Verification of the (1–δ)-Correctness Proof of CRYSTALS-KYBER with Number Theoretic Transform

Katharina Kreuzer[1]

[1] *Technical University Munich, Boltzmannstr. 3, 85748 Garching, Germany*

Abstract

This paper describes a formalization of the specification and the algorithm of the cryptographic scheme CRYSTALS-KYBER as well as the verification of its $(1 - \delta)$-correctness proof. During the formalization, a problem in the correctness proof was uncovered. In order to amend this issue, a necessary property on the modulus parameter of the CRYSTALS-KYBER algorithm was introduced. This property is already implicitly fulfilled by the structure of the modulus prime used in the number theoretic transform (NTT). The NTT and its convolution theorem in the case of CRYSTALS-KYBER was formalized as well. The formalization was realized in the theorem prover Isabelle.

Keywords

post-quantum cryptography, CRYSTALS-KYBER, number theoretic transform, verification, Isabelle

1. Introduction

With large-scale quantum computers all crypto systems based on RSA and Diffie-Hellmann can be broken using Shor's algorithm. Since recent developements in quantum computing lead to believe that these feasible quantum computers are not too far off in the future, methods for cryptography which are resistant even to attacks by quantum computers are hot research topics. In the course of the standardization process initialized by the National Institute of Standards and Technology (NIST), a variety of post-quantum crypto systems have been designed. Most prominent are the so-called lattice-based crypto schemes. One of these is the key encapsulation mechanism CRYSTALS-KYBER (abbreviated as Kyber throughout this presentation) which was developed by Roberto Avanzi *et al.* [1] [2].

We have formalized the algorithms for key encapsulation, encryption and decryption and verified the $(1 - \delta)$-correctness in the theorem prover Isabelle. In the course of the proof a necessary property on the modulus q used in the Kyber specification parameters has been found which was not explicitly mentioned in the papers [1] and [2] but follows

FAVPQC 2022: International Workshop on Formal Analysis and Verification of Post-Quantum Cryptographic Protocols, October 24, 2022, Madrid, Spain.

EMAIL: k.kreuzer@tum.de (K. Kreuzer)
URL: https://www21.in.tum.de/team/kreuzer/index.html (K. Kreuzer)
ORCID: 0000-0002-4621-734X (K. Kreuzer)

indirectly from the choice of q for the Number Theoretic Transform (NTT). Since the chosen parameter q fulfils this property, the proof remains valid. However, if the parameters need to be changed in the future, it is important to keep both the property for the $(1-\delta)$-correctness proof and the NTT in mind.

The NTT is a version of the Discrete Fourier Transform adapted to finite fields. In order to refine the algorithm, the NTT on polynomials, as used in Kyber, as well as its convolution theorem have been formalized. The use of the NTT in lattice-based cryptography is described in [3] and its connection to nega-cyclic convolutions in [4]. This leads to a refined and verified Kyber algorithm using the NTT for fast multiplication.

In [5] and [6], Meijers *et al.* announced a formalization of Kyber in EasyCrypt [7]. Furthermore, a post-quantum version of EasyCrypt called EasyPQC is being developed as introduced in [8]. However, up to now there is no publication or accessible code for a formalization of Kyber.

For this paper, Kyber was verified using the theorem prover Isabelle. In contrast to other cryptographic verification tools like EasyCrypt, Isabelle is foundational and everything is proved from the axioms of HOL. An introduction to Isabelle can be found in [9] and [10].

In this paper, we discuss the formalization and verification of Kyber and its $(1-\delta)$-correctness proof. First, we have a look at the specifications and parameters of Kyber in Section 2. We elaborate on the representation of the ring $\mathbb{Z}_q[x]/(x^n+1)$ as a type class in Isabelle. Since the formalization is independent from the actual parameters, in Section 3 we look at the instantiation of our formalization with the values given in [1]. Next, we describe the formalization of the algorithms for compression, decompression, key generation, encryption and decryption used in Kyber in Section 4. In Section 5 we prooceed with the verification of the $(1-\delta)$-correctness proof of Kyber. Here, we recognise a problem in the proof which we can solve by adding a property on the modulus q. This is discussed in Section 6. This newly found property is already fulfilled when working in the NTT domain. The formalization of the NTT on polynomials and its convolution theorem is analyzed in Section 7. In the end, we give a short outlook on further research questions. The full formalization can be found in [11].

2. Formalizing the Specifications of Kyber

Let q be a prime and n a power of two, i.e., there is an n' such that $n = 2^{n'}$. Let R_q denote the ring $\mathbb{Z}_q[x]/(x^n+1)$. Note that x^n+1 is the $2^{n'}$-th cyclotomic polynomial which is irreducible over the integers $\mathbb{Z}$, but reducible over the finite field $\mathbb{Z}_q$.

When implementing the specifications of Kyber, one first has to think how to formalize a quotient ring when factoring a polynomial ring with coeffiecients in a finite field by an ideal generated by a cyclotomic polynomial, namely $\mathbb{Z}_q[x]/(x^n+1)$. There are various concepts behind this construct which are not easy to formalise in Isabelle. Two main features in Isabelle support abstaction over a context of assumptions for theorems: The type class constraints (introduced in [12]) and explicit assumptions summarized in a context called locale (introduced in [13]). To still be able to work over these complicated spaces without too many premises, we chose to use type class constructs.

First of all, the existing formalization of the finite field uses the type class *mod_ring* over a finite type. The modulus prime is encoded as the cardinality of the finite type. It represents the residue classes of the ring $\mathbb{Z}_q$ where q is the cardinality of the finite type.

Polynomials can be easily constructed using the *poly* type constructor. The *poly* constructor defines a polynomial to be a function from the natural numbers to the coefficient space which is 0 almost everywhere. A polynomial p in $R[x]$ is thus represented by the function of coefficients $f : \mathbb{N} \longrightarrow R$ such that $p = \sum_{i=0}^{\infty} f(i)x^i$. Since p has only finitely many non-zero coefficients, f is 0 almost everywhere. For example the polynomial $p = x^2 + 2$ is represented as the function f with:

$$f(i) = \begin{cases} \text{if } i = 0 \text{ then } 2 \\ \text{if } i = 2 \text{ then } 1 \\ \text{else } 0 \end{cases}$$

There is an alternative definition which defines polynomials using a list constructor *pCons*. This allows the user to convert concrete polynomials to lists of coefficients and vice versa. Continuing our example from above, the list correponding to $p = x^2 + 2$ is $[2, 0, 1]$. However, when representing polynomials as lists, one has to be careful to always reduce redundant zero coefficients in order to guarantee a unique representation. For example, the list $[2, 0, 1, 0, 0]$ also represents the polynomial p.

The most difficult part is to construct the quotient ring R_q. First, an equivalence relation needs to be established for residue classes modulo $x^n + 1$. Then, one can factor out the equivalence relation using the command *quotient_type* as introduced in [14]. The resulting structure inherits basic properties like the zero element, addition, subtraction and multiplication from the original polynomial ring through lifting and transfer [15].

Vectors are implemented using a fixed finite type as an index set. Since Isabelle does not allow dependent types, a separate finite type for indexing is used to encode the length of a vector. This idea was introduced by Harrison in [16]. For example, when working with vectors in $\mathbb{Z}^k$, we use the type *(int, 'k) vec*, where *'k* is a finite type with cardinality exactly k used for indexing the integer coefficients.

An important fact to note when dealing with formalizations is that the functions translating between the different types always need to be stated explicitly. Often in the mathematical literature, this distinction is blurred to enable a shorter presentation.

3. Formalizing the Parameters of Kyber

The parameters of Kyber in Table 1 are taken from [1]. Since the framework for the specification of Kyber is formalized independently from the actual parameters, we can instantiate the formalization with any parameters sufficing all required properties:

- n, n', q, k, d_u, d_v, d_t are positive integers
- $n = 2^{n'}$ is a power of 2
- $q > 2$ is an odd prime with $q \mod 4 = 1$

Table 1
Parameter set of Kyber in [1]

variable	value	context
n	$256 = 2^{n'}$	degree of cyclotomic polynomial
n'	8	exponent of 2 in degree n
q	7681	prime number, modulus
k	3	dimension of vectors
d_u	11	digits for encryption of u
d_v	3	digits for encryption of v
d_t	11	digits for key generation

This is especially of interest for eventual changes in the parameter set in the future. Furthermore, different security level implementations use different parameters. For example, the initial parameter of the modulus q in [1] is 7681, but the newer specificaion for the third round of the NIST standardization process [17] uses the modulus 3329. Furthermore, different sizes k of vectors define different security levels.

In our formalization, we instantiate the locale containing the Kyber algorithm and proof of $(1 - \delta)$-correctness with the parameter set given in Table 1. Unfortunately this has been trickier than expected. The existing code generation for generating finite types of a specific cardinality does not allow the user to instantiate this type for the type class of prime cardinalities. Therefore, the type class with 7681 elements was instantiated manually for prime cardinality.

4. Formalizing the Kyber Algorithm

The cryptographic scheme Kyber is divided in three algorithms: the key generation, the encryption and the decryption. Using a randomly chosen input, the key generation produces a public and secret key pair that are applied in the en- and decryption. In order to discard some lower order bits to make the keys smaller, a compression and decompression function is added. The compression function is also used to extract the message in the decryption. For a clearer presentation, we omit explicit type casts when they are unambiguous. For example, the embedding of integers in the reals or vice versa has an explicit type cast. An important type cast that we will state explicitly is the cast from an integer to the module R_q which we denote as the function *to_module*. In the actual formalization, all type casts are stated.

4.1. Input to the Algorithm

The key generation requires an input $A \in R_q^{k\times k}$, $s \in R_q^k$ and $e \in R_q^k$ which is chosen randomly. A is chosen uniformly at random from the finite set $R_q^{k\times k}$. This matrix is part of the public key. For the secret key s and the error term e, we define the binomial distribution β_η. Choose η values c_i with $P(c_i = -1) = P(c_i = 1) = 1/4$ and $P(c_i = 0) = 1/2$ and return the value $x = \sum_{i=1}^{\eta} c_i$. In Kyber, we use $\eta = 4$ and thus: For

Table 2
Probability distribution of β_η for $\eta = 4$

$x = \sum_{i=1}^{4} c_i$	-4	-3	-2	-1	0	1	2	3	4
$P(x)$	$\frac{1}{256}$	$\frac{8}{256}$	$\frac{28}{256}$	$\frac{56}{256}$	$\frac{70}{256}$	$\frac{56}{256}$	$\frac{28}{256}$	$\frac{8}{256}$	$\frac{1}{256}$

generating a polynomial in R_q according to β_η, every coefficient is chosen independently from β_η. Similarly, a vector is generated according to β_η by independently choose all entries according to β_η. Both s and e are generated according to β_η.

The sampled values A, s and e constitute an instance of the module-Learning-With-Errors (module-LWE) problem which is defined in the following.

Definition 1 (Module-LWE)**.** Given a uniformly random $A \in R_q^{k\times k}$ and $s, e \in R_q^k$ chosen randomly according to the distribution β_η. Let $b = As + e$, then the (decision) module-LWE problem asks to distinguish (A, b) from uniformly random $(A', b') \in R_q^{k\times k} \times R_q^k$.

There is a probabilistic reduction proof for the NP-hardness of the module-LWE by Langlois and Stehlé in [18]. Using the hardness of the module-LWE, the key generation of Kyber returns a public key and secret key pair where it is NP-hard to recover the secret key from the public key alone. Note that this problem would be easy to solve without the error term using the Euclidean Algorithm. Thus, the error term cannot be reused but has to be chosen according to the distribution β_η again. The random choices, the module-LWE and its NP-hardness proof have not been formalized.

4.2. Compression and Decompression

The compression and decompression functions in Kyber are essential for smaller public and secret keys and help obscuring the message. In the decryption, the message is also extracted by a compression to one bit. In order to define these functions, we introduce a positive integer d with $2^d < q$. Thus, we have $d < \lceil log_2(q) \rceil$. In this section, we write the modulo operation with modulus 2^d to mean the unique representant in $\{0, \dots, 2^d - 1\}$.

When compressing a value x, we omit the least important bits and reduce the representation of x to d bits. Decompression rescales to the modulus q. Compression and decompression functions are defined for integers in the following way.

$$compress_d\ x = \left\lceil \frac{2^d \cdot x}{q} \right\rfloor \mod 2^d$$

$$decompress_d\ x = \left\lceil \frac{q \cdot x}{2^d} \right\rfloor$$

Note that the round function is defined as $\lceil x \rfloor = \lfloor x + \frac{1}{2} \rfloor$. The compression and decompression functions are extended as functions over $\mathbb{Z}_q$ by taking the unique representant in $\{0, \ldots, q-1\}$. We denote compression and decompression over polynomials as *compress_poly* and *decompress_poly* and over vectors as *compress_vec* and *decompress_vec*. They are defined to perform the compression or decompression coefficient- and index-wise, respectively.

We call the value $decompress_d\ (compress_d\ x) - x$ the compression error. The rounding in the compression and decompression may introduce such a compression error. For example, consider the values $d = 2$ and $q = 5$. Then, the compression of 2 is $compress_2\ 2 = \lceil 1.6 \rfloor \mod 4 = 2$ and $decompress_2\ 2 = \lceil 2.5 \rfloor = 3$. Here, the compression error is $decompress_2\ (compress_2\ 2) - 2 = 3 - 2 = 1$. Another reason for a compression error is the modulo operation in the compression function. For example consider $d = 2$ and $q = 11$. Then the compression of 10 is $compress_2\ 10 = \lceil 3.\overline{63} \rfloor \mod 4 = 0$ and $decompress_2\ 0 = 0$. Here, the compression error for integers is $decompress_2\ (compress_2\ 10) - 10 = -10$. Interpreting this as a number over $\mathbb{Z}_{11}$, we get a compression error of 1.

4.3. Key Generation, Encryption and Decryption

We now want to state the actual algorithm. The algorithm for key generation is defined in the following way using the compression depth d_t:

$$key\_gen\ d_t\ A\ s\ e = compress\_vec_{dt}\ (A \cdot s + e)$$

The output $t = key\_gen\ d_t\ A\ s\ e$ of *key_gen* together with the matrix A constitutes the public key, whereas the vector s is the secret key. Together (A, t) form an instance of the module-LWE problem. The NP-hardness of the module-LWE states, that it is hard to recuperate the secret key s from the public key (A, t). Therefore, the hardness of module-LWE underlies the security of Kyber.

To encrypt a bitstring $\bar{m}$ with at most n bits, we consider the message polynomial $m \in R_q$ obtained by $m = \sum_{i=0}^{n-1} \bar{m}(i) x^i$. Thus, the message polynomial m only has coefficients in $\{0, 1\}$. We also need to generate another secret $r \in R_q^k$ together with errors $e_1 \in R_q^k$ and $e_2 \in R_q$ according to the distribution β_4. We then calculate the encryption as the following:

$$\begin{aligned}
&encrypt\ t\ A\ r\ e1\ e2\ dt\ du\ dv\ m = \\
&\qquad (compress\_vec_{du}\ (A^T \cdot r + e1), \\
&\qquad\ compress\_poly_{dv}\ ((decompress\_vec_{dt}\ t)^T r + e2 + to\_module(\lceil q/2 \rfloor) \cdot m))
\end{aligned}$$

The encryption outputs the compressed values u and v, i.e., $(u, v) = encrypt\ t\ A\ r\ e1\ e2\ dt\ du\ dv\ m$. Using the secret key s, we can recover the message m from u and v in the decryption function. We extract the message as the highest bit in $v - s^T u$ using the compression function with depth 1.

$$\begin{aligned}
&decrypt\ u\ v\ s\ du\ dv = \\
&\qquad compress\_poly_1\ ((decompress\_poly_{dv}\ v) - s^T (decompress\_vec_{du}\ u))
\end{aligned}$$

During the algorithms, the compression and decompression induce errors which should not affect the correctness of the decryption result. This problem is investigated in the $(1-\delta)$-correctness proof of Kyber. The following section describes a verification of this proof in Isabelle.

5. Verifying the $(1-\delta)$-Correctness Proof of Kyber

To verify the $(1-\delta)$-correctness of the specification of Kyber in Isabelle, we look at the following proof from [1]. The full formalization of the correctness proof can be found in [11].

Using the module-LWE problem as in Definition 1, distinguishing between uniformly random samples and samples generated by a module-LWE instance in the key generation and encryption is NP-hard. Thus distinguishing the random values c_t, c_u and c_v from [1, Theorem 1] and the compression errors of t, u and v, respectively, is also NP-hard. Since the module-LWE and its hardness asumption have not been formalized yet, we only consider c_t, c_u and c_v to be the absolute value of the compression errors.

$$c_t = \|t - \mathit{decompress\_vec}_{d_t}\ (\mathit{compress\_vec}_{d_t}\ t)\|_\infty$$

Note that the "norm" $\|\cdot\|_\infty$ in [1] is defined to be slightly different than one would expect: Instead of using a regular modulo operation, we define the recentered operation $\bmod^\pm$ to be the representative with smallest norm. That means $\bar{a} := (a \bmod^\pm q)$ is the unique element with $-q/2 < \bar{a} \leq q/2$ such that $\bar{a} \equiv a \mod q$. As q is an odd number in our case, we get that $a \bmod^\pm q \in \{\frac{-q+1}{2}, \dots, \frac{q-1}{2}\}$. Using this recentered modulo operation, we define the function $\|\cdot\|_\infty$ on polynomials as:

$$p = \sum_{i=1}^{\deg p} p_i \cdot x^i \longmapsto \|p\|_\infty = \max_{i\in\{0,\dots,\deg p\}} |p_i \bmod^\pm q|$$

Analogously, for vectors $v \in R_q^k$ we define:

$$\|v\|_\infty = \max_{i\in\{1,\dots,k\}} \|v_i\|_\infty$$

Unfortunately with the recentering one looses the absolute homogeneity, i.e., for a scalar s and vector v only

$$\|s \cdot v\|_\infty \leq |s| \cdot \|v\|_\infty \tag{1}$$

holds with an inequality instead of equality. For example consider the case $q = 3$, $s = 2$ and $v = (2)$. We then have the strict inequality:

$$\|2\cdot(2)\|_\infty = |2\cdot(2) \bmod^\pm 3| = 1 < 2 = |2|\cdot|2 \bmod^\pm 3| = |2|\cdot\|(2)\|_\infty$$

The $\|\cdot\|_\infty$ function is not a norm, but is positive definite and fulfils the triangle inequality. This is not explicitly mentioned in the Kyber paper [1] and indeed poses a problem in the proof of the following theorem.

We state the correctness theorem for Kyber, which is formalized in [11]:

Theorem 5.1. *Given $A \in R_q^{k \times k}$, $s, r, e, e_1 \in R_q^k$, $e_2 \in R_q$ and the message $m \in R_q$ with coefficients in $\{0, 1\}$ and setting:*

- *$t =$ key_gen d_t A s e as the output of the key generation*
- *$(u, v) =$ encrypt t A r e1 e2 dt du dv m as the output of the encryption*
- *c_t, c_u and c_v as the absolute value of the compression errors of t, u and v, respectively*

If $\|e^T r + e_2 + c_v - s^T e_1 + c_t^T r - s^T c_u\|_\infty < \lceil q/4 \rfloor$, then the decryption algorithm returns the original message m:

$$\textit{decrypt } u \ v \ s \ du \ dv = m$$

A crypto system is correct, if it always returns the original message. However, in our case, we have a failure probability and can only state the $(1 - \delta)$-correctness. This is defined in the following:

Definition 2 ($(1 - \delta)$-correctness)**.** Let *key_gen*, *encrypt* and *decrypt* constitute a cryptographic algorithm where the *key_gen* outputs a public key *pk* and a secret key *sk*. Then the cryptographic algorithm is $(1 - \delta)$-correct, if and only if:

$$\mathbb{P}[\textit{decrypt}(sk, \textit{encrypt}(pk, m)) \mid (pk, sk) \leftarrow \textit{key\_gen}] \geq 1 - \delta$$

We have that Kyber is $(1 - \delta)$-correct if and only if the message is always returned correctly when assuming the inequality:

$$\|e^T r + e_2 + c_v - s^T e_1 + c_t^T r - s^T c_u\|_\infty < \lceil q/4 \rfloor$$

Using Theorem 5.1, we deduce the $(1 - \delta)$-correctness of Kyber.

Corollary. *Let $\delta = \mathbb{P}\left[\|e^T r + e_2 + c_v - s^T e_1 + c_t^T r - s^T c_u\|_\infty \geq \lceil q/4 \rfloor\right]$. Then Kyber is $(1 - \delta)$-correct.*

The formalization of the proof of Theorem 5.1 can be found in [11]. One problem encountered during the formalization was that $\|\cdot\|_\infty$ is not a norm. This is not explicitly mentioned in the Kyber paper [1] and indeed poses a problem in the proof which we will discuss in greater detail in the next section. In short: We cannot follow that *decompress_poly*$_1$ (*compress_poly*$_1$ m) $= 0$ in the last step of the correctness proof unless $q \equiv 1 \mod 4$.

Another painful step in the proof was to ensure that all calculations are conform with the residue classes modulo the polynomial $x^n + 1$. Indeed, in Isabelle the type casting is explicit, so one always has to channel through all type casts. Especially, one always has to show that the implications hold independently from the representative chosen from a residue class. In some cases, we also presume natural embeddings and isomorphisms to hold in pen-and-paper proofs which have to be stated explicitly in Isabelle (for example the *to_module* function mentioned in the previous section). Thus, formalizations are much more verbose.

Before we can start the proof of Theorem 5.1, we need to show an auxiliary lemma on the estimation of the compression error.

Lemma 5.2. *Let x be an element of $\mathbb{Z}_q$ and $x' = decompress_d\ (compress_d\ x)$ its image under compression and decompression with $2^d < q$. Then we have:*

$$|x' - x \mod^{\pm} q| \le \lceil q/2^{d+1} \rfloor$$

Proof. Let x be the representative in $\{0, \dots, q-1\}$. Then consider two cases, namely $x < \lceil q - \frac{q}{2^{d+1}} \rceil$ and $x \ge \lceil q - \frac{q}{2^{d+1}} \rceil$. These cases arise from the distinction whether the modulo reduction in the definition of the compression function is triggered or not. Indeed, we have $compress_d\ x = \lceil \frac{2^d}{q} x \rfloor \mod 2^d$ where $\frac{2^d}{q} x < 2^d$, but $\lceil \frac{2^d}{q} x \rfloor = 2^d$ if and only if $x \ge \lceil q - \frac{q}{2^{d+1}} \rceil$. In the latter case, the modulo operation in the compression function is activated and returns $compress_d\ x = 0$. In the following, we will abbreviate the compression function by *comp* and the decompression function by *decomp*.

Case 1: Let $x < \lceil q - \frac{q}{2^{d+1}} \rceil$. Then the modulo reduction in the compression function $comp_d\ x = \lceil \frac{2^d}{q} x \rfloor \mod 2^d = \lceil \frac{2^d}{q} x \rfloor$ is not triggered. Thus we get:

$$\begin{aligned}
|x' - x| &= |decomp_d\ (comp_d\ x) - x| = \\
&= \left| decomp_d\ (comp_d\ x) - \frac{q}{2^d} \cdot comp_d\ x + \frac{q}{2^d} \cdot comp_d\ x - \frac{q}{2^d} \cdot \frac{2^d}{q} \cdot x \right| \le \\
&\le \left| decomp_d\ (comp_d\ x) - \frac{q}{2^d} \cdot comp_d\ x \right| + \frac{q}{2^d} \cdot \left| comp_d\ x - \frac{2^d}{q} \cdot x \right| = \\
&= \left| \left\lceil \frac{q}{2^d} \cdot comp_d\ x \right\rfloor - \frac{q}{2^d} \cdot comp_d\ x \right| + \frac{q}{2^d} \cdot \left| \left\lceil \frac{2^d}{q} \cdot x \right\rfloor - \frac{2^d}{q} \cdot x \right| \le \\
&\le \frac{1}{2} + \frac{q}{2^d} \cdot \frac{1}{2} = \frac{q}{2^{d+1}} + \frac{1}{2}
\end{aligned}$$

Since $x' - x$ is an integer, we also get:

$$|x' - x| \le \left\lfloor \frac{q}{2^{d+1}} + \frac{1}{2} \right\rfloor = \left\lceil \frac{q}{2^{d+1}} \right\rfloor$$

Therefore also $|x' - x| \le \lfloor q/2 \rfloor$ such that the *mod*$^{\pm}$ operation does not change the outcome. Finally for this case, we get

$$|x' - x \mod^{\pm} q| \le \left\lceil \frac{q}{2^{d+1}} \right\rfloor$$

Case 2: Let $x \ge \lceil q - \frac{q}{2^{d+1}} \rceil$. Then the modulo operation in the compression results in the compression to zero, i.e., $comp_d\ x = 0$. Using the assumption on x, we get:

$$\begin{aligned}
|x' - x \mod^{\pm} q| &= |decomp_d\ 0 - x \mod^{\pm} q| = \\
&= |-x \mod^{\pm} q| = |-x + q| \le \\
&\le \left| \left\lceil q - \frac{q}{2^{d+1}} \right\rceil - q \right| = \left\lfloor \frac{q}{2^{d+1}} \right\rfloor \le \left\lceil \frac{q}{2^{d+1}} \right\rfloor
\end{aligned}$$

□

The proof of Theorem 5.1 proceeds as follows. Given A, s, r, e, e_1, e_2 and the message m, we calculate t, u and v using the key generation and encryption algorithm. We define $\tilde{t}$, $\tilde{u}$ and $\tilde{v}$ to be the decompressed values of t, u and v, respectively. With the compression errors c_t, c_u and c_v, we get the equations:

$$\begin{aligned} \tilde{t} &= As + e + c_t \\ \tilde{u} &= A^T r + e_1 + c_u \\ \tilde{v} &= \tilde{t}^T r + e_2 + \lceil q/2 \rfloor \cdot m + c_v \end{aligned}$$

This leads to the calculation in the decryption:

$$\tilde{v} - s^T \tilde{u} = e^T r + e_2 + c_v + c_t^T r - s^T e_1 - s^T c_u + \lceil q/2 \rfloor \cdot m$$

We accumulate all error terms in a new variable w:

$$w := e^T r + e_2 + c_v + c_t^T r - s^T e_1 - s^T c_u$$

and get $\|w\|_\infty < \lceil q/4 \rfloor$ from the assumptions.

Now, we need to show that $m' := \mathit{decrypt}(u, v, s)$ is indeed the original message m. We consider the value of $\tilde{v} - s^T \tilde{u}$, its compression with $d = 1$, namely m', and the decompressed value $\mathit{decompress}_1\ m'$. Since the compression depth is 1, we get $m' \in \{0, 1\}$. Thus:

$$\mathit{decompress}_1\ m' = \lceil q/2 \cdot m' \rfloor = \lceil q/2 \rfloor \cdot m'$$

Using Lemma 5.2, it follows that:

$$\|w + \lceil q/2 \rfloor (m - m')\|_\infty = \|\tilde{v} - s^T \tilde{u} - \mathit{decompress}_1\ (\mathit{compress}_1\ (\tilde{v} - s^T \tilde{u}))\|_\infty \le \lceil q/4 \rfloor$$

Using the triangle inequality on $\| \cdot \|_\infty$, we calculate

$$\begin{aligned} \|\lceil q/2 \rfloor (m - m')\|_\infty &= \|w + \lceil q/2 \rfloor (m - m') - w\|_\infty \le \\ &\le \|w + \lceil q/2 \rfloor (m - m')\|_\infty + \|w\|_\infty < \\ &< \lceil q/4 \rfloor + \lceil q/4 \rfloor = 2 \lceil q/4 \rfloor \end{aligned}$$

It remains to show that we can indeed deduce $m = m'$ which concludes the proof of Theorem 5.1. According to the last step from the proof of $(1 - \delta)$-correctness in [1], this follows directly for any odd prime q. However, therein lies a hidden problem. Indeed, for $q \equiv 3 \mod 4$, we cannot conclude the proof with this argument. In the next section, we discuss why we can only deduce this step under the assumption that $q \equiv 1 \mod 4$.

6. Additional Property $q \equiv 1 \mod 4$

Consider the following: Given the inequality $\|\lceil q/2 \rfloor \cdot (m - m')\|_\infty < 2 \cdot \lceil q/4 \rfloor$ we need to show that indeed $m = m'$. We prove this statement by contradiction. Assume that there exists a coefficient of $m - m'$ that is different from zero. Since m and m' are polynomials with coefficients in $\{0, 1\}$, a non-zero coefficient can either be 1 or -1. Then we get

$$\|\lceil q/2 \rfloor \cdot (m - m')\|_\infty = |\lceil q/2 \rfloor \cdot (\pm 1) \mod^\pm q| = \ldots$$

For all numbers greater than 2, all primes are odd numbers. Thus we have $\lceil q/2 \rfloor = (q + 1)/2$. We continue our calculation:

$$\cdots = \left|\frac{q+1}{2} \mod^\pm q\right| = \left|\frac{-q+1}{2}\right| = \frac{q-1}{2} = 2 \cdot \frac{q-1}{4} = \ldots$$

since the *mod*$^\pm$ operation reduces $\frac{q+1}{2}$ to the representative $\frac{-q+1}{2}$. Now we need to relate $\frac{q-1}{4}$ to $\lceil q/4 \rfloor$. We have two cases:

Case 1: For $q \equiv 1 \mod 4$ we indeed get the equality $\frac{q-1}{4} = \lceil q/4 \rfloor$ that we need. In this case we have

$$\|\lceil q/2 \rfloor \cdot (m - m')\|_\infty = 2 \cdot \lceil q/4 \rfloor$$

which is a contradiction to our assumption.

Case 2: For $q \equiv 3 \mod 4$ we get the strict inequality $\frac{q-1}{4} < \frac{q+1}{4} = \lceil q/4 \rfloor$ resulting in

$$\|\lceil q/2 \rfloor \cdot (m - m')\|_\infty < 2 \cdot \lceil q/4 \rfloor$$

which is no contradiction to the assumption. Indeed in this case we cannot deduce $m = m'$, since it is possible that a coefficient of $m - m'$ is non-zero.

Consider this short exapmle: Let $q = 7$ ($\equiv 3 \mod 4$, thus we are in case 2), $m = 0$ and $m' = 1$. In this case, the inequality of the assumption holds

$$\|\lceil q/2 \rfloor \cdot (m - m')\|_\infty = 3 < 4 = 2 \cdot \lceil q/4 \rfloor$$

but $m \neq m'$.

Therefore Theorem 5.1 only holds if the modulus q fulfils the property $q \equiv 1 \mod 4$.

In the specification of Kyber, concrete values for the variables of the system are given. However, for different security levels or implementations different values for the parameters in Kyber are used. For example in the status report of the third round in the post-quantum crypotography standardization process by the National Institute for Standards and Technology [17], the modulus q is chosen to be 3329, whereas in the first paper about Kyber [1] and the specification and documentations paper [2], the modulus was chosen as $q = 7681$. Considering possible changes to these variables, it is important to enable the verified proof to cover all possible cases.

During the proof, we encountered a problem for arbitrary primes q. In the last step of the correctness proof, the homogeneity of the redefined function $\|\cdot\|_\infty$ is assumed. Unfortunately, by the recentering of the infinity norm using the *mod*$^\pm$ definition instead of the regular modulo operation, the property of homogeneity was lost. This causes the problem that we cannot deduce that the message m and its computed output of the decryption m' are indeed the same for arbitrary primes. For the primes chosen in the specifications [1], [2] and [17] however, this implication is indeed true because they fulfil the property $q \equiv 1 \mod 4$.

Indeed, the modulus q is chosen according to a much more rigid scheme: In order to implement the multiplication to compute faster, the Number Theoretic Transform (NTT) is used. In the case of Kyber, the NTT is computed on $R_q = \mathbb{Z}_q[x]/(x^n+1)$. The requirement for NTT on the modulus q is:

$$q \equiv 1 \mod n$$

For $q = 7681$ we have $7681 = 30 \cdot 256 + 1$, whereas for $q = 3329$ we get $3329 = 13 \cdot 256 + 1$. Since n is a power of 2, we can automatically infer the property $q \equiv 1 \mod 4$.

We will have a more thorough look at the number theoretic transform used in Kyber in the next section.

7. NTT and the Convolution Theorem

The number theoretic transform is used to speed up the multiplication on $\mathbb{Z}_q[x]/(x^n+1) = R_q$ and is based on the concepts of the Discrete Fourier Transform. An introduction to the use of the NTT for lattice-based cryptography can be found in [3] or for the special case of the CRYSTALS suite in [19]. The NTT as a nega-cyclic convolution is described in [4].

The standard multiplication for $f = \sum_{k=0}^{n-1} f_k x^k$ and $g = \sum_{k=0}^{n-1} g_k x^k$ in R_q is given by:

$$f \cdot g = \sum_{k=0}^{n-1} \left(\sum_{j=0}^{n-1} (-1)^{k-j \text{ div } n} f_j g_{k-j \bmod n} \right) x^k$$

Thus, multiplication is done using $\mathcal{O}(n^2)$ multiplications on coefficients. Unlike multiplication, addition is calculated in $\mathcal{O}(n)$ since addition is done entry-wise. Therefore, the most expensive part of calculations in the Kyber crypto algorithms is the multiplication. Using a smarter way to multiply will make the calculations in Kyber faster.

The usual NTT requires the field $\mathbb{Z}_q$ to have a n-th root of unity, that is an element ω with $\omega^n = 1$. This can be achieved by setting $q \equiv 1 \mod n$. However, since we work over the quotient ring $Z_q[x]/(x^n+1)$, we have to consider the nega-cyclic property that $x^n \equiv -1 \mod x^n + 1$ instead of the cyclic properties required by the NTT. Moreover, Kyber uses a "twisted" alternative which is easier to implement but requires the existence of a $2n$-th root of unity.

Considering all the constraints mentioned above, let ψ be a $2n$-th root of unity in R_q. Then we define the nega-cyclic twisted NTT on R_q as used in Kyber as the following.

Definition 3 (NTT)**.** Let $f = \sum_{k=0}^{n-1} f_k x^k \in R_q$, then the NTT of f is defined by:

$$NTT(f) = \sum_{k=0}^{n-1} \left(\sum_{j=0}^{n-1} f_j \psi^{j(2k+1)} \right) x^k$$

The inverse transform is scaled by the factor of n^{-1} and is given by the following.

Definition 4 (inverse NTT)**.** Let $f = \sum_{k=0}^{n-1} g_k x^k \in R_q$ be in the image of the NTT, then the inverse NTT of f is defined by:

$$invNTT(f) = \sum_{k=0}^{n-1} n^{-1} \left(\sum_{j=0}^{n-1} g_j \psi^{-k(2j+1)} \right) x^k$$

We formalized a proof of correctness of the NTT and its inverse.

Theorem 7.1. *Let f be a polynomial in R_q and g a polynomial in NTT domain. Then NTT and $invNTT$ are inverses:*

$$invNTT(NTT(f)) = f \quad \textit{and} \quad NTT(invNTT(g)) = g$$

Proof. We show the equality for every coefficient. Here, p_k denotes the coefficient of x^k in the polynomial p.

$$\begin{aligned} invNTT(NTTf))_k &= n^{-1} \left(\sum_{i=0}^{n-1} \left(\sum_{j=0}^{n-1} f_j \psi^{j(2i+1)} \right) \psi^{-k(2i+1)} \right) = \\ &= n^{-1} \sum_{j=0}^{n-1} f_j \psi^{j-k} \left(\sum_{i=0}^{n-1} \psi^{(j-k)(2i)} \right) = \dots \end{aligned}$$

We want to compute the geometric sum $\sum_{i=0}^{n-1} (\psi^{2(j-k)})^i$. If $j = k$, then the whole sum collapses to a sum over ones, reulting in n. But if $j \neq k$, we get

$$\sum_{i=0}^{n-1} (\psi^{2(j-k)})^i = \frac{\psi^{2(j-k)n} - 1}{\psi^{2(j-k)} - 1} = 0$$

using that ψ is a n-th root of unity, that is $\psi^n = 1$. We continue our calculation:

$$\dots = n^{-1} \sum_{j=0}^{n-1} f_j \psi^{j-k} (\text{if } j = k \text{ then } n \text{ else } 0) = n^{-1} n f_k = f_k$$

This finishes the first inversion property.

The proof of the second property proceeds similarly, but with inverted roles of ψ and ψ^{-1}. ☐

Using this transformation, we can reduce multiplications to compute within $\mathcal{O}(n \log(n))$ using a fast version of the NTT. To apply the NTT to the Kyber algorithms, we need the convolution theorem. It states that multiplication of two polynomials in R_q can be done index-wise over the NTT domain.

Theorem 7.2. *Let f and g be two polynomials in R_q. Let $(\cdot)$ denote the multiplication of polynomials in R_q and $(\odot)$ the coefficient-wise multiplication of two polynomials in the NTT domain. Then the convolution theorem states:*

$$NTT(f \cdot g) = NTT(f) \odot NTT(g)$$

Proof. We show the equality for every coefficient. Again, p_k denotes the coefficient of x^k in the polynomial p.

$$\begin{aligned} NTT(f\cdot g)_k &= \sum_{i=0}^{n-1}\left(\sum_{j=0}^{n-1}(-1)^{i-j \text{ div } n} f_j g_{i-j \bmod n}\right)\psi^{i(2k+1)} = \\ &= \sum_{j=0}^{n-1} f_j \psi^{j(2k+1)}\left(\sum_{i=0}^{n-1}(-1)^{i-j \text{ div } n}\psi^{i(2k+1)}\psi^{-j(2k+1)} g_{i-j \bmod n}\right) = \dots \end{aligned}$$

Using $\psi^{n-1} = -1$, we can deduce that $(-1)^{i-j \text{ div } n}\psi^{(i-j)(2k+1)} = \psi^{(i-j \bmod n)(2k+1)}$ and thus our calculation continues:

$$\begin{aligned} \dots &= \sum_{j=0}^{n-1} f_j\psi^{j(2k+1)}\left(\sum_{i=0}^{n-1}\psi^{(i-j \bmod n)(2k+1)} g_{i-j \bmod n}\right) = \\ &= \sum_{j=0}^{n-1} f_j\psi^{j(2k+1)}\left(\sum_{i'=0}^{n-1}\psi^{i'(2k+1)} g_{i'}\right) = \\ &= \left(\sum_{j=0}^{n-1} f_j\psi^{j(2k+1)}\right)\cdot\left(\sum_{i'=0}^{n-1}\psi^{i'(2k+1)} g_{i'}\right) = \\ &= NTT(f)_k \cdot NTT(g)_k \end{aligned}$$

where we reindex the sum over i to a sum over $i - j \mod n$ using the cyclic property of the modulo function. This shows that multiplication is indeed performed coefficient-wise on the NTT domain. □

Together with Theorem 7.1 this yields the fast multiplication formula in Kyber.

Theorem 7.3. *Let f and g be two polynomials in R_q. Let $(\cdot)$ denote the multiplication of polynomials in R_q and $\odot$ the coefficient-wise multiplication of two polynomials in the NTT domain. Then multiplication in R_q can be computed by:*

$$f \cdot g = invNTT(NTT(f) \odot NTT(g))$$

The formalization of the NTT for Kyber went relatively smoothly since it is based on the formalization of the standard NTT by Ammer in [20]. The only minor hindrances were the conversion between the types and working with representatives over R_q as well as the rewriting of huge sums.

8. Conclusion

In this presentation, we described the formalization of the CRYSTALS-KYBER key-generation, encryption and decryption algorithms and verified the proof of $(1-\delta)$-correctness under the assumption $q \equiv 1 \mod 4$. As Kyber was designed to compute fast multiplications using the NTT, the property $q \equiv 1 \mod 4$ is obtained from the necessary requirements for the NTT. Therefore, the formalization of the NTT and its convolution theorem were inspected.

Building on these results, the current algorithm formalization can still be extended by the formalization of the module-LWE assumptions and the sampling techniques for random choice or other refinement steps. It would also be very interesting to formalize the hardness results of the module-LWE that Kyber is building upon.

Acknowledgments

My deepest gratitude go to Manuel Eberl for his continuous help with formalization problems in Isabelle and for listening to and helping with all my issues. Many thanks also go to my supervisor Tobias Nipkow for his continuous support and for proof reading. The author gratefully acknowledges the financial support of this work by the research training group ConVeY funded by the German Research Foundation under grant GRK 2428.

References

[1] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, D. Stehlé, CRYSTALS — Kyber: A CCA-Secure Module-Lattice-Based KEM, in: 2018 IEEE European Symposium on Security and Privacy, 2018, pp. 353–367.

[2] R. M. Avanzi, J. W. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, D. Stehlé, CRYSTALS-Kyber Algorithm Specifications And Supporting Documentation, 2017.

[3] P. Longa, M. Naehrig, Speeding up the Number Theoretic Transform for Faster Ideal Lattice-Based Cryptography, 2016, pp. 124–139.

[4] J. Klemsa, Fast and Error-Free Negacyclic Integer Convolution Using Extended Fourier Transform, 2021, pp. 282–300.

[5] M. Barbosa, A. Hülsing, M. Meijers, P. Schwabe, Formal Verification of Post-Quantum Cryptography, 2022. URL: https://csrc.nist.gov/CSRC/media/Presentations/formal-verifcation-of-post-quantum-cryptography/images-media/session-2-meijers-formal-verification-pqc.pdf.

[6] M. Barbosa, A. Hülsing, M. Meijers, P. Schwabe, Formal Verification of Post-Quantum Cryptography, 2022. URL: https://csrc.nist.gov/CSRC/media/Events/third-pqc-standardization-conference/documents/accepted-papers/meijers-formal-verification-pqc2021.pdf.

[7] GitHub, EasyCrypt, 2022. URL: https://github.com/EasyCrypt/easycrypt.

[8] M. Barbosa, G. Barthe, X. Fan, B. Grégoire, S.-H. Hung, J. Katz, P.-Y. Strub, X. Wu, L. Zhou, EasyPQC: Verifying Post-Quantum Cryptography, Cryptology ePrint Archive, Paper 2021/1253, 2021.

[9] T. Nipkow, L. Paulson, M. Wenzel, Isabelle/HOL — A Proof Assistant for Higher-Order Logic, volume 2283 of *LNCS*, Springer, 2002.

[10] T. Nipkow, G. Klein, Concrete Semantics with Isabelle/HOL, Springer, 2014. http://concrete-semantics.org.

[11] K. Kreuzer, CRYSTALS-Kyber, Archive of Formal Proofs (2022). https://isa-afp.org/entries/CRYSTALS-Kyber.html, Formal proof development.

[12] F. Haftmann, M. Wenzel, Constructive Type Classes in Isabelle, in: T. Altenkirch, C. McBride (Eds.), Types for Proofs and Programs, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 160–174.

[13] C. Ballarin, Locales and Locale Expressions in Isabelle/Isar, in: S. Berardi, M. Coppo, F. Damiani (Eds.), Types for Proofs and Programs, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 34–50.

[14] C. Kaliszyk, C. Urban, Quotients Revisited for Isabelle/HOL, 2011, pp. 1639–1644.

[15] B. Huffman, O. Kunčar, Lifting and Transfer: A Modular Design for Quotients in Isabelle/HOL, in: Certified Programs and Proofs, Springer International Publishing, 2013, pp. 131–146.

[16] J. Harrison, A HOL Theory of Euclidean space, in: J. Hurd, T. Melham (Eds.), Theorem Proving in Higher Order Logics, 18th International Conference, TPHOLs 2005, volume 3603 of *Lecture Notes in Computer Science*, Springer-Verlag, Oxford, UK, 2005.

[17] G. Alagic, D. A. Cooper, Q. Dang, T. Dang, J. M. Kelsey, J. Lichtinger, Y.-K. Liu, C. A. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, D. Smith-Tone, D. Apon, Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process, 2022.

[18] A. Langlois, D. Stehlé, Worst-case to average-case reductions for module lattices, Des. Codes Cryptogr. 75 (2015) 565–599.

[19] A. Sprenkels, The Kyber/Dilithium NTT, 2022. URL: https://electricdusk.com/ntt.html.

[20] T. Ammer, K. Kreuzer, Number Theoretic Transform, Archive of Formal Proofs (2022). https://isa-afp.org/entries/Number_Theoretic_Transform.html, Formal proof development.