Verification of Correctness and Security Properties for CRYSTALS-KYBER

Katharina Kreuzer[1][0000−0002−4621−734X]

Technical University of Munich
Boltzmannstr. 3, 85748 Garching, Germany
k.kreuzer@tum.de
https://www21.in.tum.de/team/kreuzer/index.html

Abstract. This paper describes a formalization of the specification and the algorithm of the public key encryption scheme CRYSTALS-KYBER as well as the verification of its δ-correctness and indistinguishability under chosen plaintext attack (IND-CPA) security proof. The algorithms and proofs were formalized with only minimal assumptions in a modular way to verify the proofs for all possible parameter sets. During the formalization in this flexible setting, problems in the correctness proof were uncovered. Furthermore, the security of CRYSTALS-KYBER under IND-CPA was verified using a game-based approach. As the security property does not hold for the original version of CRYSTALS-KYBER, we only show the IND-CPA security for the latest versions. The security proof was verified under the hardness assumption of the module Learning-with-Errors Problem. The formalization was realized in the theorem prover Isabelle and is foundational.

Keywords: post-quantum cryptography · CRYSTALS-KYBER · number theoretic transform · security · 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.

The winner of the NIST standardization process for public key encryption (PKE) and key encapsulation methods (KEM) was announced in July 2022. It is the key encapsulation mechanism CRYSTALS-KYBER (abbreviated as Kyber throughout this presentation) which was originally developed by Joppe Bos *et al.* [9]. In the first submission to the NIST standardisation process [4], the algorithms from the original paper are extended by sampling methods using

pseudorandom functions and an encoding and decoding function for mapping bits to polynomials and vice versa. A main change to the submission in the second round [3] was excluding the compression and decompression functions in the key generation and encryption functions. The reason is that a problem in the security proof under the indistinguishability under chosen plaintext attack (IND-CPA) was found by Jan-Pieter D'Anvers. Furthermore the use of a slightly different algorithm for fast multiplication allowed the use of a smaller prime for the finite field. For the last submission in round three [2] in October 2020, only small parameter changes have been made.

In contrast to other lattice-based crypto systems like NTRU [33] by Chen *et al.*, Kyber works over a module instead of a polynomial ring. This combines the advantages of working over polynomials and vectors. In order to exploit the structures given by the underlying module, the number theoretic transform (NTT) is used for fast multiplication. The NTT is a version of the Discrete Fourier Transform adapted to finite fields and structures thereover. An optimized version of the NTT uses butterfly schemes like Cooley-Tukey or Gentleman-Sande. The use of the NTT in lattice-based cryptography is described in [27] and its connection to nega-cyclic convolutions in [20].

1.1 Our Contribution

The formalization can be found in [21] and [**?**]. We have formalized the algorithms for key generation, encryption and decryption of the PKE scheme of Kyber both for the original [4,9] and the latest versions [2,3]. The formalization only uses minimal assumptions in order to allow for instantiations with various parameter sets. As an example, some parameter sets as used in [9] and [3] were formalized as well.

During the verification of the δ-correctness proof, we noticed two problems. Firstly, we could only verify the δ-correctness for a slightly modified δ. The claim for δ-correctness of Kyber as in [9] could not be formalized. This issue has already been pointed out by Manuel Barbosa as mentioned in [18]. We investigate the reason and explain our modifications to the error bound δ. Secondly, our modular formalization of the δ-correctness proof revealed a necessary assumption on the modulus q used in the Kyber specification parameters which was not explicitly mentioned in the papers [2–4,9]. In the parameter set of Kyber, this assumption follows indirectly from the choice of the modulus q for the number theoretic transform (NTT), an algorithm for fast multiplication used in Kyber. Since the chosen parameter q fulfils this assumption, this part of the proof remains valid. If the parameters need to be changed in the future, it is important to keep both the new assumption for the δ-correctness proof and the NTT in mind.

In order to refine the algorithm, the NTT on polynomials as used in the original version of Kyber, as well as its convolution theorem have also been formalized for this article.

In our formalization, we only verified the game-based security proof against IND-CPA of Kyber without the compression of the public key (i.e., the latest version of Kyber [2,3]). The security property against IND-CPA for the original

version of Kyber [9] and the version submitted to the first round of the NIST standardisation process [4] does not hold as remarked by the authors in [9]. This is the reason why the authors [9] chose to omit the compression of the public key from the submissions for the second and subsequent roundss of the NIST standardisation process. The proof stated in [9] is thus valid for Kyber without the compression of the public key under the module-Learning-with-Errors (module-LWE) hardness assumption. The game version for the module-LWE was formalized as a building block for the security proof. If the hardness assumption on the module-LWE problem is true, the advantage against the module-LWE game is negligible (i.e., small enough).

1.2 Related Work

A short version on the formalization of the δ-correctness of the original version of Kyber can be found in [22]. Meijers *et al.* [7, 8] announced a formalization of Kyber in EasyCrypt [11]. Furthermore, a post-quantum version of EasyCrypt called EasyPQC is being developed [6]. However, to the best of the authors' knowledge, there is, up to now, no publication or publicly accessible formalization of Kyber, its correctness proof or the IND-CPA security proof. As mentioned in [18], Manuel Barbosa noticed a problem in the pen-and-paper proof of the δ-correctness proof during the formalization.

Due to this problem, the δ-correctness fails to comply with the necessary conditions for a Fujisaki-Okamoto (FO) transform along the lines of [14] (and its formalization by Unruh [35]). In order to fill this gap, new conditions for an alternative FO transform have been found by Hövelmanns *et al.* [18]. A formalization of the FO variant is out of scope for this work.

Recently, a verification of the lattice-based post-quantum crypto system Saber [23] was published [16]. The verification is based on EasyCrypt. Saber has many similarities to Kyber: They both work over the same module. The main difference lies in the underlying hardness assumption. Saber is based on the module-Learning-with-Rounding problem. Instead of adding random errors as in Kyber, Saber generates errors by adding some fixed error and rounding. Moreover, Saber does not use any compression function on the output as Kyber does. Therefore, the δ-correctness proof of Saber does not encounter the problems noticed in Kybers correctness proofs.

Last year, the NTT was verified in CryptoLine by Hwang *et al.* in [17]. CryptoLine is a tool for low-level verification of implementations which stands in contrast to our high level verification of the mathematics behind Kyber.

1.3 Isabelle/HOL

All formalizations and verifications were implemented in the theorem prover Isabelle. An introduction to Isabelle can be found in [30] and [29]. In contrast to other cryptographic verification tools like EasyCrypt, Isabelle is foundational and everything is proved from the axioms of HOL. We formalize and verify

the mathematical principles and algorithms used in Kyber and analyze the δ-correctness and the IND-CPA security property. This is done on a high-level, abstract view of Kyber and is not restricted to an implementation.

Two main features in Isabelle support abstraction over a context of assumptions: The type class constraints (introduced in [12]) and explicit assumptions summarized in a context called locale (introduced in [5]). As in many functional languages, type classes allow ad-hoc polymorphisms. Assumptions are integrated in the type class definition and thus can be used for resoning steps. The module system in Isabelle using locales enables the proofs to be independent from concrete instantiations of variables. This flexible implementation allows the user to verify proofs for different parameter inputs. A nice example is the change of the prime underlying the finite field from the submission of round one [4] to round two [3]. Since the verified proof in Isabelle is independent of the parameters, this change in the parameter set could be verified by simply invoking another locale instance. However, the changes in the algorithms still need to be verified independently.

For the game-based security proof of the IND-CPA property, we based our formalizations on the developement of cryptographic game-hop proofs CryptHOL by Lochbihler [25, 26]. Furthermore, we make use of the extensive libraries in Isabelle for example for concepts from algebra (e.g., vectors, polynomials, quotient rings, finite fields, etc.), analysis (e.g., norms, lemmas about sums and integrals), probability theory (e.g., (sub-)probability mass functions, expectations and probabilities) and many more.

1.4 Structure

In this paper, we discuss the formalization and verification of Kyber and its δ-correctness proof, as well as the game-based IND-CPA security proof for Kyber. 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 some values given in [9] and [2]. Next, we describe the formalization of the algorithms for compression, decompression, key generation, encryption and decryption used in the original and recent versions of Kyber in Section 4. The recent version differs from the original by omitting the compression of the output of the key generation. In Section 5, we proceed with the verification of the δ-correctness proof of Kyber for both previously presented versions. Here, we recognise two problems in the proof: On the one hand, we can only show δ-correctness for a modified δ. We analyze why the original proof could not be formalized and how a modification on δ can fix this issue. On the other hand, we inspect a problem with the inequalities in the proof which we can solve by adding an assumption on the modulus q. This is discussed in Section 5.6. This newly found assumption is already fulfilled when working in the NTT domain. The formalization of the NTT on polynomials and its convolution theorem is analyzed in Section 6. In Section 7, we give a short introduction to game-based cryptography and define the game

versions of the IND-CPA security game and the module-LWE problem. As the security proof was formalized using the framework CryptHOL [25], we point out important concepts for formalizing cryptographic security proofs in Isabelle in Section 8. The formalization of the game-based security proof of Kyber against IND-CPA follows in Section 9. In the end, we give a short outlook on further research questions. The full formalization can be found in [21] and [**?**].

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 of how to formalize a quotient ring when factoring a polynomial ring with coefficients 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. 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 corresponding 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* [19]. 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 [13]. For example, when working with vectors in $\mathbb{Z}^k$, we use the type *(int, 'k) vec*. Here, *'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. In the mathematical literature, this distinction is often blurred to enable a shorter presentation.

3 Formalizing the Parameters of Kyber

The parameters of the original version of Kyber [9] are given in Table 1.

Table 1: Original parameter set of Kyber [9]

variable	value	context
n	$256 = 2^{n'}$	degree of cyclotomic polynomial
n'	8	exponent of 2 in 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

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 (and d_t for Kyber with compression of the public key) are positive integers
- $n = 2^{n'}$ is a power of 2
- $q > 2$ is an odd prime with $q \mod 4 = 1$ (the latter is an additional assumption and will be discussed in Section 5.6)

This is especially of interest for eventual changes in the parameter set. Furthermore, different security level implementations use different parameters. For example, the initial parameter of the modulus q in [9] is 7681, but since round two of the NIST standardisation process [2,3], Kyber uses the modulus 3329 and adapted d_u and d_v. Furthermore, different sizes k of vectors (and adapted d_u and d_v) define different security levels. The parameter sets for different security levels from the third round specification of Kyber [2] can be found in Table 2.

Table 2: Parameter set of round three Kyber [2]

variable	Kyber512	Kyber768	Kyber1024	context
n	256	256	256	$= 2^{n'}$ degree of cyclotomic polynomial
n'	8	8	8	exponent of 2 in degree n
q	3329	3329	3329	prime number, modulus
k	2	3	4	dimension of vectors
d_u	10	10	11	digits for encryption of u
d_v	4	4	5	digits for encryption of v

In our formalization, we instantiate the locale containing the original Kyber algorithm and proof of δ-correctness with the parameter set given in Table 1 and the algorithms without compression of the public key with the parameter set given in Table 2 for Kyber768. 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 (and 3329 elements respectively) was instantiated manually for prime cardinality.

4 Formalizing the Kyber Algorithm

The PKE scheme Kyber is divided into 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. However, the compression of the public key breaks the security proof against IND-CPA. Therefore, since the submission to round two of the NIST standardisation process, this compression of the public key was left out.

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 the inputs $A \in R_q^{k\times k}$, $s \in R_q^k$ and $e \in R_q^k$ which are 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 centered 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 the original version of Kyber [9], we use $\eta = 4$.

Note that in the more recent submissions of Kyber, the value η determining the variance of the centered binomial distribution was changed as well. Again, the formalization in locales allows us to easily change these values of η. However, in the third submission round [2], two separate values η_1 and η_2 have been introduced. This distinction has not been formalised.

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

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^k. Let $t = As + e$, then the (decision) module-LWE problem asks to distinguish (A, t) from uniformly random $(A', t') \in R_q^{k\times k} \times R_q^k$.*

There is a probabilistic reduction proof for the NP-hardness of the module-LWE by Langlois and Stehlé [24]. 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. This NP-hardness property is also called the module-LWE hardness assumption.

Note that the module-LWE problem without the error term would be easy to solve using the Euclidean Algorithm. Thus, the error term cannot be reused but has to be chosen according to the distribution β_η^k again. The random choices and the reduction to the module-LWE have been formalized in the IND-CPA security proof for Kyber. The NP-hardness proof of the module-LWE has not been formalized.

4.2 Compression and Decompression

The compression and decompression functions in Kyber help to reduce the public and secret key sizes (only in the original version) and obscure 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 "mod 2^d" to denote the modulo operation with modulus 2^d, yielding the unique representative in $\{0, \ldots, 2^d - 1\}$.

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 representative 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 c_x. 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.

In the following, for a value x, we will denote the compression of x by x^* and the decompression of the compression as x'.

4.3 Key Generation, Encryption and Decryption

We now want to state the actual algorithm. Let $t = A \cdot s + e$ and d_t be the compression depth associated to t. Then the algorithm for key generation as in the original version of Kyber [9] is defined in the following way:

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

We denote by $t^* = key\_gen\ dt\ A\ s\ e$ the output of the key generation algorithm with compression.

Since round two of the NIST standardisation process [3], the key generation is defined without the compression of the output. The new key generation algorithm reads:

$$key\_gen'\ A\ s\ e = A \cdot s + e$$

The output of the key generation without compression is t. Together, the matrix A and $t^{(*)}$ constitute the public key, whereas the vector s is the secret key. When we say that the public and secret key pair $(A, t^{(*)})$ and s are generated by the key generation algorithm, we mean the probabilistic program where A, s and e are chosen according to their distributions, $t^{(*)}$ is calculated by $key\_gen^{(\prime)}$ and $(A, t^{(*)})$ and s are the output.

The pair (A, t) also forms 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 pair (A, t). Therefore, the security of Kyber is based on the hardness of module-LWE.

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_η^k and β_η. (Note that in round three, the value of η in the distributions of e_1 and e_2 have been slightly changed.) We then calculate the encryption in the original scheme:

$$\begin{aligned}&\mathit{encrypt}\ t^*\ A\ r\ e_1\ e_2\ dt\ du\ dv\ m =\\ &\quad(\mathit{compress\_vec}_{du}\ (A^T \cdot r + e_1),\\ &\quad\ \mathit{compress\_poly}_{dv}\ ((\mathit{decompress\_vec}_{dt}\ t^*)^T r + e_2 + \mathit{to\_module}(\lceil q/2 \rfloor) \cdot m))\end{aligned}$$

Since round two of the NIST standardisation process, a decryption of the public key is no longer necessary. Thus, the encrytion algorithm reads:

$$\begin{aligned}&\mathit{encrypt}'\ t\ A\ r\ e_1\ e_2\ du\ dv\ m =\\ &\quad(\mathit{compress\_vec}_{du}\ (A^T \cdot r + e_1),\\ &\quad\ \mathit{compress\_poly}_{dv}\ (t^T r + e_2 + \mathit{to\_module}(\lceil q/2 \rfloor) \cdot m))\end{aligned}$$

Let $u = A^T \cdot r + e_1$ and $v = t^{(*)T} r + e_2 + \mathit{to\_module}(\lceil q/2 \rfloor) \cdot m$. Then, the encryption outputs the compressed values u^* and v^* in a pair (u^*, v^*). When referring to the encryption without the input of r, e_1 and e_2, we mean the probabilistic program that first generates r, e_1 and e_2 according to their distributions and then calculates the encryption functions as stated above.

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.

$$\begin{aligned}&\mathit{decrypt}\ u^*\ v^*\ s\ du\ dv =\\ &\quad\mathit{compress\_poly}_1\ ((\mathit{decompress\_poly}_{dv}\ v^*) - s^T(\mathit{decompress\_vec}_{du}\ u^*))\end{aligned}$$

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 δ-correctness proof of Kyber. The following section describes a verification of this proof in Isabelle.

5 Verifying the δ-Correctness Proof of Kyber

To verify the δ-correctness of the specification of Kyber in Isabelle, we look at the pen-and-paper proof from [9, Theorem 1]. This proof shows the correctness of the original version of Kyber, but can also be easily adapted to the recent versions omitting the compression of the public key. We will have a look at the ensuing changes in the correctness proof in Section 5.5 (formalized in [**?**]).

5.1 $\|\cdot\|_\infty$ – a Wolf in Sheep's Clothing

In order to estimate values, the authors [9] use a function $\|\cdot\|_\infty$. However, it is defined slightly differently from what one would expect: Instead of using a

regular modulo operation, we define the recentered operation $\mathrm{mod}^{\pm}$ to be the representative with smallest norm. That means $\bar{a} := (a \mod^{\pm} q)$ 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 $a \mod^{\pm} q \in \{\frac{-q+1}{2}, \ldots, \frac{q-1}{2}\}$. Using this recentered modulo operation, we define the function $\|\cdot\|_\infty$ on polynomials as:

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

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

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

Unfortunately with the recentering one looses the absolute homogeneity, i.e., for a scalar s and vector v only $\|s \cdot v\|_\infty \leq |s| \cdot \|v\|_\infty$ 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:

$$\|2 \cdot (2)\|_\infty = |2 \cdot (2) \mod^{\pm} 3| = 1 < 2 = |2| \cdot |2 \mod^{\pm} 3| = |2| \cdot \|(2)\|_\infty$$

Therefore, the $\|\cdot\|_\infty$ function is not a norm, but a pseudonorm. It is positive definite and fulfils the triangle inequality. This is not explicitly mentioned in [9] and indeed poses a problem in the proof of the following theorem.

5.2 Correctness of the Original Kyber Algorithms

A crypto system is correct, if it always returns the original message. However, this can only be guaranteed under a certain condition. Thus, we need to consider a failure probability and can only state the δ-correctness. This is defined in the following:

Definition 2 (δ-correct PKE). *Let key_gen, encrypt and decrypt constitute a public key encryption scheme $\mathcal{A}$ where key_gen outputs a public key pk and a secret key sk. Let $\mathcal{M}$ be the space of all possible messages. Then the public key encryption scheme is δ-correct, if and only if:*

$$\mathbb{E}[\max_{m \in \mathcal{M}} \mathbb{P}[\mathit{decrypt}(sk, \mathit{encrypt}(pk, m)) \neq m]] \leq \delta$$

where the expectation is taken over (pk, sk) generated by key_gen.

The notion of a δ-correct PKE was taken from the requirements for the verified Fujisaki-Okamoto transform given by Unruh in [35].

For the original Kyber algorithms [9, Theorem 1], the δ-correctness theorem is proven in two steps: First, the distributions for the compression errors are related with the module-LWE problem. Second, the main proof follows by a deterministic calculation under a assumption given by the definition of δ. If the assumption is not fulfilled, the proof cannot guarantee correctness of the Kyber scheme. The second, deterministic part is stated in the following theorem which is formalized in [21].

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

- $t^* = key\_gen\ d_t\ A\ s\ e$, *the output of the key generation*
- $(u^*, v^*) = encrypt\ t^*\ A\ r\ e1\ e2\ dt\ du\ dv\ m$, *the output of the encryption*
- c_t, c_u *and* c_v, *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:

$$decrypt\ u\ v\ s\ du\ dv = m$$

We have that Kyber is correct 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 1 and the definition of δ-correctness, we deduce the following.

Corollary 1. *Let:*

$$\delta' = \mathbb{E}\left[\max_{m\in\mathcal{M}} \mathbb{P}\left[\begin{cases} e, r, e_1 \leftarrow \beta_\eta^k; \quad e_2 \leftarrow \beta_\eta; \\ u = A^T r + e_1; \quad v = t^T r + c_t^T r + e_2 + \lceil \frac{q}{2} \rfloor m; \\ \|e^T r + e_2 + c_v - s^T e_1 + c_t^T r - s^T c_u\|_\infty \geq \lceil q/4 \rfloor \end{cases}\right]\right]$$

where the expectation is taken over $((A, t^*), s)$ generated by key_gen. Then the original version of Kyber is δ'-correct.

Note that in this proposition, the δ' is not the same as in [9, Theorem 1], where the authors set

$$\delta = \mathbb{P}\left[\begin{cases} s, e, r, e_1 \leftarrow \beta_\eta^k; \quad e_2 \leftarrow \beta_\eta; \\ c_t, c_u, c_v \leftarrow \Psi_d^k \\ \|e^T r + e_2 + c_v - s^T e_1 + c_t^T r - s^T c_u\|_\infty \geq \lceil q/4 \rfloor \end{cases}\right]$$

Here, Ψ_d^k is the distribution of the compression error of x in pseudonorm for a uniformly generated $x \leftarrow R_q^k$.

The main difference between δ' and δ is that in δ' the values of c_t, c_u and c_v are calculated as the correct compression errors whereas in δ, they are the compression errors of uniformly random values t, u and v. The authors claim in [9, Proof of Theorem 1] that this change is negligible since its value can be bounded by the advantage against module-LWE problems.

However, when formalizing this claim, this implication could not be proven. The reason is that the change from a module-LWE instance to a uniformly random instance looses all information about the secret key. As the secret key is dependent on the chosen public key but independent of a uniformly random instance, this definition of the module-LWE makes sense. In the definition of δ-correctness, we cannot omit the information about the secret key since we need it for the decryption. Therefore, we cannot separate the module-LWE instance from $\mathbb{P}[decrypt(sk, encrypt(pk, m)) \neq m]$ in order to bound this value with the advantage against the module-LWE.

The formalization shows only δ'-correctness for Kyber. δ' and δ could not be related.

5.3 Correctness of Kyber without Public Key Compression

The deterministic part of the correctness proof for the later versions of Kyber [2,3] is analogous to the previous Theorem 1, only omitting the compression of the public key t , which is formalized in [**?**]:

Theorem 2. *Let $A \in R_q^{k\times k}$, $s, r, e, e_1 \in R_q^k$, $e_2 \in R_q$ and let the message $m \in R_q$ with coefficients in $\{0,1\}$. Define:*

- *$t = key\_gen'\ A\ s\ e$, the output of the key generation*
- *$(u^*, v^*) = encrypt'\ t\ A\ r\ e1\ e2\ du\ dv\ m$, the output of the encryption*
- *c_u and c_v, the compression errors of u and v, respectively*

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

$$decrypt\ u^*\ v^*\ s\ du\ dv = m$$

Again, we have that Kyber is correct when assuming the inequality:

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

Using Theorem 2, we deduce the δ'-correctness of Kyber without compression of the public key.

Corollary 2. *Let:*

$$\delta' = \mathbb{E}\left[\max_{m\in\mathcal{M}} \mathbb{P}\left[\left\{\begin{array}{ll} e, r, e_1 \leftarrow \beta_\eta^k; & e_2 \leftarrow \beta_\eta; \\ u = A^T r + e_1; & v = t^T r + e_2 + \lceil \frac{q}{2} \rfloor m; \\ \|e^T r + e_2 + c_v - s^T e_1 - s^T c_u\|_\infty \geq \lceil q/4 \rfloor & \end{array}\right.\right]\right]$$

where the expectation is taken over $((A,t),s)$ generated by key_gen'. Then Kyber without compression of the public key is δ'-correct.

Again, the δ' from the corollary above is different from the δ claimed in [2,3]:

$$\delta = \mathbb{P}\left[\left\{\begin{array}{ll} s, e, r, e_1 \leftarrow \beta_\eta^k; & e_2 \leftarrow \beta_\eta; \\ c_u, c_v \leftarrow \Psi_d^k & \\ \|e^T r + e_2 + c_v - s^T e_1 - s^T c_u\|_\infty \geq \lceil q/4 \rfloor & \end{array}\right.\right]$$

The reason is the same as discussed at the end of Section 5.2 for the original Kyber algorithms. The formalization shows only δ'-correctness for Kyber. δ' and δ could not be related.

5.4 Auxiliary Lemma

Before we can start the proof of Theorem 1 (and Theorem 2), we need to show an auxiliary lemma on the estimation of the compression error.

Lemma 1. *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:*

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

Proof. Let x be the representative in $\{0, \ldots, q-1\}$. Then consider two cases, namely $x < \lceil q - \frac{q}{2^{d+1}} \rceil$ and $x \geq \lceil q - \frac{q}{2^{d+1}} \rceil$. 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 $x \geq \lceil q - \frac{q}{2^{d+1}} \rceil$. 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 $compress_d\ x$ by x^*.

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

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

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

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

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

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

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

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

□

A non-trivial step in the formalization of 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.

5.5 Proof of Correctness

The formalization of the proof of Theorem 1 can be found in [21], the proof of Theorem 2 in [**?**]. One problem encountered during the formalization was that $\|\cdot\|_\infty$ is only a pseudonorm (recall Section 5.1). This is not explicitly mentioned in [9] and indeed poses a problem in the proof which we will discuss in greater detail in the next section. In short: We cannot conclude a correct decryption in the last step of the correctness proof unless $q \equiv 1 \mod 4$.

The proof of Theorem 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 t', u' and 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:

$$\begin{aligned} t' &= As + e + c_t \\ u' &= A^T r + e_1 + c_u \\ v' &= t'^T r + e_2 + \lceil q/2 \rfloor \cdot m + c_v \end{aligned}$$

This leads to the calculation in the decryption:

$$v' - s^T u' = e^T r + e_2 + c_v + c_t^T r - s^T e_1 - s^T c_u + \lceil q/2 \rfloor \cdot m$$

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.

In the case of Theorem 2 where the compression is omitted, we have $v' = t^T r + e_2 + \lceil q/2 \rfloor \cdot m + c_v$. Then we calculate in the decryption:

$$v' - s^T u' = e^T r + e_2 + c_v - s^T e_1 - s^T c_u + \lceil q/2 \rfloor \cdot m$$

Again, accumulating all error terms in a new variable $\widetilde{w}$, we define:

$$\widetilde{w} := e^T r + e_2 + c_v - s^T e_1 - s^T c_u$$

and get $\|\widetilde{w}\|_\infty < \lceil q/4 \rfloor$ from the assumptions. In the rest of the proof, we calculate with w. An analogous proof for Theorem 2 can be obtained by exchanging w with $\widetilde{w}$.

Now, we need to show that $m' := decrypt(u, v, s)$ is indeed the original message m. We consider the value of $v' - s^T u'$, its compression with $d = 1$, namely m', and the decompressed value $decompress_1\ m'$. Since the compression depth is 1, we get $m' \in \{0, 1\}$. Thus:

$$decompress_1\ m' = \lceil q/2 \cdot m' \rfloor = \lceil q/2 \rfloor \cdot m'$$

Using Lemma 1, it follows that:

$$\begin{aligned} &\|w + \lceil q/2 \rfloor (m - m')\|_\infty = \\ =&\|v' - s^T u' - decompress_1\ (compress_1\ (v' - s^T u'))\|_\infty \leq \lceil q/4 \rfloor \end{aligned}$$

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

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

It remains to show that we can indeed deduce $m = m'$ which concludes the proof of Theorems 1 and 2. According to the last step of [9, Proof Thm 1], this follows directly for any odd prime q. However, therein lies a hidden problem. [9, Proof Thm 1] makes use of the homogeneity of $\|\cdot\|_\infty$. Since $\|\cdot\|_\infty$ is only a pseudonorm and not a norm, we needed to find an alternative proof in the formalization. Interestingly enough, in the case of $q \equiv 3 \mod 4$, we cannot conclude the proof. In the next section, we discuss why we can only deduce this step under the assumption that $q \equiv 1 \mod 4$ and give a counterexample for the case $q \equiv 3 \mod 4$.

5.6 Additional Assumption $q \equiv 1 \mod 4$

The following remains to be shown for the proof of Theorem 1 and 2: Given the inequality

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

we need to deduce that indeed $m = m'$.

We prove this statement by contradiction. Assume that $m - m'$, i.e., 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

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

Since we cannot use the homogeneity of $\|\cdot\|_\infty$ to pull out the absolute value of ± 1, we need to find a different proof. We break down the formula to find the remaining problems. All primes q greater than two are odd. Thus we have $\lceil q/2 \rfloor = (q + 1)/2$. We continue our calculation:

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

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. In this case, the proof of Theorems 1 and 2 is completed.

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.

Example 1. 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 $m \neq m'$. This is a counterexample for the correctness of the proof of Theorem 1 (and 2) in the case $q \equiv 3 \mod 4$.

In conclusion, Theorem 1 only holds if the modulus q fulfils the assumption $q \equiv 1 \mod 4$. □

In the specification of Kyber, concrete values for the variables of the system are given (see Section 3). For example in the recent version of Kyber [2,3], the modulus q is chosen to be 3329, whereas in early versions [4,9], the modulus was chosen as $q = 7681$. Considering possible changes to these variables (for different versions or security levels), it is important to enable the verified proof to cover all possible cases. Therefore, the implementation of the formalization was chosen to be as adaptive and flexible as possible. This resulted in the discovery of the additional assumption $q \equiv 1 \mod 4$.

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 $n = 256$ and $q = 7681$ 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 have a more thorough look at the NTT used in Kyber in the next section.

6 NTT and the Convolution Theorem

The NTT is used to speed up the multiplication on $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ 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 [27] or for the special case of the CRYSTALS suite in [34]. The NTT as a nega-cyclic convolution is described in [20].

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 in R_q 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 the calculations in Kyber crypto algorithms is 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, the original 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 for Kyber [9] as follows:

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

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

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 as:*

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

We formalized a proof of correctness of the NTT and its inverse [21] .

Theorem 3. *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, resulting 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 $2n$-th root of unity, that is $\psi^{2n} = 1$. We continue our calculation:

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

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 4. *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.

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

Using $\psi^{n(2k+1)} = -1$, 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:

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

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 3 this yields the fast multiplication formula.

Theorem 5. *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 the original Kyber [9] was realtively straightforward since it is based on the formalization of the standard NTT by Ammer in [1]. 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.

Since the NTT for the recent version of Kyber [2] was also formalized in [17], we verified only the NTT for the original Kyber specifications. Note that the NTT for the latest versions of Kyber [2,3] is a bit different, since the finite field $\mathbb{Z}_{3329}$ does not contain a $2n$-th root of unity, but only an n-th root of unity.

7 Game-Based Cryptography

An important cryptographic property of public key encryption schemes is the security against IND-CPA. This attack describes a game where an adversary tries to gain information or knowledge from the ability of choosing plaintexts.

More formally, the IND-CPA game for a PKE (given by the key generation, encryption and decryption algorithms) is defined as follows.

Definition 5 (IND-CPA game). *Two parties, the challenger and the adversary, play the following game.*

1. *The challenger generates a public and secret key pair using the key generation algorithm and publishes the public key.*

2. *The adversary sends the challenger two messages m_0 and m_1 with the same length.*
3. *The challenger chooses uniformly at random a bit b. He encrypts the message m_b with the encryption algorithm and sends the ciphertext to the adversary.*
4. *The adversary returns a guess b' which of the two given messages m_0 and m_1 the challenger has encrypted. He wins if $b = b'$.*

The advantage $Adv^{IND-CPA}$ of the adversary $\mathcal{A}$ is defined as $Adv^{IND-CPA}(\mathcal{A}) = |\mathbb{P}[b' = b] - \frac{1}{2}|$. A PKE scheme is IND-CPA secure if and only if the advantage of the adversary is negligible, that means sufficiently small.

Figure 1 depicts the IND-CPA game.

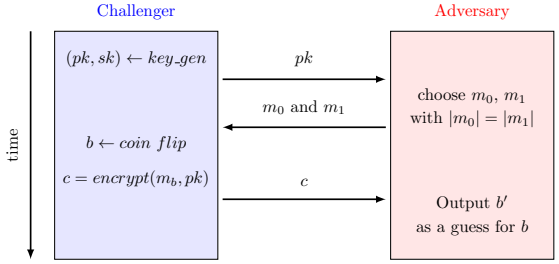

Fig. 1: A diagram of the IND-CPA game.

The formalization of the IND-CPA game was taken from the CryptHOL Tutorial [26]. The flexible formalization in an Isabelle locale allows the user to instantiate this concept in any context fulfilling the properties of the locale. In this way, the IND-CPA game definition could easily be applied to the case of Kyber by instantiating with the Kyber algorithms for key generation, encryption and decryption.

We can also state the module-LWE from Definition 1 in game form.

Definition 6 (module-LWE game). *Two parties, called the challenger and the (module-LWE) adversary, play the following game.*

1. *The challenger chooses $A_0 \in R_q^{m \times k}$ uniformly at random, s according to β_η^k and e according to β_η^m. He then computes $t_0 = A_0 s + e$. ((A_0, t_0) is an instance generated by the module-LWE problem.)*
2. *The challenger chooses $A_1 \in R_q^{m \times k}$ and $t_1 \in R_q^m$ uniformly at random. ((A_1, t_1) is a random instance.)*
3. *The challenger chooses a random bit b and sends the adversary the value of (A_b, t_b).*
4. *The adversary returns a guess b' whether the tuple (A_b, t_b) was generated as a module-LWE instance or is uniformly random. He wins, if his guess is correct.*

The advantage Adv_m^{mLWE} of the module-LWE adversary $\mathcal{A}$ is defined as

$$Adv_m^{mLWE}(\mathcal{A}) = |\mathbb{P}[b' = 0 \wedge b = 0] - \mathbb{P}[b' = 0 \wedge b = 1]|$$

The module-LWE hardness assumption states that the advantage against an adversary in the module-LWE game is negligible, that means sufficiently small.

Figure 2 depicts the module-LWE problem in game form.

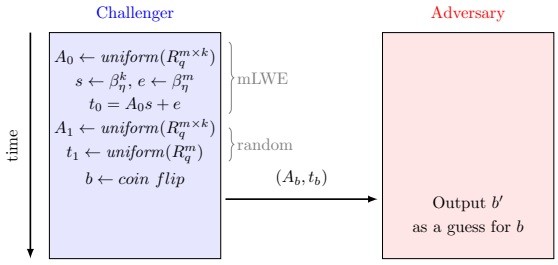

Fig. 2: A diagram of the module-LWE game.

In the proof of the IND-CPA security property for Kyber, the advantage of a module-LWE adversary is used twice, but with different dimensions m. The key generation corresponds to a module-LWE with $m = k$ such that A is a quadratic matrix. However, in the encryption, the matrix A is extended by the vector t, resulting in a $(k + 1) \times k$ matrix. This corresponds to the module-LWE with $m = k + 1$.

The module-LWE was again formalized in an Isabelle locale in order to allow for two separate instantiations (once with $m = k$ and once with $m = k + 1$). However, the instantiations needed an additional twist. Since the vector type in Isabelle has a fixed dimension implemented as a finite type (in our case type $'k$ of cardinality k), it is more difficult to work over vectors whose dimension is a function over k. In our case, this could be solved using the option type. The option type $'k\ option$ embeds elements a of type $'k$ as $Some\ a$ and adds the element $None$. Thus $'k\ option$ has exactly $k + 1$ elements. This solves our problem.

8 Using CryptHOL in Isabelle

CryptHOL [25] is a library for game-based security proofs in cryptography. It is based on the extensive libraries for probability theory in Isabelle. Its main contributions are subprobability mass function as the type class *spmf* and generative probabilistic values as the type class *gpv*. We give a short intuitive understanding of these type classes.

8.1 Subprobability Mass Functions

The *spmf* type class is a superclass of probabilistic mass functions. We consider a finite set S. A probabilistic mass function $f : S \longmapsto [0, 1]$ is the probability distribution of a discrete random variable X, i.e., $f(x) = \mathbb{P}[X = x]$ such that the weight equals one:

$$\sum_{x \in S} f(x) = 1$$

For subprobabilistic mass functions, we allow the weight to be less than one:

$$\sum_{x \in S} f(x) \leq 1$$

A subprobabilistic mass function is called lossless, if it has weight equal to one. Indeed, in our setting we need to model the probability that a security game is compromised by intentional malicious input. For example in the IND-CPA game, the adversary can intentionaly input two messages of different length and thus gain information about the ciphertext. As we only want to look at correctly played security games, we use subprobability mass functions.

8.2 Generative Probabilistic Values

To model cryptographic primitives such as hash functions, we need a method to generate and store random values. This idea is developed in the *gpv* type class which describes probabilistic algorithms. The type class *gpv* depends on three input types: the type of the algorithm, the input state type and the output state type.

When running a *gpv*, we connect it with a random oracle (that models for example a hash function) and hand through the current state. Whenever we query the oracle, we generate a new state. It needs to be included in the input for the next call to the oracle using a *gpv*.

The Kyber public key encryption does not use hash functions. Thus we could model the security proof with subprobability mass functions only. However, to stay consistent with the CryptHOL library, we generalized the formalization of the security proof to use generative probabilistic values whenever we query the adversary or the encryption algorithm. The proofs do not get significantly harder and the automation can hadle this generalization step most of the time.

8.3 Using Monads for Describing Probabilistic Algorithms

Functional programming hands us tools to easily define probabilistic algorithms and distributions. The concept of choice is the Giry-monad. Monads are a concept from category theory applied to functional programming. We give a short introduction to monads in general and the Giry monad in particular. More about monads can be found in [31] and the introduction of monads into functional programming in [28]. A good introduction to the Giry monad in the context of Isabelle is given in [10].

Monads give a pattern to design type classes. They consist of a type constructor M and two operations:

- *return*: recieves a value a and hands back a monadic value $M\ a$
- *bind*: recieves a monadic value Ma and a function $f : a \longrightarrow M\ b$ and returns the application of f to the unwrapped value a, yielding an element $M\ b$

Monads need to fulfil three laws: the left and right identities and the associativity. An example of monads is the Giry monad. It assigns to each measurable space the space of probability measures over it (see [32]).

Example 2. The type class of probability mass functions *pmf* of discrete probability is a monad. The *return* function for an element a is defined as the Dirac measure on a. The *bind* function on an probability mass function p_X using a function f is defined as:

$$(bind\ p_X\ f)(y) = \sum_x p_X(f(x)(y))$$

Thus, the Giry monad can model successive execution of random experiments and probabilistic algorithms using the *bind* and *return* functions.

Both the type class *spmf* and *gpv* are monads with respective *return* and *bind* functions. This gives us a tool to model probabilistic algorithms in Isabelle.

9 IND-CPA Security Proof for Round 2 and 3 Kyber

Since round two of the NIST standardization process [3], the compression of the public key in Kyber has been omitted. The reason was that otherwise the IND-CPA security proof [9, Theorem 2] does not hold. The problem lies in the second reduction step where the decompression of the compression of the public key is not distributed uniformly at random any more. This entails that we cannot apply the reduction to the modul-LWE. The security of Kyber without compression under IND-CPA is stated in the following theorem. Its formalization can be found in [**?**]

Theorem 6. *The public key encryption scheme Kyber without compression of the public key is IND-CPA secure against the module-LWE hardness assumption.*

More formally, this theorem states the following: Given any adversary $\mathcal{A}$ to the IND-CPA game of Kyber and assuming that $\mathcal{A}$ is lossless (i.e., does not act maliciously), the advantage of $\mathcal{A}$ in the IND-CPA game can be bounded by twice the advantage in the module-LWE game.

Proof. Let Adv^{Kyber} be the advantage in the IND-CPA game instantiated with the Kyber algorithms *key_gen*′, *encrypt*′ and *decrypt*. Let f_1 be the reduction function from $\mathcal{A}$ to the first module-LWE instance and f_2 the reduction function from $\mathcal{A}$ to the second module-LWE instance. Then the exact formula of the theorem above reads:

$$Adv^{Kyber}(\mathcal{A}) \leq Adv_k^{mLWE}(f_1(\mathcal{A})) + Adv_{k+1}^{mLWE}(f_2(\mathcal{A})) \tag{1}$$

Note that in the formalization we state the reduction functions for the adversary precisely. They need to have a polynomial running time. Since a formal

framework for analyzing the running time is out of scope for this project, we assume the running time hypothesis to be correct. The reason is that the reduction functions use only one call to the given adversary and one for f_1 the Kyber encryption algorithm. Otherwise, the functions are non-recursive, polynomial time probabilistic algorithms.

The proof of equation (1) proceeds in three steps (also called game-hops).

1. Reduction of key generation to the first module-LWE instance with $m = k$
2. Reduction of encryption to the second module-LWE instance with $m = k+1$
3. Reduction of the rest to a coin flip

In every game-hop, we define an intermediate game and analyze the difference in the advantage. The initial game $game_0$ is exactly the IND-CPA game. That implies:

$$\mathbb{P}[b = b'] = \mathbb{P}[game_0 = True]$$

The first intermediate game $game_1$ is defined by the following steps:

1. The challenger generates a public key (A, t) uniformly at random and publishes the public key.
2. The adversary sends the challenger two messages m_0 and m_1 with the same length.
3. The challenger chooses a bit b uniformly at random. He encrypts the message m_b with the encryption algorithm and sends the ciphertext to the adversary.
4. The adversary returns a guess b' for which of the two given messages m_0 and m_1 the challenger has encrypted. He wins if $b = b'$.

Figure 3 illustrates $game_1$. The change to the initial game $game_0$ (marked in green) is in the first step where the public and secret key pair is now generated uniformly at random instead of being created by the key generation algorithm.

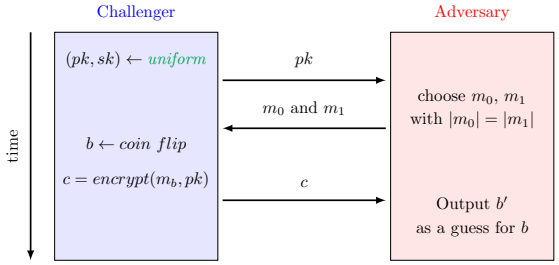

Fig. 3: A diagram of $game_1$.

The key generation algorithm creates a module-LWE instance. Distinguishing an module-LWE instance from a uniformly random instance is exactly the module-LWE game. Hence, the difference

$$|\mathbb{P}[game_0 = True] - \mathbb{P}[game_1 = True]|$$

is the same probability as the advantage Adv_k^{mLWE} for an adversary $f_1(\mathcal{A})$ where f_1 is a suitable reduction function.

The second intermediate game $game_2$ is defined by the following steps:

1. The challenger generates a public key (A, t) uniformly at random and publishes the public key.
2. The adversary sends the challenger two messages m_0 and m_1 with the same length.
3. The challenger chooses a bit b uniformly at random. He chooses a ciphertext uniformly at random from $R_q^k \times R_q$ and sends the ciphertext to the adversary.
4. The adversary returns a guess b' for b. He wins if $b = b'$.

Figure 4 illustrates $game_2$. The change to $game_1$ (marked in green) is that the ciphertext is not generated by the encryption but chosen uniformly at random.

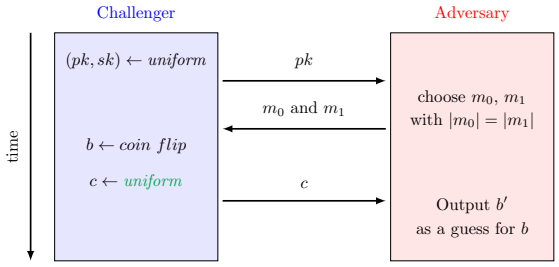

Fig. 4: A diagram of $game_2$.

In the encrpytion, the reduction to the module-LWE is not as straightforward as for the key generation. This is caused by the addition of the message m to the module-LWE instance. Indeed, in the formalization, we need to make two separate steps.

First, we show that the probility of distinguishing an instance of the form

$$(A\ t)\, r + \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}$$

and a uniformly random instance $(u\ v')^T$ is exactly the module-LWE advantage for $m = k + 1$. Note that it is important to look at $(k + 1)$-dimensional vectors instead of splitting the instance in k- and 1-dimensional parts because r is chosen to be the same for the multiplication with both A and t.

Second, we need to show that $v' + \lceil q/2 \rfloor \cdot m$ is also distributed uniformly. That is, we cannot distinguish between the probabilites of the value $v' + \lceil q/2 \rfloor \cdot m$ for a uniformly random v' and a uniformly random v. Since we are working over a finite field and v' and m are independent, we can show this property using the law of total probability.

We deduce that

$$|\mathbb{P}[game_1 = True] - \mathbb{P}[game_2 = True]|$$

equals the advantage Adv_{k+1}^{mLWE} for an adversary $f_2(\mathcal{A})$ where f_2 is a suitable reduction function.

In the last game-hop, we have a closer look at $game_2$. Since the ciphertext sent to the adversary is now independent from the chosen message, the guess of

the adversary is a coin flip. Thus the probability of guessing correctly is exactly 1/2. We get

$$\mathbb{P}[game_2 = True] = 1/2$$

Finally, we can put together all the previous steps.

$$Adv^{Kyber}(\mathcal{A}) = \left|\mathbb{P}[b = b'] - \frac{1}{2}\right| = \left|\mathbb{P}[game_0 = True] - \frac{1}{2}\right|$$

This equality is infered from the definition of the adversary for the IND-CPA game for Kyber. The $game_0$ is the initial IND-CPA game. We continue by applying the triangle inequality.

$$\begin{aligned}
&\left|\mathbb{P}[game_0 = True] - \frac{1}{2}\right| \leq \\
&\leq |\mathbb{P}[game_0 = True] - \mathbb{P}[game_1 = True]| + \left|\mathbb{P}[game_1 = True] - \frac{1}{2}\right| = \\
&= Adv_k^{mLWE}(f_1(\mathcal{A})) + \left|\mathbb{P}[game_1 = True] - \frac{1}{2}\right|
\end{aligned}$$

The last equality is deduced from the reduction of $game_0$ to $game_1$ as a module-LWE instance. We proceed by applying the triangle inequality again on the second part.

$$\begin{aligned}
&\left|\mathbb{P}[game_1 = True] - \frac{1}{2}\right| \leq \\
&\leq |\mathbb{P}[game_1 = True] - \mathbb{P}[game_2 = True]| + \left|\mathbb{P}[game_2 = True] - \frac{1}{2}\right| = \\
&= Adv_{k+1}^{mLWE}(f_2(\mathcal{A})) + \left|\mathbb{P}[game_2 = True] - \frac{1}{2}\right|
\end{aligned}$$

Here, the last equality is deduced from the reduction of $game_1$ to $game_2$ as a module-LWE instance with $m = k+1$. Finally, we have $\left|\mathbb{P}[game_2 = True] - \frac{1}{2}\right| = 0$ as $game_2$ behaves like a coin flip. In total, the claim is proven as we have shown the formula:

$$Adv^{Kyber}(\mathcal{A}) \leq Adv_k^{mLWE}(f_1(\mathcal{A})) + Adv_{k+1}^{mLWE}(f_2(\mathcal{A})) \qquad \square$$

During the formalization process, it became clear that this proof does not work for Kyber with compression of the public key as remarked by the authors of Kyber [9, Sec. Security of the real scheme]. The proof for the scheme without compression of the public key could be formalized analogously to the pen-and-paper proof. The most time-consuming parts were getting familiar with the CryptHOL library environment and working out the details of the pen-and-paper proof which was extremely short.

CryptHOL works with subprobability mass functions and generative probabilistic values and supplies a huge library of fundamental lemmas. Since the

example game-based proof of the CryptHOL Tutorial [26] is based mainly on the automation, understanding the formal proof and rewriting steps is not straightforward. However, once the necessary lemmas are located and added to the automation, the automatic proof finder can solve most rewriting steps.

Some steps where the automation fails are for example when commutativity laws need to be applied in both directions. Then the simplifier runs in loops and cannot terminate. Making smaller proof steps or explicitly initialising the commutativity laws solves these issues.

10 Conclusion

In this presentation, we described the formalization of CRYSTAL-KYBER's key-generation, encryption and decryption algorithms. Both the original version and the latest version omitting the compression of the public key were formalized.

During the formalization of the δ-correctness proof two problems were uncovered: One could be solved by modifying the value of δ, the other by adding the assumption $q \equiv 1 \mod 4$. Under these conditions, the δ'-correctness of both versions could be verified. Differences between the original proof and the formalization were discussed and counterexamples for failing proofsteps were given.

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 NTT used in the original version of Kyber and its convolution theorem were formalized as well. Moreover, a verification of the IND-CPA security proof for Kyber without compression of the public key was presented. The IND-CPA security property does not hold for the original version of Kyber.

Building on these results, the Fujisaki-Okamoto transform can be applied to the current algorithm formalization to obtain a verified key encapsulation mechanism that is secure against the indistinguishability under chosen ciphertext attack (IND-CCA). However, this is not very significant, since the value of the modified δ' has not been estimated or related to the originally claimed δ. A different approach to achieve a formally verified IND-CCA secure transform is to use a different version of the Fujisaki-Okamoto transform. In this case, both the transform and necessary assumptions need to be formalized. Another very interesting aspect is to formalize the hardness results of the module-LWE that Kyber is building on.

Acknowledgements Many thanks to Manuel Eberl and Tobias Nipkow for continuous support and fruitful discussions. 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. Ammer, T., Kreuzer, K.: Number Theoretic Transform. Archive of Formal Proofs (August 2022), `https://isa-afp.org/entries/Number_Theoretic_Transform.html`, Formal proof development

2. Avanzi, R.M., Bos, J.W., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Kyber Algorithm Specifications And Supporting Documentation (version 3.0) (01/10/2020)
3. Avanzi, R.M., Bos, J.W., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Kyber Algorithm Specifications And Supporting Documentation (version 2.0) (30/03/2019)
4. Avanzi, R.M., Bos, J.W., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Kyber Algorithm Specifications And Supporting Documentation (30/11/2017)
5. Ballarin, C.: Locales and Locale Expressions in Isabelle/Isar. In: Berardi, S., Coppo, M., Damiani, F. (eds.) Types for Proofs and Programs. pp. 34–50. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
6. Barbosa, M., Barthe, G., Fan, X., Grégoire, B., Hung, S.H., Katz, J., Strub, P.Y., Wu, X., Zhou, L.: EasyPQC: Verifying Post-Quantum Cryptography. Cryptology ePrint Archive, Paper 2021/1253 (2021)
7. Barbosa, M., Hülsing, A., Meijers, M., Schwabe, P.: Formal Verification of Post-Quantum Cryptography (2022), https://csrc.nist.gov/CSRC/media/Presentations/formal-verifcation-of-post-quantum-cryptography/images-media/session-2-meijers-formal-verification-pqc.pdf
8. Barbosa, M., Hülsing, A., Meijers, M., Schwabe, P.: Formal Verification of Post-Quantum Cryptography (2022), https://csrc.nist.gov/CSRC/media/Events/third-pqc-standardization-conference/documents/accepted-papers/meijers-formal-verification-pqc2021.pdf
9. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS — Kyber: A CCA-Secure Module-Lattice-Based KEM. In: 2018 IEEE European Symposium on Security and Privacy. pp. 353–367 (2018)
10. Eberl, M., Hölzl, J., Nipkow, T.: A Verified Compiler for Probability Density Functions. In: Vitek, J. (ed.) Programming Languages and Systems. pp. 80–104. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)
11. GitHub: EasyCrypt (2022), https://github.com/EasyCrypt/easycrypt
12. Haftmann, F., Wenzel, M.: Constructive Type Classes in Isabelle. In: Altenkirch, T., McBride, C. (eds.) Types for Proofs and Programs. pp. 160–174. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
13. Harrison, J.: A HOL Theory of Euclidean space. In: Hurd, J., Melham, T. (eds.) Theorem Proving in Higher Order Logics, 18th International Conference, TPHOLs 2005. Lecture Notes in Computer Science, vol. 3603. Springer-Verlag, Oxford, UK (2005)
14. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A Modular Analysis of the Fujisaki-Okamoto Transformation. Cryptology ePrint Archive, Paper 2017/604 (2017), https://eprint.iacr.org/2017/604, https://eprint.iacr.org/2017/604
15. Huffman, B., Kunčar, O.: Lifting and Transfer: A Modular Design for Quotients in Isabelle/HOL. In: Certified Programs and Proofs, pp. 131–146. Springer International Publishing (2013)
16. Hülsing, A., Meijers, M., Strub, P.Y.: Formal Verification of Saber's Public-Key Encryption Scheme in EasyCrypt. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology – CRYPTO 2022. pp. 622–653. Springer Nature Switzerland, Cham (2022)
17. Hwang, V., Liu, J., Seiler, G., Shi, X., Tsai, M.H., Wang, B.Y., Yang, B.Y.: Verified NTT Multiplications for NISTPQC KEM Lattice Finalists: Kyber, SABER, and

NTRU. IACR Transactions on Cryptographic Hardware and Embedded Systems **2022**(4), 718–750 (Aug 2022)
18. Hövelmanns, K., Hülsing, A., Majenz, C.: Failing gracefully: Decryption failures and the Fujisaki-Okamoto transform. Cryptology ePrint Archive, Paper 2022/365 (2022), https://eprint.iacr.org/2022/365, https://eprint.iacr.org/2022/365
19. Kaliszyk, C., Urban, C.: Quotients Revisited for Isabelle/HOL. pp. 1639–1644 (05 2011)
20. Klemsa, J.: Fast and Error-Free Negacyclic Integer Convolution Using Extended Fourier Transform, pp. 282–300 (07 2021)
21. Kreuzer, K.: CRYSTALS-Kyber. Archive of Formal Proofs (September 2022), https://isa-afp.org/entries/CRYSTALS-Kyber.html, Formal proof development
22. Kreuzer, K.: Verification of the (1–δ)-Correctness Proof of CRYSTALS-KYBER with Number Theoretic Transform. Cryptology ePrint Archive, Paper 2023/027 (2023), https://eprint.iacr.org/2023/027, https://eprint.iacr.org/2023/027
23. KU Leuven ESAT/COSIC: Saber (2022), https://www.esat.kuleuven.be/cosic/pqcrypto/saber/resources.html
24. Langlois, A., Stehlé, D.: Worst-Case to Average-Case Reductions for Module Lattices. Des. Codes Cryptogr. **75**(3), 565–599 (Jun 2015)
25. Lochbihler, A.: CryptHOL. Archive of Formal Proofs (May 2017), https://isa-afp.org/entries/CryptHOL.html, Formal proof development
26. Lochbihler, A., Sefidgar, S.R.: A tutorial introduction to CryptHOL. Cryptology ePrint Archive, Paper 2018/941 (2018), https://eprint.iacr.org/2018/941, https://eprint.iacr.org/2018/941
27. Longa, P., Naehrig, M.: Speeding up the Number Theoretic Transform for Faster Ideal Lattice-Based Cryptography. pp. 124–139 (10 2016)
28. Moggi, E.: Notions of computation and monads. Information and Computation **93**(1), 55–92 (jul 1991)
29. Nipkow, T., Klein, G.: Concrete Semantics with Isabelle/HOL. Springer (2014), http://concrete-semantics.org
30. Nipkow, T., Paulson, L., Wenzel, M.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)
31. nLab authors: monad. https://ncatlab.org/nlab/show/monad (Nov 2022), Revision 97
32. nLab authors: monads of probability, measures, and valuations. https://ncatlab.org/nlab/show/monads%20of%20probability%2C%20measures%2C%20and%20valuations (Nov 2022), Revision 32
33. Schwabe, P.: NTRU (2022), https://ntru.org/
34. Sprenkels, A.: The Kyber/Dilithium NTT (2022), https://electricdusk.com/ntt.html
35. Unruh, D.: Post-Quantum Verification of Fujisaki-Okamoto. In: Moriai, S., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2020. pp. 321–352. Springer International Publishing, Cham (2020)