NTTRU: Truly Fast NTRU Using NTT*

Vadim Lyubashevsky[1] and Gregor Seiler[2]

[1] IBM Research – Zurich, Switzerland
vadim.lyubash@gmail.com
[2] IBM Research – Zurich and ETH Zurich, Switzerland
gseiler@inf.ethz.ch

Abstract. We present NTTRU – an IND-CCA2 secure NTRU-based key encapsulation scheme that uses the number theoretic transform (NTT) over the cyclotomic ring $\mathbb{Z}_{7681}[X]/(X^{768}-X^{384}+1)$ and produces public keys and ciphertexts of approximately 1.25 KB at the 128-bit security level. The number of cycles on a Skylake CPU of our constant-time AVX2 implementation of the scheme for key generation, encapsulation and decapsulation is approximately 6.4K, 6.1K, and 7.9K, which is more than 30X, 5X, and 8X faster than these respective procedures in the NTRU schemes that were submitted to the NIST post-quantum standardization process. These running times are also, by a large margin, smaller than those for all the other schemes in the NIST process. We also give a simple transformation that allows one to provably deal with small decryption errors in OW-CPA encryption schemes (such as NTRU) when using them to construct an IND-CCA2 key encapsulation.

Keywords: NTRU · Lattice Cryptography · AVX2 · NTT

1 Introduction

Lattice-based schemes based on structured polynomial lattices [HPS98, LPR13] provide us with one of the most promising solutions for post-quantum encryption. The public key and ciphertext sizes are about 1 KB and encryption / decryption are faster than that of traditional encryption schemes based on RSA and ECDH assumptions. Lattice schemes are especially fast when they work over rings in which operations can be performed via the Number Theory Transform (NTT) [LMPR08] and many lattice-based encryption schemes indeed utilize this approach (e.g. NewHope [ADPS16], Kyber [BDK+18], LIMA[SAL+17]). Using the NTT could be particularly beneficial to NTRU because key generation (whose timing is important in ephemeral key exchange) requires inversion over a polynomial ring, which is a much more efficient operation when done over NTT-compatible rings. Despite this apparent advantage, there were no NTT-based NTRU schemes submitted to the NIST standardization process, and the key generation procedure in the proposed schemes ([HRSS17], [BCLvV17]) was thus significantly slower than in the proposals based on Ring / Module-LWE.

One of the possible reasons that NTT-based NTRU has not been proposed as a candidate is that NTT is most efficient over rings whose dimension is a power of 2 – i.e. rings of the form $\mathbb{Z}[X]/(X^d \pm 1)$ where d is a power of 2. Based on current security analysis (c.f. [ACD+18]), however, the ring dimension needs to be somewhere between 700 and 800 for 128-bit security (e.g. NTRU-HRSS [HRSS17] uses dimension 701, NTRU-Prime [BCLvV17] is in dimension 761, and Kyber / Saber [DKRV18] use dimension 768). And unlike schemes based on generalized LWE (like Kyber) that are able to use a public key

*Source code available at: https://github.com/gregorseiler/NTTRU

consisting of a matrix of smaller-degree power-of-2 rings without increasing the public key size, this approach does not work for NTRU. The reason is that the part of the public key that contains a composition of small rings in Kyber is completely random and can be generated from a seed via an XOF, whereas the entire NTRU public key is a function of the secret key and none of it can be stored as a seed. Having the public key composed of several parts would therefore significantly increase the size of the NTRU public key.

Some Advantages of NTRU. Despite these limitations on parameter selection, there are some advantages that NTRU enjoys over its counterparts based on Ring-LWE and Module-LWE. The first advantage is speed of encapsulation and decapsulation. The public key of Ring-LWE based primitives consists of a uniformly random polynomial a_1 and another polynomial $a_2 = a_1 s_1 + s_2$, where s_i are kept secret. To reduce the size of the public key, one generates $a_1 = \mathrm{H}(k)$ where k is some short random seed, and only stores k, a_2 as the public key. In encapsulation and decapsulation, however, one needs to compute $a_1 = \mathrm{H}(k)$, and this may be a somewhat costly operation (in comparison to all the other computation envolved in encapsulation / decapsulation) if H is an XOF based on SHAKE or AES.

A more interesting, in our opinion, advantage of NTRU is that NTRU-based public keys and ciphertexts may give rise to more compact (and faster) primitives that utilize zero-knowledge proofs. An NTRU ciphertext consists of just one ring element, whereas a Ring-LWE ciphertext consists of two. This is not a disadvantage for Ring-LWE if one only uses it for encryption because the second ciphertext can be drastically compressed so that only a few (e.g. 2 or 3) high-order bits need to be output. Furthermore, the first ciphertext can also be compressed so that Ring-LWE ciphertexts may actually be a little smaller than NTRU ones. But, if one is using a cryptographic primitive that includes an encryption scheme and a zero-knowledge proof that the ciphertext is correctly formed (e.g. verifiable encryption schemes, group signatures, etc.) then there does not seem to be a way around needing to prove that both parts of the Ring-LWE ciphertext are correctly formed. The proofs for Ring-LWE based schemes would therefore be twice as large as those for NTRU.

We do want to point out that one should be careful when using NTRU in "advanced" primitives. It is often the case that these types of schemes (e.g. FHE [BGV12], verifiable encryption [LN17], group signatures [dPLS18]) require the gap between the modulus and the error to be somewhat large, and it is known that due to the special structure of its public key, NTRU with a large modulus and small error is less secure than Ring-LWE [ABD16, KF17]. [1]

In short, there are reasons that both NTRU and generalized LWE-based schemes may be useful in different situations. Generalized LWE schemes like Kyber and Saber have the advantage that they are based on weaker assumptions, do not require re-implementation to increase/decrease security, and can be used as a basis for schemes requiring a large gap between error and modulus; while NTRU has the advantage of having faster encapsulation / decapsulation and may result in smaller outputs when used together with zero-knowledge proofs. One should therefore hope that both of these variants of doing lattice-based cryptography become accepted standards.

1.1 Our Results

NTT over the ring $\mathbb{Z}_{7681}[X]/(X^{768} - X^{384} + 1)$. The main result of this paper is a very fast AVX2 implementation of NTT over the ring $\mathbb{Z}_q[X]/(X^{768} - X^{384} + 1)$, which leads to an NTRU-based IND-CCA2 secure KEM with key generation, encapsulation, and

[1] This is based on the currently-best algorithms against Ring-LWE, which of course could always improve.

decapsulation algorithms being over 30X, 5X, and 8X faster than the respective procedures in [HRSS17], which is the fastest NTRU-based submission to NIST and the scheme which is being used in Google's CECPQ2 experiment [Lan18]. We show that with an appropriately chosen prime q, one can perform NTT over the ring $\mathbb{Z}_q[X]/(X^{768} - X^{384} + 1)$ essentially as fast as that over a ring that's a power-of-2. We furthermore give additional optimizations related to Montgomery modular reduction that should speed up all NTT computations (thus schemes like Kyber and NewHope should see an improvement in running time.) Our current scheme is currently the fastest (in all aspects) of any lattice-based NIST submissions.

NTT over a polynomial ring $\mathbb{Z}_q[X]/(f(X))$ crucially uses the fact that q is chosen such that the polynomial $f(X)$ can be factored as $f(X) = \prod_{i=1}^{k} f_i(X) \pmod q$, where $f_i(X)$ are polynomials of small degree (usually 1, but could be higher). Then multiplying two polynomials $a, b \in \mathbb{Z}_q[X]/(f(X))$ is done by first computing

$$a_i = a \bmod f_i(X) \text{ and } b_i = b \bmod f_i(X) \text{ for } i = 1 \text{ to } k \tag{1}$$

then computing the component-wise product $(a_1 b_1, \ldots, a_k b_k)$, and finally using the inverse operation to find the polynomial c such that $c \bmod f_i(X) = a_i b_i \bmod f_i(X)$. When the polynomial $f(X) = X^d + 1$, (where d is a power-of-2), then one can pick a modulus q such that $X^d + 1 = (X^{d/2} + r)(X^{d/2} - r)$, and then each of the terms $(X^{d/2} \pm r)$ themselves factor into $(X^{d/4} \pm r')$, and so on until one reaches linear factors. This factorization is what lends power-of-2 rings to have particularly efficient implementations of the decomposition in (1). Furthermore it is usually not necessary to do a "full" polynomial multiplication as, for example, the modular reductions in (1) may already be pre-computed.

Something similar can be achieved for the ring $\mathbb{Z}_q[X]/(X^{768} - X^{384} + 1)$. For our particular choice of $q = 7681$, the polynomial $X^{768} - X^{384} + 1$ initially splits into $(X^{384} + 684)(X^{384} - 685)$, but from that point on, the splitting tree rooted at these two factors always splits into two factors of the form $X^i \pm r'$ all the way down to irreducible polynomials $X^3 \pm r$. Therefore the very first split and the degree of the final irreducible polynomials is different than in the description in the paragraph above. The latter does not create any loss in efficiency because doing polynomial multiplication modulo $\mathbb{Z}_q[X]/(X^3 - r)$ is, or possibly even more, efficient than if one were able to split further.[2] And the fact that the original polynomial does not split into $X^{384} \pm r$ can be dealt with just one extra addition. In short, our NTT over the ring $\mathbb{Z}_q[X]/(X^{768} - X^{384} + 1)$ is as efficient as NTT over power-of-2 rings.

One of the most costly operations when performing NTT is reduction modulo q. Indeed, a significantly improved modular reduction strategy based on a modification of the Montgomery redution algorithm is used in the NTT implementation of Kyber which runs in less than 500 cycles on Skylake and Haswell processors and is more than 5X faster than the previously used floating point NTT [BDK+18, Sei18]. We improve on this modular reduction strategy and reduce the number of 16 bit integer multiplications with 16 bit results needed for one $\mathbb{Z}_q$-multiplication from 4 to 3 with the help of more precomputed constants. Our NTT implementation runs in 810 cycles on Skylake processors which is less than twice as many compared to the Kyber NTT although the input coefficient vectors are 3 times shorter in Kyber. Our improvement can also be used in Kyber, but this would not make much difference for the running time of the whole scheme unless polynomial sampling is drastically improved.

[2] Even in NTT over the ring $\mathbb{Z}_q[X]/(X^d + 1)$ where $X^d + 1$ splits into linear factors modulo q, it may be beneficial to not split all the way into irreducible factors, but instead perform multiplications in rings of the form $\mathbb{Z}_q[X]/(X^{d'} - r)$ for $d' = 2$ or 4.

Dealing with very small decryption errors. The basic building block for constructing an NTRU-based IND-CCA2 secure KEM is a *one-way chosen plaintext attack* (OW-CPA) secure encryption scheme.[3] Like all lattice-based encryption schemes, this scheme has the property that if one sets parameters to minimize public key and ciphertext sizes, valid ciphertexts may be incorrectly decrypted. If this decryption error is too high, then there are simple attacks against the OW-CPA, and the derived IND-CCA2 version, of the scheme.

While NTRU-HRSS and NTRU-Prime set the parameters of their OW-CPA NTRU schemes so as to not have any decryption errors, errors do occur in our scheme with a small probability $\approx 2^{-1230}$, when the probability is taken over the secret key and the randomness and message used in encryption. We show that one can still have a provably-secure IND-CCA2 KEM when it uses such a OW-CPA scheme.[4]

In the transformation from a OW-CPA secure encryption scheme without decryption errors to an IND-CCA2 secure KEM (see Section 2.3), the encryption algorithm only has control of the message because the randomness needed in the OW-CPA encryption scheme is derived from the message using a cryptographic hash function modeled as a random oracle. This implies that if decryption errors occur with probability ϵ and the message space is of size $|\mathcal{M}|$, then the probability over the randomness in the key-generation procedure, a decryption error is even possible is at most $\epsilon \cdot |\mathcal{M}|$. It is therefore safe to use NTRU if decryption errors are $\ll 1/|\mathcal{M}|$. Unfortunately in the NTRU one-way function, the message space is somewhat large, and so this bound may not be good enough. But we show (Section 2.4) that a OW-CPA scheme with an arbitrary-size message space can be transformed into a OW-CPA scheme in which the message space can be as small as the shared-key produced in the KEM (in our case, 256 bits). The cost of this transformation is that the ciphertext also increases by 256 bits and one needs to invoke an extra hash function mapping the original message space to 256 bits. This is, however, in our opinion a worthwhile tradeoff because it still results in a more efficient scheme than if we are forced to increase the modulus to avoid having decryption errors.

1.2 Parameters, Timing, and Comparisons

Without taking the algebraic structure of the underlying ring into account, our scheme is at least as secure as NTRU-HRSS. This is due to the fact that we use the same error distribution while having a larger ring dimension and smaller modulus.[5] Furthermore, both our scheme and NTRU-HRSS use cyclotomic rings (while NTRU-Prime purposefully avoids them), and neither scheme (nor NTRU-Prime) uses the "natural" distribution over the dual of the number field as in [LPR13]. While there are specifically-tailored examples of distributions and ring structure that can make the problem easier (c.f. [CLS16, Pei16]), we are not aware of any natural examples where the security of NTRU is degraded based on the choice of the cyclotomic ring.

In Table 1, we compare the parameters of NTTRU to some other AVX2 optimized

[3] The security definition of a OW-CPA secure encryption scheme is that it is hard, over the choice of the randomness and the message, to recover the message from the ciphertext.

[4] There are works (e.g. [HKSU18, Theorem A.4] and [HHK17]) that show how to deal with decryption errors in the encryption scheme when converting an IND-CPA encryption scheme to an IND-CCA2 KEM. There are some subtle differences, however, between starting with an IND-CPA scheme vs. a OW-CPA one which make these reductions inapplicable, and we discuss this at the end of Section 2.4. There are also works that construct an IND-CCA2 KEM from a OW-CPA scheme with decryption errors [HHK17], but decryption errors there are defined in a way that is not very convenient to compute for NTRU – we have a more detailed discussion of this issue in Section 2.2.

[5] But since the rings are different, we cannot give a formal security reduction between the two schemes.

[6] We removed the 141 bytes for the QROM proof

[7] We removed 32 bytes for the QROM proof

[8] In parentheses are the additional required resources for the message-space reduction transformation from Section 2.4 needed for dealing with small decryption errors.

Table 1: Comparison of lattice-based IND-CCA2 secure KEMs. Cycle counts are medians of many executions of key generation (**K**), encapsulation (**E**) and decapsulation (**D**). Measurements for all schemes except NTTRU (this work) were obtained in the supercop-20190110 benchmarking run on a machine called "samba" with an Intel Skylake Xeon E3-1220 CPU. NTTRU was benchmarked on a laptop with an Intel Skylake i7-6600U CPU. Bytes are given for public keys (**pk**) and ciphertexts (**c**).

Scheme		**Cycles**		**Bytes**
Streamlined NTRU Prime 4591^{761} [BCLvV17]	**K**:	888 115	**pk:**	1 218
	E:	42 073	**c:**	1 047
	D:	88 137		
NTRU-HRSS [HRSS17]	**K**:	220 331	**pk:**	1 140
	E:	34 591	**c:**	1 140[6]
	D:	65 042		
Kyber [BDK+18]	**K**:	77 456	**pk:**	1 088
	E:	105 478	**c:**	1 120[7]
	D:	102 029		
NTTRU (this paper)[8]	**K**:	6 431	**pk:**	1 248
	E:	6 101($+\approx 500$)	**c:**	1 248 (+32)
	D:	7 878($+\approx 500$)		

constant-time implementations of IND-CCA2 secure KEMs that were submitted to the NIST standardization process. We do not consider the QROM model in this paper, while several schemes performed an additional transformation which resulted in a larger ciphertext (but was computationally cheap as it just involved more XOF output).

In Table 2, for illustration purposes, we give the running times of just the NTRU one-way function (and its inversion) part of the protocol. The "mathematical" part of our algorithms requires only ≈ 5000 cycles for key generation and ≈ 2300 cycles for encapsulation / decapsulation. The rest being used for randomness expansion using an XOF and input / output formatting. Due to the extremely fast multiplication using NTT, these latter steps take up a much more significant percentage of our running time than that of NTRU-HRSS, and so we also optimized some of them. It would be interesting to optimize these parts further, as they still form the bulk of the running time.

Table 2: Skylake cycle counts of our AVX2 optimized constant-time implementation of the NTRU one-way function over the ring $\mathbb{Z}_{7681}[X]/(X^{768} - X^{384} + 1)$. The counts are the medians of 256 executions each.

	NTTRU one-way function		
	Key generation	Encryption	Decryption
Total	4933	2346	2349

We now briefly explain why the running time of NTTRU is so much faster than that of the other lattice-based schemes submitted to the NIST standardization process. In short, it's because doing NTRU using NTT does not require any trade-offs that are present in other lattice-based proposals and one can use the optimal choice in every aspect of the scheme.

1. Highly optimized AVX2 NTT is very fast, even over some non-power-of-2 rings, such as the one used in this paper. One NTT operation over a 768-dimensional ring requires only 810 cycles and at most 2 NTTs are needed in each part of the scheme (i.e. key generation, encapsulation, decapsulation). Schemes that do not support NTT require more than 5X more cycles to perform polynomial multiplication.

2. In generalized LWE schemes, every part of the scheme has a procedure that expands a small seed into a long random bitstring that is used to create a random polynomial in the ring (or several polynomials in smaller rings in the case of Kyber / Saber) that forms one part of the public key (the other part of the public key which depends on the secret must be stored in "uncompressed" form). This is done using SHAKE or AES and it can be a rather time-intensive operation which is significantly slower than the part requiring NTT. If this expansion is done via some cheaper procedure which is not a PRF (which should be OK for practical security), then this would not incur such a big penalty for generalized LWE schemes. NTRU does not require any such expansion because its public key only has one part.

3. During seed expansion, one needs to create elements that are uniformly distributed in $\mathbb{Z}_q$. If q is a power-of-2 (e.g. as in Saber), then this is very efficient, but using such a q prevents one from using NTT. If one uses an NTT-compatible q (as in NewHope and Kyber), then one needs to use "rejection sampling" to get uniformity over $\mathbb{Z}_q$. This is, unfortunately, not compatible with fast vectorization due to required branching. One solution may be to use a procedure that results in a somewhat biased (non-uniform) distribution, which should still not degrade the practical security of the scheme. This is not an issue in NTRU-based schemes because the whole public key is the quotient of two polynomials with small coefficients, and so it is simply stored in uncompressed form.

Based on the above, we conjecture that unless lattice schemes use NTT for polynomial multiplication and avoid costly methods for sampling in $\mathbb{Z}_q$, their performance will not be able to match NTRU that uses NTT.

1.3 Open Problems and Future Directions

Our OW-CPA NTRU scheme has (very small) decryption errors and our transformation to an IND-CCA2 scheme that accounts for these errors is currently proved in the standard ROM model. It would be nice to also show that this transformation (or perhaps a small modification of it) also holds in the QROM. As we discuss at the end of Section 2.4, we believe such a proof should be possible (and should even be able to tolerate a larger error) due to the fact that our transformed OW-CPA scheme shares similar properties with IND-CPA and OW-CPA schemes from which QROM transformations accounting for decryption errors do exist.

Our NTT uses a modulus reduction step that uses only one Montgomery reduction and this should apply to other NTT-based schemes. Additionally, very efficient implementations of LPR-type [LPR13] Ring-LWE schemes (e.g. NewHope, LIMA) should now be possible in dimensions 2^k3^ℓ (like e.g. 768).[9]

Acknowledgements.

This work is supported by the SNSF ERC starting transfer grant FELICITY and the Horizon2020 project FutureTPM. We thank Eike Kiltz for useful conversations about decryption errors.

[9] LIMA also proposes a KEM over safe-prime rings in which NTT does not work natively, but they use other (somewhat slower) FFT-related algorithms to perform polynomial multiplication.

2 Preliminaries

2.1 Notation

For some finite set $\mathcal{S}$, we will write $D_{\mathcal{S}}$ to denote some distribution with support on $\mathcal{S}$. We write $\mathrm{H}_{D_{\mathcal{S}}}$ to denote a cryptographic hash function (modeled as a random oracle) that outputs elements onto $\mathcal{S}$ according to the distribution $D_{\mathcal{S}}$. Just writing $\mathrm{H}_{\mathcal{S}}$ means that the distribution onto $\mathcal{S}$ is uniform. The functions $\bmod q$ and $\bmod^{\pm} q$ signify modular reductions modulo (an odd) q with the former mapping integers onto the space $[0, q-1]$ and the latter to the domain $\left[-\frac{q-1}{2}, \frac{q-1}{2}\right]$.

In this paper, we will denote by R_q the polynomial ring $\mathbb{Z}_q[X]/(X^{768} - X^{384} + 1)$ for $q = 7681$. Elements of this ring are polynomials of degree 767 with coefficients between -3840 and 3840. The modular binomial distribution β_k is generated by creating Bernoulli $a_1, \ldots, a_k, b_1, \ldots, b_k \leftarrow \{0,1\}$ and outputting $(\sum a_i - \sum b_i) \bmod^{\pm} 3$. We will write β_k^d to be the distribution over a d-dimensional vector each of whose coefficients is chosen according to β_k. In this paper, as in [HRSS17], we use the distribution β_2, which results in the distribution $\Pr[-1] = 5/16$, $\Pr[0] = 6/16$, $\Pr[1] = 5/16$.

For a d-dimensional vector v, we will abuse notation and write $v \in R_q$ to mean that this vector becomes a polynomial in R_q. In other words, a vector $\{v_0, v_1, \ldots, v_{767}\} \in R_q$ is the polynomial $\sum_{i=0}^{767} v_i X^i$.

2.2 OW-CPA Secure Encryption

An encryption scheme consists of a key-generating algorithm $\mathsf{Gen}(1^\lambda)$, an encryption function $\mathsf{ECPA}(m, r, \mathsf{pk})$, and a decryption function $\mathsf{DCPA}(c, \mathsf{sk})$. The function Gen takes a security parameter λ and outputs a secret key / public key pair $(\mathsf{sk}, \mathsf{pk})$. The encryption algorithm ECPA takes a message $m \in \mathcal{M}$, randomness $r \in \mathcal{R}$, and the public key pk and outputs a ciphertext c. The decryption function $\mathsf{DCPA}(c, \mathsf{sk})$ takes the ciphertext c and the secret key sk and outputs m. If for all (randomized) algorithms $\mathcal{A}$ running in time at most t,

$$\Pr_{(\mathsf{sk},\mathsf{pk})\leftarrow\mathsf{Gen}, m\leftarrow D_{\mathcal{M}}, r\leftarrow D_{\mathcal{R}}}[\mathcal{A}(\mathsf{ECPA}(m, r, \mathsf{pk}), \mathsf{pk}) = m] \leq \epsilon,$$

then we say that the encryption scheme is (t, ϵ)-OW-CPA.

The version of the NTRU encryption scheme we will be using is "randomness-recovering". That is, once the decryption function recovers m, it can also recover the randomness r. This property allows someone in possession of the public key to check whether a ciphertext c is the encryption of a message m using a function $\mathsf{Rec}(m, c, \mathsf{pk})$ (which works by recovering the randomness r and then checking whether $\mathsf{ECPA}(m, r, \mathsf{pk}) = c$). This function is only used in the proofs, where its existence makes the proofs tighter, and is never needed in the actual schemes. If an encryption scheme has such a function Rec, then we will say that the scheme is *message-verifiable*.

A decryption error occurs when for some $(m, r) \in \mathcal{M} \times \mathcal{R}$, we have

$$\mathsf{DCPA}(\mathsf{ECPA}(m, r, \mathsf{pk}), \mathsf{sk}) \neq m.$$

We will say that a OW-CPA scheme has probability ϵ of having a decryption error if

$$\Pr_{(\mathsf{sk},\mathsf{pk})\leftarrow\mathsf{Gen}, m\leftarrow D_{\mathcal{M}}, r\leftarrow D_{\mathcal{R}}}[\mathsf{DCPA}(\mathsf{ECPA}(m, r, \mathsf{pk}), \mathsf{sk}) \neq m] = \epsilon.$$

We point out that this definition differs from that in [HHK17] because they define their decryption error (see [HHK17, Figure 2]) as a game in which the Adversary picks the message after seeing $(\mathsf{pk}, \mathsf{sk})$. We define the error over the randomness of the message

because, in our opinion, it is easier to work with this definition in schemes (like NTRU) where the particular message has a big effect on the decryption error (unlike in LPR-type Ring-LWE schemes where the message has virtually no effect). Also, when the message is random, the decryption error can be computed fairly precisely (Section 3.2).

We now make the observation that if $\mathrm{H}_{D_\mathcal{R}}$ is modeled as a random oracle, then the decryption error stays the same if $r = \mathrm{H}_{D_\mathcal{R}}(m)$. So we can equivalently define the decryption error as:

$$\Pr_{(\mathsf{sk},\mathsf{pk})\leftarrow\mathsf{Gen},m\leftarrow D_\mathcal{M}}[\mathsf{DCPA}(\mathsf{ECPA}(m,\mathrm{H}_{D_\mathcal{R}}(m),\mathsf{pk}),\mathsf{sk}) \neq m] = \epsilon. \tag{2}$$

2.3 IND-CCA2 Secure KEMs and Decryption Errors

One can transform a OW-CPA encryption scheme into an IND-CCA2 secure KEM using standard techniques (c.f. [FO99, Den02]). The key generation is exactly the same as in the OW-CPA scheme, while the encapsulation (Enc) and decapsulation (Dec) procedures are described in Algorithms 1 and 2 following the construction in [Den02, Table 5].

It is shown in [Den02, Theorem 5, Theorem 9] that if there is an adversary who has advantage δ of winning the IND-CCA2 security game against the KEM $(\mathsf{Gen},\mathsf{Enc},\mathsf{Dec})$, then there is an algorithm who, in the same time, can break the OW-CPA security property of the scheme $(\mathsf{Gen},\mathsf{ECPA},\mathsf{DCPA})$ with probability $f(\delta)$. The function f (linearly) depends on the number of various random oracle queries $\mathcal{A}$ can perform (see [Den02, Theorem 9]).

Algorithm 1 Enc

Input: Public key pk
Output: Shared key $k \in \mathcal{K}$ and ciphertext $c \in \mathcal{C}$
1: choose $m \leftarrow D_\mathcal{M}$
2: $r := \mathrm{H}_{D_\mathcal{R}}(m)$
3: $c := \mathsf{ECPA}(m, r, \mathsf{pk})$
4: $k := \mathrm{H}_\mathcal{K}(m)$
5: **return** (k, c)

Algorithm 2 Dec

Input: Secret key sk, ciphertext $c \in \mathcal{C}$
Output: Shared key $k \in \mathcal{K}$
1: $m := \mathsf{DCPA}(c, \mathsf{sk})$. If failure, then halt.
2: $r := \mathrm{H}_{D_\mathcal{R}}(m)$
3: if $\mathsf{ECPA}(m, r, \mathsf{pk}) \neq c$, then halt.
4: **return** $k := \mathrm{H}_\mathcal{K}(m)$

Notice that in the the CCA-secure encapsulation (Algorithm 1), the randomness r passed to the OW-CPA encryption scheme is a deterministic function of the message m, and so the only input over which the encryptor has control over is the message. Therefore if the probability, over the secret key and the message, of a decryption error is ϵ as in (2), then by the union bound we can conclude that

$$\Pr_{(\mathsf{sk},\mathsf{pk})\leftarrow\mathsf{Gen}}[\ \exists m \text{ that causes a decryption error }] \leq \epsilon \cdot |\mathcal{M}|.$$

In other words, with probability $1 - \epsilon \cdot |\mathcal{M}|$ over the choice of the secret key, decryption errors are not possible. Having $\epsilon \cdot |\mathcal{M}| < 2^{-128}$ is therefore enough to discount any attacks based on decryption errors if one is aiming for 128-bit security level. We formally prove the above intuition in Lemma 2. First we show that a OW-CPA scheme remains secure when the randomness r is chosen to be $\mathrm{H}_{D_\mathcal{R}}(m)$ rather than chosen according to $D_\mathcal{R}$.

Lemma 1. *If there is an algorithm $\mathcal{A}$ who is able to break the OW-CPA property of a message-verifiable scheme* $(\mathsf{Gen}, \mathsf{ECPA}, \mathsf{DCPA})$*, when the input to* ECPA *is* $(m, H_{D_\mathcal{R}}(m), \mathsf{pk})$ *for* $m \leftarrow D_\mathcal{M}$*, with probability* δ*, then there is also an algorithm that can break the OW-CPA property of the "usual"* $(\mathsf{Gen}, \mathsf{ECPA}, \mathsf{DCPA})$ *encryption scheme with probability* δ*.*

Proof. The reduction obtains a ciphertext $c = \mathsf{ECPA}(m, r, \mathsf{pk})$ for some unknown m, r and sends the public key pk and c to $\mathcal{A}$. For all $m' \neq m \in \mathcal{M}$, the value $\mathrm{H}_{D_\mathcal{R}}(m')$ is a random value in $\mathcal{R}$. Thus whenever the reduction receives a query $\mathrm{H}_{D_\mathcal{R}}(m')$ for a new m', it uses the algorithm Rec to see whether $\mathsf{Rec}(m', c, \mathsf{pk}) = 1$. If so, it outputs m' as its answer and wins. Otherwise, it chooses a random element in $\mathcal{R}$ and sends it to $\mathcal{A}$. When $\mathcal{A}$ outputs, the answer m, the reduction outputs it as well. Thus the success probability of the reduction is at least δ.[10] □

Lemma 2. *If a message-verifiable encryption scheme* $(\mathsf{Gen}, \mathsf{ECPA}, \mathsf{DCPA})$ *has decryption error* ϵ *and there exists an algorithm* $\mathcal{A}$ *having advantage* δ *in the IND-CCA2 security game against the KEM* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *derived from the encryption scheme, then there is another algorithm that has advantage* $f(\delta) - \epsilon \cdot |\mathcal{M}|$ *of breaking the OW-CPA security of* $(\mathsf{Gen}, \mathsf{ECPA}, \mathsf{DCPA})$*, where* f *is as in the beginning of this section (i.e. the loss in the reduction from [Den02, Theorem 9]).*

Proof. If we define $r = \mathrm{H}_{D_\mathcal{R}}(m)$ with $\mathrm{H}_{D_\mathcal{R}}$ being modeled as a random function, then the condition in the Lemma implies that

$$\Pr_{(\mathsf{sk},\mathsf{pk})\leftarrow\mathsf{Gen}, m\leftarrow D_\mathcal{M}}[\mathsf{DCPA}(\mathsf{ECPA}(m, \mathrm{H}_{D_\mathcal{R}}(m), \mathsf{pk}), \mathsf{sk}) \neq m] \leq \epsilon,$$

and then by the union bound, we have that

$$\Pr_{(\mathsf{sk},\mathsf{pk})\leftarrow\mathsf{Gen}}[\ \exists m \text{ s.t. } \mathsf{DCPA}(\mathsf{ECPA}(m, \mathrm{H}_{D_\mathcal{R}}(m), \mathsf{pk}), \mathsf{sk}) \neq m] \leq \epsilon \cdot |\mathcal{M}|. \tag{3}$$

Let $\mathcal{K}' \subseteq \mathcal{K}$ be the set of "good" keys for which the OW-CPA scheme with $r = \mathrm{H}_{D_\mathcal{R}}(m)$ has no decryption errors. By (3), we know that Gen produces good keys with probability $1 - \epsilon \cdot |\mathcal{M}|$. Therefore, with probability $1 - \epsilon \cdot |\mathcal{M}|$, the OW-CPA scheme with $r = \mathrm{H}_{D_\mathcal{R}}(m)$ has no decryption errors. Therefore an adversary who breaks the IND-CCA2 KEM with probability δ breaks this OW-CPA scheme with probability $f(\delta) - \epsilon \cdot |\mathcal{M}|$. By Lemma 1, this implies that breaking the OW-CPA of the encryption scheme $(\mathsf{Gen}, \mathsf{ECPA}, \mathsf{DCPA})$ also has success probability $f(\delta) - \epsilon \cdot |\mathcal{M}|$. □

2.4 Reducing the Message Space of a OW-CPA Scheme

In light of the fact that having small message spaces helps us discount decryption-error attacks, we show that it is possible to convert a OW-CPA encryption scheme with arbitrary size message spaces to a OW-CPA scheme with a small (e.g. 256-bit) message space. This message space $\mathcal{M}'$ should, in general, be the same size as the shared keyspace $\mathcal{K}$ in the IND-CCA2 KEM. The transformation requires an additional call to a hash function and adds a small number of bits (e.g. 256) to the ciphertext. If the OW-CPA scheme is $(\mathsf{Gen}, \mathsf{ECPA}, \mathsf{DCPA})$, then we define the encryption and decryption functions ECPA' and DCPA' as in Algorithms 3 and 4.

Lemma 3. *Suppose that* $(\mathsf{Gen}, \mathsf{ECPA}, \mathsf{DCPA})$ *is a message-verifiable encryption scheme and there is an adversary against the OW-CPA security property of* $(\mathsf{Gen}, \mathsf{ECPA}', \mathsf{DCPA}')$ *running in time* τ *making* κ *queries to* $H_{\mathcal{M}'}$ *and* μ *queries to* $H_{D_\mathcal{M}}$*, and succeeding with probability* δ*. Then there is an adversary who breaks the OW-CPA security of* $(\mathsf{Gen}, \mathsf{ECPA}, \mathsf{DCPA})$ *in time* τ *with probability* $\delta - (\mu + 1)/|\mathcal{M}'|$*.*

[10] Without the oracle Rec, the reduction could guess which of the queries to $\mathrm{H}_{D_\mathcal{R}}$ is the correct message and output it, thus losing a factor of the number of queries to $\mathrm{H}_{D_\mathcal{R}}$ in the reduction.

Algorithm 3 ECPA'

Input: Randomness $r \in \mathcal{R}$, message $m' \in \mathcal{M}'$, public key pk for ECPA
Output: Ciphertext c'
1: $m := \mathrm{H}_{D_\mathcal{M}}(m')$
2: $c := \mathsf{ECPA}(m, r, \mathsf{pk})$
3: $u := m' \oplus \mathrm{H}_{\mathcal{M}'}(m)$
4: **return** $c' := (c, u)$

Algorithm 4 DCPA'

Input: Ciphertext $c' = (c, u)$, secret key sk for DCPA
Output: message m'
1: $m := \mathsf{DCPA}(c, \mathsf{sk})$
2: $m' := u \oplus \mathrm{H}_{\mathcal{M}'}(m)$
3: **return** m'

Proof. The reduction gets the public key pk and a ciphertext $c = \mathsf{ECPA}(m, r, \mathsf{pk})$ for some random, unknown r, m. It picks random $m', u \leftarrow \mathcal{M}'$ and outputs pk as the public key and $c' = (c, u)$ as the encryption of m'. Because $\mathrm{H}_{D_\mathcal{M}}$ is modeled as a random oracle,[11] (c, u) is a correct distribution – i.e. implicitly, $\mathrm{H}_{D_\mathcal{M}}(m') = m$, which is uniform in $\mathcal{M}$). Note that except for the query $\mathrm{H}_{D_\mathcal{M}}(m')$, the reduction is able to give honest random responses to queries to $\mathrm{H}_{D_\mathcal{M}}$ and $\mathrm{H}_{\mathcal{M}'}$. If $\mathcal{A}$ makes the query $\mathrm{H}_{D_\mathcal{M}}(m')$, then the reduction aborts and fails. Because everything that $\mathcal{A}$ sees is independent of m', the probability that $\mathcal{A}$ makes such a query is at most $\mu/|\mathcal{M}'|$. Furthermore, since $m' = u \oplus \mathrm{H}_{\mathcal{M}'}(m)$, if $\mathcal{A}$ never queries $\mathrm{H}_{\mathcal{M}'}(m)$, then he has at most a $1/|\mathcal{M}'|$ probability of returning the correct m'. Thus with probability at least $\delta - 1/|\mathcal{M}'|$, $\mathcal{A}$ must make the query $\mathrm{H}_{\mathcal{M}'}(m)$. Using the Rec function, the reduction can learn the m that is a valid plaintext for the ciphertext c and return it.[12] □

Note. Notice that when the encryption scheme ECPA' is plugged into the encapsulation function Enc in Algorithm 1, the encryptor no longer has control over the input (m, r, pk) to ECPA. In particular, both m and r are generated from m' using cryptographic hash functions. Thus even if the encryptor knows the secret key, it is not easy for him to come up with an m, r that cause a decryption error. This is in contrast to a generic OW-CPA scheme (and in particular the NTRU OW-CPA scheme) where the decryption error can very much depend on m and the encryptor has full control over it.

If, instead of starting from a OW-CPA scheme, we were creating an IND-CCA2 KEM from an IND-CPA encryption scheme, then the decryption error in an IND-CPA scheme is, by definition, not dependent on the message (because in the definition of IND-CPA security, the adversary has control of the message). In the transformation to an IND-CCA2 KEM, the encryptor also just has control of m (with everything else derived from it via a random oracle), which does not affect the decryption error and this allows for proofs such as [HHK17, HKSU18] to remain meaningful even if the decryption error is not smaller than $1/|\mathcal{M}|$. We therefore conjecture that when one uses ECPA', the condition that the decryption error is $\ll 1/|\mathcal{M}'|$ may also not be necessary. We leave this question, as well as proving the reduction secure in the QROM, to future work.

[11] For this proof, it is actually enough for $\mathrm{H}_{D_\mathcal{M}}$ to be a keyed pseudo-random function with the key being part of the secret key of the encryption scheme ECPA'.
[12] Without Rec, the reduction could guess the m and incur a κ factor loss in the success probability.

Table 3: NTRU Variable Definitions

Description	Variable	Definition
modulus	q	7681
ring	R_q	$\mathbb{Z}_q[X]/(X^{768}-X^{384}+1)$
message space	$\mathcal{M}$	$\{-1,0,1\}^{768} \in R_q$
randomness space	$\mathcal{R}$	$\{-1,0,1\}^{768} \in R_q$
shared key space	$\mathcal{K}$	$\{0,1\}^{256}$
binomial distribution	β_2^{768}	generate $a_1, a_2, a_3, a_4 \leftarrow \{0,1\}^{768}$ output $a_1 + a_2 - a_3 - a_4 \bmod^{\pm} 3 \in R_q$
distributions over $\mathcal{M}$, $\mathcal{R}$	$D_{\mathcal{M}}, D_{\mathcal{R}}$	β_2^{768}

3 OW-CPA NTRU

3.1 The NTRU Function

There are several ways to define a OW-CPA secure NTRU encryption scheme. In general, one chooses two polynomials g, f and sets the public key to be $h = pg/f$ where p is some small prime. The encryption function is then $c = hr + m$ and decryption first computes $fc = pgr + mf \bmod^{\pm} q$. At this point, if all the coefficients of p, g, r, m, and f are small, then $fc \bmod^{\pm} q = pgr + mf \in \mathbb{Z}[X]$ and so $fc \bmod^{\pm} q \bmod^{\pm} p = mf$ and one recovers f by dividing by f modulo p. A decryption error occurs if $pgr + mf$ when computed over $\mathbb{Z}[X]$ (i.e. without reduction modulo q) has coefficients larger, in absolute value, than $(q-1)/2$. We will compute this decryption error in Section 3.2.

In order to make it unnecessary to divide by f modulo p, a common trick is to set $f = pf' + 1$, where f' is chosen according to the same distribution as f was before, which makes f congruent to 1 modulo p. This has the disadvantage of increasing the decryption error, but because one cannot use NTT to do multiplication / division in the ring $\mathbb{Z}_p[X]/(X^{768}-X^{384}+1)$ due to the fact that p is small (i.e. 3 in our case), we believe that it is a worthwhile trade-off if one wants efficiency and can tolerate the larger decryption error.

The key generation, encryption, and decryption $(\mathsf{G_{NTRU}}, \mathsf{E_{NTRU}}, \mathsf{D_{NTRU}})$ procedures are given below based on the templates in Section 2 and the variable definitions are given in Table 3.

Algorithm 5 $\mathsf{G_{NTRU}}$

1: $f' \leftarrow \beta_2^{768}$
2: $f := 3f' + 1$
3: if f is not invertible in R_q, restart (this is done in the process of computing the NTT of f)
4: $g \leftarrow \beta_2^{768}$
5: $h := 3g/f$
6: **return** $(\mathsf{sk} = f, \mathsf{pk} = h)$ (both sk and pk are stored in NTT representation)

Algorithm 6 $\mathsf{E_{NTRU}}$

Input: message m, randomness r, public key h
Output: ciphertext c
return $c := hr + m$ (computed and sent in NTT representation)

Algorithm 7 $\mathsf{D}_{\mathsf{NTRU}}$

Input: ciphertext c, secret key f (both in NTT representation)
Output: message m
return $m := (cf \bmod^{\pm} q) \bmod^{\pm} 3$

Randomness Recovery. In some security proofs of Section 2, we used the fact that the NTRU encryption scheme is message-recovering. In other words, it is possible to recover from the ciphertext both the message and the randomness. Since the ciphertext is $c = hr + m$, once m is recovered, one can simply try to compute $(c - m)/h$. In order for this to work, we would need that h is invertible in R_q. So in addition to checking that f is invertible, we would also need to check that g is. Heuristically, each NTT coefficient of g (which is a polynomial of degree 2 over $\mathbb{Z}_q[X]$) has probability of $1/q^3$ of being 0. There are 256 such coefficients, and so the probability that all of then are non-zero is greater than $1 - 256/q^3 \approx 1 - 2^{-30}$. So one could check for invertibility and restart with a very small probability, but since randomness recovery is not crucial to the proofs (it only allows for them to be tighter), we believe that it should also be fine to ignore this issue, especially for ephemeral key exchange.

3.2 Computing the Decryption Error

The decryption algorithm takes a ciphertext of the form $c = \frac{3g}{f} r + m$ and multiplies by $f = 3f' + 1$, to obtain

$$3gr + 3f'm + m = 3(gr + f'm) + m \tag{4}$$

where all the variables are distributed according to β_2^{768}. For correctness, we need all the coefficients in (4) to be of absolute value at most $(q-1)/2$. This way (4) = (4) $\bmod^{\pm} q$. Since the coefficients of m have size at most 1, we need $gr + f'm < (q-1)/6$ in order to avoid decryption errors.

The best way to analyze the distribution of the coefficients of the result in (4) is to view polynomial multiplication in R_q as vector-matrix multiplication. For example, in the ring $\mathbb{Z}[X]/(X^6 - X^3 + 1)$, the product of $a = \sum_{i=0}^{5} a_i X^i$ and $b = \sum_{i=0}^{5} b_i X^i$ can be written as $c = \sum_{i=0}^{5} c_i X_i$ satisfying

$$\begin{bmatrix} a_0 & -a_5 & -a_4 & -a_3 & -a_2 - a_5 & -a_1 - a_4 \\ a_1 & a_0 & -a_5 & -a_4 & -a_3 & -a_2 - a_5 \\ a_2 & a_1 & a_0 & -a_5 & -a_4 & -a_3 \\ a_3 & a_2 + a_5 & a_1 + a_4 & a_0 + a_3 & a_2 & a_1 \\ a_4 & a_3 & a_2 + a_5 & a_1 + a_4 & a_0 + a_3 & a_2 \\ a_5 & a_4 & a_3 & a_2 + a_5 & a_1 + a_4 & a_0 + a_3 \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{bmatrix} \tag{5}$$

Notice that the first column in the matrix above is just the coefficients of a, the second is aX, and so on, until the last column is aX^5. In general, the multiplication of two polynomials $a = \sum_{i=0}^{d-1} a_i X^i$ and $b = \sum_{i=0}^{d-1} b_i X^i$ in the ring $\mathbb{Z}[X]/(X^d - X^{d/2} + 1)$ can be written as a matrix-vector product

$$\begin{bmatrix} \mathbf{L} - \mathbf{U} & -\mathbf{A} - \mathbf{U} \\ \mathbf{A} + \mathbf{U} & \mathbf{A} + \mathbf{L} \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ b_1 \\ \cdots \\ b_{d-1} \end{bmatrix} \tag{6}$$

where $\mathbf{L}, \mathbf{U}, \mathbf{A}$ are the following square $d/2$-dimensional Toeplitz matrices:

$$\mathbf{L} = \begin{bmatrix} a_0 & 0 & \cdots & 0 \\ a_1 & a_0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ a_{d/2-1} & a_{d/2-2} & \cdots & a_0 \end{bmatrix}, \quad \mathbf{U} = \begin{bmatrix} 0 & a_{d-1} & \cdots & a_{d/2+1} \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & a_{d-1} \\ 0 & 0 & \cdots & 0 \end{bmatrix}, \text{ and}$$

$$\mathbf{A} = \begin{bmatrix} a_{d/2} & a_{d/2-1} & \cdots & a_1 \\ a_{d/2+1} & a_{d/2} & \cdots & a_2 \\ \cdots & \cdots & \cdots & \cdots \\ a_{d-1} & a_{d-2} & \dots & a_{d/2} \end{bmatrix}.$$

The main observation is that each coefficient of the product $\begin{bmatrix} \mathbf{A}+\mathbf{U} & \mathbf{A}+\mathbf{L} \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ b_1 \\ \cdots \\ b_{d-1} \end{bmatrix}$ is the sum of $d/2$ independent random variables

$$c = ba + b'(a + a'), \text{ where } a, b, a', b' \leftarrow \beta_2. \tag{7}$$

For example, in (5), the coefficient c_5 is

$$c_5 = (b_0 a_5 + b_3(a_5 + a_2)) + (b_1 a_4 + b_4(a_4 + a_1)) + (b_2 a_3 + b_5(a_3 + a_0)),$$

and the three summands above are independent.

Similarly, the coefficient of the i^{th} row of

$$\begin{bmatrix} \mathbf{L}-\mathbf{U} & -\mathbf{A}-\mathbf{L} \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ b_1 \\ \cdots \\ b_{d-1} \end{bmatrix}$$

is the sum of $d/2 - i$ random variables c as in (7) and i independent random variables of the form $ba + b'a$ where $a, b, a', b' \leftarrow \beta_2$. It's therefore clear that the "wider" distribution is obtained in the bottom $d/2$ rows of (6) and so we will analyze the tail bounds of that one.

All the c in (7) take values between -3 and 3 with the following distribution:

Table 4: Probability distribution of $ab + b'(a + a')$, where $a, b, a', b' \leftarrow \beta_2$

± 3	± 2	± 1	0
.0190735	.0457764	.2311707	.4079590

Each coefficient of the product gr and $f'm$ is (at worst) distributed as the sum of $d/2$ random variables as in (7), and therefore the sum $gr + f'm$ is distributed as the sum of d independent random variables in (7).

Computing the probability distribution of this sum can be done via a convolution (i.e. polynomial multiplication). Define the polynomial

$$\rho(X) = \sum_{i=-3d}^{3d} \rho_i X^i = \left(\sum_{j=-3}^{3} \theta_j X^j \right)^d, \tag{8}$$

where θ_j is the probability of j in Table 4. Then ρ_i is the probability that the sum of d random variables in (7) is i. Then the probability that any coefficient of $gr + f'm$ is

greater than $(q-1)/6$ is

$$2 \cdot \sum_{i=(q-1)/6}^{3d} \rho_i, \tag{9}$$

where we used the symmetry $\rho_i = -\rho_i$. The probability in the above equation is exact for the coefficients with degree $d/2$ through $d-1$ of $gr+f'm$ (because those correspond to the bottom half of (6)), and represents an upper bound for the other coefficients. Applying the union bound, we summarize the above with the following lemma:

Lemma 4. *When f', g, r, m are chosen from the distribution β_2^d, the probability of a decryption error in the scheme $(\mathsf{G_{NTRU}}, \mathsf{E_{NTRU}}, \mathsf{D_{NTRU}})$ is at most*

$$2d \cdot \sum_{i=(q-1)/6}^{3d} \rho_i,$$

where ρ_i are as in (8). □

Decryption error and security for our parameter set. For the NTRU scheme that uses parameters in Table 3, the decryption error value from Lemma 4 is approximately $\epsilon = 2^{-1230}$. The message space $\mathcal{M}$ consists of polynomials in R_q with $\pm 1, 0$ coefficients, and is therefore of size $3^{768} \approx 2^{1217}$. Unfortunately, the product $\epsilon \cdot |\mathcal{M}| \approx 2^{-13}$ is not small enough to provably guarantee security via Lemma 2 by directly using the encryption scheme $(\mathsf{G_{NTRU}}, \mathsf{E_{NTRU}}, \mathsf{D_{NTRU}})$. In order to apply this lemma, we would need to first decrease the message space (to, say, 2^{256}) by using the construction in Section 2.4. Then the decryption error is sufficiently small to make Lemma 2 meaningful. While applying the transformation of Section 2.4 is not particularly expensive (adding 32 bytes to the ciphertext and around 500 cycles to encryption / decryption), we believe that in practice the error is small enough for this to not be strictly necessary.

4 NTT over $\mathbb{Z}_{7681}[X]/(X^{768} - X^{384} + 1)$

The Number Theoretic Transform, or NTT for short, is a special case of the Fast Fourier Transform over finite fields, see [Ber01] for an excellent survey. In lattice cryptography we need to compute in polynomial rings of the form

$$R_q = \mathbb{Z}_q[X]/(f)$$

where $f \in \mathbb{Z}[X]$ is an irreducible polynomial and q is often a prime – in this paper we only deal with prime q. The NTT starts with the observation that if f factors into a product $f = gh$ over the finite field $\mathbb{Z}_q$, then, by the Chinese remainder theorem, we have an isomorphism

$$\mathbb{Z}_q[X]/(f) \cong \mathbb{Z}_q[X]/(g) \times \mathbb{Z}_q[X]/(h).$$

Now it can be advantageous to compute multiplication or inversion in R_q by computing this map, then the corresponding operations in the two factors, and finally the inverse of the map. Furthermore if the factors g and h continue to split into more factors then one obtains a divide and conquer algorithm for computing in R_q. For the approach to be advantageous, it is of course necessary that all the maps can be computed efficiently. In the popular case where R_q is a power-of-two cyclotomic ring modulo some prime, we have

$$R_q = \mathbb{Z}_q[X]/(X^{2^k} + 1) = \mathbb{Z}_q[X]/(X^{2^{k-1}} - \zeta) \times \mathbb{Z}_q[X]/(X^{2^{k-1}} + \zeta)$$

when $\zeta \in \mathbb{Z}_q$ is a primitive 4-th root of unity. This map is easy to compute with just 2^{k-1} multiplications, 2^{k-1} additions and 2^{k-1} subtractions. Concretely, write $n = 2^k$ and let $f = f_0 + f_1 X + \cdots + f_{n-1} X^{n-1} \in R_q$. Then we have

$$f \bmod X^{n/2} - \zeta = (f_0 + \zeta f_{n/2}) + (f_1 + \zeta f_{n/2+1})X + \cdots + (f_{n/2-1} + \zeta f_{n-1})X^{n/2-1},$$
$$f \bmod X^{n/2} + \zeta = (f_0 - \zeta f_{n/2}) + (f_1 - \zeta f_{n/2+1})X + \cdots + (f_{n/2-1} - \zeta f_{n-1})X^{n/2-1}.$$

The computation of the two coefficients of X^i in the two reduced polynomials involves a multiplication, an addition and a subtraction and is called a butterfly operation. Now if there there are 8-th roots of unity we can repeat this in the same way and split the ring into a total of 4 factors and then iterate further, possibly down to rings modulo linear polynomials if there are 2^{k+1}-th roots. Note that the total cost of all the splittings in each level is exactly the same $n/2$ butterfly operations.

4.1 Our Ring

We turn to the case where the defining polynomial of the ring R_q is the m-th cyclotomic polynomial with m of the form $m = 2^k 3^l$, $k, l \geq 1$. It is given by $X^n - X^{n/2} + 1$ where $n = \varphi(m) = m/3$. In our NTRU instantiation we use the 2304-th cyclotomic polynomial $X^{768} - X^{384} + 1$. For a fast NTT algorithm, the trick is to do a first splitting into two polynomials of the form $X^{n/2} - \zeta_1$, $X^{n/2} - \zeta_2$. Then one can continue with the same radix-2 steps as in the power-of-two case above by extracting square roots of ζ_1 and ζ_2. The main observation we use is that if ζ_1 and $\zeta_2 = \zeta_1^5$ are the two primitive sixth roots of unity in the underlying field then we indeed have $X^n - X^{n/2} + 1 = (X^{n/2} - \zeta_1)(X^{n/2} - \zeta_2)$. This is because the sixth cyclotomic polynomial is $X^2 - X + 1$. Hence $\zeta_1 + \zeta_2 = \zeta_1 + \zeta_1^5 = \zeta_1 - \zeta_1^2 = 1$ and $\zeta_1 \zeta_2 = \zeta_1 \zeta_1^5 = \zeta_1^6 = 1$. So, with $\zeta = \zeta_1$,

$$\mathbb{Z}_q[X]/(X^n - X^{n/2} + 1) = \mathbb{Z}_q[X]/(X^{n/2} - \zeta) \times \mathbb{Z}[x]/(X^{n/2} - \zeta^5)$$

Next notice that we do not have to multiply coefficients by both ζ and ζ^5 to reduce modulo $X^{n/2} - \zeta$ and $X^{n/2} - \zeta^5$, i.e. to compute the Chinese remainder map, because $\zeta^5 = 1 - \zeta$. So instead of multiplying by ζ^5 we can just subtract the already computed product with ζ from the coefficient itself. This means our first level splitting only needs $n/2$ extra additions compared to an optimal radix-2 step. These additional additions do not cost much. For an example, we again write $f = f_0 + f_1 X + \cdots + f_{n-1} X^{n-1}$. Then,

$$\begin{aligned} f \bmod X^{n/2} - \zeta &= (f_0 + \zeta f_{n/2}) + (f_1 + \zeta f_{n/2+1})X + \cdots + (f_{n/2-1} + \zeta f_{n-1})X^{n/2-1} \\ f \bmod X^{n/2} - \zeta^5 &= (f_0 + f_{n/2} - \zeta f_{n/2}) + (f_1 + f_{n/2+1} - \zeta f_{n/2+1})X + \ldots \\ &\quad + (f_{n/2-1} + f_{n-1} - \zeta f_{n-1})X^{n/2-1}. \end{aligned}$$

When there is a $3 \cdot 2^k$-th root the NTT can be continued up to factors modulo polynomials of the form $X^{3^{l-1}} - \zeta$. From there on it is possible to proceed with so-called radix-3 steps down to linear factors if enough further cube roots can be extracted. In our instantiation we use the prime modulus $q = 7681$ which does not support this so we do not go into more details here. Indeed $q - 1 = 7680 = 2^9 \cdot 3 \cdot 5$ and we stop at polynomials of degree under 3.

In summary, our NTT for polynomials in $\mathbb{Z}_{7681}[X]/(X^{768} - X^{384} + 1)$ consists of 8 levels $i = 0, \ldots, 7$. In the i-th level, 2^i polynomials of degree less than $768/2^i$ are each split into two polynomials of degree less than $768/2^{i+1}$. The total cost in each level is 384 multiplications, 384 additions and 384 subtractions in $\mathbb{Z}_q$, except in level 0 where there are 384 additional $\mathbb{Z}_q$-additions. The output of our NTT consists of 256 polynomials modulo different $X^3 - \zeta_j$ with ζ_j varying over the primitive 768-th roots of unity.

Inverting the NTT is very similar to the forward NTT and also consists of 8 levels $i = 0, \ldots, 7$. In the i-th level 2^{7-i} pairs of polynomials are merged by performing 384

butterfly operations and 384 additional additions in level 7. Moreover, there is an additional division by 256 necessary at the end since each merging introduces a superfluous factor of 2.

Our AVX2 optimized implementation of the NTT in assembler language needs just 810 cycles on a Skylake CPU. This is about twice the number of cycles needed for the Kyber NTT (in the ring $\mathbb{Z}_{7681}[X]/(X^{256}+1)$) even though the coefficient vectors are 3 times longer in our case.

4.2 Modular Reduction

In the NTT, after every multiplication of a polynomial coefficient with a root of unity the result needs to be reduced modulo q so that subsequent operations do not overflow. These modular reductions crucially determine the efficiency of the NTT implementation. A particular efficient modular reduction algorithm is the Montgomery reduction algorithm [Mon85]. It does not compute proper Euclidean remainders or even representatives modulo q but instead so-called Hensel remainders. They differ by an additional factor $\beta^{-1} \mod q$ where β is the word size, $\beta = 2^{16}$ in our case. The standard Montgomery reduction algorithm for reducing a 32 bit integer modulo q needs two multiplications of 32-bit integers, one logical AND operation, one 32-bit addition and one bit-shift. So every multiplication in the finite field by a root of unity during the NTT needs in fact three multiplications in the implementation. With the same number of multiplications one can compute remainders using floating point arithmetic, but with the advantage that the results are Euclidean remainders in the range $0 \le r < q$. The Hensel remainders computed by the Montgomery reduction algorithm can go up to $2q$ and hence one needs more reductions after additions to avoid overflows. This and the fact that floating point arithmetic on modern x86 CPUs is about as fast as integer arithmetic was the reason that floating point NTTs where among the fastest NTTs used in lattice cryptography for some time. Examples of schemes relying on floating point NTTs include NewHope [ADPS16] and early versions of Kyber [BDK+18]. This has changed with the NTT used in newer versions of Kyber, including Kyber as submitted to the NIST PQC standardization process. Here an integer-arithmetic NTT implementation is used that is faster then the previous floating point NTT by a factor of more than 5 [BDK+18, Sei18]. This was achieved by using a modification of the Montgomery reduction algorithm that needs less expensive multiplications, and, more importantly, allows to operate on more densely packed vectors in a vectorized implementation. We recall some of the details here for convenience so that we can explain our improvement over the Kyber NTT.

Definition 1. Let $a \in \mathbb{Z}$ and q be an odd positive integer. The *Hensel remainder* r of a modulo q with respect to the word size $\beta = 2^l$ such that $q < \frac{\beta}{2}$ is the unique integer r such that $a = mq + r\beta$ with $-\frac{\beta}{2} \le m < \frac{\beta}{2}$.

Suppose we want to compute the Hensel remainder r of $a = mq + r\beta$ as in Definition 1. The standard Montgomery reduction algorithm computes a slightly different definition of Hensel remainder where $a \ge 0$ and $0 \le m < \beta$. It first multiplies a with $-q^{-1}$ modulo β. This gives $\beta - m$, which is then multiplied by q and added to a, resulting in $(r+q)\beta$. Finally one divides by β and obtains $r + q \equiv a\beta^{-1} \pmod q$. The reason for multiplying with $-q^{-1}$ instead of q^{-1} is that $r + q$ is non-negative.

In the Kyber NTT signed arithmetic is used and the definition as stated with $-\beta/2 \le m < \beta/2$. Then the low words of a and mq are equal, see [Sei18, Lemma 2] for more details. Therefore it is sufficient to only compute the high word of mq which can then be subtracted from the high word of a to directly obtain r without the division by β. In the AVX2 instruction set there are instructions to only compute the low or high 16 bits of all the products of the corresponding elements of two vectors of 16 bit signed integers, namely *vpmullw* and *vpmulhw*. So one can use these instructions to compute $m = aq^{-1} \mod {}^{\pm}\beta$

with a low half-product instead of a full product and a logical AND, and then a signed high half-product to obtain the high word of mq.

Moreover, this reduction algorithm operates separately on the high and low words of a. Hence, to multiply two integers modulo q it is sufficient to separately compute the low and high words of the two-word product and then reduce them as explained. When instead computing full products one needs to occupy twice the width for each coefficient in the vector registers so that there is enough space in between the coefficients for intermediate full products.

In NTT implementations the factors that need to be multiplied with are the fixed roots of unity ζ of the underlying field which are usually precomputed. So when multiplying $b \in \mathbb{Z}_q$ with a precomputed $c \in \mathbb{Z}_q$ we separately compute the two words of $a = bc$ and then multiply the low word by the precomputed $q^{-1} \bmod \beta$. The most important improvement of our NTT over the Kyber NTT is that for every root of unity ζ we also precompute $q^{-1}\zeta \bmod \beta$. Then, since the low product is associative and commutative, we save one low product in each multiplication and subsequent reduction. More concretely, to obtain $a = bc \bmod q$ we compute the low word of bcq^{-1} by doing a low half-product with the precomputed $cq^{-1} \bmod \beta$, then a signed high half-product of the result with q, which we in turn subtract from the signed high half-product of b and c.

In each level the coefficients grow by at most q in magnitude, see [Sei18, Lemma 1]. So, since q fits into 13 bits one needs to additionally reduce the coefficients at least every 4 levels. We make use of the special form $q = 2^{13} - 2^9 + 1$ of our prime which implies $2^{13} \equiv 2^9 - 1 \pmod{q}$ and use [Sei18, Algorithm 4] to perform these modular reductions. But since this reduction algorithm only leaves room for adding integers of magnitude less then $3q$ without overflowing over 16 bit we need to reduce in every third level.

In the inverse NTT some of the coefficients can double in magnitude from one level to the next so we use [Sei18, Algorithm 5] in the inverse NTT so that it is sufficient to still reduce only in every third level.

4.3 Vectorization

One of the reason for the big speed-up of AVX2 optimized NTT implementations using assembly language or intrinsics over C-only implementations, which is in the order of 16X for example in the case of Kyber, is that the NTT can be efficiently vectorized. In each level half of the coefficients need to be multiplied by roots of unity. By loading several of the coefficients into a vector register one can compute products in parallel and then also the subsequent additions and subtractions. This works without any complications as long as the degrees of the polynomials are a multiple of twice the number of coefficients fitting into a vector register. The reason is that in this case those coefficients that need to be multiplied by roots of unity, which make up the upper half of the polynomial, completely fill one or more vector registers and therefore can be multiplied with maximum efficiency. When the degrees of the polynomials have reached the vector size, vectors need to be reshuffled so that only coefficients from the upper half of a polynomial are in a particular vector register before multiplication.

4.4 Instruction Scheduling

As we have explained above, a multiplication over $\mathbb{Z}_q$ where the low product of one of the factors with $q^{-1} \bmod \beta$ is assumed to be available consists of a chain of a low half-product followed by a high half-product and in parallel another high half-product. The results are then combined by a subtraction. On current Intel processors the multiplication instructions *vpmullw* and *vpmulhw* have a latency of 5 cycles each and the subtraction instruction *vpsubw* has a latency of 1 cycle [Fog18]. So it takes 11 cycles before the finished reduced product is ready. In each cycle two multiplication instructions can be dispatched so it is

theoretically possible to multiply over $\mathbb{Z}_q$ with a throughput of $2/3 \cdot 16$ $\mathbb{Z}_q$-products per cycle. One can not rely on the out of order execution capability of the CPU to always find instructions to execute for getting near this throughput. So in our implementation we always interleave the products of 6 coefficient vectors with precomputed roots. Then there are 6 more vectors available to store the intermediate high products. In these multiplication steps of the NTT involving 6 coefficient vectors there are 12 independent multiplication instructions that can keep the multipliers busy until results for the next dependent instructions become ready.

4.5 Reducing Loads and Stores

Since we densely pack 16 coefficients of 16 bit each into a 256 bit AVX2 vector register and work on 6 registers at a time, we could in principle load one polynomial of degree under 96 into the registers and transform it completely down to polynomials of degree under 3 without any further loads and stores. Then we can handle the next polynomial of degree under 96 and so forth. We use a slightly different approach since we want to multiply 6 full vector registers at a time. There are 16 vector registers so we can indeed have 192 coefficients loaded at a time but then there are not enough spare registers for intermediate results during multiplication. Therefore we store 6 vectors registers while multiplying the other 6 registers and transform polynomials of degree less than 192 completely down to degree under 3 polynomials with only 6 stores and 6 loads in each level.

4.6 Reducing NTTs

In our implementation we make use of the standard technique of transmitting polynomials in their transformed representation when this is advantageous. So the ciphertext in our KEM, which consists of the polynomial c, is in fact transmitted in this form, which saves one inverse NTT during encapsulation and one forward NTT in decapsulation. Also the polynomial f is stored in NTT representation in the secret key, saving another NTT during decapsulation.

5 Other Implementation Functions

5.1 Base Case Multiplication

For the base case multiplication which consists of 256 products of polynomials modulo various $X^3 - \zeta$ we use quadratic schoolbook multiplication. So, we compute the following formula involving 11 products in $\mathbb{Z}_q$,

$$\begin{aligned} fg \bmod X^3 - \zeta = {} & (f_0 g_0 + \zeta(f_1 g_2 + f_2 g_1)) \\ & + (f_0 g_1 + f_1 g_0 + \zeta(f_2 g_2))X \\ & + (f_0 g_2 + f_1 g_1 + f_2 g_0)X^2. \end{aligned}$$

Here, in contrast to the $\mathbb{Z}_q$-products in the NTT, the products do not always involve precomputable constants. But still every coefficient f_i of the first polynomial f is multiplied by every coefficient g_i of the second polynomial g. So to save multiplications we compute low half-products of all the coefficients f_i with $q^{-1} \bmod \beta$ and keep them for all the products that involve these coefficients.

Our AVX2 optimized implementation of the base case multiplication needs 396 cycles on a Skylake CPU. A possible optimization to speed-up decapsulation that we did not implement is to store $q^{-1}f$ in the secret key.

5.2 Base Case Inversion

We can write the polynomial product $h = fg \bmod X^3 - \zeta$ as a matrix-vector multiplication over $\mathbb{Z}_q$ which involves the 3x3 rotation matrix corresponding to f,

$$\begin{pmatrix} h_0 \\ h_1 \\ h_2 \end{pmatrix} = \begin{pmatrix} f_0 & \zeta f_2 & \zeta f_1 \\ f_1 & f_0 & \zeta f_2 \\ f_2 & f_1 & f_0 \end{pmatrix} \begin{pmatrix} g_0 \\ g_1 \\ g_2 \end{pmatrix}.$$

Now to compute the inverse of f modulo $X^3 - \zeta$ we compute the inverse of the rotation matrix, of course under the assumption that it exists. It is given by the adjungate matrix divided by the determinant. If the determinant is zero then the polynomial is not invertible and we abort. We know that the inverse matrix will be the rotation matrix of the inverse polynomial. So we can just read off the coefficients from the first column. We find $f^{-1} = d^{-1}(f_0' + f_1' X + f_2' X^2)$ where the coefficients f_i' are given by

$$\begin{aligned} f_0' &= f_0^2 - \zeta f_1 f_2 \\ f_1' &= \zeta f_2^2 - f_0 f_1 \\ f_2' &= f_1^2 - f_0 f_2 \end{aligned}$$

and d is the determinant

$$d = f_0(f_0^2 - \zeta f_1 f_2) + \zeta f_1(f_1^2 - f_0 f_2) + \zeta f_2(\zeta f_2^2 - f_0 f_1) = f_0 f_0' + \zeta(f_1 f_2' + f_2 f_1').$$

It follows that to compute $f^{-1} \bmod X^3 - \zeta$ we need 14 multiplications in $\mathbb{Z}_q$ and one inversion of d, which we obtain by $d^{-1} = d^{q-2}$. The inversion costs another 23 multiplications.

In our vectorized assembler implementation of this function we made some effort to compute the function on as many vectors as possible while maintaining an order of the multiplications so that long dependency chains are started as early as possible under the constraint given by the number of free registers. Our implementation runs in 2200 cycles on a Skylake processor.

5.3 Dealing with Montgomery Factors

As we explained we use our modified Montgomery reductions everywhere when computing multiplications in $\mathbb{Z}_q$ and these reductions introduce additional factors of $\beta^{-1} \bmod q$. We now explain how we deal with them. Inside the NTT we use the standard method of precomputing the compile-time constant roots of unity with additional factors of β so that they cancel with the factors introduced by reduction. In base case multiplication and inversion this is not possible so we need to keep track of the additional factors introduced. Then, since the NTT and its inverse are linear operations, we can remove them together with the division by 256 at the end of the inverse NTT. In base case multiplication we see from the formulas for the coefficients of the product polynomial that there will be a Montgomery factor β^{-1} in every coefficient. In base case inversion the coefficients f_i' also have the same factor β^{-1}, while the determinant d has a factor of β^{-2} in it. When inverting d by raising it to the $(q-2)$-th power, every squaring also squares the Montgomery factor and introduces one additional factor. So the factor in d^{2^i} is equal to $(\beta^{-1})^\nu$ with $\nu = 2^{i+1} + 2^i - 1$. Moreover every multiplication of the different d^{2^i} also additionally multiplies by β^{-1}. We find the total factor in our computation of d^{-1} to be equal to $(\beta^{-1})^\nu$ where

$$\nu = 10 + \sum_{\substack{i=1 \\ i \neq 9}}^{12} (2^{i+1} + 2^i - 1) \equiv -4 \pmod{q-1}.$$

So our computation of d^{-1} differs by a factor of β^4. Then we multiply every f_i' by this and obtain $\beta^2 f^{-1}$. This aligns very nicely with the operations in NTRU. Because f^{-1} is multiplied by g which removes one of the β and then the resulting h is multiplied by r in encryption removing the other β and giving the correct c without any Montgomery factor. In decryption c is multiplied by f, so in the inverse NTT, which is only used in decryption, we need to do an additional multiplication by β together with $1/256 \bmod q$.

5.4 Sampling Binomial Distribution

The short polynomials f and g in key generation and the message m in encapsulation are sampled from the distribution where each coefficient is given by independent random variables of the form $(b_1 + b_2) - (b_3 + b_4) \bmod 3$ with independent Bernoulli variables b_i. In the NTRU-HRSS implementation this is done in two stages. First $b_1 + b_2$ and $b_3 + b_4$ are sampled by adding adjacent bits in a random string. Then the difference of two such results are reduced modulo 3 by looking up the remainder in a table of all 9 possibilities. We use a slightly simplified approach and directly look up the 4 input bits in an appropriate table. When doing this in a straight-forward way one could use a 32 bit integer as the lookup table where the 16 entries in $\{0, 1, 2\}$ are given by the 16 pairs of adjacent bits. Then one just has to shift this 32 bit integer by twice the value of the 4 random bits. We use a slightly improved version where we use symmetries in the table so that a 16 bit table is actually sufficient which is then shifted directly by the value of the 4 random bits. This could be advantageous in a vectorized implementation where one could use 16 copies of the lookup table densely packed in a 256 bit register. Then by variably shifting this register by the corresponding integers in a register where only the four low bits in each 16 bit word are non-zero, one would directly obtain 16 correctly sampled coefficients. Unfortunately, in contrast to the AVX512 instruction set, there is no variable shift of densely packed 16 bit words in AVX2. Therefore we have to resort to the slightly less efficient variable shift of 32 bit words.

5.5 Symmetric Primitives

In the KEM we need a stream cipher to expand a seed to the randomness needed for sampling the short polynomials f and g in key generation and also for the message m in encapsulation. Moreover a hash function is used to hash the message to the randomness for r and the shared secret. It turned out that the speed of arithmetic in our KEM is sufficiently fast so that the use of SHAKE becomes a serious bottleneck. Therefore instead of SHAKE we use AES256 in counter mode to expand seeds and SHA512 to hash the message to a 64 byte string from which the first 32 bytes are taken as the shared secret and the last 32 bytes as a seed for the polynomial r in the NTRU encryption function.

5.6 Vectorized Packing

Also the time needed for packing uniform polynomials modulo q in a bit string where every coefficient only occupies 13 bits turned out to be responsible for a significant time when using a straight-forward C implementation of the packing function. One reason is that when performing this task by packing 8 adjacent coefficients into a string of 13 bytes, then this is quite difficult to vectorize. Hence, instead in our packed bit strings coefficients are adjacent which are 16 places apart in coefficient vector of the corresponding polynomial. This is very easy to vectorize, one can just pack 16 times 8 coefficients simultaneously. Also it does not incur a penalty in implementations that do not use vectorization.

References

[ABD16] Martin R. Albrecht, Shi Bai, and Léo Ducas. A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. In *CRYPTO*, pages 153–178, 2016. 2

[ACD+18] Martin R. Albrecht, Benjamin R. Curtis, Amit Deo, Alex Davidson, Rachel Player, Eamonn W. Postlethwaite, Fernando Virdia, and Thomas Wunderer. Estimate all the {LWE, NTRU} schemes! In *SCN*, pages 351–367, 2018. 1

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange – a new hope. In *Proceedings of the 25th USENIX Security Symposium*, pages 327–343. USENIX Association, 2016. http://cryptojedi.org/papers/#newhope. 1, 16

[BCLvV17] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU prime: Reducing attack surface at low cost. In *SAC*, pages 235–260, 2017. 1, 5

[BDK+18] Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS - kyber: A cca-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P*, pages 353–367, 2018. 1, 3, 5, 16

[Ber01] Daniel J. Bernstein. Multidigit multiplication for mathematicians, 2001. 14

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325, 2012. 2

[CLS16] Hao Chen, Kristin E. Lauter, and Katherine E. Stange. Security considerations for galois non-dual RLWE families. In *SAC*, pages 443–462, 2016. 4

[Den02] Alexander W. Dent. A designer's guide to kems. *IACR Cryptology ePrint Archive*, 2002. http://eprint.iacr.org/2002/174. 8, 9

[DKRV18] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber: Module-lwr based key exchange, cpa-secure encryption and cca-secure KEM. In *AFRICACRYPT*, pages 282–305, 2018. 1

[dPLS18] Rafaël del Pino, Vadim Lyubashevsky, and Gregor Seiler. Lattice-based group signatures and zero-knowledge proofs of automorphism stability. In *CCS*, pages 574–591, 2018. 2

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *CRYPTO*, pages 537–554, 1999. 8

[Fog18] Agner Fog. Instruction tables, 2018. 17

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the fujisaki-okamoto transformation. In *TCC*, pages 341–371, 2017. 4, 7, 10

[HKSU18] Kathrin Hövelmanns, Eike Kiltz, Sven Schäge, and Dominique Unruh. Generic authenticated key exchange in the quantum random oracle model. *IACR Cryptology ePrint Archive*, 2018:928, 2018. 4, 10

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *ANTS*, pages 267–288, 1998. 1

[HRSS17] Andreas Hülsing, Joost Rijneveld, John M. Schanck, and Peter Schwabe. High-speed key encapsulation from NTRU. In *CHES*, pages 232–252, 2017. 1, 3, 5, 7

[KF17] Paul Kirchner and Pierre-Alain Fouque. Revisiting lattice attacks on overstretched NTRU parameters. In *EUROCRYPT (1)*, volume 10210 of *Lecture Notes in Computer Science*, pages 3–26, 2017. 2

[Lan18] Adam Langley, 2018. https://www.imperialviolet.org/2018/12/12/cecpq2.html. 3

[LMPR08] Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert, and Alon Rosen. SWIFFT: A modest proposal for FFT hashing. In *FSE*, pages 54–72, 2008. 1

[LN17] Vadim Lyubashevsky and Gregory Neven. One-shot verifiable encryption from lattices. In *EUROCRYPT*, 2017. 2

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6):43, 2013. Preliminary version appeared in EUROCRYPT 2010. 1, 4, 6

[Mon85] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985. 16

[Pei16] Chris Peikert. How (not) to instantiate ring-lwe. In *SCN*, pages 411–430, 2016. 4

[SAL+17] Nigel P. Smart, Martin R. Albrecht, Yehuda Lindell, Emmanuela Orsini, Valery Osheter, Kenneth G. Paterson, and Guy Peer. LIMA 1.1 - a PQC encryption scheme. Technical report, 2017. https://lima-pq.github.io/. 1

[Sei18] Gregor Seiler. Faster AVX2 optimized NTT multiplication for ring-lwe lattice cryptography. *IACR Cryptology ePrint Archive*, 2018:39, 2018. http://eprint.iacr.org/2018/039. 3, 16, 17