Verification of Correctness and Security Properties for CRYSTALS-KYBER

Katharina Kreuzer
School of Computation, Information and Technology
Technical University of Munich
Munich, Germany
0000-0002-4621-734X

***Abstract*—Since the post-quantum crypto system CRYSTALS-KYBER has been chosen for standardization by the National Institute for Standards and Technology (US), a formal verification of its correctness and security properties becomes even more relevant. Using the automated theorem prover Isabelle, we are able to formalize the algorithm specifications and parameter sets of Kyber's public key encryption scheme and verify the δ-correctness and indistinguishability under chosen plaintext attack property. However, during the formalization process, several gaps in the pen-and-paper proofs were discovered. All but one gap concerning the error bound δ could be filled. Calculations in smaller dimensions give examples where the bound δ is less than the actual error term, violating the correctness property. Since the correctness proof could be formalized up to an application of the module-Learning-with-Errors assumption, we believe that the discrepancy of the original error bound and the formalized version is relatively small. Thus the correctness could be formalized up to a minimal change to the error bound.**

***Index Terms*—post-quantum cryptography, CRYSTALS-KYBER, number theoretic transform, security, verification, Isabelle.**

I. Introduction

With large-scale quantum computers all crypto systems based on RSA and Diffie-Hellman can be broken using Shor's algorithm. Since recent developments 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) of the US, a variety of post-quantum crypto systems have been designed [33]. 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 KEM CRYSTALS-KYBER (abbreviated as Kyber throughout this presentation) which was originally developed by Bos *et al.* [11]. In the first submission to the NIST standardization process [6], 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 [5] was excluding the compression and decompression functions in the key generation and encryption functions. The reason is that a problem in the security proof for the indistinguishability under chosen plaintext attack (IND-CPA) was found by D'Anvers [11, footnote 6]. 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 [4] in October 2020, some parameter changes have been made. Most notable is the change of splitting the variances of the centred binomial distribution for the error terms in the encryption. This could not be formalized since the underlying hardness assumption requires the errors to be of the same distribution. However, this only affects the proofs for Kyber512, since in Kyber 768 and Kyber1024 there is no such split. Throughout this paper, we focus on the formalization of the most recent version (namely 3rd round with security levels Kyber768 and Kyber1024) for Kyber's PKE scheme which we refer to as Kyber if not stated otherwise.

The underlying hard problem for Kyber is the module-Learning-with-Errors (module-LWE) problem. It states that it is hard to recompute a small vector when given a matrix and the matrix-vector-product perturbed by additional small errors. Without the error term, this problem can be solved by Gaussian elimination, but with the error it becomes NP-hard under certain conditions [25].

Since Kyber's key generation and encryption are based on masking the output with an error using module-LWE instances, this may result in a positive probability that the errors get too large so that we cannot decrypt correctly. We therefore need to consider δ-correctness, where δ bounds the correctness error. The correctness error is defined as the probability of an incorrect decryption in the worst case over all messages and in the mean over the generated public and secret key pairs.

As cryptography is used in many safety critical areas, security of the schemes and correctness of their mathematical proofs is crucial. The standard to ensure correctness of proofs for many years was to check and recheck proofs manually. However since humans are inevitably prone to errors, flaws in proofs may go unnoticed for years. Formalization in automated theorem provers can help to uncover such flaws, inconsistencies or simple calculus errors. Especially in cryptography, formal analysis and verification can uncover a number of

This work was supported by the Research Training Group GRK 2428 CONVEY of the German Research Council (DFG).

vulnerabilities of crypto systems or protocols. Some examples are vulnerabilities found in the Matrix messenger [1] and the Jitsi video conference tool [29] during a formalization. In recent years, formalizing proofs in cryptography has gained more and more attention. This motivates checking correctness and security properties also for post-quantum crypto systems like Kyber.

A. Our Contribution

With this paper, we introduce a formalization of Kyber's PKE scheme, its correctness and IND-CPA security property in Isabelle. The formalization includes the algorithms for key generation, encryption and decryption of both the original [6], [11] and the latest versions [4], [5]. Using minimal assumptions in the formalization, we allow for instantiations with various parameter sets.

During the formalization of the correctness of Kyber, we encountered two problems: Firstly, we could only verify the δ-correctness for a modified δ'. We give a counter-example for a small parameter set where the originally claimed δ [11] violates the δ-correctness property. Further experiments with different parameter sets result in similar findings. This issue has been acknowledged by Kyber authors in private communication. Secondly, we notice that the function $\|\cdot\|_\infty$ as defined in [11] is not the usual maximum norm, but only a pseudo-norm. This results in a failing proof step which can be resolved by adding an assumption on the modulus q. The additional assumption is fulfilled by all Kyber parameter sets. Overall, the correctness of Kyber could be formally proved with only a small change on the error bound.

Kyber uses the number theoretic transform (NTT) for fast multiplication. The aforementioned additional assumption is implied by assumptions on Kyber for the NTT. The NTT in the case of Kyber, as well as its convolution theorem have also been formalized for this article. However, we will not go into detail in this presentation, but include a short overview in the Appendix for the readers convenience.

The formalization is foundational, i.e. that everything is proven with respect to the higher order logic (HOL) kernel of Isabelle/HOL. The only computational assumption we make is that the underlying hardness assumption of the module-Learning-with-Errors problem (module-LWE) holds.

B. Related Work

A short version on the formalization of the δ-correctness of the original version of Kyber can be found in [24]. Meijers *et al.* [9], [10] announced a formalization of Kyber in EasyCrypt [14]. Furthermore, a post-quantum version of EasyCrypt called EasyPQC is being developed [8]. Recently, Almeida *et al.* [2] introduced a formalization of the implementation code to the specification in the frameworks EasyCrypt and Jasmin. The formalization in EasyCrypt/Jasmin is complementary to this presentation, since it does not verify the mathematical proofs of correctness or security properties of the specifications. To the best of the authors' knowledge, there is, up to now, no publication or publicly accessible formalization of Kybers correctness proof or the IND-CPA security proof. Private conversation with Kyber authors showed that the flaw uncovered by this formalization effort in the correctness proof was known, but a solution was not yet found.

In 2022, the NTT was verified in CryptoLine by Hwang *et al.* in [18]. CryptoLine is a tool for low-level verification of implementations which stands in contrast to our high level verification of the mathematics behind Kyber.

C. Isabelle

All formalizations and verifications were implemented in the theorem prover Isabelle. An introduction to Isabelle can be found in [32] and [31]. In contrast to other cryptographic verification tools, Isabelle is foundational meaning everything is proved from the axioms of higher order logic. The formalizations for this work are performed on the specification level of Kyber and are not restricted to an implementation.

Two main features in Isabelle support abstraction over a context of assumptions: The type class constraints (introduced in [15]) and explicit assumptions summarized in a context called locale (introduced in [7]). These abstractions allow instantiations with several parameter sets, making changes for example on the underlying prime (e.g. [6] to [5]) easy.

For our formalizations, we make extensive use of several libraries for Isabelle, including algebra, analysis, probability theory and CryptHOL [26] (a library for cryptography). Tutorials on the latter can be found in [27].

D. 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 II-A. 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 II-B we look at the instantiation of our formalization for some parameter sets. Next, we describe the formalization of the algorithms for compression, decompression, key generation, encryption and decryption of Kyber in Section III. In Section IV, we proceed with the verification of the δ-correctness proof of Kyber. Here, we recognize two problems in the proof: On the one hand, we can only show δ-correctness for a modified δ' as described in Section IV-C. We analyse why the original proof could not be formalized and how a modification on δ can fix this issue. Indeed, we showcase small dimensional examples where the proof fails for the original δ. 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 IV-F. This newly found assumption is already fulfilled when working in the NTT domain. More on the formalization of the NTT on polynomials and its convolution theorem can be found in the Appendix X-D. In Section V, 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 [26], we point out important concepts for formalizing cryptographic security proofs in Isabelle in Section VI. The formalization of the game-based security proof of Kyber against IND-CPA follows in Section VII. In the end, we give a short outlook on further research questions. The full formalization can be found in [21] and [22].

Throughout this paper, we will use bold font to highlight vectors and matrices (e.g. $\mathbf{v}$, $\mathbf{A}$) and roman font for polynomials (e.g. x).

II. Formalizing the Context of Kyber

Starting a formalization of the specification of Kyber requires a framework to state and calculate with Kyber's polynomial quotient ring. Isabelle offers possibilities to implement the framework and parameter set in a flexible way using type classes and locales.

A. Formalizing the Polynomial Quotient Ring

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 $\mathbb{Z}_q[x]/(x^n + 1)$. This is the space where the Kyber algorithms work in. Note that $x^n + 1$ is the $2^{n'+1}$-th cyclotomic polynomial which is irreducible over the integers $\mathbb{Z}$, but reducible over the finite field $\mathbb{Z}_q$.

There are various concepts behind this construct which are not easy to formalize 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 $\mathrm{p} = \sum_{i=0}^{\infty} f(i)x^i$. Since p has only finitely many non-zero coefficients, f is 0 almost everywhere. For example the polynomial $\mathrm{p} = x^2 + 2$ is represented as the function f with:

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

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 concrete Isabelle formalization is explained in Appendix X-A1. The resulting structure inherits basic properties like the zero element, addition, subtraction and multiplication from the original polynomial ring through lifting and transfer [17].

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 [16]. 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 abstracted away to enable a shorter presentation.

B. Formalizing the Parameters of Kyber

Kyber depends on a number of parameters defining the module, the compression and decompression. These are:

- $n = 2^{n'}$, the degree of the cyclotomic polynomial
- q, the prime number and modulus
- k, the dimension of vectors in the module
- d_u and d_v, the number of digits for compression and decompression of u and v, respectively

Since the framework for the context 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 are positive integers
- $n = 2^{n'}$ is a power of 2
- $q > 2$ is a prime with $q \bmod 4 = 1$ (the latter is an additional assumption and will be discussed in Section IV-F)

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 [11] is 7681, but since round two of the NIST standardization process [4], [5], 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 second (and third) round specification of Kyber [4], [5] can be found in Table I.

The Isabelle formalization of the parameter set can be found in Appendix X-A2. In our formalization, we instantiate the locale containing the Kyber algorithm and proof of δ-correctness with the parameter set given in Table I for Kyber768.

III. Formalizing the Kyber Algorithm

The PKE scheme Kyber consists of three algorithms: the key generation, the encryption and the decryption. The key generation produces a public and secret key pair given a random input. The keys are then applied in the en- and decryption. In order to discard some lower order bits to make the ciphertext smaller, a compression and decompression function is added. The compression function is also used to extract the message in the decryption. In the first versions of Kyber, the compression of the public key invalidates the IND-CPA security proof. Therefore, since the submission to round

Table I: Parameter set of round two and three Kyber [4], [5]

	n	n'	q	k	d_u	d_v
Kyber512 (round 2)	256	8	3329	2	10	3
Kyber768 (round 2 & 3)	256	8	3329	3	10	4
Kyber1024 (round 2 & 3)	256	8	3329	4	11	5

two of the NIST standardization process, this compression of the public key was left out. We focus on the newer versions in this presentation.

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.

A. Input to the Algorithm

The key generation requires the inputs $\mathbf{A} \in R_q^{k \times k}$, $\mathbf{s} \in R_q^k$ and $\mathbf{e} \in R_q^k$ which are chosen randomly. $\mathbf{A}$ is chosen uniformly at random from the finite set $R_q^{k \times k}$. In the implementation, the matrix $\mathbf{A}$ is generated from a uniformly random seed via an XOF. This expansion has not been formalized. Instead, we require that $\mathbf{A}$ itself is uniformly random. $\mathbf{A}$ is also part of the public key. For elements of the secret key $\mathbf{s}$ and the error term $\mathbf{e}$, we define the centred 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 $x = \sum_{i=1}^{\eta} c_i$. 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 $\mathbf{s}$ and $\mathbf{e}$ are generated according to β_η^k.

For our formalization of Kyber, we use $\eta = 2$ [4], [5]. Note that in the more recent submissions of Kyber, the value η determining the variance of the centred binomial distribution was changed as well. Again, the formalization in locales allows us to easily change these values of η. However, for Kyber 512 in the third submission round [4], two separate values η_1 and η_2 have been introduced. This distinction has not been formalized. The reason is that the following definition of the module-LWE problem only allows the distribution on the elements of the error vector to be the same. Since the security proofs reduces a module-LWE instance where e_1 and e_2 appear in one vector, the formalization does not allow the splitting of η_1 and η_2.

The sampled values $\mathbf{A}$, $\mathbf{s}$ and $\mathbf{e}$ constitute an instance of the module-LWE problem which is defined in the following.

Definition 1 (Module-LWE)**.** Given a uniformly random $\mathbf{A} \in R_q^{k \times k}$ and $\mathbf{s}, \mathbf{e} \in R_q^k$ chosen randomly according to the distribution β_η^k. Let $\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e}$, then the (decision) module-LWE problem asks to distinguish $(\mathbf{A}, \mathbf{t})$ from uniformly random $(\mathbf{A}', \mathbf{t}') \in R_q^{k \times k} \times R_q^k$.

There is a probabilistic reduction proof for the average-case NP-hardness of the module-LWE by Langlois and Stehlé [25]. Therefore, the key generation of Kyber returns a public key and secret key pair where it is (in average) NP-hard to recover the secret key from the public key alone. This 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's PKE scheme. The NP-hardness proof of the module-LWE has not been formalized.

B. Compression and Decompression

The compression and decompression functions in Kyber help to reduce ciphertext size 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.

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

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

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

We call the value $decomp_d\ (comp_d\ x) - x$ 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 $comp_2\ 2 = \lceil 1.6 \rfloor \mod 4 = 2$ and $decomp_2\ 2 = \lceil 2.5 \rfloor = 3$. Here, the compression error is $decomp_2\ (comp_2\ 2) - 2 = 3 - 2 = 1$. 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 $comp_2\ 10 = \lceil 3.\overline{63} \rfloor \mod 4 = 0$ and $decomp_2\ 0 = 0$. Here, the compression error for integers is $decomp_2\ (comp_2\ 10) - 10 = -10$. 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' to avoid overly lengthy expressions.

C. Key Generation, Encryption and Decryption

We now want to state the actual algorithms. For the convenience of the reader, we append the formal definitions of key generation, encryption and decryption in Isabelle in the Appendix X-A3. The calculation of the key generation is defined in the following way:

$$key\_gen\ \mathbf{A}\ \mathbf{s}\ \mathbf{e} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$$

We denote by $\mathbf{t} = key\_gen\ \mathbf{A}\ \mathbf{s}\ \mathbf{e}$ the output of the key generation. Together, the matrix $\mathbf{A}$ and the vector $\mathbf{t}$ constitute the public key, whereas the vector $\mathbf{s}$ is the secret key. When we say that the public and secret key pair $(\mathbf{A}, \mathbf{t})$ and $\mathbf{s}$ are generated by the key generation algorithm, we mean the probabilistic program where $\mathbf{A}$, $\mathbf{s}$ and $\mathbf{e}$ are chosen according to their distributions, $\mathbf{t}$ is calculated by key_gen and $(\mathbf{A}, \mathbf{t})$ and $\mathbf{s}$ are the output.

Note that in the original version of Kyber [11], the key generation included a compression of $\mathbf{t}$. However, this resulted in a major flaw of the IND-CPA proof. Thus, since the second round of NIST's standardization [5], the compression in the key generation was omitted.

The pair $(\mathbf{A}, \mathbf{t})$ also forms an instance of the module-LWE problem. The module-LWE hardness assumption states that in average cases it is hard to recuperate the secret key $\mathbf{s}$ from the pair $(\mathbf{A}, \mathbf{t})$.

To encrypt a bit-string $\bar{m}$ with at most n bits, we consider the message polynomial $\mathrm{m} \in R_q$ obtained by $\mathrm{m} = \sum_{i=0}^{n-1} \bar{m}(i)x^i$. Thus, the message polynomial m only has coefficients in $\{0, 1\}$. For the encryption, we also need to generate another secret $\mathbf{r} \in R_q^k$ together with errors $\mathbf{e_1} \in R_q^k$ and $\mathrm{e}_2 \in R_q$ according to the distribution β_η^k and β_η. We then calculate the encryption:

$$\begin{aligned} &encrypt\ \mathbf{t}\ \mathbf{A}\ \mathbf{r}\ \mathbf{e_1}\ \mathrm{e}_2\ du\ dv\ \mathrm{m} = \\ &\quad (\mathbf{comp}_{du}\ (\mathbf{A}^T \cdot \mathbf{r} + \mathbf{e_1}), \\ &\quad \mathrm{comp}_{dv}\ (\mathbf{t}^T\mathbf{r} + \mathrm{e}_2 + \\ &\quad + to\_module(\lceil q/2 \rfloor) \cdot \mathrm{m})) \end{aligned}$$

Let $\mathbf{u} = \mathbf{A}^T \cdot \mathbf{r} + \mathbf{e_1}$ and $\mathrm{v} = \mathbf{t}^T\mathbf{r} + \mathrm{e}_2 + to\_module(\lceil q/2 \rfloor) \cdot \mathrm{m}$. Then, the encryption outputs the compressed values $\mathbf{u}^*$ and v^* in a pair $(\mathbf{u}^*, \mathrm{v}^*)$. When referring to the encryption without the input of $\mathbf{r}$, $\mathbf{e_1}$ and e_2, we mean the probabilistic program that first generates $\mathbf{r}$, $\mathbf{e_1}$ and e_2 according to their distributions and then calculates the encryption function as stated above.

Using the secret key $\mathbf{s}$, we can recover the message m from $\mathbf{u}^*$ and v^* in the decryption function. We extract the message as the highest bit in $\mathrm{v}' - \mathbf{s}^T\mathbf{u}'$ using the compression function with depth 1.

$$\begin{aligned} &decrypt\ \mathbf{u}^*\ \mathrm{v}^*\ \mathbf{s}\ du\ dv = \\ &\quad \mathrm{comp}_1\ ((\mathrm{decomp}_{dv}\ \mathrm{v}^*) - \\ &\quad \mathbf{s}^T(\mathbf{decomp}_{du}\ \mathbf{u}^*)) \end{aligned}$$

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.

IV. 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 [11, 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. Formalizations can be found in [21] and [22].

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

In order to estimate values, the authors of Kyber [11] use a function $\|\cdot\|_\infty$. However, it is defined slightly differently from what one would expect: Instead of using a regular modulo operation, the re-centred operation $\mod^\pm$ is defined as 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 Kyber, we get that $a \mod^\pm q \in \{\frac{-q+1}{2}, \ldots, \frac{q-1}{2}\}$. Using this re-centred modulo operation, we define the function $\|\cdot\|_\infty$ on polynomials as:

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

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

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

Unfortunately with the re-centring one loses the absolute homogeneity, i.e., for a scalar $\mathbf{s}$ and vector $\mathbf{v}$ only $\|s \cdot \mathbf{v}\|_\infty \leq |s| \cdot \|\mathbf{v}\|_\infty$ holds with an inequality instead of equality. For example consider the case $q = 3$, $s = 2$ and $\mathbf{v} = (2)$. We then have the strict inequality:

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

Therefore, the $\|\cdot\|_\infty$ function is not a norm, but a pseudo-norm. It is positive definite and fulfils the triangle inequality. This is not explicitly mentioned in [11] and indeed poses a problem in the proof of the following correctness theorem.

B. Correctness of the Kyber Algorithms

A crypto system is correct, if it always returns the original message. However, since Kyber uses errors to mask the ciphertext, there is a chance that the error may be too large to decipher correctly. 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:

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

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

The intuition is that the probability of a decryption failure in the worst-case scenario over the message space and in a mean over the secret and public key pair should be bounded by a constant δ.

The δ-correctness of the PKE is a necessary requirement for the Fujisaki-Okamoto transform (verified by Unruh [37] in qrhl-tool). When connected with Unruh's formalization, the formalization presented in this paper results in a formal verification of Kyber's KEM with a verified indistinguishability under chosen ciphertext attack security property. A connection to Unruh's formalization was out of scope.

For the Kyber algorithms [11, Theorem 1], the δ-correctness theorem is proved in two steps:

1) An assumption sufficient for the correct decryption can be calculated deterministically. This is the main argument of the proof. The assumption is incorporated in the definition of δ.
2) The distributions in the compression errors are claimed to be uniformly random due to a reduction using the module-LWE problem.

The first, deterministic part is stated in the following theorem. Its formalization can be found in the Appendix X-A4.

Theorem IV.1. *Let* $\mathbf{A} \in R_q^{k \times k}$, $\mathbf{s}, \mathbf{r}, \mathbf{e}, \mathbf{e_1} \in R_q^k$, $\mathrm{e}_2 \in R_q$ *and let the message* $\mathrm{m} \in R_q$ *with coefficients in* $\{0, 1\}$. *Define:*

- $\mathbf{t} = key\_gen\ \mathbf{A}\ \mathbf{s}\ \mathbf{e}$*, the output of the key generation*
- $(\mathbf{u}^*, \mathrm{v}^*) = encrypt\ \mathbf{t}\ \mathbf{A}\ \mathbf{r}\ \mathbf{e1}\ \mathrm{e}2\ du\ dv\ \mathrm{m}$*, the output of the encryption*
- $\mathbf{c_u}$ *and* $\mathrm{c_v}$*, the compression errors of* $\mathbf{u}$ *and* v*, respectively*

If $\|\mathbf{e}^T\mathbf{r} + \mathrm{e}_2 + \mathrm{c_v} - \mathbf{s}^T\mathbf{e_1} - \mathbf{s}^T\mathbf{c_u}\|_\infty < \lceil q/4 \rfloor$*, then the decryption algorithm returns the original message* m*:*

$$decrypt\ \mathbf{u}^*\ \mathrm{v}^*\ \mathbf{s}\ du\ dv = \mathrm{m}$$

We have that Kyber is correct when assuming the inequality:

$$\|\mathbf{e}^T\mathbf{r} + \mathrm{e}_2 + \mathrm{c_v} - \mathbf{s}^T\mathbf{e_1} - \mathbf{s}^T\mathbf{c_u}\|_\infty < \lceil q/4 \rfloor \tag{1}$$

C. Modifying the Error Bound

Using Theorem IV.1 and the definition of δ-correctness, we deduce the following.

Corollary. *Let:*

$$\delta' = \mathbb{E}\left[\max_{\mathrm{m} \in \mathcal{M}} \mathbb{P}\left[\begin{cases} \mathbf{e}, \mathbf{r}, \mathbf{e_1} \leftarrow \beta_\eta^k; \quad \mathrm{e}_2 \leftarrow \beta_\eta; \\ \mathbf{u} = \mathbf{A}^T\mathbf{r} + \mathbf{e_1}; \\ \mathrm{v} = \mathbf{t}^T\mathbf{r} + \mathrm{e}_2 + \lceil \frac{q}{2} \rfloor \mathrm{m}; \\ \|\mathbf{e}^T\mathbf{r} + \mathrm{e}_2 + \mathrm{c_v} - \mathbf{s}^T\mathbf{e_1} - \\ \quad -\mathbf{s}^T\mathbf{c_u}\|_\infty \geq \lceil q/4 \rfloor \end{cases}\right]\right] \tag{2}$$

where the expectation is taken over $((\mathbf{A}, \mathbf{t}), \mathbf{s})$ *generated by* key_gen*. Then Kyber is* δ'*-correct.*

Note that in this proposition, the δ' is not the same as in [11, Theorem 1]. Using the second proving step, [11] claims δ-correctness for:

$$\delta = \mathbb{P}\left[\begin{cases} \mathbf{s}, \mathbf{e}, \mathbf{r}, \mathbf{e_1} \leftarrow \beta_\eta^k; \quad \mathrm{e}_2 \leftarrow \beta_\eta; \\ \mathbf{c_u} \leftarrow \Psi_{du}^k, \mathrm{c_v} \leftarrow \Psi_{dv} \\ \|\mathbf{e}^T\mathbf{r} + \mathrm{e}_2 + \mathrm{c_v} - \mathbf{s}^T\mathbf{e_1} + \\ \quad -\mathbf{s}^T\mathbf{c_u}\|_\infty \geq \lceil q/4 \rfloor \end{cases}\right] \tag{3}$$

Here, Ψ_d is the distribution of the compression error of x to d bits for a uniformly generated $x \leftarrow R_q$.

The main difference between δ' and δ is that in δ' the values of $\mathbf{c_u}$ and $\mathrm{c_v}$ are calculated as the correct compression errors, whereas in δ they are the compression errors of uniformly random values $\mathbf{u}$ and v. The intuitive idea given in [11, Proof of Theorem 1] is that this change is negligible since its value can be bounded by the advantage against module-LWE problems. A detailed, formal proof is missing at this point.

Despite this idea making sense intuitively, we were unable to formalize this reduction. Indeed, we claim that this reduction is incorrect in our general framework. The reason is that the change from a module-LWE instance to a uniformly random instance loses all information about the secret key. However, in the definition of δ-correctness, we cannot omit the information about the secret key during the encryption 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.

To substantiate the claim that this reduction to δ does not respect the inequality of Definition 2, we perform a comparative analysis of δ' (eq. (2)), δ (eq. (3)) and the actual correctness error:

$$\mathbb{E}[\max_{m \in \mathcal{M}} \mathbb{P}[decrypt(sk, encrypt(pk, m)) \neq m]] \tag{4}$$

In the following, the value δ is calculated using a Python script by Léo Ducas [12] that was also used for the evaluation in [11].

We showcase two comparisons: First, we calculate the exact values of δ, δ' and the correctness error for very small parameters. We set $n = 2$, $q = 17$, $k = \eta = d_v = 1$ and $d_u = 3$. The expectation, maximum and probabilities can be

computed by considering all possible values. Using a simple Python script [23], the outcomes are the following:

$$\delta = 0.211$$
$$corr\_error = 0.223$$
$$\delta' = 0.267$$

The experiment shows, that for these small parameters, the inequality between the correctness error and δ is violated. This is a counterexample to the claimed proof in [11] since it should hold for any parameters sufficing Kyber's assumptions.

Second, we substantiate our claim also for (slightly) bigger parameter sets. In this experiment, we approximate δ' and the correctness error using Monte-Carlo sampling. Python scripts for the calculation of δ' and the correctness error can be found in [23]. For all the parameter sets that we tested, the inequality

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

is violated. Results are shown in Figure 1 (and Appendix X-B). As parameters, we consider n between 5 and 16 and choose

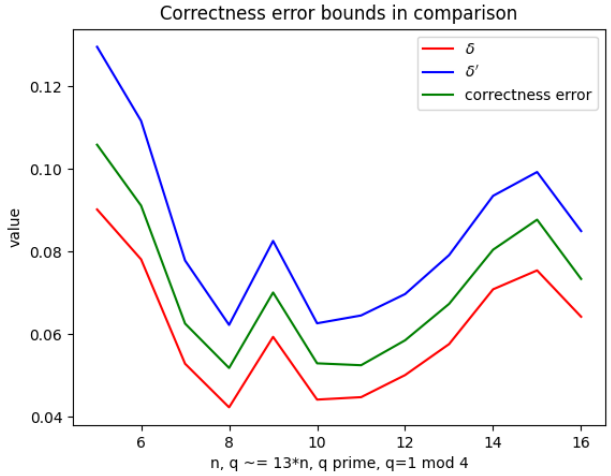

Figure 1: Comparison of absolute values of δ, δ' and the correctness error for small dimensional examples over variation on n

q to be a prime with approximatively the same ratio to n as the original parameters for Kyber ($q/n = 3329/256 \approx 13$) and $q \equiv 1 \mod 4$. Furthermore, we set $k = \eta = d_v = 2$ and $d_u = 5$. In Figure 1, we see that the correctness error (green) consistently lies below our proposed δ' (blue), but violates the relation to the calculated δ (red) from [11].

Private correspondence with an author of Kyber confirmed that this problem with δ was known. Indeed, they explained that the module-LWE reduction was more of a heuristic nature. With this heuristic module-LWE reduction, the error terms can be easily approximated using union bounds, as in [12]. However, since this proof was not formalized in detail, the dependency relations with the secret key may have been overlooked. The above calculations give a counter-example disproving this reduction via module-LWE for our general setting. It may be the case that additional assumptions for the Kyber parameters make the module-LWE reduction valid. An interesting future research question is to find suitable hypotheses to allow the module-LWE reduction or find a counter-example with the actual Kyber parameters.

Fortunately, our findings do not invalidate the correctness of the scheme itself since we could prove correctness with the bound δ'. Still, this issue may affect the level of security of Kyber. This leads us to two more important research questions:

1) Can we estimate/approximate δ' or the relation between δ and δ'?
2) Can we find another more easily calculable bound on the correctness error?

For this paper, our main focus was a foundational formalization of Kyber: indeed, we succeeded to show δ'-correctness.

D. Auxiliary Lemma

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

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

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

The proof of the auxiliary lemma can be found in Appendix X-C.

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.

E. Proof of Correctness

The formalization of the proof of Theorem IV.1 can be found in [22] (and the version with compression of the public key in [21]). One problem encountered during the formalization was that $\|\cdot\|_\infty$ is only a pseudo-norm (recall Section IV-A). This is not explicitly mentioned in [11] 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 IV.1 proceeds as follows. Given $\mathbf{A}$, $\mathbf{s}$, $\mathbf{r}$, $\mathbf{e}$, $\mathbf{e_1}$, e_2 and the message m, we calculate $\mathbf{t}$, $\mathbf{u}^*$ and v^* using the key generation and encryption algorithm. We define $\mathbf{u}'$ and v' to be the decompressed values of $\mathbf{u}^*$ and

v^*, respectively. With the compression errors $\mathbf{c_u}$ and $\mathrm{c_v}$, we get the equations:

$$\mathbf{u}' = \mathbf{A}^T\mathbf{r} + \mathbf{e_1} + \mathbf{c_u}$$
$$\mathrm{v}' = \mathbf{t}'^T\mathbf{r} + \mathrm{e}_2 + \lceil q/2 \rfloor \cdot \mathrm{m} + \mathrm{c_v}$$

This leads to the calculation in the decryption:

$$\mathrm{v}' - \mathbf{s}^T\mathbf{u}' = \mathbf{e}^T\mathbf{r} + \mathrm{e}_2 + \mathrm{c_v} - \mathbf{s}^T\mathbf{e_1} - \mathbf{s}^T\mathbf{c_u} + \lceil q/2 \rfloor \cdot \mathrm{m}$$

We accumulate all error terms in a new variable w:

$$\mathrm{w} := \mathbf{e}^T\mathbf{r} + \mathrm{e}_2 + \mathrm{c_v} - \mathbf{s}^T\mathbf{e_1} - \mathbf{s}^T\mathbf{c_u}$$

and get $\|\mathrm{w}\|_\infty < \lceil q/4 \rfloor$ from the assumptions of Theorem IV.1.

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

$$\mathrm{decomp}_1\ \mathrm{m}' = \lceil q/2 \cdot \mathrm{m}' \rfloor = \lceil q/2 \rfloor \cdot \mathrm{m}'$$

Using Lemma IV.2, it follows that:

$$\begin{aligned}&\|\mathrm{w} + \lceil q/2 \rfloor(\mathrm{m} - \mathrm{m}')\|_\infty \\ =&\|\mathrm{v}' - \mathbf{s}^T\mathbf{u}' - \mathit{decomp}_1\ (\mathit{comp}_1\ (\mathrm{v}' - \mathbf{s}^T\mathbf{u}'))\|_\infty \\ \leq&\lceil q/4 \rfloor\end{aligned}$$

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

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

It remains to show that we can indeed deduce $\mathrm{m} = \mathrm{m}'$ which concludes the proof of Theorems IV.1. According to the last step of [11, Proof Thm 1], this follows directly for any odd q. However, therein lies a hidden problem. [11, Proof Thm 1] makes use of the homogeneity of $\|\cdot\|_\infty$. Since $\|\cdot\|_\infty$ is only a pseudo-norm 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 $q \equiv 3 \mod 4$.

F. Additional Assumption $q \equiv 1 \mod 4$

The following remains to be shown for the proof of Theorem IV.1: Given the inequality

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

we need to deduce that indeed $\mathrm{m} = \mathrm{m}'$.

We prove this statement by contradiction. Assume that m is not equal m', i.e., there exists a coefficient of $\mathrm{m} - \mathrm{m}'$ 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

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

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:

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

since the $\mathrm{mod}^\pm$ 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

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

which is a contradiction to our assumption. In this case, the proof of Theorems IV.1 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

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

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

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

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

but $\mathrm{m} \neq \mathrm{m}'$. This is a counterexample for the correctness of the proof of Theorem IV.1 in the case $q \equiv 3 \mod 4$.

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

In the specification of Kyber, concrete values for the parameters of the system are given (see Section II-B). For example in the recent version of Kyber [4], [5], the modulus q is chosen to be 3329, whereas in early versions [6], [11], the modulus was chosen as 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 2^2, we can automatically infer the property $q \equiv 1 \mod 4$.

The NTT is analysed in more detail in Appendix X-D. A formalization for the NTT in the case of Kyber was included in this project. There is also a formalization of the NTT for the third round Kyber by Hwang et al. [18] in the low-level tool CryptoLine.

V. Game-Based Cryptography

An important cryptographic property of public key encryption schemes is IND-CPA security. This attack describes a game where an adversary tries to gain information about self-chosen plaintexts.

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

Definition 3 (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 2 depicts the IND-CPA game.

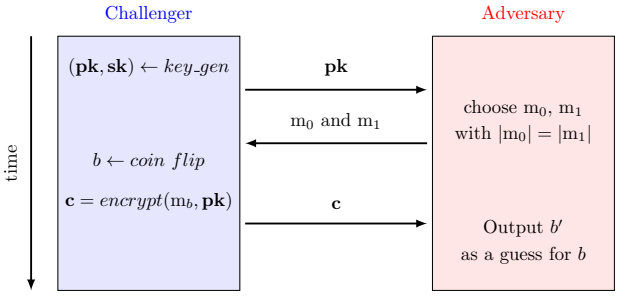

Figure 2: A diagram of the IND-CPA game.

The formalization of the IND-CPA game was taken from the CryptHOL Tutorial [27]. 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 4 (module-LWE game)**.** Two parties, called the challenger and the (module-LWE) adversary, play the following game.

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

Figure 3 depicts the module-LWE problem in game form.

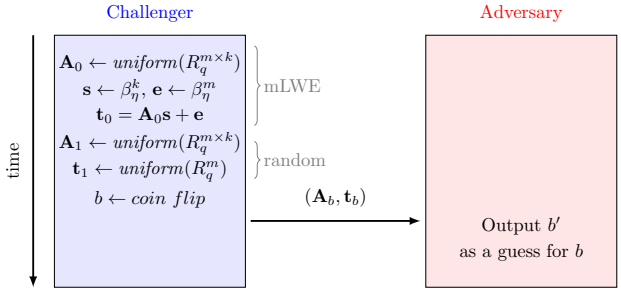

Figure 3: A diagram of the module-LWE game.

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 $\mathbf{A}$ is a quadratic matrix. However, in the encryption, the matrix $\mathbf{A}$ is extended by the vector $\mathbf{t}$, 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.

VI. Using CryptHOL in Isabelle

CryptHOL [26] 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 sub-probability 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.

A. Sub-probability Mass Functions

The *spmf* type class is a superclass of probability mass functions. We consider a finite set S. A probability 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 sub-probability mass functions, we allow the weight to be less than one:

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

A sub-probability 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 and may not terminate. For example in the IND-CPA game, the adversary can intentionally input two messages of different length and thus gain information about the ciphertext or simply not answer at all.

B. 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 sub-probability 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 handle this generalization step most of the time.

C. 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 [34] and the introduction of monads into functional programming in [30]. A good introduction to the Giry-monad in the context of Isabelle is given in [13].

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

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

Monads need to fulfil three laws: the left and right identities and associativity. Let us look at a short example.

Example VI.1. The $option$ type class is a monad. As described at the end of Section V, a type $'a\ option$ takes the values $Some\ a$ or $None$. In this case, the $option$ monad is defined over the type $'a$. The *return* function takes an element a of type $'a$ and returns an element $Some\ a$ of type $'a\ option$. The *bind* function on a function f is defined by:

$$bind\ None\ f = None$$
$$bind\ (Some\ a)\ f = f(a)$$

Another notation for the *bind* function is:

$$bind\ a\ f \equiv a \gg\!= f$$

Another example is the Giry-monad. It assigns to each measurable space the space of probability measures over it (see [35]).

Example VI.2. The type class of probability mass functions *pmf* for discrete distributions is a monad, called the Giry-monad. The *return* function for an element $\mathbf{A}$ 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.

VII. IND-CPA Security Proof for Kyber

Since round two of the NIST standardization process [5], the compression of the public key in Kyber has been omitted. The reason was that otherwise the IND-CPA security proof [11, 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 from the module-LWE. The security of Kyber without compression under IND-CPA is stated in the following theorem. Its formalization can be found in [22]

Theorem VII.1. *Given any adversary $\mathcal{A}$ to the IND-CPA game of Kyber and assuming that $\mathcal{A}$ is lossless, the advantage of $\mathcal{A}$ in the IND-CPA game can be bounded by twice the advantage in the module-LWE game.*

Loosely speaking: the public key encryption scheme Kyber without compression of the public key is IND-CPA secure against the module-LWE hardness assumption. The formalization in Isabelle can be found in Appendix X-A5.

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:

$$Adv^{Kyber}(\mathcal{A}) \leq Adv_k^{mLWE}(f_1(\mathcal{A})) + Adv_{k+1}^{mLWE}(f_2(\mathcal{A})) \tag{5}$$

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 analysing 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 (for f_1) one to the Kyber encryption algorithm. Otherwise, the functions are non-recursive, polynomial time probabilistic algorithms.

The proof of equation (5) proceeds in three steps (also called game-hops).

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

In every game-hop, we define an intermediate game and analyse 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 $(\mathbf{A}, \mathbf{t})$ 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 4 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.

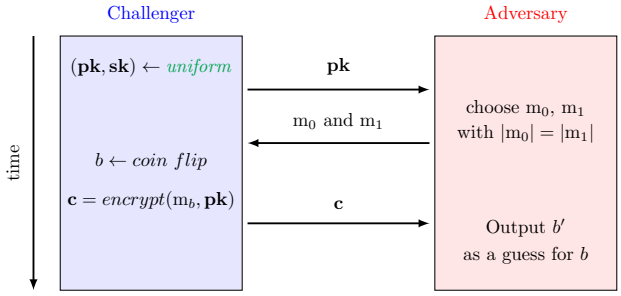

Figure 4: A diagram of $game_1$.

The key generation algorithm creates a module-LWE instance. Distinguishing a module-LWE instance from a uniformly random instance is exactly the module-LWE game. Hence, for a suitable reduction function f_1 have:

$$|\mathbb{P}[game_0 = true] - \mathbb{P}[game_1 = true]| = Adv_k^{mLWE}(f_1(\mathcal{A}))$$

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

1) The challenger generates a public key $(\mathbf{A}, \mathbf{t})$ 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 5 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.

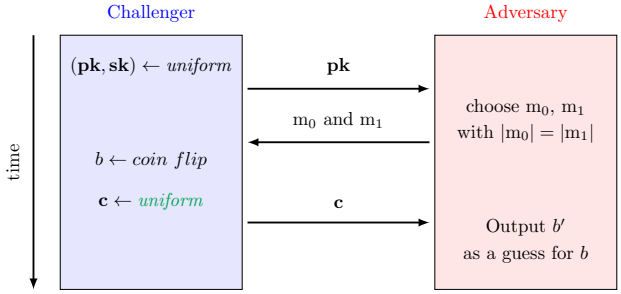

Figure 5: A diagram of $game_2$.

In the encryption, 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 probability of distinguishing an instance of the form

$$\begin{pmatrix} \mathbf{A} \\ \mathbf{t} \end{pmatrix} \mathbf{r} + \begin{pmatrix} \mathbf{e_1} \\ \mathrm{e}_2 \end{pmatrix}$$

and a uniformly random instance $(\mathbf{u}\ \mathrm{v}')^T$ 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 $\mathbf{r}$ is chosen to be the same for the multiplication with both $\mathbf{A}$ and $\mathbf{t}$. This is also the reason, why we cannot split the variance for the centred binomial distribution into η_1 and η_2, since e_1 and e_2 together form the error vector of the module-LWE instance, thus needing the same distribution.

Second, we need to show that $\mathrm{v}' + \lceil q/2 \rfloor \cdot \mathrm{m}$ is also distributed uniformly. That is, we cannot distinguish between the probabilities of the value $\mathrm{v}' + \lceil q/2 \rfloor \cdot \mathrm{m}$ 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.

For a suitable reduction function f_2, we deduce:

$$\left|\mathbb{P}[game_1 = true] - \mathbb{P}[game_2 = true]\right| = Adv_{k+1}^{mLWE}(f_2(\mathcal{A}))$$

In the last step, 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 inferred 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.

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

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.

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

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:

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

□

During the formalization process, it became clear that this proof does not work for the first version of Kyber as remarked by the authors of Kyber [11, Sec. Security of the real scheme]. The proof for the current scheme 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 sub-probability mass functions and generative probabilistic values and supplies a huge library of fundamental lemmas. Since the example game-based proof of the CryptHOL Tutorial [27] 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 initializing the commutativity laws solves these issues.

VIII. Implementation Details

The implementation in Isabelle comprises about $6.7k$ lines of code. The proportions on the topics is depicted in Figure 6. Since many concepts from algebra, analysis, probability theory and cryptographic primitives could be reused, the authors could focus solely on the formalization of Kyber. Furthermore, the automation greatly helped shortening the proofs.

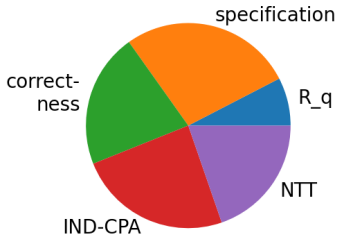

Figure 6: Distribution of lines of code on different topics

Due to the various dependencies and invocations of the library, loading the formalization theories might take some time, especially when the required theories on analysis and probability need to be built for the first time.

IX. Conclusion

In this presentation, we described the formalization of key-generation, encryption and decryption algorithms of CRYSTAL-KYBER's public key encryption scheme.

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 could be verified. Differences between the original proof

and the formalization were discussed and counterexamples for failing proof-steps were given. The additional assumption $q \equiv 1 \mod 4$ is already fulfilled by necessary properties for the number theoretic transform. Therefore, the correctness of Kyber itself is not compromised but minimal changes to the error bound δ are needed. However, the authors of Kyber acknowledged the need for an alternative bound in private communication. Moreover, a verification of the IND-CPA security proof for Kyber was presented.

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, our proposed δ' cannot be approximated as easily as the original δ. Finding a calculable bound or an approximation on δ' remains an important question. Another very interesting aspect is to formalize the hardness results of the module-LWE that Kyber is building on.

Acknowledgment

Many thanks to Tobias Nipkow and Manuel Eberl for their continuous support. Thanks also to Manuel Barbosa, Peter Schwabe and Andreas Hülsing for their insightful 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] M. R. Albrecht, S. Celi, B. Dowling, and D. Jones. Practically-exploitable Cryptographic Vulnerabilities in Matrix. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, May 2023.

[2] J. B. Almeida, M. Barbosa, G. Barthe, B. Grégoire, V. Laporte, J.-C. Léchenet, T. Oliveira, H. Pacheco, M. Quaresma, P. Schwabe, A. Séré, and P.-Y. Strub. Formally verifying Kyber Episode IV: Implementation Correctness. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, page 164–193, June 2023.

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

[4] R. M. Avanzi, J. W. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé. CRYSTALS-Kyber Algorithm Specifications And Supporting Documentation (version 3.0). 01/10/2020.

[5] R. M. Avanzi, J. W. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé. CRYSTALS-Kyber Algorithm Specifications And Supporting Documentation (version 2.0). 30/03/2019.

[6] R. M. Avanzi, J. W. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé. CRYSTALS-Kyber Algorithm Specifications And Supporting Documentation. 30/11/2017.

[7] C. Ballarin. Locales and Locale Expressions in Isabelle/Isar. In S. Berardi, M. Coppo, and F. Damiani, editors, *Types for Proofs and Programs*, pages 34–50, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[8] M. Barbosa, G. Barthe, X. Fan, B. Grégoire, S.-H. Hung, J. Katz, P.-Y. Strub, X. Wu, and L. Zhou. EasyPQC: Verifying Post-Quantum Cryptography. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 2564–2586, New York, NY, USA, 2021. Association for Computing Machinery.

[9] M. Barbosa, A. Hülsing, M. Meijers, and P. Schwabe. Formal Verification of Post-Quantum Cryptography. https://csrc.nist.gov/CSRC/media/Presentations/formal-verifcation-of-post-quantum-cryptography/images-media/session-2-meijers-formal-verification-pqc.pdf, accessed: 2022-09-06.

[10] M. Barbosa, A. Hülsing, M. Meijers, and P. Schwabe. Formal Verification of Post-Quantum Cryptography. https://csrc.nist.gov/CSRC/media/Events/third-pqc-standardization-conference/documents/accepted-papers/meijers-formal-verification-pqc2021.pdf, accessed: 2022-09-06.

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

[12] L. Ducas and J. Schanck. pq-crystals/security-estimates, 2021. https://github.com/pq-crystals/security-estimates/tree/master, accessed: 2023-08-09.

[13] M. Eberl, J. Hölzl, and T. Nipkow. A Verified Compiler for Probability Density Functions. In J. Vitek, editor, *Programming Languages and Systems*, pages 80–104, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[14] GitHub. EasyCrypt, 2022. https://github.com/EasyCrypt/easycrypt, accessed: 2022-07-26.

[15] F. Haftmann and M. Wenzel. Constructive Type Classes in Isabelle. In T. Altenkirch and C. McBride, editors, *Types for Proofs and Programs*, pages 160–174, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[16] J. Harrison. A HOL Theory of Euclidean space. In J. Hurd and T. Melham, editors, *Theorem Proving in Higher Order Logics, 18th International Conference, TPHOLs 2005*, volume 3603 of *LNCS*, pages 114–129, Oxford, UK, 2005. Springer-Verlag.

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

[18] V. Hwang, J. Liu, G. Seiler, X. Shi, M.-H. Tsai, B.-Y. Wang, and B.-Y. Yang. Verified NTT Multiplications for NISTPQC KEM Lattice Finalists: Kyber, SABER, and NTRU. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):718–750, August 2022.

[19] C. Kaliszyk and C. Urban. Quotients Revisited for Isabelle/HOL. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 1639–1644, New York, NY, USA, 2011. Association for Computing Machinery.

[20] J. Klemsa. *Fast and Error-Free Negacyclic Integer Convolution Using Extended Fourier Transform*, page 282–300. Springer International Publishing, 2021.

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

[22] K. Kreuzer. CRYSTALS-Kyber Security. *Archive of Formal Proofs*, December 2023. https://isa-afp.org/entries/CRYSTALS-Kyber_Security.html, Formal proof development.

[23] K. Kreuzer. Kyber error bound, 2023. https://github.com/ThikaXer/Kyber_error_bound, accessed: 2023-12-19.

[24] K. Kreuzer. 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.

[25] A. Langlois and D. Stehlé. Worst-Case to Average-Case Reductions for Module Lattices. *Des. Codes Cryptogr.*, 75(3):565–599, June 2015.

[26] A. Lochbihler. CryptHOL. *Archive of Formal Proofs*, May 2017. https://isa-afp.org/entries/CryptHOL.html, Formal proof development.

[27] A. Lochbihler and S. R. Sefidgar. A tutorial introduction to CryptHOL. Cryptology ePrint Archive, Paper 2018/941, 2018. https://eprint.iacr.org/2018/941.

[28] P. Longa and M. Naehrig. Speeding up the Number Theoretic Transform for Faster Ideal Lattice-Based Cryptography. In *Lecture Notes in Computer Science*, pages 124–139. Springer International Publishing, 10 2016.

[29] R. Maleckas, K. G. Paterson, and M. R. Albrecht. Practically-exploitable Vulnerabilities in the Jitsi Video Conferencing System. Cryptology ePrint Archive, Paper 2023/1118, 2023. https://eprint.iacr.org/2023/1118.

[30] E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, July 1991.

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

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

[33] NIST. NIST - Post-Quantum Cryptography, 2023. https://csrc.nist.gov/projects/post-quantum-cryptography, accessed: 2023-05-12.

[34] nLab authors. monad. https://ncatlab.org/nlab/show/monad, Nov. 2022. Revision 97.
[35] 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.
[36] A. Sprenkels. The Kyber/Dilithium NTT, 2022. https://electricdusk.com/ntt.html, accessed: 2022-10-17.
[37] D. Unruh. Post-Quantum Verification of Fujisaki-Okamoto. In S. Moriai and H. Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 321–352, Cham, 2020. Springer International Publishing.

X. Appendix

A. Isabelle Formalizations

1) Isabelle Code for the Quotient Ring R_q: The quotient ring R_q in Isabelle is defined in three steps:

1) Define a type class containing the modulus q and the polynomial $x^n + 1$ as `qr_poly'` and ascertain their compatibility.
2) Define the equivalence relation for polynomials in $\mathbb{Z}_q[x]$ modulo `qr_poly'`
3) Define the final type class `qr` of the quotient ring R_q using the constructor `quotient_type` by modding out the equivalence relation

```
class qr_spec = prime_card +
  fixes qr_poly' :: 'a itself ⇒ int poly
  assumes ¬ int CARD('a) dvd
            lead_coeff (qr_poly' TYPE('a))
  and degree (qr_poly' TYPE('a)) > 0

definition qr_rel where
  qr_rel P Q ↔ [P = Q] (mod qr_poly)

quotient_type (overloaded)
  'a qr = 'a :: qr_spec mod_ring poly / qr_rel
```

2) Isabelle Code for Parameter Sets: The parameter set of Kyber with the required properties is encoded as a locale.

```
locale kyber_spec =
fixes type_a :: ('a :: qr_spec) itself
  and type_k :: ('k ::finite) itself
  and n q::int and k n'::nat
assumes n = 2 ^ n'
    and n' > 0
    and q > 2
    and prime q
    and int (CARD('a :: qr_spec)) = q
    and int (CARD('k :: finite)) = k
    and qr_poly' TYPE('a) =
              Polynomial.monom 1 (nat n) + 1
    and q mod 4 = 1
```

3) Isabelle Code for Kyber Algorithms: The key generation in Isabelle is defined in two steps: First we sample the inputs A, s, e according to their distributions, and second we calculate the formula $As + e$. The second part is implemented in the function `key_gen`, whereas the sampling in the first step is implemented in the function `pmf_key_gen`. Here `pmf_of_set` return a uniform distribution on an input set and `beta_vec` samples a vector of polynomials according to the distribution β_η. The latter returns a probability mass function on the output, corresponding to the probability mass function on the key generation.

```
definition key_gen where
key_gen A s e = A * s + e

definition pmf_key_gen  where
"pmf_key_gen = do {
  A ← pmf_of_set (UNIV::
        (('a qr,'k) vec,'k) vec set);
  s ← beta_vec;
  e ← beta_vec;
  let t = key_gen A s e;
  return_pmf ((A, t),s)
}"
```

As with the key generation, the encryption is also split into the calculation and the sampling part. The calculation is implemented in the function `encrypt` and the sampling in the function `pmf_encrypt`. Again, `pmf_encrypt` returns a probability mass function on the ciphertext. One important fact on the formalization is that the types cast always have to be included, for example `to_module` casts the integer $\lfloor q/2 \rceil$ to the type of the module R_q and `bitstring_to_module` casts the bit-string of the message to an element in R_q.

```
definition encrypt where
encrypt t A r e1 e2 du dv m =
  (compress_vec du (A^T * r + e1),
   compress_poly dv (t^T * r + e2 +
      to_module (round(q/2)) *
      bitstring_to_module m))

definition pmf_encrypt where
"pmf_encrypt pk m = do{
  r ← beta_vec;
  e1 ← beta_vec;
  e2 ← beta;
  let c = encrypt (snd pk) (fst pk)
                r e1 e2 dt du dv m;
  return_pmf c
}"
```

Note that the compression and decompression on R_q^k are defined as an index- and coefficient-wise application. However, we need to separate these definitions in Isabelle using the suffix `_vec` and `_poly`.

Since the decryption is purely deterministic, we only implement its calculation in `decrypt`.

```
definition decrypt where
decrypt u v s du dv =
  compress_poly 1 ((decompress_poly dv v)
    - s^T * (decompress_vec du u))
```

```
lemma kyber_correct:
  fixes A s r e e1 e2 du dv cu cv t u v
  assumes t = key_gen A s e
  and (u,v) = encrypt t A r e1 e2 du dv m
  and cu = compress_error_vec du
        ((transpose A) *v r + e1)
  and cv = compress_error_poly dv
        (scalar_product t r + e2 +
       to_module (round(q/2)) * m)"
  and abs_infty_poly (scalar_product e r
       + e2 + cv - scalar_product s e1
       - scalar_product s cu)
      < round (q / 4)
  and set ((coeffs ∘ of_qr) m) ⊆ {0,1}
  shows decrypt u v s du dv = m
```

Then the corollary that Kyber is `delta_kyber`-correct is formalized as follows:

```
lemma
shows
  expectation pmf_key_gen (λ(pk, sk).
    MAX m ∈ Msgs. pmf (do {
      (u,v) ← pmf_encrypt pk m;
      return_pmf (decrypt u v sk du dv ≠ m)
    })) True)
  ≤ delta_kyber
```

5) Isabelle Code for IND-CPA property: The Isabelle formalization of Theorem VII.1 is the following:

```
theorem concrete_security_kyber:
assumes lossless: ind_cpa.lossless 𝒜
shows
  ind_cpa.advantage (ro.oracle, ro.initial) 𝒜 ≤
  mlwe.advantage (kyber_reduction1 𝒜) +
  mlwe.advantage (kyber_reduction2 𝒜)
```

Here `ro` initialized the random oracle that needs to be specified for the existing definition of the IND-CPA security game in CryptHOL. The lossless property states that the adversary $\mathcal{A}$ terminates.

B. Comparative results on δ, δ' and the correctness error

Figure 7 shows the relations between δ, δ' and the actual correctness error for small parameter sets. Two values are portrayed on the x- and y-axis. If the experimental results of a parameter set lies above the diagonal, then the value on the y axis is bigger. If the results lie below the diagonal, then the value on the x-axis is bigger. These plots show that the inequality between δ and the correctness error is violated, whereas with δ', the inequality is preserved.

C. Proof of the Auxiliary Lemma IV.2

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 $x \geq \lceil q - \frac{q}{2^{d+1}} \rceil$. These cases arise from the distinction whether the modulo reduction in the definition of the compression function is triggered or not. Indeed, we have $comp_d\ x = \lceil \frac{2^d}{q} x \rfloor \mod 2^d$ 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 $comp_d\ x = 0$. In the following, we will abbreviate $comp_d\ x$ 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:

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

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 $\mathrm{mod}^{\pm}$ 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., $comp_d\ x = 0$. Using the assumption on x, we get:

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

□

D. 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 [28] or for the special case of the CRYSTALS suite in [36]. The NTT as a nega-cyclic convolution is described in [20]. To shorten this presentation, we omit all proofs which can be found in the aforementioned references.

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

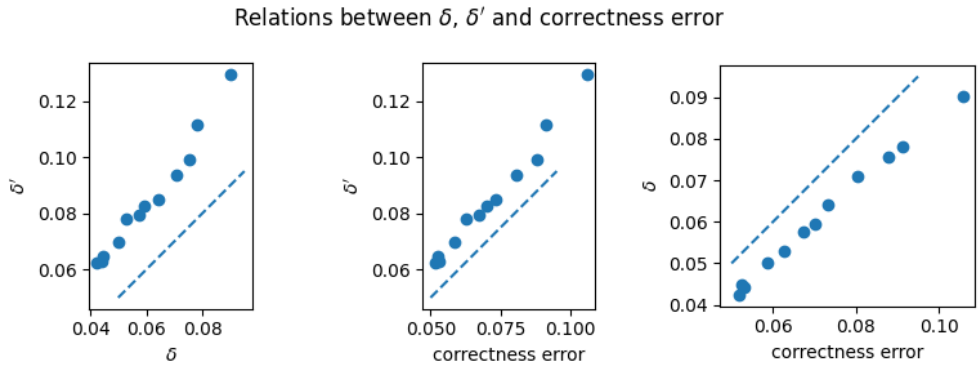

Figure 7: Relational comparisons between δ, δ' and the correctness error for small parameters

$\mathcal{O}(n)$ 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 [11] as follows:

Definition 5 (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 6 (inverse NTT)**.** Let $g = \sum_{k=0}^{n-1} g_k x^k \in R_q$ be in the image of the NTT, then the inverse NTT of g is defined as:

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

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

Theorem X.1. *Let f be a polynomial in R_q and g a polynomial in NTT domain. Then NTT and $invNTT$ are inverses:*

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

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 X.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:*

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

Together with Theorem X.1 this yields the fast multiplication formula.

Theorem X.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 the original Kyber [11] was relatively straight-forward since it is based on the formalization of the standard NTT by Ammer in [3]. 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 [4] was also formalized in [18], we verified only the NTT for the original Kyber specifications. Note that the NTT for the latest versions of Kyber [4], [5] 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.