A Practical Key Exchange for the Internet using Lattice Cryptography

Vikram Singh*

Abstract

In [21], Peikert presents an efficient and provably secure set of lower level primitives for practical post-quantum cryptography. These primitives also give the first lattice-based scheme to provide perfect forward secrecy, and thus represent a major advancement in providing the same sort of security guarantees that are now expected for modern internet traffic protection. However, the presentation in [21] might prove a bit daunting for the slightly less mathematical reader. Here we provide what we hope will be a clear and self-contained exposition of how the algorithm can be implemented, along with sample code and some initial analysis for potential parameter sizes.

We focus on the simpler case, as chosen by Bos et al in [1], of cyclotomic rings whose degree is a power of two. We describe the necessary arithmetic setup and choices regarding error sampling, and give a possibly cleaner mechanism for reconciliation of the shared secrets. Then we present Peikert's Diffie-Hellman-like key exchange algorithms along with security, correctness and implementation analysis. We demonstrate parameter choices that outperform [1] by a factor of up to 13 for equivalent security.

Keywords: Cryptography, Lattice, Ring-LWE, Ring Learning With Errors, Key Exchange, IKE, TLS

1 Introduction

Lattice-based cryptographic protocols, particularly those presented by Peikert in [21], have great promise as components of Internet standards and other open protocols such as Internet Key Exchange (IKE) and Transport Layer Security (TLS). This is due both to their efficiency and practicality, and that they provide security guarantees against classical and quantum adversaries. Indeed, Peikert expressly defines his protocols as drop-in replacements for the classical key exchange and key transport primitives of Diffie-Hellman and RSA as used in real-world networks, and *adhering as closely as possible* to the abstract protocols underlying existing and proposed standards, *e.g.* IETF RFCs like [11, 26, 10, 12, 13]. Adapting lattice-based mechanisms to have the desired properties of these primitives had been a challenge, but Peikert was

*vs77814@gmail.com. Principal Consultant, VS Communications.

able to demonstrate concrete ways in which existing protocols could be generalised so as to yield secure lattice-based instantiations, without substantially affecting their overall form or security analysis. Unlike earlier lattice-based constructions such as NTRUEncrypt, the Peikert scheme fits neatly into the TLS protocols with proofs of security in each case ([21, 1]).

Each of the schemes proposed by Peikert is based on the *learning with errors over rings* (ring-LWE) problem [19, 20]. In overview, the ring-LWE problem in a ring R may be defined by fixing a certain error distribution χ over R that is concentrated on "small" elements, and randomly picking a secret ring element $s \in R$ from which to generate "random noisy ring equations". Each such equation is generated by letting $a_i \in R$ be a random ring element, and $e_i \in R$ be drawn from χ as a random small perturbation, then publishing pairs $(a_i, b_i = a_i \cdot s + e_i)$. The *decision* version of ring-LWE is to distinguish independent "random noisy ring equations" from truly uniform pairs. That is, given pairs $(a_i, b_i) \in R \times R$ either of the form $(a_i, b_i = a_i \cdot s + e_i)$ or uniform random in $R \times R$, distinguish from which of the two distributions the pairs have been drawn. There is also a *search* version of ring-LWE: given pairs of the form $(a_i, b_i) \in R \times R$ as above, recover the secret s.

Ring-LWE enjoys strong provable hardness guarantees: it is hard on average so long as the Short Vector Problem (SVP) is hard to approximate on so-called *ideal* lattices in the corresponding ring *in the worst case*. These results provide good theoretical evidence that ring-LWE is a solid foundation on which to design cryptosystems, and this evidence has been reinforced by concrete cryptanalytic efforts.

In [21], Peikert presents a ring-LWE based Diffie-Hellman-like key exchange algorithm in which two users each exchange a single ring-LWE "sample" or public key to arrive at *approximate* or "noisy" agreement on a ring element. Let $b = a \cdot s_0 + e_0$ be the public key of the first party, and $c = a \cdot s_1 + e_1$ be the public key of the second. Then the two parties can arrive at an approximate shared secret $s_0 \cdot c = s_0 \cdot a \cdot s_1 + s_0 \cdot e_1 \approx s_0 \cdot a \cdot s_1 + s_1 \cdot e_0 = s_1 \cdot b$, assuming that s_0, e_0, s_1, e_1 are all "small" for a suitable definition of small. In order to (non-interactively) reach *exact* agreement, the second party will derive the binary key stream from its version of the shared secret value and also derive and send an additional bit-string referred to as the "masking bits." These masking bits provide the extra hint which can be then fed into the "reconciliation" technique for the first party to arrive at exact agreement with the key stream derived by the second.

Throughout this paper we aim to give a readable and self-contained description of the Peikert algorithms and our implementation. We restrict to the case of cyclotomic rings whose degree is a power of two, as do Bos et al in [1]. This case hides complexities of ring arithmetic which it is clearer to do without at this stage, yet also provides a reasonable diversity of practical security levels.

In Section 2 we establish the necessary setup, defining subgaussian random variables and how we will use them to bound the accumulation of error terms. We state the ring-LWE problem and define the cyclotomic rings we shall use, including the Chinese Remainder basis required

for efficient arithmetic. We define the error distribution we shall use to randomly draw small ring elements. In Section 3 we sketch the key exchange, and describe how masking bits and the reconciliation mechanism are used to generate key from the approximate shared secrets. The reconciliation mechanism is a slight variation on Peikert's, which is cleaner but has the same effect.

In Section 4 we present Peikert's basic key exchange algorithm in detail in the context of our implementation choices. We then present Peikert's development of provably actively secure key exchanges in order to protect against an adversary that can choose ciphertexts, and his development of provably secure authenticated key exchanges in order to assure each party that the key is agreed only with the holder of the desired certified identity.

In Section 5 we describe our parameter choices along with a consideration of their security, an analysis of correctness, and performance timings for the basic key exchange. We provide example `sage` code in Appendix A, and higher performance `C` code at

`https://github.com/vscrypto/ringlwe_power`.

We suggest that our parameters are both more efficient and more secure than any previous work.

2 Preliminaries and definitions

For any integer q, let $\mathbb{Z}_q$ denote the quotient ring $\mathbb{Z}/q\mathbb{Z}$, *i.e.* the ring of integers modulo q. The elements of this ring can be considered in terms of a distinguished set of representatives, *e.g.* the set $\{0, 1, \ldots q-1\}$.

For any two subsets X, Y of an additive group, we define $-X := \{-x : x \in X\}$ and $X + Y := \{x + y : x \in X, y \in Y\}$. Similarly, we define $x + Y := \{x + y : y \in Y\}$ for a fixed element x.

We define the *infinity norm* on a ring R with basis $Y = \{y_j\}$ to be $||r||_\infty := \max_j(r_j)$ for $r = \sum_j r_j \cdot y_j \in R$. We will be interested in the growth of individual basis coefficients of elements of R and will perform our analysis on this infinity norm $|| \cdot ||_\infty$ rather the Euclidean norm $|| \cdot ||_2$.

2.1 Gaussian and subgaussian distributions

Early exposition of ring-LWE ([19]) relied on Gaussian distributions for all error sampling. For $r > 0$, the Gaussian distribution D_r over $\mathbb{R}$ with parameter r has probability distribution function $\frac{1}{r}e^{(-\pi x^2/r^2)}$. More recent works [20, 21] have updated these ideas to be based on the notion of a *subgaussian distribution*. For any $\delta \geq 0$, we say that a random variable X (or

its distribution) over $\mathbb{R}$ is δ-*subgaussian* with parameter $r > 0$ if for all $t \in \mathbb{R}$, the (scaled) moment-generating function satisfies

$$\mathbb{E}[\exp(2\pi tX)] \leq \exp(\delta) \cdot \exp(\pi r^2 t^2).$$

Subgaussians are useful to simplify the analysis and have many nice features, such as the sum of independent subgaussians is another subgaussian whose parameters can be calculated. Most importantly, as noted in [21], any B-bounded centered random variable X (i.e., $\mathbb{E}[X] = 0$ and $|X| \leq B$ always) is 0-subgaussian with parameter $B\sqrt{2\pi}$. Thus simple bounded distributions, such as uniform random from an interval, are subgaussian. We will utilise this fact to provide significant efficiency gains and simplification to our key establishment.

2.2 Ring-LWE

We now recall the ring-LWE probability distribution and (decisional) problem as presented in [21]; see [19] for a more general form.

Definition 1 (Ring-LWE Distribution)**.** For an $s \in R$ and a distribution χ over R, a sample from the Ring-LWE Distribution $A_{s,\chi}$ over $R_q \times R_q$ is generated by choosing $a \leftarrow R_q$ uniformly at random, choosing $e \leftarrow \chi$, and outputting $(a, b = a \cdot s + e)$.

Definition 2 (Ring-LWE Decision)**.** The *decision* version of the ring-LWE problem, denoted R-$\mathsf{DLWE}_{q,\chi}$,is to distinguish with non-negligible advantage between independent samples from $A_{s,\chi}$, where $s \leftarrow \chi$ is chosen once and for all, and the same number of *uniformly random* and independent samples from $R_q \times R_q$.

The main theorem of [19] can be stated informally as follows:

Theorem 1. Suppose that it is hard for polynomial-time quantum algorithms to approximate (the search version of) the shortest vector problem (SVP) in the worst case on ideal lattices in R to within a fixed poly(n) factor. Then any poly(n) number of samples drawn from the R-LWE distribution are pseudorandom to any polynomial-time (possibly quantum) attacker.

This tells us that distinguishing the Ring-LWE Distribution from random or recovering its secret is hard, provided SVP is hard.

A main benefit of ring-LWE over traditional LWE and other lattice-based techniques is in efficiency. Whereas LWE requires the use of $\Omega(n)$ samples $(a_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, for ring-LWE, typically only a very small number of samples $(a, b) \in R \times R$ are used.[1] Essentially, ring-LWE offers a compact representation for the lattice in question. The reason for this is that each polynomial $v \in R$ represents n vectors in the lattice, one for each multiple $v \cdot x^i$ for

[1] A careful statement of the above worst-case hardness result shows that it deteriorates with the number of samples; fortunately, all our applications require only a small number of samples.

$i \in \{0, \ldots, n-1\}$.[2] If a is fixed for all users, then the public key is just the element $b \in R$. We see that this results in smaller public keys for ring-LWE because LWE would have $\Omega(n^2)$ modular values whereas ring-LWE can have only n.

2.3 Cyclotomic rings

Let $R := \mathbb{Z}[\zeta_m] \cong \mathbb{Z}[x]/\langle \Phi_m(x) \rangle$ be the mth cyclotomic ring, where Φ is defined by $x^m - 1 = \prod_{d|m} \Phi_d(x)$ and ζ_m is a primitive mth root of unity. The degree of R is the degree of Φ, which is given by the Euler totient function $n := \phi(m)$. We shall mainly be interested in the quotient ring $R_q := R/qR \cong \mathbb{Z}_q[\zeta_m]$ where all of our operations can be defined.

There are a variety of simplifications and efficiencies that arise when considering the special case of $m = 2^\ell$ a power of two. In this paper, we will focus on this special case in order to invoke those results. In this case, $n = \phi(m) = m/2$ and $\Phi_m(x) = 1 + x^n$. We shall utilise prime q such that $q \equiv 1 \pmod{m}$, as is required by the Security Proof stated in Theorem 2.7 of [19]. This also allows arithmetic speed-ups using the Chinese Remainder basis: the ideal $< q >:= qR$ factors as $m/2$ prime ideals which provide the orthogonal components required by the Chinese Remainder Theorem. For the remainder of this paper, we consider m and q of this form unless explicitly stated. We also drop the subscript m on ζ_m.

2.3.1 Bases for cyclotomic rings

In [20], multiple bases for the cyclotomic ring R are defined. As we choose to work with $m = 2^\ell$, the power, powerful, and decoding bases will all coincide, a nice simplification that stems from choosing m of this form.[3]

The power basis for R or R_q is the natural basis of powers of ζ:

$$\vec{p} := (\zeta^i)_{i=0,\ldots,n-1}$$

If we write $a \in R$ in the power basis as $a = \vec{p} \cdot \mathbf{a}$, then the canonical embedding[4] defines the Chinese Remainder Theorem matrix CRT as the square matrix

$$\sigma(a) = \mathrm{CRT} \cdot \mathbf{a} = \begin{pmatrix} 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^3 & \omega^6 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{m-1} & \omega^{(m-1)2} & \cdots & \omega^{(m-1)(n-1)} \end{pmatrix} \cdot \mathbf{a}$$

[2] In ring-LWE, each ring element actually represents a so-called *ideal lattice* - a lattice L is an ideal lattice if for all $\mathbf{v} \in L$, $x \cdot \mathbf{v}$ is also in L. Thus one polynomial in R defines a space of n vectors in L.

[3] Simplifying equation 6.2 of [20] to the $m = 2^\ell$ case, we find that $\vec{d} = (2/m)\vec{p}$ for $\vec{d}$ the decoding basis of $R^\vee$. Thus, performing the scaling described in Section 2.3.2 of [21] to move to the decoding basis of R simply recovers the power basis of R.

[4] The canonical embedding $\sigma : R \to H \subset \mathbb{C}^n$ is given by $v \mapsto (v(\omega_m^i))_{i \in \mathbb{Z}_m^\times}$ for $\omega_m := e^{2\pi i/m}$.

where the rows are indexed by $\mathbb{Z}_m^*$ and the columns by $\{0, 1, \ldots, n-1\}$ for $n = m/2$. Note that multiplication by CRT can be done by fast Fourier transform methods in $O(n \log n)$ complexity.

In [20], the CRT matrix is defined as a general object over any commutative ring containing a primitive mth root of unity ω. By fixing a specific primitive mth root of unity ζ in $\mathbb{Z}_q$, we can turn the CRT matrix into an important and powerful tool for working in R_q. As q was chosen with $q \equiv 1 \pmod{m}$, the cyclotomic polynomial $x^n + 1$ will factor completely modulo q:

$$x^n + 1 = (x - \zeta)(x - \zeta^3)(x - \zeta^5) \cdots (x - \zeta^{m-1})$$

for ζ a primitive mth root of unity in $\mathbb{Z}_q$. Thus, multiplication by CRT gives evaluation of the polynomial at each of the mth roots of unity, that is, at the primitive root ζ and each of its odd powers. Each polynomial in R_q has a unique representation from evaluating at the roots of unity, so this form can be used as a basis, the Chinese Remainder basis. While addition can be performed component-wise in all bases, the unique thing about the Chinese Remainder basis is that multiplication is also component-wise: for any two polynomials $z, z' \in R_q$ and any $c \in \mathbb{Z}_q$, $z(c) \cdot z'(c) = (z \cdot z')(c)$. Naively multiplication would require summing $O(n^2)$ cross-products of all pairs of coefficients but use of the Chinese Remainder basis brings that to $O(n)$-complexity.

The Chinese Remainder basis $\vec{c}$ is defined for R_q by

$$\vec{c}^T := \vec{p}^T \cdot \mathrm{CRT}_q^{-1} \text{ , where } \mathrm{CRT}_q := \mathrm{CRT} \pmod{q}$$

and thus admits $O(n \log n)$ fast Fourier transform method conversion between itself and the power basis. We have shown we can convert between any of these bases and the Chinese Remainder basis in $O(n \log n)$, hence can perform multiplication in $O(n \log n)$.

2.4 Error distributions

In order that each party has some secret information that can be combined in Diffie-Hellman fashion to obtain a shared secret value accessible only to the two parties, it is necessary to produce the short error elements that act as private keys. The aim is to produce errors that are subgaussian with parameters as tight as possible, so that the security parameters of these errors can be nicely controlled once they have been combined to form shared secret values.

As in [1], since this paper restricts to the case of $m = 2^\ell$ being a power of 2, sampling from a discrete Gaussian can be performed by sampling each coefficient from a 1-dimensional discrete Gaussian $D_{\mathbb{Z},\sigma}$with parameter σ.[5] The discrete Gaussian assigns to each $x \in \mathbb{Z}$ a probability proportional to $e^{-x^2/(2\sigma^2)}$, normalized by the factor $S = 1 + 2\sum_{k=1}^{\infty} e^{-k^2/(2\sigma^2)}$, given by

[5] In [20], the short secret and error vectors are produced by sampling the conjugate symmetry space $H \subset \mathbb{C}^n$ and then mapping into R by left-multiplying by the matrix CRT_m^*. If $m = 2^\ell$, the matrix $\mathrm{CRT}_m^* \cdot \mathrm{CRT}_m$ is diagonal. Let us consider the covariance matrix $\Sigma = E[(\mathrm{CRT}_m^* \cdot v)(\mathrm{CRT}_m^* \cdot v)^*] = E[\mathrm{CRT}_m^* \cdot v \cdot v^* \cdot \mathrm{CRT}_m] = \mathrm{CRT}_m^* \cdot E[vv^*] \cdot \mathrm{CRT}_m$ since CRT_m^* is a constant and so can be removed from the expectation. Since the elements of v are independent, the covariance matrix $E[vv^*]$ is diagonal, and so we can see that Σ is diagonal, demonstrating that the entries of $\mathrm{CRT}_m^* \cdot v$ are independent and so can be sampled independently.

$D_{\mathbb{Z},\sigma}(x) = \frac{1}{S}e^{-x^2/(2\sigma^2)}$. To sample from R_q according to a discretised Gaussian distribution, we use the method in [1]: precompute a lookup table T of size 52 where $T[0] := \lfloor 2^{192}/S \rfloor$ and

$$T[i] := \left\lfloor 2^{192} \cdot \left(\frac{1}{S} + 2\sum_{x=1}^{i} D_{\mathbb{Z},\sigma}(x)\right)\right\rfloor$$

for $i = 1, \ldots, 50$, and where $T[51] := 2^{192}$. We sample each coefficient by generating a 192-bit integer t at random, finding the index $\texttt{ind} \in [0, 50]$ such that $T[\texttt{ind}] \leq t < T[\texttt{ind}+1]$, and then generating an additional random bit for the sign $\texttt{sign} \in \{-1, 1\}$:

GaussSample
Input: standard deviation σ
Output: error vector e
for $i = 1, \ldots, n$
$t \xleftarrow{\$} (0, 2^{192})$
$\texttt{ind}_i = 0$
repeat $\texttt{ind}_i$ += 1 until
$T[\texttt{ind}_i] \leq t < T[\texttt{ind}_i + 1]$
$\texttt{sign}_i \xleftarrow{\$} \{-1, 1\}$
$e_i = \texttt{sign}_i \cdot \texttt{ind}_i$
$e \leftarrow \vec{p} \cdot (e_1, \ldots, e_n)$

To accord with [24, 18, 17, 1], we will set $\sigma = 8/\sqrt{2\pi} \approx 3.192$ so that the discrete Gaussian $D_{\mathbb{Z}^n,\sigma}$ approximates the continuous Gaussian D_σ extremely well.

In instantiations of ring-LWE, sampling the error terms from Gaussian distributions is typically by far the most expensive operation. As noted in Section 2.2 of [21], uniform sampling from a B-bounded interval is subgaussian with parameter $B\sqrt{2\pi}$, and uniform sampling of each coefficient is both simpler and significantly more efficient:

UniformSample
Input: bound B
Output: error vector e
for $i = 1, \ldots, n$
$e_i \xleftarrow{\$} \{-B, \ldots, B\}$
$e \leftarrow \vec{p} \cdot (e_1, \ldots, e_n)$

As the discrete uniform distribution is itself subgaussian and need not attempt to closely approximate a continuous distribution, we have more flexibility in setting the parameter B. In order to provide a fair comparison between the two alternatives, we set $B = 5$ so that the standard deviation of the two distributions is approximately equal. Thus we can use a uniform random distribution choosing the coefficients from the set $\{-5, \ldots, 5\}$.

3 Key generation and reconciliation mechanism

In [21], Peikert presents a ring-LWE based Diffie-Hellman-type key exchange algorithm in which two users exchange ring-LWE public keys to arrive at *approximate* or "noisy" agreement on a ring element. In order to (non-interactively) reach *exact* agreement, the second party sends along an additional bit-string which can be fed into the "reconciliation" technique from [21] which we will develop in this section.

3.1 Description of the exchange

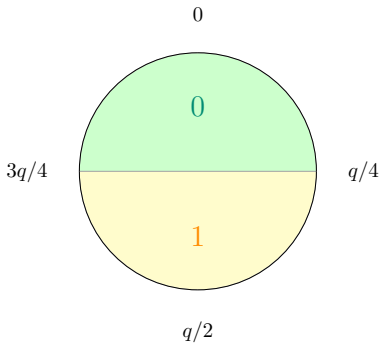
We begin with an informal sketch of how the key exchange works. In the basic key exchange, the first party creates a public key $b = a \cdot s_1 + s_0$ and transmits that to the second party. Upon receipt of the public key b, the second party creates his public key $u = e_0 \cdot a + e_1$ and his version of the approximate shared secret $v = e_0 \cdot b + e_2 = e_0 \cdot a \cdot s_1 + e_0 \cdot s_0 + e_2$. Upon receipt of the public key u and masking bits $\langle v \rangle_2$, the first party forms her version of the shared secret $w = u \cdot s_1 = e_0 \cdot a \cdot s_1 + e_1 \cdot s_1$, and feeds w and the received masking bits $\langle v \rangle_2$ into the reconciliation function to recover the key stream. Since s_0, s_1, e_0, e_1, e_2 are all small, $w \approx v$ and the masking bits $\langle v \rangle_2$ provide sufficient information for the two parties to exactly agree; however, for adversaries, the masking bits do not give any help in determining what the underlying key bit will be.

Party A		Party B
$pk = b = a \cdot s_1 + s_0$	$\xrightarrow{b}$	
		$pk = u = e_0 \cdot a + e_1$
		$SSV = v = e_0 \cdot b + e_2 =$
		$\quad e_0 \cdot a \cdot s_1 + e_0 \cdot s_0 + e_2$
		key $= \lfloor v \rceil_2$
	$\xleftarrow{(u, \langle v \rangle_2)}$	mask $= \langle v \rangle_2$
$w = u \cdot s_1$		
key $= \mathsf{rec}(w, \langle v \rangle_2)$		

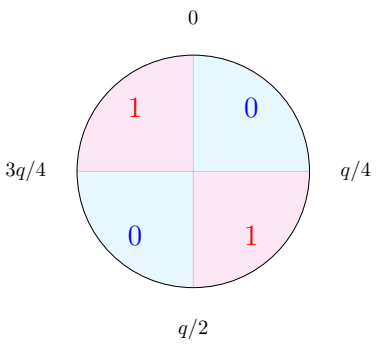
Figure 1: Basic key exchange algorithm.

The key stream that the two users will agree upon will be generated by applying the modular rounding function $\lfloor \cdot \rceil_2$ to each coefficient of the shared secret to round to the closer of 0 or $\frac{q}{2}$. In order for the two parties to achieve exact agreement, the second party also sends over the "masking bits" for his version of the approximate shared secret as generated by the function $\langle \cdot \rangle_2$ which labels which "quadrant" modulo q a coefficient falls into. If an equal number of elements of $\mathbb{Z}_q$ were in each quadrant, then key would be unbiased and the masking bits would give no information about the key. However, q is odd so there is an imbalance. Randomization is used

to correct this.



(a) The key stream as generated by the function $\lfloor \cdot \rceil_2$



(b) The masking bits as generated by the function $\langle \cdot \rangle_2$

Figure 2: The key stream and mask bits generating functions.

3.2 Randomized Rounding

We will now explain in detail each of the functions required to generate the key stream and masking bits, as well as the reconciliation function.

In order to create the new reconciliation technique in [21], Peikert works with even q, and generalises to the more common setting of odd q by introducing a step wherein all elements of $\mathbb{Z}_q$ are mapped into $\mathbb{Z}_{2q}$ in a randomized way. However, there are two basic criteria that need to be achieved, namely that the key stream be unbiased and that the masking bits achieve perfect hiding, each of which can be realised without resorting to mapping up into an even modulus by utilising a simple randomized rounding procedure common in engineering. By utilising the randomized rounding procedure, no larger internal states need be generated and in fact nothing at all needs to be done except in a pair of edge cases where the flip of a coin decides how to modify the value. This randomized rounding is thus more efficient.

We begin by defining the intervals I_0, I'_1, I'_0, I_1 to partition the elements of $v \in \mathbb{Z}_q$ according to the four respective quadrants as in Figure 3 as:

$$
\begin{aligned}
I_0 &:= \mathbb{Z}_q \cap [0, \tfrac{q}{4}) \\
I'_1 &:= \mathbb{Z}_q \cap [\tfrac{q}{4}, \tfrac{q}{2}) \\
I'_0 &:= \mathbb{Z}_q \cap [\tfrac{q}{2}, \tfrac{3q}{4}) \\
I_1 &:= \mathbb{Z}_q \cap [\tfrac{3q}{4}, q)
\end{aligned}
$$

For odd q the number of elements in each set is:

$$\begin{pmatrix} \#I_1 & \#I_0 \\ \#I'_0 & \#I'_1 \end{pmatrix} = \begin{cases} \begin{pmatrix} \frac{q-1}{4} & \frac{q+3}{4} \\ \frac{q-1}{4} & \frac{q-1}{4} \end{pmatrix} & \text{for } q \equiv 1 \pmod 4 \\ \begin{pmatrix} \frac{q-3}{4} & \frac{q+1}{4} \\ \frac{q+1}{4} & \frac{q+1}{4} \end{pmatrix} & \text{for } q \equiv 3 \pmod 4. \end{cases}$$

We will define key to be 0 if an element is in the top row $I_0 \cup I_1$ and 1 for the bottom row $I'_0 \cup I'_1$. We will define the masking bit as 0 for the anti-diagonal $I_0 \cup I'_0$ and 1 for the lead-diagonal $I_1 \cup I'_1$. We want the key to be unbiased, which will require that the top and bottom rows have equal counts. Furthermore, the masking bit tells us the diagonal; conditional on this, we shall require that no information is given about the key, i.e. both cells of the diagonal have equal counts. In order to meet these two criteria on the count array, we can perform a preprocessing stage which randomizes the value of elements in some edge cases. To minimise the amount of randomization and the distance an element moves, a natural choice is to probabilistically nudge the two edge elements out of the unique large cell for $q \equiv 1 \pmod 4$, or into the unique small cell for $q \equiv 3 \pmod 4$.

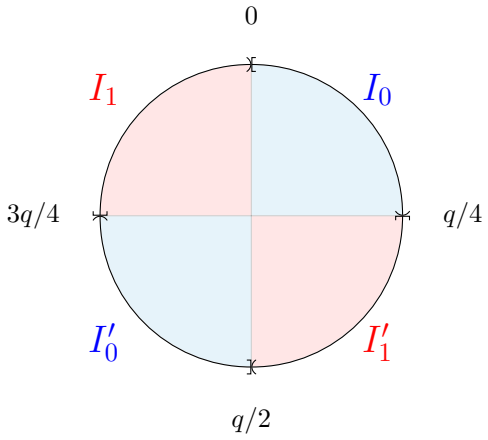


Figure 3: The values modulo q and their partition into I_0, I'_1, I'_0, I_1.

Thus we can define our *randomized rounding* procedure to affect this, for the two separate cases:

- $\mathbf{q \equiv 1} \pmod{\mathbf{4}}$. We map out of I_0. If $v = 0$, we will flip a coin or draw a uniform random bit and map 0 to either itself or $q - 1$ depending on the random bit. This has the effect of, with 50% probability, moving an element from I_0 to I_1. Independently, if $v = \frac{q-1}{4}$, we map $\frac{q-1}{4}$ to either itself or $\frac{q+3}{4}$ depending on a random bit, thus moving an element from I_0 to I'_1 with 50% probability. The count array representing the relative probability of

each set becomes

$$\begin{pmatrix} \frac{q+1}{4} & \frac{q-1}{4} \\ \frac{q-1}{4} & \frac{q+1}{4} \end{pmatrix} \text{ for } q \equiv 1 \pmod 4$$

which satisfies the two criteria of equal row sums for unbiased key, and equal cells on the lead-diagonal and anti-diagonal for perfect hiding of key by the mask. Note that the mask bit is biased, but this does not impact security.

- $\mathbf{q \equiv 3} \pmod{\mathbf{4}}$. We map into I_1. If $v = 0$, we will flip a coin or draw a uniform random bit and map 0 to either itself or $q-1$ depending on the random bit. This has the effect of, with 50% probability, moving an element from I_0 to I_1. Independently, if $v = \frac{3q-1}{4}$, we map $\frac{3q-1}{4}$ to either itself or $\frac{3q+3}{4}$ depending on a random bit, thus moving an element from I'_0 to I_1 with 50% probability. The count array representing the relative probability of each set becomes

$$\begin{pmatrix} \frac{q+1}{4} & \frac{q-1}{4} \\ \frac{q-1}{4} & \frac{q+1}{4} \end{pmatrix} \text{ for } q \equiv 3 \pmod 4$$

which satisfies the two criteria of equal row sums for unbiased key, and equal cells on the lead-diagonal and anti-diagonal for perfect hiding of key by the mask. Note that the mask bit is biased, but this does not impact security.

This randomized rounding procedure produces a similar effect to the dbl$(\cdot)$ procedure from [21], without the additional computational burden of mapping all elements of $\mathbb{Z}_q$ into $\mathbb{Z}_{2q}$. We will denote the randomized rounding of v by $\bar{v}$ and apply it before deriving mask and key bits.

3.3 Reconciliation mechanism

Given the above exposition, we can now define the *modular rounding* function $\lfloor \cdot \rceil_2 : \mathbb{Z}_q \to \mathbb{Z}_2$ as $\lfloor x \rceil_2 := \lfloor \frac{2}{q} \cdot x \rceil \pmod 2$. We will use $\lfloor \bar{v} \rceil_2$ to generate the key stream, which will not be biased due to the use of randomized rounding.

As in [21], we define the *cross rounding* function $\langle \cdot \rangle_2 : \mathbb{Z}_q \to \mathbb{Z}_2$ as

$$\langle v \rangle_2 := \lfloor \frac{4}{q} \cdot v \rfloor \pmod 2.$$

Equivalently, $\langle \bar{v} \rangle_2$ is the $b \in \{0,1\}$ such that $\bar{v} \in I_b \cup I'_b$. We use $\langle \bar{v} \rangle_2$ to generate the masking bit-string. If v is uniform random in $\mathbb{Z}_q$, then $\langle \bar{v} \rangle_2$ will be biased in $\mathbb{Z}_2$ despite the randomized rounding. Regardless of this bias, however, $\langle \bar{v} \rangle_2$ hides $\lfloor \bar{v} \rceil_2$ perfectly:

Claim 1. If $v \in \mathbb{Z}_q$ is uniform random and randomized rounded as outlined above, then $\lfloor v \rceil_2$ is uniform random given $\langle v \rangle_2$.

Proof: For any $b \in \{0,1\}$, if we condition on $\langle v \rangle_2 = b$, then v is uniform over $I_b \cup I'_b$. By the definition of I_b, if $v \in I_b$ then $\lfloor v \rceil_2 = 0$, whereas if $v \in I'_b$ then $\lfloor v \rceil_2 = 1$, so $\lfloor v \rceil_2$ is uniform random given $\langle v \rangle_2$. □

We can now see how reconciliation of the recipient's derived shared secret w using the mask bit $\langle \bar{v} \rangle_2$ must work. The mask bit tells the recipient which diagonal quadrant $\bar{v}$ is in. Provided the difference between shared secrets $|w - \bar{v}|$ is no more than one-eighth q then the shared secrets are sufficiently close that the recipient can infer which quadrant $\bar{v}$ was in, hence infer the key bit $\lfloor \bar{v} \rceil_2$.

Let $E := [-q/8, q/8) \cap \mathbb{Z}$, and define the *reconciliation* function $\mathrm{rec} : \mathbb{Z}_q \times \mathbb{Z}_2 \to \mathbb{Z}_2$ as

$$\mathrm{rec}(w, b) := \begin{cases} 0 & \text{if } w \in I_b + E \pmod{q} \\ 1 & \text{otherwise.} \end{cases}$$

Claim 2. If $w = v + e \pmod{q}$ for some $v \in \mathbb{Z}_q$ and $e \in E$, then $\mathrm{rec}(w, \langle v \rangle_2) = \lfloor v \rceil_2$.

To see that the claim is true, we will consider what the reconciliation function does in greater detail.

The variable w is a coefficient of the receiving party's version of the shared secret and $\langle v \rangle_2$ is the received hint based on which quadrant that coefficient of the sending party's version of the shared secret fell into. Suppose that $b = \langle v \rangle_2 = 0$. Then $v \in I_0 \cup I_0'$, and the region of interest, $I_0 + E$, can be pictorially represented as the shaded area in Figure 4.

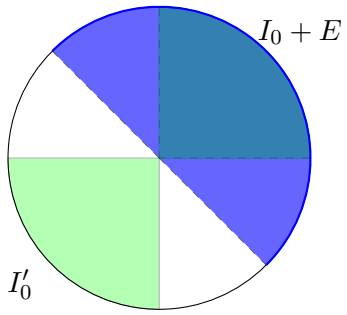


Figure 4: The region $I_0 + E$.

Now suppose that we have a guaranteed bound on how far apart w and v can be, namely that $w = v + e \pmod{q}$ for some $v \in \mathbb{Z}_q$ and $e \in E$, that is $|w - v| < \frac{q}{8}$. Then if $w \in I_0 + E$, v must be in I_0 (recall $v \in I_0 \cup I_0'$) or else be farther than $\frac{q}{8}$ away from w. Then by the definition of I_0, we have that $\lfloor v \rceil_2 = 0$ and $\mathrm{rec}(w, 0) = 0$. Otherwise, if $w \notin I_0 + E$, v must be in I_0' since those are the only possible v values within $\frac{q}{8}$ of a w value that is not in $I_0 + E$, and this would mean that $\lfloor v \rceil_2 = 1$.

Similarly, suppose that $b = \langle v \rangle_2 = 1$. Then $v \in I_1 \cup I_1'$ and the region of interest $I_1 + E$ modulo q is as in Figure 5.

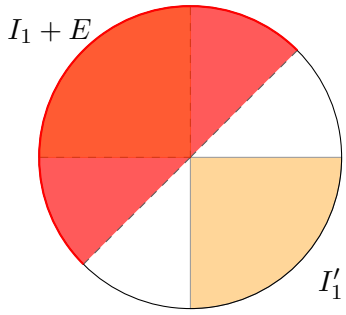


Figure 5: The region $I_1 + E$.

If $w \in I_1 + E$, then v must be in I_1 to have $|w - v| < \frac{q}{8}$, and so $\lfloor v \rceil_2 = 0$ by the definition of I_1. If $w \notin I_1 + E$, then v must be in I_1' and so $\lfloor v \rceil_2 = 1$.

We can extend the rounding and reconciliation functions to any basis for the cyclotomic ring R by simply applying the functions coordinate-wise to the $\mathbb{Z}_q$-coefficients of the inputs in the chosen basis. Formally, if $Y = \{y_i\} \subset R$ is a basis for R and $v = \sum_j v_j y_j \in R_q$ for coefficients $v_j \in \mathbb{Z}_q$, then $\lfloor v \rceil_2 := \sum_j \lfloor v_j \rceil_2 \cdot y_j \in R_2$ and similarly for $\langle v \rangle_2$. The reconciliation function can be extended to $\mathrm{rec} : R_q \times \{0,1\}^n \to R_2$ by setting $\mathrm{rec}(w, b) := \sum_j \mathrm{rec}(w_j, b_j) \cdot y_j$ where $w = \sum_j w_j y_j$ and $b = b_0 b_1 \ldots b_n \in \{0,1\}^n$.

4 Main algorithm and variants

In [21], Peikert presents a ring-LWE based Diffie-Hellman-type key exchange algorithm that is provably passively secure, and then shows how to use this basic primitive to construct provably actively secure key exchanges and provably secure authenticated key exchanges. We will present each of these variants in detail for completeness.

4.1 Ring-LWE ephemeral Diffie-Hellman

In this section we present an efficient *key encapsulation mechanism* (KEM1) or key exchange that is provably secure against passive attacks. The system is constructed using the reconciliation technique from Section 3 to allow the two parties to derive an ephemeral key from a pseudorandom value in R_q on which they approximately agree. Previous ring-LWE cryptosystems needed to have a ciphertext consisting of a pair of ring elements, but the reconciliation mechanism reduces this to a single ring element and a bit string of length n.

The parameters for KEM1 will be (m, q, a, χ) where

- $m = 2^\ell$, and so $n = \phi(m) = m/2$;
- q is an odd prime integer such that $q \equiv 1 \pmod{m}$;
- a is a fixed element of the ring $R_q := \mathbb{Z}/q\mathbb{Z}[\zeta_m] \cong \mathbb{Z}_q[x]/\langle \Phi_m(x) \rangle$; and
- χ is an error distribution over R_q.

We will denote by $\textbf{Sample}(\chi)$ the function used to sample error terms from χ. For example, this could be $\textbf{GaussSample}(\sigma)$ for the discrete Gaussian with standard deviation σ, or $\textbf{UniformSample}(B)$ for the uniform distribution on the interval $\{-B, \ldots, B\}$.

Key generation samples error terms from χ to produce a private key (s_0, s_1), and combines it with the generator a to form the public key. Note that decapsulation only needs s_1 so we can discard s_0.

KEM1.Generate
Input: Domain parameters (m, q, a, χ)
Output: Private key s_1 and public key b
$s_0 \leftarrow \textbf{Sample}(\chi)$
$s_1 \leftarrow \textbf{Sample}(\chi)$
$b \leftarrow s_1 \cdot a + s_0 \in R_q$

Encapsulation[6] generates a public key for the second party and takes the received public key b to produce a shared key μ and a ciphertext c. The shared keys will belong to $\mathcal{K} = \{0,1\}^n$ and their corresponding encapsulations will belong to $\mathcal{C} = R_q \times \{0,1\}^n$. Here we identify R_2 with bit strings in $\{0,1\}^n$ in the obvious way.

KEM1.Encapsulate
Input: Recipient's public key b
Output: Shared key μ and encapsulation c
$e_0 \leftarrow \textbf{Sample}(\chi)$
$e_1 \leftarrow \textbf{Sample}(\chi)$
$e_2 \leftarrow \textbf{Sample}(\chi)$
$u \leftarrow e_0 \cdot a + e_1 \in R_q$
$v \leftarrow e_0 \cdot b + e_2 \in R_q$
$\bar{v} \leftarrow \textbf{RandomizedRound}(v)$
$\mu \leftarrow \lfloor \bar{v} \rceil_2 \in \{0,1\}^n$
$c \leftarrow (u, \langle \bar{v} \rangle_2) \in R_q \times \{0,1\}^n$

[6]The reader may notice that the encapsulation and decapsulation functions in [21] include a term g which ours omit. In [20], g is defined as $g := \prod_p (1 - \zeta_p)$ for all *odd* primes p dividing m. Thus in the special case of $m = 2^\ell$, $g = 1$ and can safely be omitted.

Decapsulation forms the approximate shared secret

$$w = u \cdot s_1 = e_0 \cdot a \cdot s_1 + e_1 \cdot s_1 \approx e_0 \cdot a \cdot s_1 + e_0 \cdot s_0 + e_2 = e_0 \cdot b + e_2 = v$$

and uses the reconciliation mechanism from Section 3 to derive the shared secret key μ.

KEM1.Decapsulate
Input: Recipient's private key s_1 and encapsulation c Output: Shared key μ
$(u, v') \leftarrow c$ $w \leftarrow u \cdot s_1 \in R_q$ $\mu \leftarrow \mathbf{Rec}(w, v')$

Lemma 1 (Lemma 4.1 from [21])**.** The KEM1 is IND-CPA secure, assuming the hardness of R-DLWE$_{q,\chi}$ given two samples.

4.2 Actively secure Ring-LWE key exchange

In this section we present the transformation of KEM1 into an secure encryption scheme (PKC2) which is secure under chosen-ciphertext attack (*i.e.,* IND-CCA secure). In order to do so, we will utilise the second Fujisaki-Okamoto transformation [7] which converts any passively secure encryption scheme into one which is actively secure in the random-oracle model.[7]

To apply the Fujisaki-Okamoto transformation to KEM1 there are two main changes that we need to make. Firstly, the transformation uses an encryption scheme where the key is an explicit input while KEM1 derives the key as part of the encapsulation. However, as we shall see it is straightforward to convert a key encapsulation mechanism into an encryption scheme, by using the derived key to mask the plaintext. Secondly, we need the derived encryption scheme to be deterministic. This means that the error terms must be reproducable and so the sampling function needs to be modified to use the output of a deterministic pseudorandom number generator **PRG**.

The domain parameters for PKC2 will be (m, q, a, χ, N, ℓ) where

- $m = 2^\ell$, and so $n = \phi(m) = m/2$;
- q is an odd prime integer such that $q \equiv 1 \pmod{m}$;
- a is a fixed element of the ring $R_q := \mathbb{Z}/q\mathbb{Z}[\zeta_m] \cong \mathbb{Z}_q[x]/\langle \Phi_m(x) \rangle$;
- χ is an error distribution over R_q;

[7] The FO transform requires the use of the random oracle model. However, the basic unauthenticated KEM is proved secure in the standard model, and can be combined into other protocols in the standard model; see [1] for a proof of security in the standard model incorporating Peikert's scheme into TLS.

- $N \geq 0$ is the bit length of the plaintext; and
- ℓ is the length of the seed for the pseudorandom generator **PRG**.

We will denote by **Sample**$(\chi; \mathbf{PRG})$ the sampling function modified to use output from **PRG** and by **PRG**.**Seed**$(seed)$ the seeding function for **PRG**.

Key generation for the actively secure protocol PKC2 does not require deterministic sampling so is identical to key generation for KEM1.

PKC2.Generate
Input: Domain parameters (m, q, a, χ) Output: Private key s_1 and public key b
$s_0 \leftarrow \mathbf{Sample}(\chi)$ $s_1 \leftarrow \mathbf{Sample}(\chi)$ $b \leftarrow s_1 \cdot a + s_0 \in R_q$

The encryption function PKC2 chooses a random value σ, and uses this to both mask the message and seed the pseudorandom generator. The seed σ is then masked by the key derived from the deterministic KEM1. This requires a pair of hash functions $\mathbf{G} : \{0,1\}^n \to \{0,1\}^N$ and $\mathbf{H} : \{0,1\}^{n+N} \to \{0,1\}^\ell$.

PKC2.Encrypt
Input: Recipient's public key b and plaintext m Output: Ciphertext (c_1, c_2)
$\sigma \xleftarrow{\$} \{0,1\}^n$ $c_1 \leftarrow \mathbf{KEM1}.\mathbf{Encrypt}(b, \sigma, \mathbf{H}(\sigma \| m))$ $c_2 \leftarrow \mathbf{G}(\sigma) \oplus m \in \{0,1\}^N$

As mentioned above, PKC2 requires a deterministic public key encryption routine. Thus we will modify KEM1 to create a deterministic encryption function which will be used to wrap the seed value $\sigma \in \{0,1\}^n$ using the"coins" $r = H(\sigma||m)$.

KEM1.Encrypt
Input: Recipient's public key b, plaintext $\sigma \in \{0,1\}^n$, and seed $r \in \{0,1\}^\ell$
Output: Ciphertext c_1
PRG.**Seed**(r)
$e_0 \leftarrow$ **Sample**$(\chi;$ **PRG**$)$
$e_1 \leftarrow$ **Sample**$(\chi;$ **PRG**$)$
$e_2 \leftarrow$ **Sample**$(\chi;$ **PRG**$)$
$u \leftarrow e_0 \cdot a + e_1 \in R_q$
$v \leftarrow e_0 \cdot b + e_2 \in R_q$
$\bar{v} \leftarrow$ **RandomizedRound**(v)
$c_1 \leftarrow (u, \langle \bar{v} \rangle_2, \lfloor \bar{v} \rceil_2 \oplus \sigma) \in R_q \times \{0,1\}^n \times \{0,1\}^n$

Decryption uses the reconciliation mechanism from Section 3 to recover the seed and unmask the message. However, to provide protection against chosen ciphertext attacks it is necessary to check that c_1 was validly generated from the seed value σ.

PKC2.Decrypt
Input: Recipient's private key s_1, and ciphertext (c_1, c_2)
Output: Plaintext m or $\perp$
$(u, v', \sigma') \leftarrow c_1$
$w \leftarrow u \cdot s_1 \in R_q$
$\hat{\sigma} \leftarrow \sigma' \oplus$ **Rec**$(w, v') \in \{0,1\}^n$
$m \leftarrow$ **G**$(\hat{\sigma}) \oplus c_2$
$\hat{c_1} \leftarrow$ **KEM1**.**Encrypt**$(b, \hat{\sigma},$ **H**$(\hat{\sigma}\|\|m))$
if $\hat{c_1} = c_1$
return m
else
return $\perp$

Theorem 2 (Theorem 5.1 from [21])**.** PKC2 is IND-CCA secure, assuming that PKC is passively *one-way* secure, PRG is a secure pseudorandom generator, and G and H are modelled as random oracles.

4.3 Ring-LWE authenticated key exchange

Key exchange protocols, such as Diffie-Hellman [6], are secure against passive adversaries, but are susceptible to *man-in-the-middle* attacks where an active adversary can modify, insert or delete messages. An *authenticated key exchange* (AKE) protocol provides mutual authentication of the two parties' identities and protects the integrity of the enitre message flow. In this section, we give a protocol for authenticated key exchange derived from the passively secure

KEM1 together with a signature; we will use the signature construction from [23] as it seems to fit best with KEM1.

4.3.1 Signature

We begin by briefly presenting the signature from [23]. The signature scheme is parametrized by $(m, q, a, \chi, \chi', \kappa)$ where

- $m = 2^\ell$ so that $n = \phi(m) = m/2$;
- q is an odd prime integer such that $q \equiv 1 \pmod{m}$;
- a is a fixed element of the ring $R_q := \mathbb{Z}/q\mathbb{Z}[\zeta_m] \cong \mathbb{Z}_q[x]/\langle \Phi_m(x) \rangle$;
- χ and χ' are error distributions over R_q; and
- κ is a bound on the size of the ring elements in the signature.

Key generation uses the first distribution χ to produce the static signing key in essentially the same way as for KEM1 and PKC2, except that both s_0 and s_1 will be needed to sign a message.

SIG.Generate
Input: Domain parameters (m, q, a, χ)
Output: Private key (s_0, s_1) and public key t
$s_0 \leftarrow \mathbf{Sample}(\chi)$
$s_1 \leftarrow \mathbf{Sample}(\chi)$
$t \leftarrow s_1 \cdot a + s_0 \in R_q$

Signing generates an ephemeral key from the second distribution χ' and uses this to construct two "small" elements of the ring that are related by the signing key and the hash of the message. This means that we will need a hash function $\mathbf{H} : \{0,1\}^* \to R_q$ that maps onto "small" elements of R_q, say as drawn from a third distribution χ'' over R_q. Further, to avoid leaking information via the signatures we will need a *rejection sampling* algorithm $\mathbf{Reject} : R_q^4 \to (0, 1)$ which gives the probability with which the putative signature will be rejected; see Section 4 of [23].

SIG.Sign
Input: Private key (s_0, s_1) and message μ
Output: Signature σ
$y_0 \leftarrow \textbf{Sample}(\chi')$
$y_1 \leftarrow \textbf{Sample}(\chi')$
$c \leftarrow \textbf{H}(a \cdot y_1 + y_0 \| \mu)$
$z_0 \leftarrow s_0 \cdot c + y_0$
$z_1 \leftarrow s_1 \cdot c + y_1$
if $\textbf{Reject}(z_0, z_1, s_0 \cdot c, s_1 \cdot c)$
start again
else
return $\sigma \leftarrow (z_0, z_1, c)$

Verification consists of checking that the elements z_0 and z_1 are small enough and satisfy the relation

$$a \cdot z_1 + z_0 - t \cdot c = a \cdot y_1 + y_0.$$

Note that since $a \cdot y_1 + y_0$ is not part of the signature we instead need to use c to check that

$$\textbf{H}(a \cdot y_1 + y_0 \| \mu) = \textbf{H}(a \cdot z_1 + z_0 - t \cdot c \| \mu).$$

SIG.Verify
Input: Public key t, signature σ and message μ
Output: True or false
$(z_0, z_1, c) \leftarrow \sigma$
if $\|z_0\| \leq \kappa$, $\|z_1\| \leq \kappa$ and $c = \textbf{H}(a \cdot z_1 + z_0 - t \cdot c \| \mu)$
return true
else
return false

4.3.2 Sign-and-MAC protocol

The signature-based authentication mode in Internet Key Exchange (IKE) is designed around the "SIGn-and-MAc" (SIGMA) family of protocols specified by Krawczyk in [16]. The Krawczyk paper gives an excellent in-depth, yet still quite readable, explanation of the design considerations for AKE protocols and other associated issues.

In [21], Peikert presents a protocol Σ_0' which is a slight generalisation of the Σ_0 protocol from [2], which itself uses the SIGMA design [16] underlying the IKE protocol. The difference comes from using ring-LWE's "noisy" Diffie-Hellman-like key-agreement, which is, however, readily incorporated into the initiator and responder roles. As such, an existing Diffie-Hellman-based implementation can be modified to use the new protocol with relative ease and with few, if any,

structural changes. Thus, an implementation of the new protocol can work alongside alternative existing protocols facilitating backwards compatibility and user choice.

The parameters of the Σ_0' protocol are

- A digital signature scheme SIG;
- A key exchange or key encapsulation KEM with key space $\mathcal{K}$;
- A pseudorandom function $F : \mathcal{K} \times \{0,1\} \to \mathcal{K}'$; and
- A message authentication code MAC with key space $\mathcal{K}'$ and message space $\{0,1\}^*$.

A successful execution of the protocol outputs a secret key in $\mathcal{K}'$.

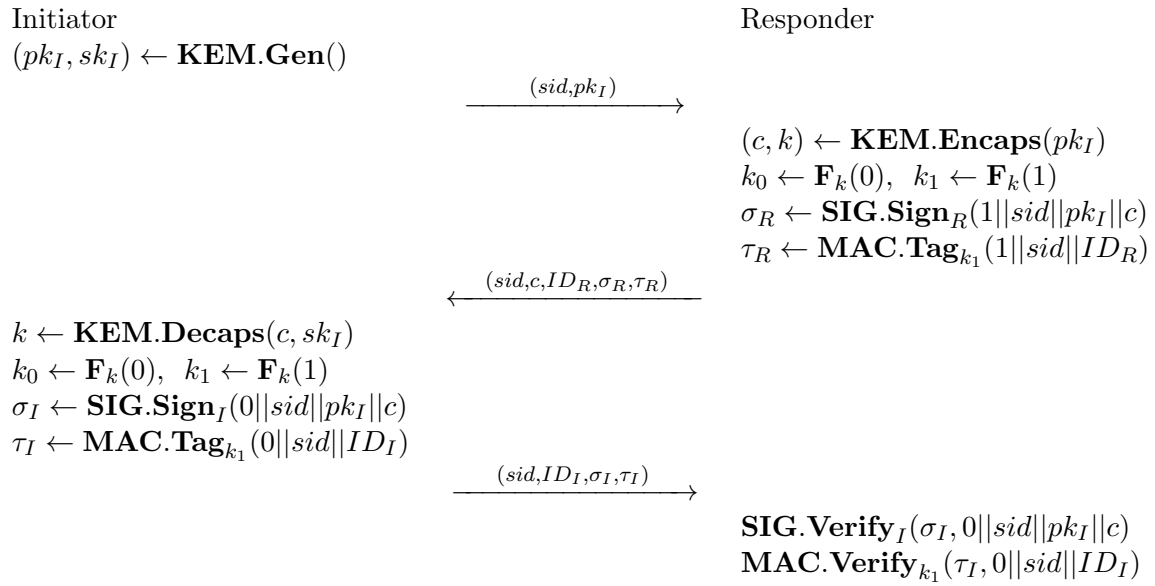


Figure 6: Sign-and-MAC protocol.

We assume that the initiator and responder have static signing keys whose corresponding public verifications are bound to their identities ID_I and ID_R in, for example, certificates issued by a trusted certificate authority. The protocol Σ_0' proceeds as follows:

1. **Start message** $(I \to R) : (sid, pk_I)$.

   The initiator generates an ephemeral keypair and sends the public key pk_I and session identifier sid to the responder. The session identifier must be distinct from all previous sessions initiated by ID_I.

2. **Response message** $(R \to I) : (sid, c, ID_R, \mathbf{SIG.Sign}_R(1, sid, pk_I, c), \mathbf{MAC.Tag}_{k_1}(1, sid, ID_R))$.

   The recipient uses the KEM to generate a master key k, and derives a session key k_0 and a MAC key k_1 from it. The recipient then signs the initiator's public key along with the encapsulated master key, and authenticates their own identity using the MAC.

3. **Finish message** $(I \to R) : (sid, ID_I, \mathbf{SIG.Sign}_I(0, sid, pk_I, c), \mathbf{MAC.Tag}_{k_1}(0, sid, ID_I))$.

   The initiator decapsulates to recover the session key k_0 and MAC key k_1, and verifies the signature and authenticated identity. If either verification fails, the session is aborted. Otherwise, the initiator signs their own public key and the encapsulated master key, and authenticates their identity.

4. **Responder completion**:

   The recipient verifies the signature and authenticated identity. If either verification fails, the session is aborted.

Theorem 3. The Σ_0' protocol is SK-secure in the post-specified peer model of [2], assuming that SIG and MAC are existentially unforgeable under chosen-message attack, that KEM is IND-CPA secure, and that F is a secure pseudorandom function.

Proof: See Theorem 6.1 from [21], which in turn references the proof from [2]. □

5 Analysis

5.1 Instantiating the parameters

We wish to choose concrete parameters for testing as practical candidates. Our aim shall be to reduce the public key length while maintaining security and correctness. In order to do so, we refer to the guidance in Section 4.4 of [21]. To begin we consider how to set the lattice dimension $n = \phi(m)$ and the modulus q. For the special case of $m = 2^\ell$, the only practical choices for n are 512 and 1024, and so we will consider each along with a toy of dimension 256. The security analysis in [21] gives a practical bound on the size of q of $q \approx n^{(3/2)}$ (see Section 4.4 of [21]). Thus in order to satisfy both this practical bound and the proof of security for ring-LWE, we begin with $q \geq n^{(3/2)}$ and search for q a prime which is congruent to 1 modulo m and as close to $n^{(3/2)}$ as failure rates will allow in order to provide the best practical security.

Security	n	q	public key size
toy	256	15361	3584 bits
regular	512	25601	7680 bits
high	1024	40961	16384 bits

Figure 7: Our choices of parameters. Public key size computed as $n \cdot \lceil \log_2 q \rceil$ bits.

The authors of [1] mention that their choice of q is conservative in that it provides a very large margin for correctness and can be reduced. If we compare our parameters from Table 7 to those of [1], we can see that by decreasing q, our parameter choices offer increased security with smaller bandwidth, a feature that is unthinkable in traditional public key systems but can be enjoyed in lattice-based schemes.[8]

We have chosen to keep q prime of the form $q \equiv 1 \pmod{m}$ to maintain consistency with the proof of security for ring-LWE. By following the analysis in [4, 17] we feel confident in saying that our regular security parameters will provide at least 128-bits of security and our high security parameters will provide over 256-bits of security.[9] We also note that the maximum length of the data field of a packet sent over Ethernet is 1500 bytes. If public keys are too large, they will not fit into a single packet, increasing the possibility of key establishment failures based on transmission problems. Our 128-bit secure parameters will fit into a single Ethernet packet, unlike the 128-bit secure parameters proposed by [1] which would require three maximum sized Ethernet packets. Moreover, even our 256-bit secure parameters fit in two Ethernet packets, thus providing greater security with less data overhead than the choice in [1]. Many important applications, including Domain Name Server (DNS) and much voice and video traffic, use User Datagram Protocol (UDP) instead of Transmission Control Protocol (TCP). Unlike TCP, UDP has no concept of acknowledgement, retransmission, or timeout, and thus the more Ethernet packets required to transmit a public key, the greater the potential for key exchange difficulties.

In the next section, we show that this choice of parameters still offers an extremely low failure rate, leading to what we believe is an extremely attractive and practical parameter option.

5.2 Correctness of the scheme

For each of the three systems described in Section 4, we need each coefficient of the error term to be bounded by $\frac{q}{8}$, that is, we must have that $||s_0 e_0 + e_2 - s_1 e_1||_\infty < \lfloor \frac{q}{8} \rfloor$ in order to be guaranteed correctness of the algorithms. This comes from Claim 2 of Section 3, being the maximum difference between shared secret values such that the reconciliation function still produces the correct key. To conduct a direct analysis of the growth of the coefficients of the error term, we will begin by considering what multiplication of elements looks like.

For $m = 2^\ell$, multiplication is achieved by negacyclic convolution. We recall that $R := \mathbb{Z}[\zeta] = \mathbb{Z}[x]/\langle \Phi_m(x) \rangle = \mathbb{Z}[x]/\langle x^n + 1 \rangle$ for ζ a primitive mth root of unity and $n = \phi(m) = m/2$. If we

[8] As the modulus of reduction q is reduced, errors become larger *relative to* the size of the problem, thus making the lattice problem harder.

[9] Lepoint and Naehrig [17] improve on the distinguishing attacks considered in [18, 27] by embedding the Ring-LWE problem in an LWE problem and allowing the number of samples to vary. Using the BKZ 2.0 simulator from [3], and the estimated cost of pruned enumeration from [17], we determine the number of samples and block size which minimise the cost of the attack. We found that in order to achieve a distinguishing advantage greater than $\epsilon = 2^{-128}$ with the $n = 512$ parameters, enumeration would need to visit at least 2^{128} nodes. Similarly, for a distinguishing advantage greater than $\epsilon = 2^{-256}$ with the $n = 1024$ parameters, enumeration would need to visit substantially more than 2^{256} nodes.

multiply $\zeta^i \cdot \zeta^j$ then we just need to reduce modulo the equation $\Phi_m(x) = 1 + x^n$, so $x^n = -1$. Let $a \in R_q$ be written as $\sum_{i=0}^{n-1} a_i \zeta^i$, and similarly let $b = \sum_{j=0}^{n-1} b_j \zeta^j \in R_q$. The product of any two elements $a, b \in R_q$ is given by

$$\begin{aligned}
\sum_{i=0}^{n-1} a_i \zeta^i \cdot \sum_{j=0}^{n-1} b_j \zeta^j &= \sum_{k=0}^{2n-2} \big(\sum_{i+j=k} a_i b_j \big) \zeta^k \\
&= \sum_{k=0}^{n-1} \big(\sum_{i+j=k} a_i b_j \big) \zeta^k - \sum_{k=n}^{2n-2} \big(\sum_{i+j=k} a_i b_j \big) \zeta^{(k-n)} \\
&= \sum_{k=0}^{n-1} \big(\sum_{i=0}^{k} a_i b_{k-i} - \sum_{i=k+1}^{n-1} a_i b_{n+k-i} \big) \zeta^k
\end{aligned}$$

Thus we see that each coefficient of a product of two elements consists of a sum of n terms, each term being the product of two independent random variables.

If error terms are generated by sampling a Gaussian distribution, then each of the independent random variables a_i, b_j comes from a discretised Gaussian distribution centered at 0 and with variance σ^2. By invoking (a generalisation of) the Central Limit Theorem[10], we can say that each of these coefficients is well approximated by a Gaussian distribution centered at 0 with variance $n\sigma^4$, and to approximate the coefficients of $s_0 e_0 + e_2 - s_1 e_1$ we can consider a Gaussian distribution centered at 0 with variance $2n\sigma^4 + \sigma^2$. As the reconciliation function will only fail if the absolute value of one or more of these coefficients exceeds $q/8$, we will estimate the probability of that occurring for each of our parameter sets. Fixing $\sigma = 8/\sqrt{(2\pi)}$, if we use the parameters $n = 512, q = 25601$, we find a probability of a coefficient of the error term exceeding the $q/8$ bound is less than 2^{-71}. For $n = 1024$ and $q = 40961$, the probability is less than 2^{-91}.

If we create the private keys and error vectors by drawing coefficients of the power basis uniformly at random from $\{-B, \ldots, 0, \ldots, B\}$, then it is fairly simple to directly analyse the correctness of the system and compute specific probabilities of failure. In this case, each of the a_i and b_j are independent random variables following a uniform distribution over the set $\{-B, \ldots, 0, \ldots, B\}$. The product of two such random variables is itself a random variable over $\{-B^2, \ldots, 0, \ldots, B^2\}$ and the sum of sum variables yields a convolution of their distributions.[10] Thus the coefficients of a product $a \cdot b$ are random variables with support $\{-nB^2, \ldots, 0, \ldots, nB^2\}$. We can calculate the probability that $||s_0 e_0 + e_2 - s_1 e_1||_\infty < \lfloor \frac{q}{8} \rfloor$ by considering the correct convolution of probability distributions. For the parameter set $n = 512, q = 25601$, we find a probability of a coefficient of the error term exceeding the $q/8$ bound is $2^{-75.72}$. For $n = 1024$ and $q = 40961$, the probability is $2^{-96.11}$. If we wish to consider the probability of failure of the key exchange, we must recognise that there are n coefficients, each of which could exceed the $q/8$ bound. However, the $q/8$ bound is only tight

[10] Each of the a_i and b_j is independent; however, when summing up the products of such terms, a given a_i or b_j will appear on average in two such products, creating a weak dependence between the summands.

for coefficients falling at the boundary of a quadrant: $\{0, \pm\lfloor q/4 \rceil, \lfloor q/2 \rceil\}$. Most coefficients will not lie on or near these boundaries, which will lead to a higher permissible bound. We calculate the probability that a failure of the key exchange will occur to be less than $2^{-74.37}$ for $n = 512$ and to be less than $2^{-94.11}$ for $n = 1024$. Thus we can see that these parameters provide more than sufficient soundness for practical applications.

5.3 Timings

We have implemented the Ring-LWE ephemeral Diffie-Hellman key encapsulation mechanism of Section 4.1 in the lower level language `C` in order to assess its performance. This code is available on the GitHub repository at `https://github.com/vscrypto/ringlwe_power` along with its benchmarking routines. We are grateful to the authors of [1] for the code accompanying their paper, from which we borrow the overall structure as well as the functions for performance timing, random number generation, and Gaussian sampling. Timings of each operation are presented in Tables 8 and 9, made using a 1.9GHz Intel Core i5 4300U[11] with code compiled using `gcc` version 4.9.2 with the `-O3` optimisation flag.

Sampling	**Gaussian**			**Uniform**		
n	**256**	**512**	**1024**	**256**	**512**	**1024**
Sample	169400	333500	663400	12300	21600	41800
FFT Forward	19900	43700	95300	20800	45000	95300
FFT Backward	20700	45400	98800	21300	45900	98800
FFT Multiply	63100	138300	300500	63200	138300	300900
Round $\lfloor . \rceil_2$ and Cross-Round $\langle . \rangle_2$	5300	7900	12900	5300	7700	12900
Reconcile Rec(., .)	1100	2100	4200	1000	2100	4200
KEM1.Generate	377100	757200	1527400	66300	135200	280600
KEM1.Encapsulate	577900	1156800	2331300	111800	222100	457600
KEM1.Decapsulate	24400	52900	113800	24500	52800	113900
Total runtime	979400	1967000	3972500	202700	410200	852200

Figure 8: Average cycle count of operations

[11] Processor has two cores which can each run two threads at 1.9GHz using Hyper Threading. The clock rate on a single thread can be increased to 2.9GHz using Turbo Boost, in the absence of competition from another thread on that core. Performance tests were run as a single thread so Turbo Boost could be taken advantage of.

Sampling	**Gaussian**			**Uniform**		
n	**256**	**512**	**1024**	**256**	**512**	**1024**
Sample	67	133	265	4	8	16
FFT Forward	8	18	38	8	18	38
FFT Backward	8	18	39	8	18	39
FFT Multiply	25	55	120	25	55	120
Round $\lfloor . \rceil_2$ and Cross-Round $\langle . \rangle_2$	2	3	5	2	3	5
Reconcile Rec(., .)	0.4	0.8	1	0.4	0.8	1
KEM1.Generate	151	303	612	26	54	112
KEM1.Encapsulate	231	463	934	44	89	183
KEM1.Decapsulate	9	21	45	9	21	45
Total runtime	391	787	1591	79	164	340

Figure 9: Average operation time in micro seconds ($1\ \mu s = 10^{-6}\ s$)

In comparison, the code of [1], which uses Gaussian sampling with $n = 1024$, $q = 2^{32} - 1$ to achieve the 128-bit security level, has an equivalent total runtime of 2180 μs (5444000 clock cycles) on our machine under the same compilation conditions. Our 256-bit secure $n = 1024$ parameter set with Gaussian sampling outperforms this with a total runtime of 1591 μs. If we compare to our 128-bit secure $n = 512$ parameter set, with Gaussian sampling we outperform [1] by a factor of nearly three with a total runtime of 787 μs while with uniform sampling, this outperforms [1] by a factor of 13 with a total runtime of 164 μs. Given that [1] integrates their key exchange into OpenSSL and finds that the performance is acceptable for TLS, we are also able to make this claim for any of our parameter sets.

The dominant operations of the key exchange are sampling and multiplication of ring elements. We found that the Gaussian sampling function took over 16 times longer than the uniform sampling. Since a total of 5 samples are required for the key exchange, this makes Gaussian sampling responsible for about 83% of the total runtime compared to about 24% for uniform sampling. While we readily acknowledge that significant work has been done to speed up the computation of discrete Gaussian distributions, we think it unlikely that any amount of optimisation will ever make Gaussian sampling in line with the speed of uniform sampling. Moreover, many of the fastest Gaussian sampling routines require a significant amount of memory and utilise a large number of random bits as input. Thus we believe that uniform sampling will provide the best combination of practical security and efficiency in real world settings.

We optimise arithmetic on ring elements using the fast fourier transform as described in Section 2.3.1, and keeping elements in the Fourier domain where possible, because both multiplication and addition are pointwise. Thus we transmit the recipient's public key b and the sender's public key u in the Fourier domain, and store the sender's private key s_1 and recipient's private key e_0 in the Fourier domain. This way we never perform the FFT Multiply operation directly, which is the sequence of two FFT Forward operations, a pointwise modular multiply,

and an FFT Backward operation. For our implementation of the fast fourier transform we perform a Discrete Weighted Fourier Transform using the decimation-in-frequency algorithm of Gentleman-Sande [8] in the forward direction, and the decimation-in-time algorithm of Cooley-Tukey [5] in the reverse.

6 Conclusions

The ring learning with errors problem is a promising cryptographic primitive that is believed to be resistant to attacks by quantum computers. The decision ring-LWE problem naturally leads to a passively secure Diffie-Hellman-like unauthenticated key exchange protocol, which can readily be extended to an actively secure version and an authenticated version. We have examined these key exchange mechanisms, and provided both practical analysis and more practical parameter choices than were previously available. We provided example implementations in `sage` in Appendix A and in `C` at `https://github.com/vscrypto/ringlwe_power`. Using the `C` implementation, we have demonstrated that our parameter choices outperform [1] by as much as a factor of 13 for equivalent security and can thus claim, like [1], that this is acceptable runtime for TLS. Furthermore, our 128-bit secure public keys fit into a single Ethernet packet, and our 256-bit secure public keys require two, thus providing greater reliability of transmission over the Internet and less data overhead.

We plan to continue and expand upon this work. In [20] and [21], the authors develop the theory in the most general setting for the form of m. We would like to make a careful examination of other specialised forms of m, starting with the prime form, to see if any practical benefit can be derived from selecting an m that is not a power of two. We also plan to explore ring-LWE-based signature schemes.

7 Acknowledgments

We are grateful to our associate Arjun Chopra for support and advice on the paper and implementation.

References

[1] J. W. Bos, C. Costello, M. Naehrig, and D. Stebila. Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. `http://eprint.iacr.org/2014/599`.

[2] R. Canetti and H. Krawczyk. Security analysis of IKE's signature-based key-exchange protocol. In *CRYPTO'02*, pages 143-161. 2002. Full version at `http://eprint.iacr.org/2002/120`.

[3] Y. Chen and P. Q. Nguyen. BKZ 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT*, volume 7073 of LNCS, pages 1-20. Springer, 2011.

[4] Y. Chen and P. Q. Nguyen. BKZ 2.0: Better lattice security estimates. 2013. Full version. Available at `http://www.di.ens.fr/ ychen/research/Full_BKZ.pdf`.

[5] J. Cooley and J. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Comp.*, 19:297-301, 1965.

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

[7] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *CRYPTO'99*, pages 537-554. 1999.

[8] W. Gentleman and G. Sande. Fast Fourier transforms - for fun and profit. In *Proceedings of the AFIPS*, volume 29, pages 563-578, 1966.

[9] T. Güneysu, V. Lyubashevsky, and T. Pöppelmann. Practical Lattice-Based Cryptography: A Signature Scheme for Embedded Systems. In E.Prouff and P. Schaumont, editors, *CHES 2012*, LNCS 7428, pages 530-547. Springer, 2012.

[10] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). RFC 2409 (Proposed standard), November 1998. Obsoleted by RFC 4306, updated by RFC 4109.

[11] R. Housley. Use of the RSAES-OAEP Key Transport Algorithm in Cryptographic Message Syntax (CMS). RFC 3560 (Proposed Stadard), July 2003.

[12] C. Kaufman. Internet Key Exchange (IKEv2) Protocol. RFC 4306 (Proposed Standard), December 2005. Obsoleted by RFC 5996, updated by RFC 5282.

[13] C. Kaufman, P. Hoffman, Y. Nir, and P. Eronen. Internet Key Exchange Protocol Version 2 (IKEv2). RFC 5996 (proposed Standard), September 2010. Updated by RFCs 5998, 6989.

[14] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. RFC 2401 (Proposed Standard), November 1998. Obsoleted by RFC 4301, updated by RFC 3168.

[15] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), December 2005. Updated by RFC 6040.

[16] H. Krawczyk. SIGMA: The 'SIGn-and-MAc' approach to authenticated Diffie-Hellman and its use in the IKE-protocols. In *CRYPTO'03*, pages 400-425. 2003. Full version at `http://webee.technion.ac.il/ hugo/sigma.html`.

[17] T. Lepoint and M. Naehrig. A comparison of the homomorphic encryption schemes FV and YASHE. In David Pointcheval and Damien Vergaud, editors, *AFRICACRYPT*, volume 8469 of *Lecture Notes in Computer Science*, pages 318-335. Springer, 2014.

[18] R. Linder and C. Peikert. Better key sizes (and attacks) for LWE-based encryption. In *CT-RSA'11*, pages 319-339, 2011.

[19] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. *Journal of the AMC*, 60(6):43:1-43:35, November, 2013. Preliminary version in *EUROCRYPT*, pages 1-23, 2010.

[20] V. Lyubashevsky, C. Peikert, and O. Regev. A Toolkit for Ring-LWE Cryptography. In *EUROCRYPT'13*, pages 35-54. 2013.

[21] C. Peikert. Lattice Cryptography for the Internet. In Michele Mosca, editor, *Proc. 6th International Conference on Post-Quantum Cryptography (PQCrypto) 2014*, LNCS 8772, pages 197-219. Springer, 2014. Full version available at `http://eprint.iacr.org/2014/070`

[22] V. Lyubashevsky. Fiat-Shamir With Aborts: Applications to Lattice and Factoring-Based Signatures. In M. Matsui, editor, *Asiacrypt 2009*, volume 5912 of LNCS,pages 598-616. Springer, Dec. 2009.

[23] V. Lyubashevsky. Lattice Signatures Without Trapdoors. In *EUROCRYPT'12*, pages 738-755, 2012. `http://eprint.iacr.org/2011/537`

[24] D. Micciancio and O. Regev. Lattice-based cryptography. In Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmaen, editors, *Post-Quantum Cryptography*, pages 147-191. Springer Berlin Heidelberg, 2009.

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

[26] J. Randall, B. Kaliski, J. Brainard, and S. Turner. Use of the RSA-KEM Key Transport Algorithm in the Cryptographic Message Syntax (CMS). RFC 5990 (Proposed Standard), September 2010.

[27] J. van de Pol and N. P. Smart. Estimating key sizes for high dimensional lattice-based systems. In *IMA Int. Conf.*, pages 290-303, 2013.

A Code

```
"""

An implementation of the Peikert scheme in sage.

"""

m = 2048; q = 40961; n = 1024;
m = 1024; q = 25601; n = 512;


Phi = cyclotomic_polynomial(m)    # Phi is the mth cyclotomic polynomial
Pq = PolynomialRing(Integers(q),'xx'); xx = Pq.gen()
Rq = -Pq.quotient(Phi,'zz'); zz = Rq.gen()        # The ring R_q used in R-LWE
B = 5                             # parameter used for the width of uniform sampling
sigma =  8/sqrt(2*pi)             # parameter for the width of the Gaussian sampling


##################################
# Sampling Functions
##################################

T = []                            # Precompute table for Gaussian  width sigma
T.append(2^189)
for i in range(1,52):
   sum = 0
   for j in range(1,i+1):
      sum += 1/8*exp(-j^2/(2*sigma^2))
   T.append(floor(2^192*(1/8+2*sum)))
T.append(2^192)


def Gauss_sample_Rq(sigma):       # discretised Gaussian sampling on Rq
   tmp = []
   for i in range(n):
      t = randrange(2^189,2^192)
      j = 0
      while t < T[j]:
         j +=1
      j -= 1
      sign = randrange(2)
      if sign == 0:
         sign = -1
      tmp.append(sign*j)
   return tmp


def uniform_sample_Rq(B):          # create element of Rq sampled uniformly from {-B..B}
   tmp = []
   for i in range(n+1):
      tmp.append(randrange(-B,B+1))
   return tmp



##################################
# Rounding functions
##################################
a = Rq.random_element()
```

```
def randomized_round(a):          # input an element of Rq
   """ Randomized rounding for creating the bitstring to send inside the protocol
       (simplification eliminating the need to map into 2q)  """
   coeff=Rq(a).list(); round_coeff = [];
   for i in range(n):
      if coeff[i] == 0 :
         tmp=randrange(0,2)
         if tmp == 0 :
            round_coeff.append(1)
         else:
            round_coeff.append(q-1)
      elif coeff[i] == (q-1)/4 :
         tmp=randrange(0,2)
         if tmp == 0 :
            round_coeff.append(coeff[i])
         else:
            round_coeff.append(coeff[i]+1)
      else:
         round_coeff.append(coeff[i])
   return round_coeff


def modular_round(a):
   """ Input randomized rounded a in Rq as a vector of coefficients. """
   round_coeff = []
   for i in range(n):
      round_coeff.append(round(2*(a[i].lift()-round(q/4))/q) % 2);
   return round_coeff


def cross_round(a):
   """ Input randomized rounded a in Rq as a vector of coefficients. """
   round_coeff = []
   for i in range(n):
      round_coeff.append(floor(4*a[i].lift()/q) % 2)
   return round_coeff


def rec(a,b):                     #Input an element of Rq and a bitstring
   coeff=Rq(a).list(); key = [];
   for i in range(n):
      if b[i] == 0 :
         if coeff[i] in range(0,floor(q/4+q/8)+1) or coeff[i] in range(q-1-floor(q/8),q):
            key.append(0)
         else: key.append(1)
      if b[i] == 1 :
         if coeff[i] in range(floor(q/4-q/8),(q-1)/2+floor(q/8)+1):
            key.append(0)
         else: key.append(1)
   return key


##################################
# Key agreement with Uniform sampling
##################################

#Gen:
s0 = uniform_sample_Rq(B)
s1 = uniform_sample_Rq(B)

b = Rq(a)*Rq(s1) + Rq(s0)
```

```
#Encaps
e0 = uniform_sample_Rq(B)
e1 = uniform_sample_Rq(B)
e2 = uniform_sample_Rq(B)

u = Rq(a)*Rq(e0) + Rq(e1)
v = Rq(b)*Rq(e0) + Rq(e2)

vr = randomized_round(v)
vp = cross_round(vr)
mu = modular_round(vr)

#Decaps
w = u*Rq(s1)
mup = rec(w,vp)

assert mu == mup


##################################
# Key agreement with Gaussian sampling
##################################


#Gen:
s0 = Gauss_sample_Rq(sigma)
s1 = Gauss_sample_Rq(sigma)

b = Rq(a)*Rq(s1) + Rq(s0)

#Encaps
e0 = Gauss_sample_Rq(sigma)
e1 = Gauss_sample_Rq(sigma)
e2 = Gauss_sample_Rq(sigma)

u = Rq(a)*Rq(e0) + Rq(e1)
v = Rq(b)*Rq(e0) + Rq(e2)

vr = randomized_round(v)
vp = cross_round(vr)
mu = modular_round(vr)

#Decaps
w = u*Rq(s1)
mup = rec(w,vp)

assert mu == mup
```