://eprint.iacr.org/2017/295`.

A Upper bound on Failure Probability

In this section, we present an upper bound on the probability that Alice and Bob do not agree on a common key. In order to do so, we derive an upper bound on the probability that for given i, j, entry (i, j) of $\{\boldsymbol{K}_a - \boldsymbol{K}_b\}_p$ is at least $\Delta + 1$. Note that $\boldsymbol{K}_a - \boldsymbol{K}_b \equiv \boldsymbol{E}_b \cdot \boldsymbol{S}_a - \boldsymbol{S}_b^{\mathsf{T}} \cdot \boldsymbol{E}_a \pmod{p}$. We consider entry (i, j) of $\boldsymbol{K}_a - \boldsymbol{K}_b$. We denote the i-th row of $\boldsymbol{E}_b$ by $\boldsymbol{e}'$, the j-th column of $\boldsymbol{S}_a$ by $\boldsymbol{s}$, the i-th row of $\boldsymbol{S}_b^{\mathsf{T}}$ by $\boldsymbol{s}'$, and the j-th column of $\boldsymbol{E}_a$ by $\boldsymbol{e}$. Note that $\boldsymbol{s}$ and $\boldsymbol{s'}$ are ternary vectors of length n and both have Hamming weight h. We are thus interested in finding an upper bound for

$$\Pr[|(\boldsymbol{s}, \boldsymbol{e}') - (\boldsymbol{s}', \boldsymbol{e})| \geq \Delta + 1.] \tag{24}$$

From the definition, it follows that $\boldsymbol{e}'$ is determined by $\boldsymbol{A}$ and $\boldsymbol{s'}$, and that $\boldsymbol{e}$ is determined by $\boldsymbol{A}$ and $\boldsymbol{s}$. We can assume that $\boldsymbol{e}'$ and $\boldsymbol{s}$ are independent, as well as $\boldsymbol{e}$ and $\boldsymbol{s}'$. The inner products $(\boldsymbol{s}, \boldsymbol{e}')$ and $(\boldsymbol{s}', \boldsymbol{e})$, however, are *not* independent. Indeed, $\boldsymbol{e}$ has non-integer coefficients, so that $(\boldsymbol{s}', \boldsymbol{e})$ need not be integer; similarly the inner product $(\boldsymbol{s}, \boldsymbol{e}')$ need not be integer. The difference

between these two inner products however, is integer. We take a conservative approach to work around this dependence.

By the triangle inequality, $|(\boldsymbol{s}, \boldsymbol{e}') - (\boldsymbol{s}', \boldsymbol{e})|$ can only be at least $\Delta + 1$ if at least one of $|(\boldsymbol{s}, \boldsymbol{e}')|$ and $|(\boldsymbol{s}', \boldsymbol{e})|$ is at least $(\Delta + 1)/2$. It follows that

$$\Pr[|(\boldsymbol{s}, \boldsymbol{e}') - (\boldsymbol{s}', \boldsymbol{e})| \geq \Delta + 1] \ \leq \ \Pr[|\boldsymbol{s}, \boldsymbol{e}')| \geq \frac{1}{2}(\Delta + 1)) \ \vee \ |(\boldsymbol{s}', \boldsymbol{e})| \geq \frac{1}{2}(\Delta + 1)]$$

$$\leq \Pr[|(\boldsymbol{s}, \boldsymbol{e}')| \geq \frac{1}{2}(\Delta + 1)] + \Pr[|(\boldsymbol{s}', \boldsymbol{e})| \geq \frac{1}{2}(\Delta + 1)].$$

We now assume that the entries of $\boldsymbol{e}'$ are independently and uniformly distributed on $(-1/2, 1/2]$; note that $\boldsymbol{s}$ is not used in the computation of $\boldsymbol{e}'$. Similarly, we assume that the entries of $\boldsymbol{e}'$ are independently and uniformly distributed on $(-1/2, 1/2]$. From the symmetry of the distribution of $\boldsymbol{e}'$ around 0, and the fact that $\boldsymbol{s}$ has entries from $\{-1, 0, 1\}$, it follows that $(\boldsymbol{s}, \boldsymbol{e}')$ is distributed as the sum of h uniform variables on $(-1/2, 1/2]$, where h is the number of non-zero entries of $\boldsymbol{s}$. We thus wish to find the upper bound for the probability that a sum of h uniform variables on $(-1/2, 1/2]$ is large. To do so, we apply the Chernoff bound.

If $X_1, \ldots, X_m$ are independent variables, then according to the Chernoff bound, for each $a \geq 0$

$$\Pr(\sum_{i=1}^{m} X_i \geq a) \leq \min_{t>0} e^{-ta} \prod_{i=1}^{m} E[e^{tX_i}]. \tag{25}$$

If each X_i is uniformly distributed on $(-1/2, 1/2]$, then for each $t > 0$

$$E[e^{tX_i}] = \int_{-1/2}^{1/2} e^{tx} dx = \frac{1}{t}(e^{t/2} - e^{-t/2}) \tag{26}$$

For each $t > 0$, we thus have that

$$\Pr(\sum_{i=1}^{m} X_i \geq a) \leq e^{-ta} \left(\frac{e^{t/2} - e^{-t/2}}{t} \right)^m. \tag{27}$$

As each X_i is in $(-1/2, 1/2]$, the only interesting case is $a \leq \frac{1}{2}m$. In (27), we write $a = \frac{1}{2}\alpha m$, set $\tau = t/2$, and obtain that for each $\tau > 0$

$$\frac{1}{m} \ln(\Pr(\sum_{i=1}^{m} X_i \geq \frac{1}{2}\alpha m)) \leq -\alpha\tau + \ln(\frac{e^{\tau} - e^{-\tau}}{2\tau}). \tag{28}$$

Hence, if we define

$$f(\alpha) := \frac{1}{\ln 2} \min\{-\alpha\tau + \ln \frac{e^{\tau} - e^{-\tau}}{2\tau}) \mid \tau > 0\},$$

then for each $a > 0$

$$\Pr[\sum_{i=1}^{m} X_i > a] \leq 2^{m \cdot f(\frac{2a}{m})}.$$

By differentiating to τ, we infer that for fixed $\alpha > 0$, the function $-\alpha\tau + \ln(\frac{e^\tau - e^{-\tau}}{2\tau})$ is minimized for the solution $\tau^*(\alpha)$ to the equation

$$\frac{e^\tau + e^{-\tau}}{e^\tau - e^{-\tau}} - \frac{1}{\tau} = \alpha.$$

We have not been able to find an analytical expression for $\tau^*(\alpha)$ nor for $f(\alpha)$. The minimization can be carried out easily in `SAGE` by running `find_local_minimum((ln(sinh(x)/x)-(α x))/ln(2.0),0,5)`. Here (0,5) denotes the interval over which to minimize. In case the location of the maximum is found at $x = 5$, it is best to increase the end point of the interval, and see if the location of the maximum changes.

Because of the symmetry of the probability distribution of $X_1, \ldots, X_m$, we have that $\Pr(|\sum_{i=1}^m X_i| \geq a) = 2(\Pr\sum_{i=1}^m X_i > a)$. Combining all the above, with $m = h$ and $a = \frac{1}{2}(\Delta + 1)$, we infer that the probability that Alice and Bob do not agree on entry (i, j) of the key is at most

$$4 \cdot 2^{h \cdot f((\Delta+1)/h)}.$$

The union bound implies that the probability that Alice and Bob do not agree on some entry of the key is at most $\overline{m} \cdot \overline{n} \cdot (4 \cdot 2^{h \cdot f((\Delta+1)/h)})$.

B Distinguishing Advantage in Dual Attack

The dual attack against the Learning with Rounding (LWR) problem aims at distinguishing the LWR distribution from a uniform distribution modulo q. The advantage of the attack, i.e., the distinguishing advantage can be directly expressed in terms of the statistical distance between these two distributions. In general, for two probability distributions p_0 and p_1 on some domain $D \subset \mathbb{R}$, a distinguishing game can be described in terms of a challenger who chooses a uniformly random bit $b \in \{0, 1\}$, draws a random number $x \in D$ according to distribution p_b, and reveals x to the adversary, who must then guess b. The advantage ϵ of the adversary in winning this game can be expressed in terms of the advantage in distinguishing between the probability distributions p_0 and p_1.

$$\epsilon = \frac{1}{2}\int_D |p_0(x) - p_1(x)|dx$$

For the case that p_0 is a Gaussian distribution: $p_0 = G_q(x, s^2)$ and p_1 is the uniform distribution on $[-q/2, q/2)$, i.e., $p_1 = 1/q$ the advantage is

$$\begin{aligned}
\epsilon &= \frac{1}{2}\int_{-q/2}^{q/2} \left| \sum_{k=-\infty}^{\infty} \frac{1}{\sqrt{2\pi s^2}} \mathbf{e}^{-(x-kq)^2/2s^2} - \frac{1}{q} \right| dx \\
&= \frac{1}{2}\int_{-1/2}^{1/2} \left| \sum_{k=-\infty}^{\infty} \frac{\alpha}{\sqrt{2\pi}} \mathbf{e}^{-\alpha^2(\xi-k)^2/2} - 1 \right| d\xi, \qquad \text{where } \alpha = q/s.
\end{aligned}$$

$$\epsilon \approx \int_{-a}^{a} \frac{\alpha}{\sqrt{2\pi}} \mathbf{e}^{-\alpha^2\xi^2/2} - 1 \; d\xi \quad = \operatorname{erf}\left(\frac{\alpha a}{\sqrt{2}}\right) - 2a = \operatorname{erf}\left(\ln\left(\frac{\alpha}{\sqrt{2\pi}}\right)\right) - \frac{2}{\alpha}\sqrt{\ln\left(\frac{\alpha^2}{2\pi}\right)}$$

For the general case however, i.e., as the standard deviation s of the Gaussian distribution increases, α decreases and the terms where $k \neq 0$ must also be considered in the summation. We can find an approximate upper bound for the advantage using the Cauchy-Schwarz inequality as:

$$\begin{aligned}
\epsilon &= \frac{1}{2}\int_{-1/2}^{1/2} \left| \sum_{k=-\infty}^{\infty} \frac{\alpha}{\sqrt{2\pi}} \mathbf{e}^{-\alpha^2(\xi-k)^2/2} - 1 \right| d\xi \\
&\leq \frac{1}{2}\left(\int_{-1/2}^{1/2} \left(\sum_{k=-\infty}^{\infty} \frac{\alpha}{\sqrt{2\pi}} \mathbf{e}^{-\alpha^2(\xi-k)^2/2} - 1 \right)^2 d\xi \right)^{1/2} \\
&= \frac{1}{2}\left(\int_{-1/2}^{1/2} \left(\sum_{k=-\infty}^{\infty} \frac{\alpha}{\sqrt{2\pi}} \mathbf{e}^{-\alpha^2(\xi-k)^2/2} \right)^2 d\xi - 1 \right)^{1/2} \\
&= \frac{1}{2}\left(\sum_{m=-\infty}^{\infty} \left|\widehat{\phi}_m\right|^2 - 1 \right)^{1/2} \qquad \text{(using Parseval's theorem).}
\end{aligned}$$

The numbers $\widehat{\phi}_m$ are the Fourier components of the periodic function $\xi \to qG_s(q\xi, s^2)$:

$$\widehat{\phi}_m = \int_{-1/2}^{1/2} \mathbf{e}^{-2\pi i m\xi} \sum_{k=-\infty}^{\infty} \frac{\alpha}{\sqrt{2\pi}} \mathbf{e}^{-\alpha^2(\xi-k)^2/2} d\xi = \mathbf{e}^{-2\pi^2 m^2/\alpha^2}$$

This gives the following upper bound for the distinguishing advantage ϵ:

$$\epsilon \leq \frac{1}{\sqrt{2}}\left(\sum_{m=1}^{\infty} \mathbf{e}^{-4\pi^2 m^2/\alpha^2} \right)^{1/2}$$

For general standard deviation s and large α, the $m = 1$ term dominates the summation, giving the approximate upper bound of $\epsilon \leq \exp(2\pi^2 s^2/q^2)/\sqrt{2}$.

C Estimating Lattice Attack Costs: Source Code

We provide here our `Python` script for obtaining the security levels for a given spKEX parameter set, both for the primal and dual lattice reduction-based attacks. Our script, originally based on a security estimation script by the authors of [2] optimizes over possible choices of BKZ block-size and number of sparse-ternary LWR samples in order to obtain the minimum attack cost (primal and/or dual) for the given parameters. This minimum attack cost is consequently the security level for the parameter set, for the attack type considered.

```
from __future__ import division
from math import sqrt,log,pi,exp,floor,ceil
log_infinity = 9999

## Calculate log base 2 of an input
def log2(n):
    res=float(log(n)/log(2))
    return res

## log_2 of best plausible Quantum Cost of SVP in dimension b.
## Source: https://github.com/tpoeppelmann/newhope.
## Parameter b: block-size of the BKZ lattice reduction algorithm used.
def svp_paranoid(b):
    c=log(sqrt(4./3))/log(2)
    return (float(log2(b)) + float(b*c))


## log_2 of best plausible Quantum Cost of SVP in dimension b.
## Source: https://github.com/tpoeppelmann/newhope.
## Parameter b: block-size of the BKZ lattice reduction algorithm used.
def svp_quantum(b):
    c=log(sqrt(13./9))/log(2)
    return (float(log2(b)) + float(b*c))


## log_2 of best known Quantum Cost of SVP in dimension b.
## Source: https://github.com/tpoeppelmann/newhope.
## Parameter b: block-size of the BKZ lattice reduction algorithm used.
def svp_classical(b):
    c=log(sqrt(3./2))/log(2)
    return (float(log2(b)) + float(b*c))


## Calculate the root Hermite factor for a given BKZ blocksize.
## Parameters:
## b: block-size of the BKZ lattice reduction algorithm used.
def get_rhf(b):
    return ( (pi*b)**(1./b) * b / (2*pi*exp(1)))**(1./(2.*b-2.))


## Calculate the log of the scaling/weighting factor used
## to build weighted lattices for attacks on sparse-ternary LWR.
## Parameters:
## sigma: Standard deviation of the LWR rounding error.
## theta: Sparseness of the sparse-ternary LWR problem.
##        theta = Hamming weight of LWR secret / n.
def log_scaling_factor(sigma,theta):
    return float(log2(sigma)-(0.5*log2(theta)))


## Calculation of the distinguishing advantage of the Dual Attack
## and the corresponding attack cost, in log_2, against sparse-ternary LWR.
## Parameters:
## b: block-size of the BKZ lattice reduction algorithm used.
## q: Large modulus of the (sparse-ternary) LWR problem.
## p: Rounding modulus of the LWR problem.
## n: dimension of the (sparse-ternary) LWR problem.
## m: number of LWR samples used in the dual attack.
## theta: Sparseness of the sparse-ternary LWR problem.
##        theta = Hamming weight of LWR secret / n.
## sec_level: 1 (classical), 2 (post-quantum), 3 (paranoid).
##            Determines sieving exponent of SVP oracle subroutine.
def dual_cost(b,q,p,n,m,theta,sec_level):
    ## Dimension of the dual lattice
    d=m+n
    ## Root-Hermite factor for a BKZ lattice reduction algorithm running with given block-size b.
    rhf=get_rhf(b)
    ## Calculate the standard deviation of the LWR rounding error
```

```
    sd=float(q/(2*sqrt(3.0)*p))
    ## Log2 of scaling factor for rescaling the lattice in case of small-secrets.
    log_omega=log_scaling_factor(sd,theta)
    omega=float(2**log_omega)
    ## Norm of the shortest vector in the scaled dual lattice.
    norm_v=(rhf**(d-1)) * ((q/omega)**(1. * n/d))
    tau= norm_v * sd / q
    ## Distinguishing advantage of the dual attack, in log_2
    log2_eps = -0.5 + (- 2 * pi * pi * tau * tau) / log(2)
    if(sec_level==1):
        svp_cost=svp_classical(b)
    elif(sec_level==2):
        svp_cost=svp_quantum(b)
    elif(sec_level==3):
        svp_cost=svp_paranoid(b)
    total_cost = max(0, - 2 * log2_eps - svp_paranoid(b)) + svp_cost
    return total_cost


## Calculate the norm of the unique shortest vector (s,e,1) in the Primal Embedding lattice
## Parameters:
## n: dimension of the (sparse-ternary) LWR problem.
## m: number of LWR samples used in the dual attack.
## sigma: Standard deviation of the LWR rounding error.
## theta: Sparseness of the sparse-ternary LWR problem.
##        theta = Hamming weight of LWR secret / n.
def min_norm(n,m,sigma,theta,omega):
    h=floor(n*theta)
    return sqrt(float((omega**2)*h) + float(m*(sigma**2)))


## Calculate the norm of the projection \widetilde(v) of the vector
## v=(s,e,1) on the vector space spanned by the last b-Gram-Schmidt
## vectors of the embedding lattice.
## Parameters:
## v_norm: The norm of the vector v=(s,e,1); v_dim: its dimension.
## b: block-size of the BKZ lattice reduction algorithm used.
## v_dim: Dimension of the vector v=(s,e,1).
def projected_norm(v_norm,b,v_dim):
    #print "v_norm is",v_norm,"and b is",b,"and v_dim is",v_dim,"and sqrt(b/v_dim) is",float(b/v_dim)
    proj_norm=float(v_norm*sqrt(b/v_dim))
    #print "proj_norm is",proj_norm
    return proj_norm

## Calculate the log of the volume of the lattice into which (primal attack) embedding is done.
## Calculating in log and not absolute values because q^d is huge and might quickly go out of hand.
## Parameters:
## q: Large modulus of the sparse-ternary LWR problem.
## d: dimension of the lattice considered in the primal embedding attack.
## n: dimension of the sparse-ternary LWR problem.
## omega: scaling factor to rescale the lattice for exploiting the ‘‘small’’-ness of secrets.
def log_embedLatticeVol(q,d,n,omega):
    log_vol=float((d-n-1)*(log2(q))) + float(n*log2(omega))
    return log_vol


## Success condition of Primal attack against sparse-ternary LWR.
## Returns true if attack succeeds for given parameters and false otherwise.
## Parameters:
## sd: standard deviation of the LWR rounding error.
## b: block-size of the BKZ lattice reduction algorithm used.
## n: dimension of the (sparse-ternary) LWR problem.
## m: number of LWR samples used in the dual attack.
## theta: Sparseness of the sparse-ternary LWR problem.
##        theta = Hamming weight of LWR secret / n.
def primal_attack_success(sd,b,q,n,m,theta):
    ## Dimension of the lattice considered in the Primal attack.
    d=m+n+1
```

```
    ## Root-Hermite factor for a BKZ lattice reduction algorithm running with given block-size b.
    log_rhf=log2(get_rhf(b))
    ## Log2 of scaling factor for rescaling the lattice in case of small-secrets.
    log_omega=log_scaling_factor(sd,theta)
    omega=float(2**log_omega)
    ## Norm of the unique shortest vector (s,e,1) that the primal attack searches for.
    norm_v=min_norm(n,m,sd,theta,omega)
    dim_v=m+n
    ## Norm of the projection of (s,e,1) on the span of the last b Gram-Schmidt vectors
    #print "norm_v is",norm_v,"and b is",b,"and dim_v is",dim_v
    proj_norm_v=projected_norm(norm_v,b,dim_v)
    ## Log2 of volume of the lattice considered in the Primal attack, scaled due to small-secrets.
    log_latt_vol=log_embedLatticeVol(q,d,n,omega)
    ## Check primal embedding attack success condition (Equation 20 of spKEX document)
    #print "proj_norm_v is",proj_norm_v
    lhs=log2(proj_norm_v)
    rhs=float((b+b-d-1)*log_rhf) + float(log_latt_vol/d)
    if(lhs<=rhs):
        #Primal attack succeeded
        return True
    else:
        #Primal attack failed
        return False


## For a given q,p,n calculate the cost of running the primal attack (if it succeeds),
## otherwise return cost=infinity.
## Parameters:
## b: block-size of the BKZ lattice reduction algorithm used.
## q: Large modulus of the (sparse-ternary) LWR problem.
## p: Rounding modulus of the LWR problem.
## n: dimension of the (sparse-ternary) LWR problem.
## m: number of LWR samples used in the dual attack.
## theta: Sparseness of the sparse-ternary LWR problem.
##        theta = Hamming weight of LWR secret / n.
## sec_level: 1 (classical), 2 (post-quantum), 3 (paranoid).
##            Determines sieving exponent of SVP oracle subroutine.
def primal_cost(b,q,p,n,m,theta,sec_level):
    if(sec_level>3):
        print "Undefined security level. Abort."
        return -1
    ## Calculate the root Hermite factor for the given BKZ block-size
    rhf=get_rhf(b)
    ## Calculate the standard deviation of the LWR rounding error
    sigma=float(q/(2*sqrt(3.0)*p))
    ## Calculate primal attack success condition, Equation 20 of spKEX document.
    if(primal_attack_success(sigma,b,q,n,m,theta)):
        #Primal attack succeeds. Calculate the actual cost of running it, assuming one SVP call
        if(sec_level==1):
            return svp_classical(b)
        elif(sec_level==2):
            return svp_quantum(b)
        elif(sec_level==3):
            return svp_paranoid(b)
    else:
        return log_infinity     #Primal attack fails. Cost of running it is undefinably large.


## For a given q,p,n, find the BKZ blocksize b and number of samples m
## that result in minimal attack cost to the attacker.
## Parameters:
## q: Large modulus of the (sparse-ternary) LWR problem.
## p: Rounding modulus of the LWR problem.
## n: dimension of the (sparse-ternary) LWR problem.
## max_m: Maximum number of (sparse-ternary) LWR samples
##        that a single spKEX key-exchange session provides to the attacker.
## m: number of LWR samples used in the dual attack.
## theta: Sparseness of the sparse-ternary LWR problem.
```

```
##        theta = Hamming weight of LWR secret / n.
## sec_level: 1 (classical), 2 (post-quantum), 3 (paranoid).
##            Determines sieving exponent of SVP oracle subroutine.
def optimize_attack_b_m(q,p,n,max_m,theta,sec_level,attack_type):
    if(sec_level>3):
        print "Undefined security level. Abort."
        return -1
    best_cost=log_infinity
    best_b=0
    best_m=0
    b_min=60
    b_max=(2*n)+max_m+1
    if(b_max<b_min):
        return (0,0,0)
    for b in range(b_min,b_max):
        if(sec_level==1):
            current_cost=svp_classical(b)
        elif(sec_level==2):
            current_cost=svp_quantum(b)
        elif(sec_level==3):
            current_cost=svp_paranoid(b)
        if(current_cost>best_cost):
            ## Cost of running the attakc is higher than the (previously) best cost.
            ## No point checking any more values of b,m since minimal cost has already been found
            print "Current cost > Best cost. Breaking loop."
            break
        for m in range(max(1,(b-n)),max_m):
            if(attack_type==1):
                #Primal attack
                attack_cost=primal_cost(b,q,p,n,m,theta,sec_level)
            elif(attack_type==2):
                #Dual attack
                attack_cost=dual_cost(b,q,p,n,m,theta,sec_level)
            else:
                #Undefined attack type
                attack_cost=log_infinity
            if(attack_cost<best_cost):
                #Have found a b,m pair for which the attack cost is even cheaper than what it was so far.
                best_cost=attack_cost
                best_b=b
                best_m=m
    if(attack_type==1):
        correct_cost=primal_cost(best_b,q,p,n,best_m,theta,sec_level)
    else:
        correct_cost=dual_cost(best_b,q,p,n,best_m,theta,sec_level)
    if((best_b != 0) and (best_cost != correct_cost)):
        print "Error. Calculated best cost not equal to the primal/dual cost for best_b and best_m."
        return (-1,-1,-1)
    else:
        print "Best_b: ",best_b," Best m: ",best_m," Minimal cost: ",best_cost
        return (best_b,best_m,best_cost)

################################################################################

## Calculate optimal BKZ block-size and number of samples for a given q,p,n
## for which attacking spKEX (either via primal attack, or via dual attack) has the minimum cost.
## spKEX parameters for Quantum security level

q=float(2**14)
p=float(2**11)
n=710
theta=0.2          #Hamming weight is 0.2n
sd=float(q/(2*sqrt(3)*p))
n_bar=8
max_m=n+n_bar
sec_level=2        #1:Classical, 2: Post-quantum, 3:Paranoid.
if sec_level is 1:
        print "Security level is Classical."
```

```
elif sec_level is 2:
        print "Security level is Quantum."
elif sec_level is 3:
        print "Security level is Paranoid."
else:
        print "Undefined security level. Abort."
        exit()

print "\nParameters:\n n:",n," q: ",q," p:",p,"Standard deviation of error:",sd

attack_type=1      #1 for primal attack, 2 for dual attack
print "\nOptimizing PRIMAL attack cost on LWR over multiple choices of BKZ blocksize and LWR samples."
(b,m,cost)=optimize_attack_b_m(q,p,n,max_m,theta,sec_level,attack_type)

attack_type=2      #1 for primal attack, 2 for dual attack
print "\n\n\nOptimizing DUAL attack cost on LWR over multiple choices of BKZ blocksize and LWR samples."
(b,m,cost)=optimize_attack_b_m(q,p,n,max_m,theta,sec_level,attack_type)
```