Post-quantum key exchange – a new hope*

Erdem Alkim
Department of Mathemathics, Ege University, Turkey

Léo Ducas
Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands

Thomas Pöppelmann
Infineon Technologies AG, Munich, Germany

Peter Schwabe
Digital Security Group, Radboud University, The Netherlands

Abstract

At IEEE Security & Privacy 2015, Bos, Costello, Naehrig, and Stebila proposed an instantiation of Peikert's ring-learning-with-errors–based (Ring-LWE) key-exchange protocol (PQCrypto 2014), together with an implementation integrated into OpenSSL, with the affirmed goal of providing post-quantum security for TLS. In this work we revisit their instantiation and stand-alone implementation. Specifically, we propose new parameters and a better suited error distribution, analyze the scheme's hardness against attacks by quantum computers in a conservative way, introduce a new and more efficient error-reconciliation mechanism, and propose a defense against backdoors and all-for-the-price-of-one attacks. By these measures and for the same lattice dimension, we more than double the security parameter, halve the communication overhead, and speed up computation by more than a factor of 9 in a portable C implementation and by more than a factor of 24 in an optimized implementation targeting current Intel CPUs. These speedups are achieved with comprehensive protection against timing attacks.

1 Introduction

The last decade in cryptography has seen the birth of numerous constructions of cryptosystems based on lattice problems, achieving functionalities that were previously unreachable (e.g., fully homomorphic cryptography [35]). But even for the simplest tasks in asymmetric cryptography, namely public-key encryption, signatures, and key exchange, lattice-based cryptography offers an important feature: resistance to all known quantum algorithms. In those times of *quantum nervousness* [68, 69], the time has come for the community to deliver and optimize concrete schemes, and to get involved in the standardization of a lattice-based cipher-suite via an open process.

For encryption and signatures, several competitive schemes have been proposed; examples are NTRU encryption [46, 79], Ring-LWE encryption [62] as well as the signature schemes BLISS [28], PASS [44] or the proposal by Bai and Galbraith presented in [6]. To complete the lattice-based cipher-suite, Bos et al. [18] recently proposed a concrete instantiation of the key-exchange scheme of Peikert's improved version of the original protocol of Ding, Xie and Lin [48, 72]. Bos et al. proved its practicality by integrating their implementation as additional cipher-suite into the transport layer security (TLS) protocol in OpenSSL. In the following we will refer to this proposal as BCNS.

Unfortunately, the performance of BCNS seemed rather disappointing. We identify two main sources for this inefficiency. First the analysis of the failure probability was far from tight, resulting in a very large modulus $q \approx 2^{32}$. As a side effect, the security is also significantly lower than what one could achieve with Ring-LWE for a ring of rank $n = 1024$. Second the Gaussian sampler, used to generate the secret parameters, is fairly inefficient and hard to protect against timing attacks. This second source of inefficiency stems from the fundamental misconception that high-quality Gaussian noise is crucial for encryption based on LWE[1], which has also made various other implementations [27, 74] slower and more complex than they would have to be.

*This work was initiated while Thomas Pöppelmann was a Ph.D. student at Ruhr-University Bochum with support from the European Union H2020 SAFEcrypto project (grant no. 644729). This work has furthermore been supported by TÜBITAK under 2214-A Doctoral Research Program Grant, by the European Commission through the ICT program under contract ICT-645622 (PQCRYPTO), and by the Netherlands Organisation for Scientific Research (NWO) through Veni 2013 project 13114 and through a Free Competition Grant. Permanent ID of this document: `0462d84a3d34b12b75e8f5e4ca032869`. Date: 2016-03-28.

[1] This is very different for lattice-based signatures or trapdoors, where distributions need to be meticulously crafted to prevent any leak of information on a secret basis.

1.1 Contributions

In this work, we propose solutions to the performance and security issues of the aforementioned BCNS proposal [18]. Our improvements are possible through a combination of multiple contributions:

- Our first contribution is an improved analysis of the failure probability of the protocol. To push the scheme even further, inspired by analog error-correcting codes, we make use of the lattice D_4 to allow error reconciliation beyond the original bounds of [72]. This drastically decreases the modulus to $q = 12289 < 2^{14}$, which improves both efficiency and security.
- Our second contribution is a more detailed security analysis against quantum attacks. We provide a lower bound on all known (or even presupposed) quantum algorithms solving the shortest-vector problem (SVP), and deduce the potential performance of a quantum BKZ algorithm. According to this analysis, our improved proposal provides 128 bits of post-quantum security with a comfortable margin.
- We furthermore propose to replace the almost-perfect discrete Gaussian distribution by something relatively close, but much easier to sample, and prove that this can only affect the security marginally.
- We replace the fixed parameter $\mathbf{a}$ of the original scheme by a freshly chosen random one in each key exchange. This incurs an acceptable overhead but prevents backdoors embedded in the choice of this parameter and all-for-the-price-of-one attacks.
- We specify an encoding of polynomials in the number-theoretic transform (NTT) domain which allows us to eliminate half of the NTT transformations inside the protocol computation.
- To demonstrate the applicability and performance of our design we provide a portable reference implementation written in C and a highly optimized vectorized implementation that targets recent Intel CPUs and is compatible with recent AMD CPUs. We describe an efficient approach to lazy reduction inside the NTT, which is based on a combination of Montgomery reductions and short Barrett reductions.

Availability of software. We place all software described in this paper into the public domain and make it available online at `https://cryptojedi.org/crypto/#newhope` and `https://github.com/tpoeppelmann/newhope`.

Acknowledgments. We are thankful to Mike Hamburg and to Paul Crowley for pointing out mistakes in a previous version of this paper, and we are thankful to Isis Lovecruft for thoroughly proofreading the paper and for suggesting the name JARJAR for the low-security variant of our proposal.

2 Lattice-based key exchange

Let $\mathbb{Z}$ be the ring of rational integers. We define for an $x \in \mathbb{R}$ the rounding function $\lfloor x \rceil = \lfloor x + \frac{1}{2} \rfloor \in \mathbb{Z}$. Let $\mathbb{Z}_q$, for an integer $q \geq 1$, denote the quotient ring $\mathbb{Z}/q\mathbb{Z}$. We define $\mathscr{R} = \mathbb{Z}[X]/(X^n + 1)$ as the ring of integer polynomials modulo $X^n + 1$. By $\mathscr{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$ we mean the ring of integer polynomials modulo $X^n + 1$ where each coefficient is reduced modulo q. In case χ is a probability distribution over $\mathscr{R}$, then $x \xleftarrow{\$} \chi$ means the sampling of $x \in \mathscr{R}$ according to χ. When we write $\mathbf{a} \xleftarrow{\$} \mathscr{R}_q$ this means that all coefficients of $\mathbf{a}$ are chosen uniformly at random from $\mathbb{Z}_q$. For a probabilistic algorithm $\mathscr{A}$ we denote by $y \xleftarrow{\$} \mathscr{A}$ that the output of $\mathscr{A}$ is assigned to y and that $\mathscr{A}$ is running with randomly chosen coins. We recall the discrete Gaussian distribution $D_{\mathbb{Z},\sigma}$ which is parametrized by the Gaussian parameter $\sigma \in \mathbb{R}$ and defined by assigning a weight proportional to $\exp(\frac{-x^2}{2\sigma^2})$ to all integers x.

2.1 The scheme of Peikert

In this section we briefly revisit the passively secure key-encapsulation mechanism (KEM) that was proposed by Peikert [72] and instantiated in [18] (BCNS). Peikert's KEM scheme is defined by the algorithms (Setup, Gen, Encaps, Decaps) and after a successful protocol run both parties share an ephemeral secret key that can be used to protect further communication (see Protocol 1).

The KEM scheme by Peikert closely resembles a previously introduced Ring-LWE encryption scheme [61] but due to a new error-reconciliation mechanism, one $\mathscr{R}_q$ component of the ciphertext can be replaced by a more compact element in $\mathscr{R}_2$. This efficiency gain is possible due to the observation that it is not necessary to transmit an explicitly chosen key to establish a secure ephemeral session key. In Peikert's scheme, the reconciliation just allows both parties to derive the session key from an approximately agreed pseudorandom ring element. For Alice, this ring element is $\mathbf{us} = \mathbf{as}' + \mathbf{e}'\mathbf{s}$ and for Bob it is $\mathbf{v} = \mathbf{bs}' + \mathbf{e}'' = \mathbf{ass}' + \mathbf{es}' + \mathbf{e}''$. For a full explanation of the reconciliation we refer to the original paper [72] but briefly recall the cross-rounding function

$\langle\cdot\rangle_2$ defined as $\langle v\rangle_2 := \lfloor \frac{4}{q}\cdot v \rceil \mod 2$ and the randomized function $\mathsf{dbl}(v) := 2v - \bar{e}$ for some random $\bar{e}$ where $\bar{e}=0$ with probability $\frac{1}{2}$, $\bar{e}=1$ with probability $\frac{1}{4}$, and $\bar{e}=-1$ with probability $\frac{1}{4}$. Let $I_0 = \{0,1,\ldots,\lfloor\frac{q}{2}\rceil - 1\}$, $I_1 = \{-\lfloor\frac{q}{2}\rfloor,\ldots,-1\}$, and $E = [-\frac{q}{4},\frac{q}{4})$ then the reconciliation function $\mathsf{rec}(w,b)$ is defined as

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

If these functions are applied to polynomials this means they are applied to each of the coefficients separately.

Parameters: q,n,χ $\mathsf{KEM.Setup}()$: $\mathbf{a} \xleftarrow{\$} \mathcal{R}_q$		
Alice (server)		Bob (client)
$\mathsf{KEM.Gen}(\mathbf{a})$:		$\mathsf{KEM.Encaps}(\mathbf{a},\mathbf{b})$:
$\mathbf{s},\mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}',\mathbf{e}',\mathbf{e}'' \xleftarrow{\$} \chi$
$\mathbf{b}\leftarrow\mathbf{as}+\mathbf{e}$	$\xrightarrow{\mathbf{b}}$	$\mathbf{u}\leftarrow\mathbf{as}'+\mathbf{e}'$
		$\mathbf{v}\leftarrow\mathbf{bs}'+\mathbf{e}''$
		$\bar{\mathbf{v}} \xleftarrow{\$} \mathsf{dbl}(\mathbf{v})$
$\mathsf{KEM.Decaps}(\mathbf{s},(\mathbf{u},\mathbf{v}'))$:	$\xleftarrow{\mathbf{u},\mathbf{v}'}$	$\mathbf{v}' = \langle\bar{\mathbf{v}}\rangle_2$
$\mu\leftarrow\mathsf{rec}(2\mathbf{us},\mathbf{v}')$		$\mu\leftarrow\lfloor\bar{\mathbf{v}}\rceil_2$

Protocol 1: Peikert's KEM mechanism.

2.2 The BCNS proposal

In a work by Bos, Costello, Naehrig, and Stebila [18] (BCNS), Peikert's KEM [72] was phrased as a key-exchange protocol (see again Protocol 1), instantiated for a concrete parameter set, and integrated into OpenSSL (see Section 8 for a performance comparison). Selection of parameters was necessary as Peikert's original work does not contain concrete parameters and the security as well as error estimation are based on asymptotics. The authors of [18] chose a dimension $n = 1024$, a modulus $q = 2^{32}-1$, $\chi = D_{\mathbb{Z},\sigma}$ and the Gaussian parameter $\sigma = 8/\sqrt{2\pi} \approx 3.192$. It is claimed that these parameters provide a classical security level of at least 128 bits considering the distinguishing attack [58] with distinguishing advantage less than 2^{-128} and $2^{81.9}$ bits of security against an optimistic instantiation of a quantum adversary. The probability of a wrong key being established is less than $2^{-2^{17}} = 2^{-131072}$. The message $\mathbf{b}$ sent by Alice is a ring element and thus requires at least $\log_2(q)n = 32$ kbits while Bob's response $(\mathbf{u},\mathbf{r})$ is a ring element R_q and an element from R_2 and thus requires at least 33 kbits. As the polynomial $\mathbf{a} \in \mathcal{R}_q$ is shared between all parties this ring element has to be stored or generated on-the-fly. For timings of their implementation we refer to Table 2. We would also like to note that besides its aim for securing classical TLS, the BCNS protocol has already been proposed as a building block for Tor [80] on top of existing elliptic-curve infrastructure [38].

2.3 Our proposal: NewHope

In this section we detail our proposal and modifications of Peikert's protocol[2]. For the same reasons as described in [18] we opt for an unauthenticated key-exchange protocol; the protection of stored transcripts against future decryption using quantum computers is much more urgent than post-quantum authentication. Authenticity will most likely be achievable in the foreseeable future using proven pre-quantum signatures and attacks on the signature will not compromise previous communication. Additionally, by not designing or instantiating a lattice-based authenticated key-exchange protocol (see [30,82]) we reduce the complexity of the key-exchange protocol and simplify the choice of parameters. We actually see it as an advantage to decouple key exchange and authentication as it allows a protocol designer to choose the optimal algorithm for both tasks (e.g., an ideal-lattice-based key exchange and a hash-based signature like [14] for authentication). Moreover, this way the design, security level, and parameters of the key-exchange scheme are not constrained by requirements introduced by the authentication part.

Parameter choices. A high-level description of our proposal is given in Protocol 2 and as in [18,72] all polynomials except for $\mathbf{r} \in \mathcal{R}_4$ are defined in the ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n+1)$ with $n = 1024$ and $q = 12289$. We decided to keep the dimension $n = 1024$ as in [18] to be able to achieve appropriate long-term security. As polynomial arithmetic is fast and also scales better (doubling n roughly doubles the time required for a polynomial multiplication), our choice of n appears to be acceptable from a performance point of view. We chose the modulus $q = 12289$ as it is the smallest prime for which it holds that $q \equiv 1 \mod 2n$ so that the number-theoretic transform (NTT) can be realized efficiently and that we can transfer polynomials in NTT encoding (see Section 7).

As the security level grows with the noise-to-modulus ratio, it makes sense to choose the modulus as small as possible, improving compactness and efficiency together with security. The choice is also appealing as the prime is already used by some implementations of Ring-LWE encryption [27,59,77] and BLISS signatures [28,73]; thus sharing of some code (or hardware modules) between our

[2] For the TLS use-case and for compatibility with BNCS [18] the key exchange is initiated by the server. However, in different scenarios the roles of the server and client can be exchanged.

proposal and an implementation of BLISS would be possible.

Noise distribution and reconciliation. Notably, we also change the distribution of the LWE secret and error and replace discrete Gaussians by the centered binomial distribution ψ_k of parameter $k = 16$ (see Section 4). The reason is that it turned out to be challenging to implement a discrete Gaussian sampler efficiently *and* protected against timing attacks (see [18] and Section 5). On the other hand, sampling from the centered binomial distribution is easy and does not require high-precision computations or large tables as one may sample from ψ_k by computing $\sum_{i=0}^{k} b_i - b'_i$, where the $b_i, b'_i \in \{0,1\}$ are uniform independent bits. The distribution ψ_k is centered (its mean is 0), has variance $k/2$ and for $k = 16$ this gives a standard deviation of $\varsigma = \sqrt{16/2}$. Contrary to [18, 72] we hash the output of the reconciliation mechanism, which makes a distinguishing attack irrelevant and allows us to argue security for the modified error distribution.

Moreover, we generalize Peikert's reconciliation mechanism using an analog error-correction approach (see Section 5). The design rationale is that we only want to transmit a 256-bit key but have $n = 1024$ coefficients to encode data into. Thus we encode one key bit into four coefficients; by doing so we achieve increased error resilience which in turn allows us to use larger noise for better security.

Short-term public parameters. NEWHOPE does not rely on a globally chosen public parameter $\mathbf{a}$ as the efficiency increase in doing so is not worth the measures that have to be taken to allow trusted generation of this value and the defense against backdoors [12]. Moreover, this approach avoids the rather uncomfortable situation that all connections rely on a single instance of a lattice problem (see Section 3) in the flavor of the "Logjam" DLP attack [1].

No key caching. For ephemeral Diffie-Hellman key-exchange in TLS it is common for servers to cache a key pair for a short time to increase performance. For example, according to [21], Microsoft's SChannel library caches ephemeral keys for 2 hours. We remark that for the lattice-based key exchange described in [72], for the key exchange described in [18], and also for the key exchange described in this paper, such short-term caching would be disastrous for security. Indeed, it is crucial that both parties use fresh secrets for each instantiation (thus the performance of the noise sampling is crucial). As short-term key caching typically happens on higher layers of TLS libraries than the key-exchange implementation itself, we stress that particular care needs to be taken to eliminate such caching when switching from ephemeral (elliptic-curve) Diffie-Hellman key exchange to post-quantum lattice-based key exchange. This issue is discussed in more detail in [29].

One *could* enable key caching with a transformation from the CPA-secure key exchange to a CCA-secure key exchange as outlined by Peikert in [72, Section 5]. Note that such a transform would furthermore require changes to the noise distribution to obtain a failure probability that is negligible in the cryptographic sense.

3 Preventing backdoors and all-for-the-price-of-one attacks

One serious concern about the original design [18] is the presence of the polynomial $\mathbf{a}$ as a fixed system parameter. As described in Protocol 2, our proposal includes pseudorandom generation of this parameter for every key exchange. In the following we discuss the reasons for this decision.

Backdoor. In the worst scenario, the fixed parameter $\mathbf{a}$ could be backdoored. For example, inspired by NTRU trapdoors [46, 79], a dishonest authority may choose mildly small $\mathbf{f}, \mathbf{g}$ such that $\mathbf{f} = \mathbf{g} = 1 \bmod p$ for some prime $p \geq 4 \cdot 16 + 1$ and set $\mathbf{a} = \mathbf{g}\mathbf{f}^{-1} \bmod q$. Then, given $(\mathbf{a}, \mathbf{b} = \mathbf{as} + \mathbf{e})$, the attacker can compute $\mathbf{bf} = \mathbf{afs} + \mathbf{fe} = \mathbf{gs} + \mathbf{fe} \mod q$, and, because $\mathbf{g}, \mathbf{s}, \mathbf{f}, \mathbf{e}$ are small enough, compute $\mathbf{gs} + \mathbf{fe}$ in $\mathbb{Z}$. From this he can compute $\mathbf{t} = \mathbf{s} + \mathbf{e} \bmod p$ and, because the coefficients of $\mathbf{s}$ and $\mathbf{e}$ are smaller than 16, their sums are in $[-2 \cdot 16, 2 \cdot 16]$: knowing them modulo $p \geq 4 \cdot 16 + 1$ is knowing them in $\mathbb{Z}$. It now only remains to compute $(\mathbf{b} - \mathbf{t}) \cdot (\mathbf{a} - 1)^{-1} = (\mathbf{as} - \mathbf{s}) \cdot (\mathbf{a} - 1)^{-1} = \mathbf{s} \bmod q$ to recover the secret $\mathbf{s}$.

One countermeasure against such backdoors is the "nothing-up-my-sleeve" process, which would, for example, choose $\mathbf{a}$ as the output of a hash function on a common universal string like the digits of π. Yet, even this process may be partially abused [12], and when not strictly required it seems preferable to avoid it.

All-for-the-price-of-one attacks. Even if this common parameter has been honestly generated, it is still rather uncomfortable to have the security of all connections rely on a single instance of a lattice problem. The scenario is an entity that discovers an unforeseen cryptanalytic algorithm, making the required lattice reduction still very costly, but say, not impossible in a year of computation, given its outstanding computational power. By finding *once* a good enough basis of the lattice $\Lambda = \{(a,1)x + (q,0)y | x, y \in \mathscr{R}\}$, this entity could then compromise *all* communications, using for example Babai's decoding algorithm [5].

This idea of massive precomputation that is only dependent on a fixed parameter $\mathbf{a}$ and then afterwards can be used to break all key exchanges is similar in flavor to the 512-bit "Logjam" DLP attack [1]. This at-

Parameters: $q = 12289 < 2^{14}$, $n = 1024$ Error distribution: ψ_{16}		
Alice (server)		**Bob (client)**
$seed \xleftarrow{\$} \{0,1\}^{256}$		
$\mathbf{a} \leftarrow \mathsf{Parse}(\mathsf{SHAKE\text{-}128}(seed))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \psi_{16}^n$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \psi_{16}^n$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, seed)}$	$\mathbf{a} \leftarrow \mathsf{Parse}(\mathsf{SHAKE\text{-}128}(seed))$
		$\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{bs}' + \mathbf{e}''$
$\mathbf{v}' \leftarrow \mathbf{us}$	$\xleftarrow{(\mathbf{u}, \mathbf{r})}$	$\mathbf{r} \xleftarrow{\$} \mathsf{HelpRec}(\mathbf{v})$
$\nu \leftarrow \mathsf{Rec}(\mathbf{v}', \mathbf{r})$		$\nu \leftarrow \mathsf{Rec}(\mathbf{v}, \mathbf{r})$
$\mu \leftarrow \mathsf{SHA3\text{-}256}(\nu)$		$\mu \leftarrow \mathsf{SHA3\text{-}256}(\nu)$

Protocol 2: Our Scheme. For the definitions of HelpRec and Rec see Section 5. For the definition of encodings and the definition of Parse see Section 7.

tack was only possible in the required time limit because most TLS implementations use fixed primes for Diffie-Hellman. One of the recommended mitigations by the authors of [1] is to avoid fixed primes.

Against all authority. Fortunately, all those pitfalls can be avoided by having the communicating parties generate a fresh **a** at each instance of the protocol (as we propose). If in practice it turns out to be too expensive to generate **a** for every connection, it is also possible to cache **a** on the server side[3] for, say a few hours without significantly weakening the protection against all-for-the-price-of-one attacks. Additionally, the performance impact of generating **a** is reduced by sampling **a** uniformly directly in NTT format (recalling that the NTT is a one-to-one map), and by transferring only a short 256-bit seed for **a** (see Section 7).

A subtle question is to choose an appropriate primitive to generate a "random-looking" polynomial **a** out of a short seed. For a security reduction, it seems to the authors that there is no way around the (non-programmable) random oracle model (ROM). It is argued in [31] that such a requirement is in practice an overkill, and that any pseudorandom generator (PRG) should also work. And while it is an interesting question how such a reasonable pseudo-random generator would interact with our lattice assumption, the cryptographic notion of a PRG *is not* helpful to argue security. Indeed, it is an easy exercise[4] to build (under the NTRU assumption) a "backdoored" PRG that is, formally, a legitimate PRG, but that makes our scheme insecure.

Instead, we prefer to base ourselves on a standard cryptographic hash-function, which is the typical choice of an "instantiation" of the ROM. As a suitable option we see Keccak [17], which has recently been standardized as SHA3 in FIPS-202 [67], and which offers extendable-output functions (XOF) named SHAKE. This avoids costly external iteration of a regular hash function and directly fits our needs.

We use SHAKE-128 for the generation of **a**, which offers 128-bits of (post-quantum) security against collisions and preimage attacks. With only a small performance penalty we could have also chosen SHAKE-256, but we do not see any reason for such a choice, in particular because neither collisions nor preimages lead to an attack against the proposed scheme.

4 Choice of the error distribution

On non-Gaussian errors. In works like [18, 27, 77], a significant algorithmic effort is devoted to sample from a discrete Gaussian distribution to a rather high precision. In the following we argue that such effort is not necessary and motivate our choice of a centered binomial ψ_k as error distribution.

Indeed, we recall that the original worst-case to average-case reductions for LWE [75] and Ring-LWE [62] state hardness for *continuous Gaussian* distributions (and therefore also trivially apply to *rounded Gaussian*, which differ from discrete Gaussians). This also extends to discrete Gaussians [19] but such proofs are not necessarily intended for direct implementations. We recall that the use of discrete Gaussians (or other distributions with very high-precision sampling) is only cru-

[3] But recall that the secrets $\mathbf{s}, \mathbf{e}, \mathbf{s}', \mathbf{s}', \mathbf{e}''$ have to be sampled fresh for every connection.

[4] Consider a secure PRG p, and parse its output $p(\text{seed})$ as two small polynomial $(\mathbf{f}, \mathbf{g})$: an NTRU secret-key. Define $p'(\text{seed}) = \mathbf{gf}^{-1} \bmod q$: under the decisional NTRU assumption, p' is still a secure PRG. Yet revealing the seed does reveal $(\mathbf{f}, \mathbf{g})$ and provides a backdoor as detailed above.

cial for signatures [60] and lattice trapdoors [36], to provide zero-knowledgeness.

The following Theorem states that choosing ψ_k as error distribution in Protocol 2 does not significantly decrease security compared to a rounded Gaussian distribution with the same standard deviation $\sigma = \sqrt{16/2}$.

Theorem 4.1 *Let ξ be the rounded Gaussian distribution of parameter $\sigma = \sqrt{8}$, that is, the distribution of $\lfloor \sqrt{8} \cdot x \rceil$ where x follows the standard normal distribution. Let $\mathscr{P}$ be the idealized version of Protocol 2, where the distribution ψ_{16} is replaced by ξ. If an (unbounded) algorithm, given as input the transcript of an instance of Protocol 2 succeeds in recovering the pre-hash key ν with probability p, then it would also succeed against $\mathscr{P}$ with probability at least*

$$q \geq p^{9/8}/26.$$

Proof *See Appendix B.*

As explained in Section 6, our choice of parameters leaves a comfortable margin to the targeted 128 bits of post-quantum security, which accommodates for the slight loss in security indicated by Theorem 4.1. Even more important from a practical point of view is that no known attack makes use of the difference in error distribution; what matters for attacks are entropy and standard deviation.

Simple implementation. We remark that sampling from the centered binomial distribution ψ_{16} is rather trivial in hardware and software, given the availability of a uniform binary source. Additionally, the implementation of this sampling algorithm is much easier to protect against timing attacks as no large tables or data-dependent branches are required (cf. to the issues caused by the table-based approach used in [18]).

5 Improved error-recovery mechanism

In most of the literature, Ring-LWE encryption allows to encrypt one bit per coordinate of the ciphertext. It is also well known how to encrypt multiple bits per coordinate by using a larger modulus-to-error ratio (and therefore decreasing the security for a fixed dimension n). However, in the context of exchanging a symmetric key (of, say, 256 bits), we end up having a message space larger than necessary and thus want to encrypt *one bit in multiple coordinates*.

In [74] Pöppelmann and Güneysu introduced a technique to encode one bit into two coordinates, and verified experimentally that it led to a better error tolerance. This allows to either increase the error and therefore improve the security of the resulting scheme or to decrease the

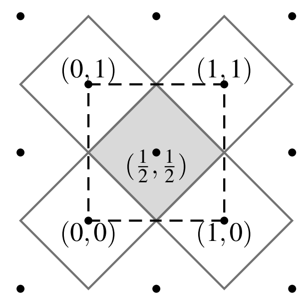

Figure 1: The lattice $\tilde{D}_2$ with Voronoi cells

probability of decryption failures. In this section we propose a generalization of this technique in dimension 4. We start with an intuitive description of the approach in 2 dimensions and then explain what changes in 4 dimensions. Appendices C and D give a thorough mathematical description together with a rigorous analysis.

Let us first assume that both client and server have the same vector $\mathbf{x} \in [0,1)^2 \subset \mathbb{R}^2$ and want to map this vector to a single bit. Mapping polynomial coefficients from $\{0,\ldots,q-1\}$ to $[0,1)$ is easily accomplished through a division by q.

Now consider the lattice $\tilde{D}_2$ with basis $\{(0,1),(\frac{1}{2},\frac{1}{2})\}$. This lattice is a scaled version of the root lattice D_2, specifically, $\tilde{D}_2 = \frac{1}{2} \cdot D_2$. Part of $\tilde{D}_2$ is depicted in Figure 1; lattice points are shown together with their Voronoi cells and the possible range of the vector $\mathbf{x}$ is marked with dashed lines. Mapping $\mathbf{x}$ to one bit is done by finding the closest-vector $v \in \tilde{D}_2$. If $v = (\frac{1}{2},\frac{1}{2})$ (i.e., $\mathbf{x}$ is in the grey Voronoi cell), then the output bit is 1; if $v \in \{(0,0),(0,1),(1,0),(1,1)\}$ (i.e., $\mathbf{x}$ is in a white Voronoi cell) then the output bit is 0.

This map may seem like a fairly complex way to map from a vector to a bit. However, recall that client and server only have a noisy version of $\mathbf{x}$, i.e., the client has a vector $\mathbf{x}_c$ and the server has a vector $\mathbf{x}_s$. Those two vectors are close, but they are not the same and can be on different sides of a Voronoi cell border.

Error reconciliation. The approach described above now allows for an efficient solution to solve this agreement-from-noisy-data problem. The idea is that one of the two participants (in our case the client) sends as a *reconciliation vector* the difference of his vector $\mathbf{x}_c$ and the center of its Voronoi cell (i.e., the point in the lattice). The server adds this difference vector to $\mathbf{x}_s$ and thus moves away from the border towards the center of the correct Voronoi cell. Note that an eavesdropper does not learn anything from the reconciliation information: the client tells the difference to a lattice point, but not whether this is a lattice point producing a zero bit or a one bit.

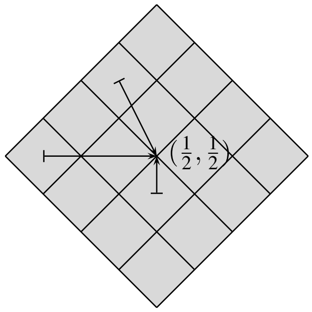

Figure 2: Splitting of the Voronoi cell of $(\frac{1}{2},\frac{1}{2})$ into $2^{rd} = 16$ sub-cells, some with their corresponding difference vector to the center

This approach would require sending a full additional vector; we can reduce the amount of reconciliation information through r-bit discretization. The idea is to split each Voronoi cell into 2^{dr} sub-cells and only send in which of those sub-cells the vector x_c is. Both participants then add the difference of the center of the sub-cell and the lattice point. This is illustrated for $r = 2$ and $d = 2$ in Figure 2.

Blurring the edges. Figure 1 may suggest that the probability of $\mathbf{x}$ being in a white Voronoi cell is the same as for $\mathbf{x}$ being in the grey Voronoi cell. This would be the case if $\mathbf{x}$ actually followed a continuous uniform distribution. However, the coefficients of $\mathbf{x}$ are discrete values in $\{0, \frac{1}{q}, \ldots, \frac{q-1}{q}\}$ and with the protocol described so far, the bits of ν would have a small bias. The solution is to add, with probability $\frac{1}{2}$, the vector $(\frac{1}{2q}, \frac{1}{2q})$ to $\mathbf{x}$ before running the error reconciliation. This has close to no effect for most values of $\mathbf{x}$, but, with probability $\frac{1}{2}$ moves $\mathbf{x}$ to another Voronoi cell if it is very close to one side of a border. Appendix E gives a graphical intuition for this trick in two dimensions and with $q = 9$. The proof that it indeed removes all biases in the key is given in Lemma C.2.

From 2 to 4 dimensions. When moving from the 2-dimensional case considered above to the 4-dimensional case used in our protocol, not very much needs to change. The lattice $\tilde{D}_2$ becomes the lattice $\tilde{D}_4$ with basis $\mathbf{B} = (\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2, \mathbf{g})$, where $\mathbf{u}_i$ are the canonical basis vectors of $\mathbb{Z}^4$ and $\mathbf{g}^t = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$. The lattice $\tilde{D}_4$ is a rotated and scaled version of the root lattice D_4. The Voronoi cells of this lattice are no longer 2-dimensional "diamonds", but 4-dimensional objects called icositetrachoron or 24-cells [57]. Determining in which cell a target point lies in is done using the closest vector algorithm $\mathsf{CVP}_{\tilde{D}_4}$, and a simplified version of it, which we call Decode, gives the result modulo $\mathbb{Z}^4$.

As in the 2-dimensional illustration in Figure 2, we are using 2-bit discretization; we are thus sending $r \cdot d = 8$ bits of reconciliation information per key bit.

Putting all of this together, we obtain the HelpRec function to compute the r-bit reconciliation information as

$$\mathsf{HelpRec}(\mathbf{x}; b) = \mathsf{CVP}_{\tilde{D}_4}\left(\frac{2^r}{q}(\mathbf{x} + b\mathbf{g})\right) \bmod 2^r,$$

where $b \in \{0, 1\}$ is a uniformly chosen random bit. The corresponding function $\mathsf{Rec}(\mathbf{x}, \mathbf{r}) = \mathsf{Decode}(\frac{1}{q}\mathbf{x} - \frac{1}{2^r}\mathbf{Br})$ computes one key bit from a vector $\mathbf{x}$ with 4 coefficients in $\mathbb{Z}_q$ and a reconciliation vector $\mathbf{r} \in \{0, 1, 2, 3\}^4$. The algorithms $\mathsf{CVP}_{\tilde{D}_4}$ and Decode are listed as Algorithm 1 and Algorithm 2, respectively.

Algorithm 1 $\mathsf{CVP}_{\tilde{D}_4}(\mathbf{x} \in \mathbb{R}^4)$

Ensure: An integer vector $\mathbf{z}$ such that $\mathbf{Bz}$ is a closest vector to $\mathbf{x}$: $\mathbf{x} - \mathbf{Bz} \in \mathcal{V}$

$\mathbf{v}_0 \leftarrow \lfloor \mathbf{x} \rceil$
$\mathbf{v}_1 \leftarrow \lfloor \mathbf{x} - \mathbf{g} \rceil$
$k \leftarrow (\|\mathbf{x} - \mathbf{v}_0\|_1 < 1)\ ?\ 0 : 1$
$(v_0, v_1, v_2, v_3)^t \leftarrow \mathbf{v}_k$
return $(v_0, v_1, v_2, k)^t + v_3 \cdot (-1, -1, -1, 2)^t$

Algorithm 2 $\mathsf{Decode}(\mathbf{x} \in \mathbb{R}^4/\mathbb{Z}^4)$

Ensure: A bit k such that $k\mathbf{g}$ is a closest vector to $\mathbf{x} + \mathbb{Z}^4$: $\mathbf{x} - k\mathbf{g} \in \mathcal{V} + \mathbb{Z}^4$

$\mathbf{v} = \mathbf{x} - \lfloor \mathbf{x} \rceil$
return 0 if $\|\mathbf{v}\|_1 \leq 1$ and 1 otherwise

Finally it remains to remark that even with this reconciliation mechanism client and server do not always agree on the same key. Appendix D provides a detailed analysis of the failure probability of the key agreement and shows that it is smaller than 2^{-60}.

6 Post-quantum security analysis

In [18] the authors chose Ring-LWE for a ring of rank $n = 1024$, while most previous instantiations of the Ring-LWE encryption scheme, like the ones in [27, 39, 59, 74], chose substantially smaller rank $n = 256$ or $n = 512$. It is argued that it is unclear if dimension 512 can offer post-quantum security. Yet, the concrete post-quantum security of LWE-based schemes has not been thoroughly studied, as far as we know. In this section we propose such a (very pessimistic) concrete analysis. In particular, our analysis reminds us that the security depends as much on q and its ratio with the error standard deviation

ς as it does on the dimension n. That means that our effort of optimizing the error recovery and its analysis not only improves efficiency but also offers superior security.

Security level over-shoot? With all our improvements, it would be possible to build a scheme with $n = 512$ (and $k = 24$, $q = 12289$) and to obtain security somewhat similar to the one of [18, 39], and therefore further improve efficiency. We call this variant JARJAR and details are provided in Appendix A. Nevertheless, as history showed us with RSA-512 [26], the standardization and deployment of a scheme awakens further cryptanalytic effort. In particular, NEWHOPE could withstand a dimension-halving attack in the line of [33, Sec 8.8.1] based on the Gentry-Szydlo algorithm [37, 56] or the subfield approach of [2]. Note that so far, such attacks are only known for principal ideal lattices or NTRU lattices, and there are serious obstructions to extend them to Ring-LWE, but such precaution seems reasonable until lattice cryptanalysis stabilizes.

We provide the security and performance analysis of JARJAR in Appendix A only for comparison with other lower-security proposals. We strongly recommend NEWHOPE for any application, and advise against using JARJAR or any other similarly aggressive proposal.

6.1 Methodology: the core SVP hardness

We analyze the hardness of Ring-LWE as an LWE problem, since, so far, the best known attacks do not make use of the ring structure. There are many algorithms to consider in general (see the survey [3]), yet many of those are irrelevant for our parameter set. In particular, because there are only $m = n$ samples available one may rule out BKW types of attacks [49] and linearization attacks [4]. This essentially leaves us with two BKZ [23, 78] attacks, usually referred to as primal and dual attacks that we will briefly recall below.

The algorithm BKZ proceeds by reducing a lattice basis using an SVP oracle in a smaller dimension b. It is known [43] that the number of calls to that oracle remains polynomial, yet concretely evaluating the number of calls is rather painful, and this is subject to new heuristic ideas [22, 23]. We choose to ignore this polynomial factor, and rather evaluate only the *core SVP hardness*, that is the cost of *one call* to an SVP oracle in dimension b, which is clearly a pessimistic estimation (from the defender's point of view).

6.2 Enumeration versus quantum sieve

Typical implementations [20, 23, 32] use an enumeration algorithm as this SVP oracle, yet this algorithm runs in super-exponential time. On the other hand, the sieve algorithms are known to run in exponential time, but are so far slower in practice for accessible dimensions $b \approx 130$. We choose the latter to predict the core hardness and will argue that for the targeted dimension, enumerations are expected to be greatly slower than sieving.

Quantum sieve. A lot of recent work has pushed the efficiency of the original lattice sieve algorithms [64, 70], improving the heuristic complexity from $(4/3)^{b+o(b)} \approx 2^{0.415b}$ down to $\sqrt{3/2}^{b+o(b)} \approx 2^{0.292b}$ (see [9, 51]). The hidden sub-exponential factor is known to be much greater than one in practice, so again, estimating the cost ignoring this factor leaves us with a significant pessimistic margin.

Most of those algorithms have been shown [50, 52] to benefit from Grover's quantum search algorithm, bringing the complexity down to $2^{0.262b}$. It is unclear if further improvements are to be expected, yet, because all those algorithms require classically building lists of size $\sqrt{4/3}^{b+o(b)} \approx 2^{0.2075b}$, it is very plausible that the best quantum SVP algorithm would run in time greater than $2^{0.2075b}$.

Irrelevance of enumeration for our analysis. In [23], predictions of the cost of solving SVP classically using the most sophisticated heuristic enumeration algorithms are given. For example, solving SVP in dimension 100 requires visiting about 2^{40} nodes, and 2^{103} nodes in dimension 190. Because this enumeration is a backtracking algorithm, it does benefit from the recent quasi-quadratic speedup [65], decreasing the quantum cost to about 2^{20} to 2^{51} operations as the dimension increases from 100 to 190.

On the other hand, our best-known attack bound $2^{0.262b}$ gives a cost of 2^{51} in dimension 190, and the best plausible attack bound $2^{0.2075b} \approx 2^{39}$. Because enumeration is super-exponential (both in theory and practice), its cost will be worse than our bounds in dimension larger than 200 and we may safely ignore this kind of algorithm.

6.3 Primal attack

The primal attack consists of constructing a unique-SVP instance from the LWE problem and solving it using BKZ. We examine how large the block dimension b is required to be for BKZ to find the unique solution. Given the matrix LWE instance $(\mathbf{A}, \mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e})$ one builds the lattice $\Lambda = \{\mathbf{x} \in \mathbb{Z}^{m+n+1} : (\mathbf{A}| - \mathbf{I}_m| - \mathbf{b})\mathbf{x} = \mathbf{0} \bmod q\}$ of dimension $d = m + n + 1$, volume q^m, and with a unique-SVP solution $\mathbf{v} = (\mathbf{s}, \mathbf{e}, 1)$ of norm $\lambda \approx \varsigma\sqrt{n+m}$. Note that the number of used samples m may be chosen between 0 and $2n$ in our case and we numerically optimize this choice.

Success condition. We model the behavior of BKZ using the geometric series assumption (which is known to be optimistic from the attacker's point of view), that finds a basis whose Gram-Schmidt norms are given by $\|\mathbf{b}_i^\star\| = \delta^{d-2i-1} \cdot \mathrm{Vol}(\Lambda)^{1/d}$ where $\delta = ((\pi b)^{1/b} \cdot b/2\pi e)^{1/2(b-1)}$ [3, 22]. The unique short vector $\mathbf{v}$ will be detected if the projection of $\mathbf{v}$ onto the vector space spanned by the last b Gram-Schmidt vectors is shorter than $\mathbf{b}_{d-b}^\star$. Its projected norm is expected to be $\varsigma\sqrt{b}$, that is the attack is successful if and only if

$$\varsigma\sqrt{b} \le \delta^{2b-d-1} \cdot q^{m/d}. \tag{1}$$

6.4 Dual attack

The dual attack consists of finding a short vector in the dual lattice $\mathbf{w} \in \Lambda' = \{(\mathbf{x},\mathbf{y}) \in \mathbb{Z}^m \times \mathbb{Z}^n : \mathbf{A}^t\mathbf{x} = \mathbf{y} \bmod q\}$. Assume we have found a vector $(\mathbf{x},\mathbf{y})$ of length ℓ and compute $z = \mathbf{v}^t \cdot \mathbf{b} = \mathbf{v}^t\mathbf{A}\mathbf{s} + \mathbf{v}^t\mathbf{e} = \mathbf{w}^t\mathbf{s} + \mathbf{v}^t\mathbf{e} \bmod q$ which is distributed as a Gaussian of standard deviation $\ell\varsigma$ if $(\mathbf{A},\mathbf{b})$ is indeed an LWE sample (otherwise it is uniform mod q). Those two distributions have maximal variation distance bounded by $\varepsilon = \exp(-\pi\tau^2)$ where $\tau = \ell\varsigma/q$, that is, given such a vector of length ℓ one has an advantage ε against decision-LWE.

The length ℓ of a vector given by the BKZ algorithm is given by $\ell = \|\mathbf{b}_0\|$. Knowing that Λ' has dimension $d = m+n$ and volume q^n we get $\ell = \delta^{d-1}q^{n/d}$. Therefore, obtaining an ε-distinguisher requires running BKZ with block dimension b where

$$-\pi\tau^2 \ge \ln\varepsilon. \tag{2}$$

Note that small advantages ε are not relevant since the agreed key is hashed: an attacker needs an advantage of at least $1/2$ to significantly decrease the search space of the agreed key. He must therefore amplify his success probability by building about $1/\varepsilon^2$ many such short vectors. Because the sieve algorithms provide $2^{0.2075b}$ vectors, the attack must be repeated at least R times where

$$R = \max(1, 1/(2^{0.2075b}\varepsilon^2)).$$

This makes the conservative assumption that all the vectors provided by the Sieve algorithm are as short as the shortest one.

6.5 Security claims

According to our analysis, we claim that our proposed parameters offer at least (and quite likely with a large margin) a post-quantum security of 128 bits. The cost of the primal attack and dual attacks (estimated by our script `scripts/PQsecurity.py`) are given in Table 1. For comparison we also give a lower bound on the security of [18] and do notice a significantly improved security in our proposal. Yet, because of the numerous pessimistic assumption made in our analysis, we do not claim any quantum attacks reaching those bounds.

Attack	m	b	Known Classical	Known Quantum	Best Plausible
BCNS proposal [18]: $q = 2^{32}-1$, $n = 1024$, $\varsigma = 3.192$					
Primal	1062	296	86	77	61
Dual	1042	259	84	**76**	62
NTRUENCRYPT [45]: $q = 2^{12}$, $n = 743$, $\varsigma \approx \sqrt{2/3}$					
Primal	613	603	176	157	125
Dual	510	459	142	**128**	103
JARJAR: $q = 12289$, $n = 512$, $\varsigma = \sqrt{12}$					
Primal	623	449	131	117	93
Dual	531	341	106	**95**	77
NEWHOPE: $q = 12289$, $n = 1024$, $\varsigma = \sqrt{8}$					
Primal	1100	967	282	253	200
Dual	938	761	229	**206**	165

Table 1: Core hardness of NEWHOPE and JARJAR and selected other proposals from the literature. The value b denotes the block dimension of BKZ, and m the number of used samples. Cost is given in $\log_2$ and is the smallest cost for all possible choices of m and b. Note that our estimation is very optimistic about the abilities of the attacker so that our result for the parameter set from [18] *does not* indicate that it can be broken with $\approx 2^{80}$ bit operations, given today's state-of-the-art in cryptanalysis.

Most other RLWE proposals achieve considerably lower security than NEWHOPE; for example, the highest-security parameter set used for RLWE encryption in [39] is very similar to the parameters of JARJAR. The situation is different for NTRUENCRYPT, which has been instantiated with parameters that achieve about 128 bits of security according to our analysis[5].

Specifically, we refer to NTRUENCRYPT with $n = 743$ as suggested in [45]. A possible advantage of NTRUENCRYPT compared to NEWHOPE is somewhat smaller message sizes, however, this advantage becomes very small when scaling parameters to achieve a similar security margin as NEWHOPE. The large downside of using NTRUENCRYPT for ephemeral key exchange is the cost for key generation. The implementation of NTRUENCRYPT with $n = 743$ in eBACS [15] takes about an order of magnitude longer for key generation alone than NEWHOPE takes in total. Also, unlike our NEWHOPE software, this NTRUENCRYPT software is not protected against timing attacks; adding such protection would presumably incur a significant overhead.

[5] For comparison we view the NTRU key-recovery as an homogeneous Ring-LWE instance. We do not take into account the combinatorial vulnerabilities [47] induced by the fact that secrets are ternary. We note that NTRU is a potentially a weaker problem than Ring-LWE: it is in principle subject to a subfield-lattice attack [2], but the parameters proposed for NTRUENCRYPT seem immune.

7 Implementation

In this section we provide details on the encodings of messages and describe our portable reference implementation written in C, as well as an optimized implementation targeting architectures with AVX vector instructions.

7.1 Encodings and generation of a

The key-exchange protocol described in Protocol 1 and also our protocol as described in Protocol 2 exchange messages that contain mathematical objects (in particular, polynomials in $\mathscr{R}_q$). *Implementations* of these protocols need to exchange messages in terms of byte arrays. As we will describe in the following, the choice of encodings of polynomials to byte arrays has a serious impact on performance. We use an encoding of messages that is particularly well-suited for implementations that make use of quasi-linear NTT-based polynomial multiplication.

Definition of NTT and NTT^{-1}. The NTT is a tool commonly used in implementations of ideal lattice-based cryptography [27, 39, 59, 74]. For some background on the NTT and the description of fast software implementations we refer to [42, 63]. In general, fast quasi-logarithmic algorithms exist for the computation of the NTT and a polynomial multiplication can be performed by computing $\mathbf{c} = \mathsf{NTT}^{-1}(\mathsf{NTT}(\mathbf{a}) \circ \mathsf{NTT}(\mathbf{b}))$ for $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathscr{R}$. An NTT targeting ideal lattices defined in $\mathscr{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$ can be implemented very efficiently if n is a power of two and q is a prime for which it holds that $q \equiv 1 \mod 2n$. This way a primitive n-th root of unity ω and its square root γ exist. By multiplying coefficient-wise by powers of $\gamma = \sqrt{\omega} \mod q$ before the NTT computation and after the reverse transformation by powers of γ^{-1}, no zero padding is required and an n-point NTT can be used to transform a polynomial with n coefficients.

For a polynomial $\mathbf{g} = \sum_{i=0}^{1023} g_i X^i \in \mathscr{R}_q$ we define

$$\mathsf{NTT}(\mathbf{g}) = \hat{\mathbf{g}} = \sum_{i=0}^{1023} \hat{g}_i X^i, \text{ with}$$
$$\hat{g}_i = \sum_{j=0}^{1023} \gamma^j g_j \omega^{ij},$$

where we fix the n-th primitive root of unity to $\omega = 49$ and thus $\gamma = \sqrt{\omega} = 7$. Note that in our implementation we use an in-place NTT algorithm which requires bit-reversal operations. As an optimization, our implementations skips these bit-reversals for the forward transformation as all inputs are only random noise. This optimization is transparent to the protocol and for simplicity omitted in the description here.

The function NTT^{-1} is the inverse of the function NTT. The computation of NTT^{-1} is essentially the same as the computation of NTT, except that it uses $\omega^{-1} \mod q = 1254$, multiplies by powers of $\gamma^{-1} \mod q = 8778$ after the summation, and also multiplies each coefficient by the scalar $n^{-1} \mod q = 12277$ so that

$$\mathsf{NTT}^{-1}(\hat{\mathbf{g}}) = \mathbf{g} = \sum_{i=0}^{1023} g_i X^i, \text{ with}$$
$$g_i = n^{-1} \gamma^{-i} \sum_{j=0}^{1023} \hat{g}_j \omega^{-ij}.$$

The inputs to NTT^{-1} are *not* just random noise, so inside NTT^{-1} our software has to perform the initial bit reversal, making NTT^{-1} slightly more costly than NTT.

Definition of Parse. The public parameter $\mathbf{a}$ is generated from a 256-bit seed through the extendable-output function SHAKE-128 [67, Sec. 6.2]. The output of SHAKE-128 is considered as an array of 16-bit, unsigned, little-endian integers. Each of those integers is reduced modulo 2^{14} (the two most-significant bits are set to zero) and then used as a coefficient of $\mathbf{a}$ if it is smaller than q and rejected otherwise. The first such 16-bit integer is used as the coefficient of X^0, the next one as coefficient of X^1 and so on. Due to a small probability of rejections, the amount of output required from SHAKE-128 depends on the seed – what is required is $n = 1024$ coefficients that are smaller than q. The minimal amount of output is thus 2 KB; the average amount is 2730.66 bytes. Using a variable amount of output from SHAKE-128 does not create a timing leak, simply because inputs and outputs are public.The resulting polynomial $\mathbf{a}$ (denoted as $\hat{\mathbf{a}}$) is considered to be in NTT domain. This is possible because the NTT transforms uniform noise to uniform noise.

The message format of $(\mathbf{b}, seed)$ and $(\mathbf{u}, \mathbf{r})$. With the definition of the NTT, we can now define the format of the exchanged messages. In both $(\mathbf{b}, seed)$ and $(\mathbf{u}, \mathbf{r})$ the polynomial is transmitted in the NTT domain (as in works like [74,77]). Polynomials are encoded as an array of 1792 bytes, in a compressed little-endian format. The encoding of *seed* is straight-forward as an array of 32 bytes, which is simply concatenated with the encoding of $\mathbf{b}$. Also the encoding of $\mathbf{r}$ is fairly straight-forward: it packs four 2-bit coefficients into one byte for a total of 256 bytes, which are again simply concatenated with the encoding of $\mathbf{u}$. We denote these encodings to byte arrays as encodeA and encodeB and their inverses as decodeA and decodeB. For a description of our key-exchange protocol including encodings and with explicit NTT and NTT^{-1} transformations, see Protocol 3.

Parameters: $q = 12289 < 2^{14}$, $n = 1024$ Error distribution: ψ_{16}^n		
Alice (server)		**Bob (client)**
$seed \xleftarrow{\$} \{0,\ldots,255\}^{32}$		
$\hat{\mathbf{a}} \leftarrow \mathsf{Parse}(\mathsf{SHAKE\text{-}128}(seed))$		
$\mathbf{s},\mathbf{e} \xleftarrow{\$} \psi_{16}^n$		$\mathbf{s}',\mathbf{e}',\mathbf{e}'' \xleftarrow{\$} \psi_{16}^n$
$\hat{\mathbf{s}} \leftarrow \mathsf{NTT}(\mathbf{s})$		
$\hat{\mathbf{b}} \leftarrow \hat{\mathbf{a}} \circ \hat{\mathbf{s}} + \mathsf{NTT}(\mathbf{e})$	$\xrightarrow[\text{1824 Bytes}]{m_a = \mathsf{encodeA}(seed, \hat{\mathbf{b}})}$	$(\hat{\mathbf{b}}, seed) \leftarrow \mathsf{decodeA}(m_a)$
		$\hat{\mathbf{a}} \leftarrow \mathsf{Parse}(\mathsf{SHAKE\text{-}128}(seed))$
		$\hat{\mathbf{t}} \leftarrow \mathsf{NTT}(\mathbf{s}')$
		$\hat{\mathbf{u}} \leftarrow \hat{\mathbf{a}} \circ \hat{\mathbf{t}} + \mathsf{NTT}(\mathbf{e}')$
		$\mathbf{v} \leftarrow \mathsf{NTT}^{-1}(\hat{\mathbf{b}} \circ \hat{\mathbf{t}}) + \mathbf{e}''$
$(\hat{\mathbf{u}}, \mathbf{r}) \leftarrow \mathsf{decodeB}(m_b)$	$\xleftarrow[\text{2048 Bytes}]{m_b = \mathsf{encodeB}(\hat{\mathbf{u}}, \mathbf{r})}$	$\mathbf{r} \xleftarrow{\$} \mathsf{HelpRec}(\mathbf{v})$
$\mathbf{v}' \leftarrow \mathsf{NTT}^{-1}(\hat{\mathbf{u}} \circ \hat{\mathbf{s}})$		$\nu \leftarrow \mathsf{Rec}(\mathbf{v}, \mathbf{r})$
$\nu \leftarrow \mathsf{Rec}(\mathbf{v}', \mathbf{r})$		$\mu \leftarrow \mathsf{SHA3\text{-}256}(\nu)$
$\mu \leftarrow \mathsf{SHA3\text{-}256}(\nu)$		

Protocol 3: Our proposed protocol including NTT and NTT^{-1} computations and sizes of exchanged messages; $\circ$ denotes pointwise multiplication; elements in NTT domain are denoted with a hat $(\hat{\ })$

7.2 Portable C implementation

This paper is accompanied by a C reference implementation described in this section and an optimized implementation for Intel and AMD CPUs described in the next section. The main emphasis in the C reference implementation is on simplicity and portability. It does not use any floating-point arithmetic and outside the Keccak (SHA3-256 and SHAKE-128) implementation only needs 16-bit and 32-bit integer arithmetic. In particular, the error-recovery mechanism described in Section 5 is implemented with fixed-point (i.e., integer-) arithmetic. Furthermore, the C reference implementation does not make use of the division operator (/) and the modulo operator (%). The focus on simplicity and portability does not mean that the implementation is not optimized at all. On the contrary, we use it to illustrate various optimization techniques that are helpful to speed up the key exchange and are also of independent interest for implementers of other ideal-lattice-based schemes.

NTT optimizations. All polynomial coefficients are represented as unsigned 16-bit integers. Our in-place NTT implementation transforms from bit-reversed to natural order using Gentleman-Sande butterfly operations [24, 34]. One would usually expect that each NTT is preceded by a bit-reversal, but all inputs to NTT are noise polynomials that we can simply consider as being already bit-reversed; as explained earlier, the NTT^{-1} operation still involves a bit-reversal. The core of the NTT and NTT^{-1} operation consists of 10 layers of transformations, each consisting of 512 butterfly operations of the form described in Listing 2.

Montgomery arithmetic and lazy reductions. The performance of operations on polynomials is largely determined by the performance of NTT and NTT^{-1}. The main computational bottleneck of those operations are 5120 butterfly operations, each consisting of one addition, one subtraction and one multiplication by a precomputed constant. Those operations are in $\mathbb{Z}_q$; recall that q is a 14-bit prime. To speed up the modular-arithmetic operations, we store all precomputed constants in Montgomery representation [66] with $R = 2^{18}$, i.e., instead of storing ω^i, we store $2^{18}\omega^i \pmod q$. After a multiplication of a coefficient g by some constant $2^{18}\omega^i$, we can then reduce the result r to $g\omega^i \pmod q$ with the fast Montgomery reduction approach. In fact, we do not always fully reduce modulo q, it is sufficient if the result of the reduction has at most 14 bits. The fast Montgomery reduction routine given in Listing 1a computes such a reduction to a 14-bit integer for any unsigned 32-bit integer in $\{0,\ldots,2^{32} - q(R-1) - 1\}$. Note that the specific implementation does not work for *any* 32-bit integer; for example, for the input $2^{32} - q(R-1) = 1073491969$ the addition `a=a+u` causes an overflow and the function returns 0 instead of the correct result 4095. In the following we establish that this is not a problem for our software.

Aside from reductions after multiplication, we also need modular reductions after addition. For this task we use the "short Barrett reduction" [8] detailed in Listing 1b. Again, this routine does not fully reduce modulo q, but reduces any 16-bit unsigned integer to an integer of at most 14 bits which is congruent modulo q.

In the context of the NTT and NTT^{-1}, we make sure that inputs have coefficients of at most 14 bits. This allows us to avoid Barrett reductions after addition on every second level, because coefficients grow by at most one bit per level and the short Barrett reduction can handle 16-bit inputs. Let us turn our focus to the input of the Montgomery reduction (see Listing 2). Before subtracting `a[j+d]` from `t` we need to add a multiple of q to avoid unsigned underflow. Coefficients never grow larger than 15 bits and $3 \cdot q = 36867 > 2^{15}$, so adding $3 \cdot q$ is sufficient. An upper bound on the expression `((uint32_t)t + 3*12289 - a[j+d])` is obtained if `t` is $2^{15}-1$ and `a[j+d]` is zero; we thus obtain $2^{15} + 3 \cdot q = 69634$. All precomputed constants are in $\{0,\ldots,q-1\}$, so the expression `(W * ((uint32_t)t + 3*12289 - a[j+d]))`, the input to the Montgomery reduction, is at most $69634 \cdot (q-1) = 855662592$ and thus safely below the maximum input that the Montgomery reduction can handle.

Listing 1 Reduction routines used in the reference implementation.

(a) Montgomery reduction ($R = 2^{18}$).

```
uint16_t mred(uint32_t a) {
  uint32_t u;
  u = (a * 12287);
  u &= ((1 << 18) - 1);
  a += u * 12289;
  return a >> 18;
}
```

(b) Short Barrett reduction.

```
uint16_t bred(uint16_t a) {
  uint32_t u;
  u = ((uint32_t) a * 5) >> 16;
  a -= u * 12289;
  return a;
}
```

Listing 2 The Gentleman-Sande butterfly inside odd levels of our NTT computation. All `a[j]` and `W` are of type `uint16_t`.

```
  W = omega[jTwiddle++];
  t = a[j];
  a[j] = bred(t + a[j+d]);
  a[j+d] = mred(W * ((uint32_t)t + 3*12289 - a[j+d]));
```

Fast random sampling. As a first step before performing any operations on polynomials, both Alice and Bob need to expand the seed to the polynomial **a** using SHAKE-128. The implementation we use is based on the "simple" implementation by Van Keer for the Keccak permutation and slightly modified code taken from the "TweetFIPS202" implementation [16] for everything else.

The sampling of centered binomial noise polynomials is based on a fast PRG with a random seed from `/dev/urandom` followed by a quick summation of 16-bit chunks of the PRG output. Note that the choice of the PRG is a purely local choice that every user can pick independently based on the target hardware architecture and based on routines that are available anyway (for example, for symmetric encryption following the key exchange). Our C reference implementation uses ChaCha20 [11], which is fast, trivially protected against timing attacks, and is already in use by many TLS clients and servers [53, 54].

7.3 Optimized AVX2 implementation

Intel processors since the "Sandy Bridge" generation support Advanced Vector Extensions (AVX) that operate on vectors of 8 single-precision or 4 double-precision floating-point values in parallel. With the introduction of the "Haswell" generation of CPUs, this support was extended also to 256-bit vectors of integers of various sizes (AVX2). It is not surprising that the enormous computational power of these vector instructions has been used before to implement very high-speed crypto (see, for example, [13, 14, 40]) and also our optimized reference implementation targeting Intel Haswell processors uses those instructions to speed up multiple components of the key exchange.

NTT optimizations. The AVX instruction set has been used before to speed up the computation of lattice-based cryptography, and in particular the number-theoretic transform. Most notably, Güneysu, Oder, Pöppelmann and Schwabe achieve a performance of only 4480 cycles for a dimension-512 NTT on Intel Sandy Bridge [42]. For arithmetic modulo a 23-bit prime, they represent coefficients as double-precision integers.

We experimented with multiple different approaches to speed up the NTT in AVX. For example, we vectorized the Montgomery arithmetic approach of our C reference implementation and also adapted it to a 32-bit-signed-integer approach. In the end it turned out that floating-point arithmetic beats all of those more sophisticated approaches, so we are now using an approach that is very similar to the approach in [42]. One computation of a dimension-1024 NTT takes 10 968 cycles, unlike the numbers in [42] this does include multiplication by the powers of γ and unlike the numbers in [42], this excludes a bit-reversal.

Fast sampling. Intel Haswell processors support the AES-NI instruction set and for the local choice of noise sampling it is obvious to use those. More specifically, we use the public-domain implementation of AES-256 in counter mode written by Dolbeau, which is included in the SUPERCOP benchmarking framework [15]. Transformation from uniform noise to the centered binomial is optimized in AVX2 vector instructions operating on vectors of bytes and 16-bit integers.

For the computation of SHAKE-128 we use the same code as in the C reference implementation. One might expect that architecture-specific optimizations (for example, using AVX instructions) are able to offer significant speedups, but the benchmarks of the eBACS project [15] indicate that on Intel Haswell, the fastest implementation is the "simple" implementation by Van Keer that our C reference implementation is based on. The reasons that vector instructions are not very helpful for speeding up SHAKE (or, more generally, Keccak) are the inherently sequential nature and the 5×5 dimension of the state matrix that makes internal vectorization hard.

Error recovery. The 32-bit integer arithmetic used by the C reference implementation for HelpRec and Rec is trivially 8-way parallelized with AVX2 instructions. With this vectorization, the cost for HelpRec is only 3 440 cycles, the cost for Rec is only 2 816 cycles.

8 Benchmarks and comparison

In the following we present benchmark results of our software. All benchmark results reported in Table 2 were obtained on an Intel Core i7-4770K (Haswell) running at 3491.953 MHz with Turbo Boost and Hyperthreading disabled. We compiled our software with gcc-4.9.2 and flags `-O3 -fomit-frame-pointer -march=corei7-avx -msse2avx`. As described in Section 7, the sampling of $\mathbf{a}$ is not running in constant time; we report the median running time and (in parentheses) the average running time for this generation, the server-side key-pair generation and client-side shared-key computation; both over 1000 runs. For all other routines we report the median of 1000 runs. We built the software from [18] on the same machine as ours and—like the authors of [18]—used `openssl speed` for benchmarking their software and converted the reported results to approximate cycle counts as given in Table 2.

Comparison with BCNS and RSA/ECDH. As previously mentioned, the BCNS implementation [18] also uses the dimension $n = 1024$ but the larger modulus $q = 2^{32} - 1$ and the Gaussian error distribution with Gaussian parameter $\sigma = 8/\sqrt{2\pi} = 3.192$. When the authors of BCNS integrated their implementation into SSL it only incurred a slowdown by a factor of 1.27 compared to ECDH when using ECDSA signatures and a factor of 1.08 when using RSA signatures with respect to the number of connections that could be handled by the server. As a reference, the reported cycle counts in [18] for a `nistp256` ECDH on the client side are 2 160 000 cycles (0.8 ms @2.77 GHz) and on the server side 3 221 288 cycles (1.4 ms @2.33 GHz). These numbers are obviously not state of the art for ECDH software. Even on the `nistp256` curve, which is known to be a far-from-optimal choice, it is possible to achieve cycle counts of less than 300 000 cycles for a variable-basepoint scalar multiplication on an Intel Haswell [41]. Also OpenSSL optionally includes fast software for `nistp256` ECDH by Käsper and Langley and we assume that the authors of [18] omitted enabling it. Compared to BCNS, our C implementation is more than 9 times faster and our AVX implementation even achieves a speedup factor of more than 24. At this performance it is in the same ballpark as state-of-the-art ECDH software, even when TLS switches to faster 128-bit secure ECDH key exchange based on Curve25519 [10], as recently specified in RFC 7748 [55].

In comparison to the BCNS proposal we see a large performance advantage from switching to the binomial error distribution. The BCNS software uses a large precomputed table to sample from a discrete Gaussian distribution with a high precision. This approach takes 1 042 700 cycles to samples one polynomial in constant time. Our C implementation requires only 33 840 cycles to sample from the binomial distribution. Another factor is that we use the NTT in combination with a smaller modulus. Polynomial multiplication in [18] is using Nussbaumer's symbolic approach based on recursive negacyclic convolutions [71]. The implementation in [18] only achieves a performance of 342 800 cycles for a constant-time multiplication. Additionally, the authors of [18] did not perform pre-transformation of constants (e.g., $\mathbf{a}$) or transmission of coefficients in FFT/Nussbaumer representation.

References

[1] Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, J. A., Heninger, N., Springall, D., Thomé, E., Valenta, L., VanderSloot, B., Wustrow, E., Béguelin, S. Z., and Zimmermann, P. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *CCS '15 Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015), ACM, pp. 5–17. https://weakdh.org/. 4, 5

[2] Albrecht, M., Bai, S., and Ducas, L. A subfield lattice attack on overstretched NTRU assumptions. IACR Cryptology ePrint Archive report 2016/127, 2016. http://eprint.iacr.org/2016/127. 8, 9

[3] Albrecht, M. R., Player, R., and Scott, S. On the concrete hardness of learning with errors. IACR Cryptology ePrint Archive report 2015/046, 2015. http://eprint.iacr.org/2015/046/. 8, 9

[4] Arora, S., and Ge, R. New algorithms for learning in presence of errors. In *Automata, Languages and Programming*

Table 2: Intel Haswell cycle counts of our proposal as compared to the BCNS proposal from [18].

	BCNS [18]	Ours (C ref)	Ours (AVX2)
Generation of **a**		57 006[a] (56 808)[a]	57 304[a] (57 145)[a]
NTT		55 868	10 968
NTT^{-1}		60 288[b]	12 128[b]
Sampling of a noise polynomial		33 840[c]	6 244[c]
HelpRec		14 892	3 440
Rec		10 148	2 816
Key generation (server)	≈ 2 477 958	271 650 (272 174)	115 414 (115 746)
Key gen + shared key (client)	≈ 3 995 977	402 058 (402 285)	144 788 (144 957)
Shared key (server)	≈ 481 937	86 584	23 988

[a] Includes reading a seed from `/dev/urandom`
[b] Includes one bit reversal
[c] Excludes reading a seed from `/dev/urandom`, which is shared across multiple calls to the noise generation

(2011), L. Aceto, M. Henzingeri, and J. Sgall, Eds., vol. 6755 of *LNCS*, Springer, pp. 403–415. https://www.cs.duke.edu/~rongge/LPSN.pdf. 8, 17

[5] Babai, L. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica 6*, 1 (1986), 1–13. http://www.csie.nuk.edu.tw/~cychen/Lattices/On%20lovasz%20lattice%20reduction%20and%20the%20nearest%20lattice%20point%20problem.pdf. 4

[6] Bai, S., and Galbraith, S. D. An improved compression technique for signatures based on learning with errors. In *Topics in Cryptology – CT-RSA 2014* (2014), J. Benaloh, Ed., vol. 8366 of *LNCS*, Springer, pp. 28–47. https://eprint.iacr.org/2013/838/. 1

[7] Bai, S., Langlois, A., Lepoint, T., Stehlé, D., and Steinfeld, R. Improved security proofs in lattice-based cryptography: using the Rényi divergence rather than the statistical distance. In *Advances in Cryptology – ASIACRYPT 2015* (2015), T. Iwata and J. H. Cheon, Eds., LNCS, Springer. http://eprint.iacr.org/2015/483/. 17

[8] Barrett, P. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In *Advances in Cryptology – CRYPTO '86* (1987), A. M. Odlyzko, Ed., vol. 263 of *Lecture Notes in Computer Science*, Springer-Verlag Berlin Heidelberg, pp. 311–323. 11

[9] Becker, A., Ducas, L., Gama, N., and Laarhoven, T. New directions in nearest neighbor searching with applications to lattice sieving. In *SODA '16 Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete Algorithms* (2016 (to appear)), SIAM. 8

[10] Bernstein, D. J. Curve25519: new Diffie-Hellman speed records. In *Public Key Cryptography – PKC 2006* (2006), M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds., vol. 3958 of *LNCS*, Springer, pp. 207–228. http://cr.yp.to/papers.html#curve25519. 13

[11] Bernstein, D. J. ChaCha, a variant of Salsa20. In *Workshop Record of SASC 2008: The State of the Art of Stream Ciphers* (2008). http://cr.yp.to/papers.html#chacha. 12

[12] Bernstein, D. J., Chou, T., Chuengsatiansup, C., Hülsing, A., Lange, T., Niederhagen, R., and van Vredendaal, C. How to manipulate curve standards: a white paper for the black hat. IACR Cryptology ePrint Archive report 2014/571, 2014. http://eprint.iacr.org/2014/571/. 4

[13] Bernstein, D. J., Chuengsatiansup, C., Lange, T., and Schwabe, P. Kummer strikes back: new DH speed records. In *Advances in Cryptology – EUROCRYPT 2015* (2014), T. Iwata and P. Sarkar, Eds., vol. 8873 of *LNCS*, Springer, pp. 317–337. full version: http://cryptojedi.org/papers/#kummer. 12

[14] Bernstein, D. J., Hopwood, D., Hülsing, A., Lange, T., Niederhagen, R., Papachristodoulou, L., Schneider, M., Schwabe, P., and Wilcox-O'Hearn, Z. SPHINCS: practical stateless hash-based signatures. In *Advances in Cryptology – EUROCRYPT 2015* (2015), E. Oswald and M. Fischlin, Eds., vol. 9056 of *LNCS*, Springer, pp. 368–397. https://cryptojedi.org/papers/#sphincs. 3, 12

[15] Bernstein, D. J., and Lange, T. eBACS: ECRYPT benchmarking of cryptographic systems. http://bench.cr.yp.to (accessed 2015-10-07). 9, 13

[16] Bernstein, D. J., Schwabe, P., and Assche, G. V. Tweetable FIPS 202, 2015. http://keccak).noekeon.org/tweetfips202.html (accessed 2016-03-21). 12

[17] Bertoni, G., Daemen, J., Peeters, M., and Assche, G. V. Keccak. In *Advances in Cryptology – EUROCRYPT 2013* (2013), T. Johansson and P. Q. Nguyen, Eds., vol. 7881 of *LNCS*, Springer, pp. 313–314. 5

[18] Bos, J. W., Costello, C., Naehrig, M., and Stebila, D. Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In *2015 IEEE Symposium on Security and Privacy* (2015), pp. 553–570. http://eprint.iacr.org/2014/599. 1, 2, 3, 4, 5, 6, 7, 8, 9, 13, 14

[19] Brakerski, Z., Langlois, A., Peikert, C., Regev, O., and Stehlé, D. Classical hardness of learning with errors. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing* (2013), ACM, pp. 575–584. http://arxiv.org/pdf/1306.0281. 5

[20] Cadé, D., Pujol, X., and Stehlé, D. fplll 4.0.4, 2013. https://github.com/dstehle/fplll (accessed 2015-10-13). 8

[21] Checkoway, S., Fredrikson, M., Niederhagen, R., Everspaugh, A., Green, M., Lange, T., Ristenpart, T., Bernstein, D. J., Maskiewicz, J., and Shacham, H. On the practical exploitability of Dual EC in TLS implementations. In *Proceedings of the 23rd USENIX security symposium* (2014). https://projectbullrun.org/dual-ec/index.html. 4

[22] Chen, Y. *Lattice reduction and concrete security of fully homomorphic encryption*. PhD thesis, l'Université Paris Diderot, 2013. Available at http://www.di.ens.fr/~ychen/research/these.pdf. 8, 9

[23] Chen, Y., and Nguyen, P. Q. BKZ 2.0: Better lattice security estimates. In *Advances in Cryptology – ASIACRYPT 2011*, D. H. Lee and X. Wang, Eds., vol. 7073 of *LNCS*. Springer, 2011, pp. 1–20. http://www.iacr.org/archive/asiacrypt2011/70730001/70730001.pdf. 8

[24] Chu, E., and George, A. *Inside the FFT Black Box Serial and Parallel Fast Fourier Transform Algorithms*. CRC Press, Boca Raton, FL, USA, 2000. 11

[25] Conway, J., and Sloane, N. J. A. *Sphere packings, lattices and groups*, 3rd ed., vol. 290 of *Grundlehren der mathematischen Wissenschaften*. Springer Science & Business Media, 1999. 18

[26] Cowie, J., Dodson, B., Elkenbracht-Huizing, R. M., Lenstra, A. K., Montgomery, P. L., and Zayer, J. A world wide number field sieve factoring record: on to 512 bits. In *Advances in Cryptology – ASIACRYPT'96* (1996), K. Kim and T. Matsumoto, Eds., vol. 1163 of *LNCS*, Springer, pp. 382–394. http://oai.cwi.nl/oai/asset/1940/1940A.pdf. 8

[27] de Clercq, R., Roy, S. S., Vercauteren, F., and Verbauwhede, I. Efficient software implementation of ring-LWE encryption. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2015* (2015), EDA Consortium, pp. 339–344. http://eprint.iacr.org/2014/725. 1, 3, 5, 7, 10

[28] Ducas, L., Durmus, A., Lepoint, T., and Lyubashevsky, V. Lattice signatures and bimodal Gaussians. In *Advances in Cryptology – CRYPTO 2013* (2013), R. Canetti and J. A. Garay, Eds., vol. 8042 of *LNCS*, Springer, pp. 40–56. https://eprint.iacr.org/2013/383/. 1, 3

[29] Fluhrer, S. Cryptanalysis of ring-LWE based key exchange with key share reuse. IACR Cryptology ePrint Archive report 2016/085, 2016. http://eprint.iacr.org/2016/085. 4

[30] Fujioka, A., Suzuki, K., Xagawa, K., and Yoneyama, K. Practical and post-quantum authenticated key exchange from one-way secure key encapsulation mechanism. In *Symposium on Information, Computer and Communications Security, ASIA CCS 2013* (2013), K. Chen, Q. Xie, W. Qiu, N. Li, and W. Tzeng, Eds., ACM, pp. 83–94. 3

[31] Galbraith, S. D. Space-efficient variants of cryptosystems based on learning with errors, 2013. https://www.math.auckland.ac.nz/~sgal018/compact-LWE.pdf. 5

[32] Gama, N., Nguyen, P. Q., and Regev, O. Lattice enumeration using extreme pruning. In *Advances in Cryptology – EUROCRYPT 2010* (2010), H. Gilbert, Ed., vol. 6110 of *LNCS*, Springer, pp. 257–278. http://www.iacr.org/archive/eurocrypt2010/66320257/66320257.pdf. 8

[33] Garg, S., Gentry, C., and Halevi, S. Candidate multilinear maps from ideal lattices. In *Advances in Cryptology – EUROCRYPT 2013* (2013), vol. 7881, Springer, pp. 1–17. https://eprint.iacr.org/2012/610. 8

[34] Gentleman, W. M., and Sande, G. Fast Fourier transforms: for fun and profit. In *Fall Joint Computer Conference* (1966), vol. 29 of *AFIPS Proceedings*, pp. 563–578. http://cis.rit.edu/class/simg716/FFT_Fun_Profit.pdf. 11

[35] Gentry, C. Fully homomorphic encryption using ideal lattices. In *STOC '09 Proceedings of the forty-first annual ACM symposium on Theory of computing* (2009), ACM, pp. 169–178. https://www.cs.cmu.edu/~odonnell/hits09/gentry-homomorphic-encryption.pdf. 1

[36] Gentry, C., Peikert, C., and Vaikuntanathan, V. Trapdoors for hard lattices and new cryptographic constructions. In *STOC '08 Proceedings of the fortieth annual ACM symposium on Theory of computing* (2008), ACM, pp. 197–206. https://eprint.iacr.org/2007/432/. 6

[37] Gentry, C., and Szydlo, M. Cryptanalysis of the revised NTRU signature scheme. In *Advances in Cryptology – EUROCRYPT 2002* (2002), XXX, Ed., vol. XXX of *LNCS*, Springer, pp. 299–320. https://www.iacr.org/archive/eurocrypt2002/23320295/nsssign_short3.pdf. 8

[38] Ghosh, S., and Kate, A. Post-quantum secure onion routing (future anonymity in today's budget). IACR Cryptology ePrint Archive report 2015/008, 2015. http://eprint.iacr.org/2015/008. 3

[39] Göttert, N., Feller, T., Schneider, M., Buchmann, J. A., and Huss, S. A. On the design of hardware building blocks for modern lattice-based encryption schemes. In *Cryptographic Hardware and Embedded Systems - CHES 2012* (2012), E. Prouff and P. Schaumont, Eds., vol. 7428 of *LNCS*, Springer, pp. 512–529. http://www.iacr.org/archive/ches2012/74280511/74280511.pdf. 7, 8, 9, 10

[40] Gueron, S. Parallelized hashing via j-lanes and j-pointers tree modes, with applications to SHA-256. IACR Cryptology ePrint Archive report 2014/170, 2014. https://eprint.iacr.org/2014/170. 12

[41] Gueron, S., and Krasnov, V. Fast prime field elliptic-curve cryptography with 256-bit primes. *Journal of Cryptographic Engineering 5*, 2 (2014), 141–151. https://eprint.iacr.org/2013/816/. 13

[42] Güneysu, T., Oder, T., Pöppelmann, T., and Schwabe, P. Software speed records for lattice-based signatures. In *Post-Quantum Cryptography* (2013), P. Gaborit, Ed., vol. 7932 of *LNCS*, Springer, pp. 67–82. http://cryptojedi.org/papers/#lattisigns. 10, 12

[43] Hanrot, G., Pujol, X., and Stehléé, D. Terminating BKZ. IACR Cryptology ePrint Archive report 2011/198, 2011. http://eprint.iacr.org/2011/198/. 8

[44] Hoffstein, J., Pipher, J., Schanck, J. M., Silverman, J. H., and Whyte, W. Practical signatures from the partial Fourier recovery problem. In *Applied Cryptography and Network Security* (2014), I. Boureanu, P. Owesarski, and S. Vaudenay, Eds., vol. 8479 of *LNCS*, Springer, pp. 476–493. https://eprint.iacr.org/2013/757/. 1

[45] Hoffstein, J., Pipher, J., Schanck, J. M., Silverman, J. H., Whyte, W., and Zhang, Z. Choosing parameters for NTRUEncrypt. IACR Cryptology ePrint Archive report 2015/708, 2015. http://eprint.iacr.org/2015/708. 9

[46] Hoffstein, J., Pipher, J., and Silverman, J. H. NTRU: a ring-based public key cryptosystem. In *Algorithmic number theory* (1998), J. P. Buhler, Ed., vol. 1423 of *LNCS*, Springer, pp. 267–288. https://www.securityinnovation.com/uploads/Crypto/ANTS97.ps.gz. 1, 4

[47] Howgrave-Graham, N. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In *Advances in Cryptology-CRYPTO 2007*. Springer, 2007, pp. 150–169. http://www.iacr.org/archive/crypto2007/46220150/46220150.pdf. 9

[48] JINTAI DING, XIANG XIE, X. L. A simple provably secure key exchange scheme based on the learning with errors problem. IACR Cryptology ePrint Archive report 2012/688, 2012. http://eprint.iacr.org/2012/688. 1

[49] KIRCHNER, P., AND FOUQUE, P.-A. An improved BKW algorithm for LWE with applications to cryptography and lattices. In *Advances in Cryptology – CRYPTO 2015* (2015), R. Gennaro and M. Robshaw, Eds., vol. 9215 of *LNCS*, Springer, pp. 43–62. https://eprint.iacr.org/2015/552/. 8, 17

[50] LAARHOVEN, T. *Search problems in cryptography*. PhD thesis, Eindhoven University of Technology, 2015. http://www.thijs.com/docs/phd-final.pdf. 8

[51] LAARHOVEN, T. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In *Advances in Cryptology – CRYPTO 2015* (2015), R. Gennaro and M. Robshaw, Eds., vol. 9216 of *LNCS*, Springer, pp. 3–22. https://eprint.iacr.org/2014/744/. 8

[52] LAARHOVEN, T., MOSCA, M., AND VAN DE POL, J. Finding shortest lattice vectors faster using quantum search. *Designs, Codes and Cryptography 77*, 2 (2015), 375–400. https://eprint.iacr.org/2014/907/. 8

[53] LANGLEY, A. TLS symmetric crypto. Blog post on imperialviolet.org, 2014. https://www.imperialviolet.org/2014/02/27/tlssymmetriccrypto.html (accessed 2015-10-07). 12

[54] LANGLEY, A., AND CHANG, W.-T. ChaCha20 and Poly1305 based cipher suites for TLS: Internet draft. https://tools.ietf.org/html/draft-agl-tls-chacha20poly1305-04 (accessed 2015-02-01). 12

[55] LANGLEY, A., HAMBURG, M., AND TURNER, S. RFC 7748: Elliptic curves for security, 2016. https://www.rfc-editor.org/rfc/rfc7748.txt. 13

[56] LENSTRA, H. W., AND SILVERBERG, A. Revisiting the gentry-szydlo algorithm. In *Advances in Cryptology – CRYPTO 2014*, J. A. Garay and R. Gennaro, Eds., vol. 8616 of *LNCS*. Springer, 2014, pp. 280–296. https://eprint.iacr.org/2014/430. 8

[57] LEYS, J., GHYS, E., AND ALVAREZ, A. Dimensions, 2010. http://www.dimensions-math.org/ (accessed 2015-10-19). 7, 18

[58] LINDNER, R., AND PEIKERT, C. Better key sizes (and attacks) for LWE-based encryption. In *Topics in Cryptology - CT-RSA 2011* (2011), A. Kiayias, Ed., vol. 6558 of *LNCS*, Springer, pp. 319–339. https://eprint.iacr.org/2010/613/. 3

[59] LIU, Z., SEO, H., ROY, S. S., GROSSSCHÄDL, J., KIM, H., AND VERBAUWHEDE, I. Efficient Ring-LWE encryption on 8-bit AVR processors. In *Cryptographic Hardware and Embedded Systems - CHES 2015* (2015), T. Güneysu and H. Handschuh, Eds., vol. 9293 of *LNCS*, Springer, pp. 663–682. https://eprint.iacr.org/2015/410/. 3, 7, 10

[60] LYUBASHEVSKY, V. Lattice signatures without trapdoors. In *Advances in Cryptology – EUROCRYPT 2012* (2012), D. Pointcheval and T. Johansson, Eds., vol. 7237 of *LNCS*, Springer, pp. 738–755. https://eprint.iacr.org/2011/537/. 6

[61] LYUBASHEVSKY, V., PEIKERT, C., AND REGEV, O. On ideal lattices and learning with errors over rings. In *Advances in Cryptology – EUROCRYPT 2010* (2010), H. Gilbert, Ed., vol. 6110 of *LNCS*, Springer, pp. 1–23. http://www.di.ens.fr/~lyubash/papers/ringLWE.pdf. 2

[62] LYUBASHEVSKY, V., PEIKERT, C., AND REGEV, O. On ideal lattices and learning with errors over rings. *Journal of the ACM (JACM) 60*, 6 (2013), 43:1–43:35. http://www.cims.nyu.edu/~regev/papers/ideal-lwe.pdf. 1, 5

[63] MELCHOR, C. A., BARRIER, J., FOUSSE, L., AND KILLIJIAN, M. XPIRe: Private information retrieval for everyone. IACR Cryptology ePrint Archive report 2014/1025, 2014. http://eprint.iacr.org/2014/1025. 10

[64] MICCIANCIO, D., AND VOULGARIS, P. Faster exponential time algorithms for the shortest vector problem. In *SODA '10 Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms* (2010), SIAM, pp. 1468–1480. http://dl.acm.org/citation.cfm?id=1873720. 8

[65] MONTANARO, A. Quantum walk speedup of backtracking algorithms. arXiv preprint arXiv:1509.02374, 2015. http://arxiv.org/pdf/1509.02374v2. 8

[66] MONTGOMERY, P. L. Modular multiplication without trial division. *Mathematics of Computation 44*, 170 (1985), 519–521. http://www.ams.org/journals/mcom/1985-44-170/S0025-5718-1985-0777282-X/S0025-5718-1985-0777282-X.pdf. 11

[67] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. FIPS PUB 202 – SHA-3 standard: Permutation-based hash and extendable-output functions, 2015. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf. 5, 10

[68] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. Workshop on cybersecurity in a post-quantum world, 2015. http://www.nist.gov/itl/csd/ct/post-quantum-crypto-workshop-2015.cfm. 1

[69] NATIONAL SECURITY AGENCY. NSA suite B cryptography. https://www.nsa.gov/ia/programs/suiteb_cryptography/, Updated on August 19, 2015. 1

[70] NGUYEN, P. Q., AND VIDICK, T. Sieve algorithms for the shortest vector problem are practical. *Journal of Mathematical Cryptology 2*, 2 (2008), 181–207. ftp://ftp.di.ens.fr/pub/users/pnguyen/JoMC08.pdf. 8

[71] NUSSBAUMER, H. J. Fast polynomial transform algorithms for digital convolution. *IEEE Transactions on Acoustics, Speech and Signal Processing 28*, 2 (1980), 205–215. 13

[72] PEIKERT, C. Lattice cryptography for the Internet. In *Post-Quantum Cryptography* (2014), M. Mosca, Ed., vol. 8772 of *LNCS*, Springer, pp. 197–219. http://web.eecs.umich.edu/~cpeikert/pubs/suite.pdf. 1, 2, 3, 4, 18, 20

[73] PÖPPELMANN, T., DUCAS, L., AND GÜNEYSU, T. Enhanced lattice-based signatures on reconfigurable hardware. In *Cryptographic Hardware and Embedded Systems – CHES 2014* (2014), L. Batina and M. Robshaw, Eds., vol. 8731 of *LNCS*, Springer, pp. 353–370. https://eprint.iacr.org/2014/254/. 3

[74] PÖPPELMANN, T., AND GÜNEYSU, T. Towards practical lattice-based public-key encryption on reconfigurable hardware. In *Selected Areas in Cryptography – SAC 2013* (2013), T. Lange, K. Lauter, and P. Lisoněk, Eds., vol. 8282 of *LNCS*, Springer, pp. 68–85. https://www.ei.rub.de/media/sh/veroeffentlichungen/2013/08/14/lwe_encrypt.pdf. 1, 6, 7, 10

[75] REGEV, O. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM 56*, 6 (2009), 34. http://www.cims.nyu.edu/~regev/papers/qcrypto.pdf. 5

[76] RÉNYI, A. On measures of entropy and information. In *Fourth Berkeley symposium on mathematical statistics and probability* (1961), vol. 1, pp. 547–561. http://projecteuclid.org/euclid.bsmsp/1200512181. 17

[77] ROY, S. S., VERCAUTEREN, F., MENTENS, N., CHEN, D. D., AND VERBAUWHEDE, I. Compact Ring-LWE cryptoprocessor. In *Cryptographic Hardware and Embedded Systems – CHES 2014* (2014), L. Batina and M. Robshaw, Eds., vol. 8731 of *LNCS*, Springer, pp. 371–391. https://eprint.iacr.org/2013/866/. 3, 5, 10

[78] SCHNORR, C.-P., AND EUCHNER, M. Lattice basis reduction: improved practical algorithms and solving subset sum problems. *Mathematical programming 66*, 1-3 (1994), 181–199. http://www.csie.nuk.edu.tw/~cychen/Lattices/Lattice%20Basis%20Reduction_%20Improved%20Practical%20Algorithms%20and%20Solving%20Subset%20Sum%20Problems.pdf. 8

[79] STEHLÉ, D., AND STEINFELD, R. Making NTRU as secure as worst-case problems over ideal lattices. In *Advances in Cryptology – EUROCRYPT 2011* (2011), K. G. Paterson, Ed., vol. 6632 of *LNCS*, Springer, pp. 27–47. http://www.iacr.org/archive/eurocrypt2011/66320027/66320027.pdf. 1, 4

[80] Tor project: Anonymity online. https://www.torproject.org/. 3

[81] VERSHYNIN, R. Introduction to the non-asymptotic analysis of random matrices. In *Compressed sensing*. Cambridge University Press, 2012, pp. 210–268. http://www-personal.umich.edu/~romanv/papers/non-asymptotic-rmt-plain.pdf. 19

[82] ZHANG, J., ZHANG, Z., DING, J., SNOOK, M., AND DAGDELEN, Ö. Authenticated key exchange from ideal lattices. In *Advances in Cryptology – EUROCRYPT 2015* (2015), E. Oswald and M. Fischlin, Eds., vol. 9057 of *LNCS*, Springer, pp. 719–751. https://eprint.iacr.org/2014/589/. 3

A JarJar

As discussed in Section 6, it is also possible to instantiate our scheme with dimension $n = 512$, modulus $q = 12289$ and noise parameter $k = 24$, to obtain a decent security level against currently known attacks: our analysis leads to a claim of 94 bits of post quantum security. Most of the scheme is similar, except that one should use $\tilde{D}_2$ as reconciliation lattice, and the provable failure bound drops to 2^{-55}. It is quite likely that a more aggressive analysis would lead to a claim of 128 bits of security against the best known attacks.

The performance of almost all computations in NewHope scales linearly in n (except for the NTT, which scales quasi-linear and a small constant overhead, for example, for reading a seed from `/dev/urandom`). Also, noise sampling scales linearly in n and k, so JarJar with $k = 24$ is expected to take $3/4$ of the time for sampling of a noise polynomial compared to NewHope. Message sizes scale linearly in n (except for the constant-size 32-byte seed). The performance of JarJar is thus expected to be about a factor of 2 better in terms of size and slightly less than a factor of 2 in terms of speed. We confirmed this by adapting our implementations of NewHope to the JarJar parameters. The C reference implementation of JarJar takes 167 102 Haswell cycles for the key generation on the server, 234 268 Haswell cycles for the key generation and joint-key computation on the client, and 45 712 Haswell cycles for the joint-key computation on the server. The AVX2 optimized implementation of JarJar takes 71 608 Haswell cycles for the key generation on the server, 84 316 Haswell cycles for the key generation and joint-key computation on the client, and 12 944 Haswell cycles for the joint-key computation on the server. As stated before, we do not recommend to use JarJar, but only provide these numbers as a target for comparison to other more aggressive RLWE cryptosystems in the literature.

B Proof of Theorem 4.1

A simple security reduction to rounded Gaussians. In [7], Bai et al. identify Rényi divergence as a powerful tool to improve or generalize security reductions in lattice-based cryptography. We review the key properties. The Rényi divergence [7, 76] is parametrized by a real $a > 1$, and defined for two distributions P, Q by:

$$R_a(P\|Q) = \left(\sum_{x \in \mathrm{Supp}(P)} \frac{P(x)^a}{Q(x)^{a-1}} \right)^{\frac{1}{a-1}}.$$

It is multiplicative: if P, P' are independents, and Q, Q' are also independents, then $R_a(P \times P' \| Q \times Q') \leq R_a(P\|Q) \cdot R_a(P'\|Q')$. Finally, Rényi divergence relates the probabilities of the same event E under two different distributions P and Q:

$$Q(E) \geq P(E)^{a/(a-1)} / R_a(P\|Q).$$

For our argument, recall that because the final shared key μ is obtained through hashing as $\mu \leftarrow \mathsf{SHA3\text{-}256}(\nu)$ before being used, then, in the random oracle model (ROM), any successful attacker must recover ν exactly. We call this event E. We also define ξ to be the rounded Gaussian distribution of parameter $\sigma = \sqrt{k/2} = \sqrt{8}$, that is the distribution of $\lfloor \sqrt{8} \cdot x \rceil$ where x follows the standard normal distribution.

A simple script (`scripts/Renyi.py`) computes $R_9(\psi_{16}\|\xi) \approx 1.00063$. Yet because $5n = 5120$ samples are used per instance of the protocol, we need to consider the divergence $R_9(P\|Q) = R_9(\psi_{16}, \xi)^{5n} \leq 26$ where $P = \psi_{16}^{5n}$ and $Q = \xi^{5n}$. We conclude as follows.

The choice $a = 9$ is rather arbitrary but seemed a good trade-off between the coefficient $1/R_a(\psi_{16}\|\xi)$ and the exponent $a/(a-1)$. This reduction is provided as a safeguard: switching from Gaussian to binomial distributions can not dramatically decrease the security of the scheme. With practicality in mind, we will simply ignore the loss factor induced by the above reduction, since the best-known attacks against LWE do not exploit the structure of the error distribution, and seem to depend only on the standard deviation of the error (except in extreme cases [4, 49]).

C Details on Reconciliation

In this section, we provide more technical details on the recovery mechanism involving the lattice $\tilde{D}_4$. For reader's convenience, some details are repeated from Section 5.

Splitting for recovery. By $\mathscr{S} = \mathbb{Z}[X]/(X^4+1)$ we denote the 8-th cyclotomic ring, having rank 4 over $\mathbb{Z}$. Recall that our full ring is $\mathscr{R} = \mathbb{Z}[X]/(X^n+1)$ for $n = 1024$. An element of the full ring $\mathscr{R}$ can be identified to a vector $(f'_0, \dots, f'_{255}) \in \mathscr{S}^{n/4}$ such that

$$f(X) = f'_0(X^{256}) + X f'_1(X^{256}) + \cdots + X^{255} f'_{255}(X^{256}).$$

In other words, the coefficients of f'_i are the coefficients $f_i, f_{i+256}, f_{i+512}, f_{i+768}$.

The lattice $\tilde{D}_4$. One may construct the lattice $\tilde{D}_4$ as two shifted copies of $\mathbb{Z}^4$ using a glue vector $\mathbf{g}$:

$$\tilde{D}_4 = \mathbb{Z}^4 \cup \mathbf{g} + \mathbb{Z}^4 \text{ where } \mathbf{g}^t = \left(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right).$$

We recall that $\tilde{D}_4$ provides the densest lattice sphere packing in dimension 4 [25], this suggest it is optimal for error correction purposes. The Voronoi cell $\mathscr{V}$ of $\tilde{D}_4$ is the icositetrachoron [57] (a.k.a. the 24-cell, the convex regular 4-polytope with 24 octahedral cells) and the Voronoi relevant vectors of two types: 8 type-A vectors $(\pm 1,0,0,0), (0,\pm 1,0,0), (0,0,\pm 1,0), (0,0,0,\pm 1)$, and 16 type-B vectors $(\pm\frac{1}{2}, \pm\frac{1}{2}, \pm\frac{1}{2}, \pm\frac{1}{2})$. The natural condition to correct decoding in $\tilde{D}_4$ should therefore be $e \in \mathscr{V}$, and this can be expressed as $\langle e, v\rangle \leq 1/2$ for all Voronoi relevant vectors v. Interestingly, those 24-linear inequalities can be split as $\|e\|_1 \leq 1$ (providing the 16 inequalities for the type-B vectors) and $\|e\|_\infty \leq 1/2$ (providing the 8 inequalities for the type-A vectors). In other words, the 24-cell $\mathscr{V}$ is the intersection of an ℓ_1-ball (an hexadecachoron) and an ℓ_∞-ball (a tessaract).

As our basis for $\tilde{D}_4$, we will choose $\mathbf{B} = (\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2, \mathbf{g})$ where $\mathbf{u}_i$ are the canonical basis vectors of $\mathbb{Z}^4$. The construction of $\tilde{D}_4$ with a glue vector gives a simple and efficient algorithm for finding closest vectors in $\tilde{D}_4$. Note that we have $\mathbf{u}_3 = -\mathbf{u}_0 - \mathbf{u}_1 - \mathbf{u}_2 + 2\mathbf{g} = \mathbf{B}\cdot(-1,-1,-1,2)^t$.

Algorithm 1 (restated) $\mathsf{CVP}_{\tilde{D}_4}(\mathbf{x} \in \mathbb{R}^4)$

Ensure: An integer vector $\mathbf{z}$ such that $\mathbf{Bz}$ is a closest vector to $\mathbf{x}$: $\mathbf{x} - \mathbf{Bz} \in \mathscr{V}$

$\mathbf{v}_0 \leftarrow \lfloor \mathbf{x} \rceil$

$\mathbf{v}_1 \leftarrow \lfloor \mathbf{x} - \mathbf{g} \rceil$

$k \leftarrow (\|\mathbf{x} - \mathbf{v}_0\|_1 < 1)\ ?\ 0 : 1$

$(v_0, v_1, v_2, v_3)^t \leftarrow \mathbf{v}_k$

return $(v_0, v_1, v_2, k)^t + v_3 \cdot (-1,-1,-1,2)^t$

Decoding in $\tilde{D}_4/\mathbb{Z}^4$. Because $\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2$ and $2\mathbf{g}$ belong to $\mathbb{Z}^4$, a vector in $\tilde{D}_4/\mathbb{Z}^4$ is simply given by the parity of its last coordinate in base $\mathbf{B}$. This gives an even simpler algorithm to encode and decode a bit in $\tilde{D}_4/\mathbb{Z}^4$. The encoding is given by $\mathsf{Encode}(k \in \{0,1\}) = k\mathbf{g}$ and the decoding is given below.

Algorithm 2 (restated) $\mathsf{Decode}(\mathbf{x} \in \mathbb{R}^4/\mathbb{Z}^4)$

Ensure: A bit k such that $k\mathbf{g}$ is a closest vector to $\mathbf{x} + \mathbb{Z}^4$: $\mathbf{x} - k\mathbf{g} \in \mathscr{V} + \mathbb{Z}^4$

$\mathbf{v} = \mathbf{x} - \lfloor \mathbf{x} \rceil$

return 0 if $\|\mathbf{v}\|_1 \leq 1$ and 1 otherwise

When we want to decode to $\tilde{D}_4/\mathbb{Z}^4$ rather than to $\tilde{D}_4$, the 8 inequalities given by type-A vectors are irrelevant since those vectors belong to $\mathbb{Z}^4$. It follows that:

Lemma C.1 *For any $k \in \{0,1\}$ and any $\mathbf{e} \in \mathbb{R}^4$ such that $\|\mathbf{e}\|_1 < 1$, we have $\mathsf{Decode}(k\mathbf{g} + \mathbf{e}) = k$.*

C.1 Reconciliation

We define the following r-bit reconciliation function:

$$\mathsf{HelpRec}(\mathbf{x}; b) = \mathsf{CVP}_{\tilde{D}_4}\left(\frac{2^r}{q}(\mathbf{x} + b\mathbf{g})\right) \bmod 2^r,$$

where $b \in \{0,1\}$ is a uniformly chosen random bit. This random vector is equivalent to the "doubling" trick of Peikert [72]. Indeed, because q is odd, it is not possible to map deterministically the uniform distribution from $\mathbb{Z}_q^4$ to $\mathbb{Z}_2$, which necessary results in a final bias.

Lemma C.2 *Assume $r \geq 1$ and $q \geq 9$. For any $\mathbf{x} \in \mathbb{Z}_q^4$, set $\mathbf{r} := \mathsf{HelpRec}(\mathbf{x}) \in \mathbb{Z}_{2^r}^4$. Then, $\frac{1}{q}\mathbf{x} - \frac{1}{2^r}\mathbf{Br} \mod 1$ is close to a point of $\tilde{D}_4/\mathbb{Z}^4$, precisely, for $\alpha = 1/2^r + 2/q$:*

$$\frac{1}{q}\mathbf{x} - \frac{1}{2^r}\mathbf{Br} \in \alpha\mathscr{V} + \mathbb{Z}^4 \quad \text{or} \quad \frac{1}{q}\mathbf{x} - \frac{1}{2^r}\mathbf{Br} \in \mathbf{g} + \alpha\mathscr{V} + \mathbb{Z}^4.$$

Additionally, for $\mathbf{x}$ uniformly chosen in $\mathbb{Z}_q^4$ we have $\mathsf{Decode}(\frac{1}{q}\mathbf{x} - \frac{1}{2^r}\mathbf{Br})$ is uniform in $\{0,1\}$ and independent of $\mathbf{r}$.

Proof One can easily check the correctness of the $\mathsf{CVP}_{\tilde{D}_4}$ algorithm: for any $\mathbf{y}$, $\mathbf{y} - \mathbf{B}\cdot\mathsf{CVP}_{\tilde{D}_4}(\mathbf{y}) \in \mathscr{V}$. To conclude with the first property, it remains to note that $\mathbf{g} \in 2\mathscr{V}$, and that $\mathscr{V}$ is convex.

For the second property, we show that there is a permutation $\pi : (\mathbf{x}, b) \mapsto (\mathbf{x}', b')$ of $\mathbb{Z}_q^4 \times \mathbb{Z}_2$, such that, forall $(\mathbf{x}, b)$ it holds that:

$$\mathsf{HelpRec}(\mathbf{x}; b) = \mathsf{HelpRec}(\mathbf{x}'; b') \quad (= \mathbf{r}) \tag{3}$$

$$\mathsf{Decode}\left(\frac{1}{q}\mathbf{x} - \frac{1}{2^r}\mathbf{Br}\right) = \mathsf{Decode}\left(\frac{1}{q}\mathbf{x}' - \frac{1}{2^r}\mathbf{Br}\right) \oplus 1 \tag{4}$$

where $\oplus$ denotes the xor operation. We construct π as follows: set $b' := b \oplus 1$ and $\mathbf{x}' = \mathbf{x} + (b - b' + q)\mathbf{g} \bmod q$. Note that $b - b' + q$ is always even so $(b - b' + q)\mathbf{g}$ is always well defined in $\mathbb{Z}_q$ (recall that $\mathbf{g} = (1/2, 1/2, 1/2, 1/2)^t$). It follows straightforwardly that $\frac{2^r}{q}(\mathbf{x} + b\mathbf{g}) - \frac{2^r}{q}(\mathbf{x}' + b'\mathbf{g}) = -2^r\mathbf{g} \mod 2^r$. Since $\mathbf{g} \in \tilde{D}_4$, condition (3) holds. For condition (4), notice that:

$$\frac{1}{q}\mathbf{x} - \frac{1}{2^r}\mathbf{Br} \in k\mathbf{g} + \alpha\mathscr{V} \mod 1$$
$$\Longrightarrow \frac{1}{q}\mathbf{x}' - \frac{1}{2^r}\mathbf{Br} \in (k \oplus 1)\mathbf{g} + \alpha'\mathscr{V} \mod 1$$

for $\alpha' = \alpha + 2/q$. Because $r \geq 1$ and $q \geq 9$, we have $\alpha' = 1/2^r + 4/q < 1$, and remembering that $\mathbf{e} \in \mathscr{V} \Rightarrow \|\mathbf{e}\|_1 \leq 1$ (inequalities for type-B vectors), one concludes by Lemma C.1. ∎

It remains to define $\mathsf{Rec}(\mathbf{x}, \mathbf{r}) = \mathsf{Decode}(\frac{1}{q}\mathbf{x} - \frac{1}{2^r}\mathbf{Br})$ to describe a 1-bit-out-of-4-dimensions reconciliation protocol (Protocol 4). Those functions are extended to 256-bits out of 1024-dimensions by the splitting described at the beginning of this section.

Alice		Bob
$\mathbf{x}' \in \mathbb{Z}_q^4$	$\mathbf{x}' \approx \mathbf{x}$	$\mathbf{x} \in \mathbb{Z}_q^4$
	$\xleftarrow{\mathbf{r}}$	$\mathbf{r} \leftarrow \mathsf{HelpRec}(\mathbf{x}) \in \mathbb{Z}_{2^r}^4$
$k' \leftarrow \mathsf{Rec}(\mathbf{x}', \mathbf{r})$		$k \leftarrow \mathsf{Rec}(\mathbf{x}, \mathbf{r})$

Protocol 4: Reconciliation protocol in $q\tilde{D}_4/q\mathbb{Z}^4$.

Lemma C.3 *If $\|\mathbf{x} - \mathbf{x}'\|_1 < (1 - 1/2^r) \cdot q - 2$, then by the above protocol 4 $k = k'$. Additionally, if $\mathbf{x}$ is uniform, then k is uniform independently of $\mathbf{r}$.*

Fixed-point implementation. One remarks that, while we described our algorithm in $\mathbb{R}$ for readability, floating-point-arithmetic is not required in practice. Indeed, all computation can be performed using integer arithmetic modulo $2^r q$. Our parameters ($r = 2$, $q = 12289$) are such that $2^r q < 2^{16}$, which offers a good setting also for small embedded devices.

D Analysis of the failure probability

To proceed with the task of bounding—as tightly as possible—the failure probability, we rely on the notion of moments of a distribution and of subgaussian random variables [81]. We recall that the moment-generating function of a real random variable $\mathscr{X}$ is defined as follows:

$$M_{\mathscr{X}}(t) := \mathbb{E}[\exp(t(\mathscr{X} - \mathbb{E}[\mathscr{X}]))].$$

We extend the definition to distributions over $\mathbb{R}$: $M_\phi := M_{\mathscr{X}}$ where $\mathscr{X} \leftarrow \phi$. Note that $M_\phi(t)$ is not necessary finite for all t, but it is the case if the support of ϕ is bounded. We also recall that if $\mathscr{X}$ and $\mathscr{Y}$ are independent, then the moment-generating functions verify the identity $M_{\mathscr{X}+\mathscr{Y}}(t) = M_{\mathscr{X}}(t) \cdot M_{\mathscr{Y}}(t)$.

Theorem D.1 (Chernoff-Cramer inequality) *Let ϕ be a distribution over $\mathbb{R}$ and let $\mathscr{X}_1 \ldots \mathscr{X}_n$ be i.i.d. random variable of law ϕ, with average μ. Then, for any t such that $M_\phi(t) < \infty$ it holds that*

$$\mathbb{P}\left[\sum_{i=1}^{n} \mathscr{X}_i \geq n\mu + \beta\right] \leq \exp\left(-\beta t + n\ln(M_\phi(t))\right).$$

Definition A centered distribution ϕ over $\mathbb{R}$ is said to be σ-subgaussian if its moment-generating function verifies $\mathbb{E}_{\mathscr{X} \leftarrow \phi}[\exp(t\mathscr{X})] \leq \exp(2t^2\sigma^2)$.

A special case of Chernoff-Cramer bound follows by choosing t appropriately.

Lemma D.2 (Adapted from [81]) *If $\mathbf{x}$ has independently chosen coordinates from ϕ, a σ-subgaussian distribution, then, for any vector $\mathbf{v} \in \mathbb{R}^n$, except with probability less than $\exp(-\tau^2/2)$, we have:*

$$\langle \mathbf{x}, \mathbf{v} \rangle \leq \|\mathbf{v}\|\sigma\tau.$$

The centered binomial distribution ψ_k of parameter k is $\sqrt{k/2}$-subgaussian. This is established from the fact that $b_0 - b'_0$ is $\frac{1}{2}$-subgaussian (which is easy to check), and by Euclidean additivity of k independent subgaussian variables. Therefore, the binomial distribution used (of parameter $k = 16$) is $\sqrt{8}$-subgaussian.

We here propose a rather tight tail-bound on the error term. We recall that the difference $\mathbf{d}$ in the agreed key before key reconciliation is $\mathbf{d} = \mathbf{es}' - \mathbf{e}'\mathbf{s} + \mathbf{e}''$. We wish to bound $\|d'_i\|_1$ for all $i \leq 255$, where the $d'_i \in \mathscr{S}$ form the decomposition of d described in Section 5. Note that, seen as a vector over $\mathbb{R}$, we have

$$\|x\|_1 = \max_y \langle x, y \rangle,$$

where y ranges over $\{\pm 1\}^4$. We also remark that for $\mathbf{a}, \mathbf{b} \in \mathscr{R}$, one may rewrite $(\mathbf{ab})_i \in \mathscr{S}$

$$(\mathbf{ab})_i = \sum_{j=0}^{255} \pm a_j b_{(i-j) \mod 256},$$

where the sign $\pm$ depends only on the indices i, j. This allows to rewrite

$$\|(\mathbf{es}' - \mathbf{e}'\mathbf{s})_i\|_1 = \max_y \sum_{j=0}^{255} \pm \left(\langle e_i, s_{i-j}y \rangle + \langle e'_i, s'_{i-j}y \rangle\right) \tag{5}$$

where $y \in \mathscr{S}$ ranges over all polynomials with ± 1 coefficients.

Lemma D.3 *Let $s, s' \in \mathscr{S}$ be drawn with independent coefficients from ψ_{16}, then, except with probability 2^{-64}, the vectors $\mathbf{v}_y = (s_0 y, \ldots, s_{255} y, s'_0 y, \ldots, s'_{255} y) \in \mathbb{Z}^{2048}$ verify simultaneously $\|\mathbf{v}_y\|_2^2 \leq 102500$ For all $y \in \mathscr{S}$ with ± 1 coefficients.*

Proof Let us first remark that $\|\mathbf{v}\|_2^2 = \sum_{i=0}^{512} \|s_i y\|_2^2$ where the s_i'ss are i.i.d. random variables following distribution ψ_{16}^4. Because the support of ψ_{16}^4 is reasonably small (of size $(216+1)^4 \approx 2^{20}$), we numerically compute the distribution $\varphi_y : \|sy\|_2^2$ where $s \leftarrow \psi_{16}^4$ for each y (see `scripts/failure.py`). Note that such numerical computation of the probability density function does not raise numerical stability concern, because it involves a depth-2 circuit of multiplication and addition of *positive* reals: the relative error growth remain at most quadratic in the length of the computation.

From there, one may compute $\mu = \mathbb{E}_{\mathscr{X} \leftarrow \varphi_y}[\mathscr{X}]$ and $M_{\varphi_y}(t)$ (similarly this computation has polynomial relative error growth assuming exp is computed with constant relative error growth).

We apply Chernoff-Cramer inequality with parameters $n = 512$, $n\mu + \beta = 102500$ and $t = 0.0055$, and obtain the desired inequality for each given y, except with probability at most $2^{-68.56}$. We conclude by union-bound over the 2^4 choices of y. ∎

Corollary D.4 *For $\mathbf{s}\mathbf{e}, \mathbf{e}', \mathbf{e}'', \mathbf{s}, \mathbf{s}' \in \mathscr{R}$ drawn with independent coefficients according to ψ_{16}, except with probability at most 2^{-61} we have simultaneously for all $i \leq 255$ that*

$$\|(\mathbf{e}'\mathbf{s} + \mathbf{e}\mathbf{s}' + \mathbf{e}'')_i\|_1 \leq 9210 < \lfloor 3q/4 \rfloor - 2 = 9214.$$

Proof First, we know that $\|(\mathbf{e}'')_i\|_1 \leq 4 \cdot 16 = 64$ for all i's, because the support of ψ is $[-16, 16]$. Now, for each i, write

$$\|(\mathbf{e}'\mathbf{s} + \mathbf{e}\mathbf{s}')_i\|_1 = \max_y \langle \mathbf{v}_{i,y}, \mathbf{e}^+ \rangle$$

according to Equation 5, where $\mathbf{v}_y$ depends on $\mathbf{s}, \mathbf{s}'$, and $\mathbf{e}^+$ is a permutation of the 2048 coefficients of $\mathbf{e}$ and $\mathbf{e}'$, each of them drawn independently from ψ_{16}. By Lemma D.3 $\|\mathbf{v}_{0,y}\|^2 \leq 102500$ for all y with probability less that 2^{-64}. Because $\mathbf{v}_{i,y}$ is equal to $\mathbf{v}_{0,y}$ up to a signed permutation of the coefficient, we have $\|\mathbf{v}_{i,y}\| \leq 102500$ for all i, y.

Now, for each i, y, we apply the tail bound of Lemma D.2 with $\tau = 10.1$ knowing that ψ_{16} is σ-subgaussian for $\sigma = \sqrt{16/2}$, and we obtain, except with probability at most $2 \cdot 2^{-73.5}$ that

$$|\langle \mathbf{v}_{i,y}, \mathbf{e} \rangle| \leq 9146.$$

By union bound, we conclude that the claimed inequality holds except with probability less than $2^{-64} + 2 \cdot 8 \cdot 256 \cdot 2^{-73} \leq 2^{-61}$. ∎

E A small example of edge blurring

We illustrate what it means to "blur the edges" in the reconciliation mechanism (cf. Section 5). This is an adaptation of the 1-dimensional randomized doubling trick of Peikert [72]. Formally, the proof that this process results in an unbiased key agreement is provided in Lemma C.2.

Consider the two-dimensional example with $q = 9$ depicted in Figure 3. All possible values of a vector are depicted with red or black dots. All red dots (in the grey Voronoi cell) are mapped to 1 in the key bit; the black dots are mapped to a zero. There is a total of $9 \cdot 9 = 81$ dots, 40 red ones, and 41 black ones. This obviously leads to a biased key. To address this we use one random bit to determine whether we add the vector $(\frac{1}{2q}, \frac{1}{2q})$ before running error reconciliation. For most dots this makes no difference; they stay in the same Voronoi cell and the "directed noise" is negligible for larger values of q. However, some dots right at the border are moved to another Voronoi cell with probability $\frac{1}{2}$ (depending on the value of the random bit). In the example in Figure 3 we see that 72 dots (36 red and 36 black ones) remain in their Voronoi cell; the other 9 dots (4 red and 5 black ones) change Voronoi cell with probability $\frac{1}{2}$ which precisely eliminates the bias in the key.

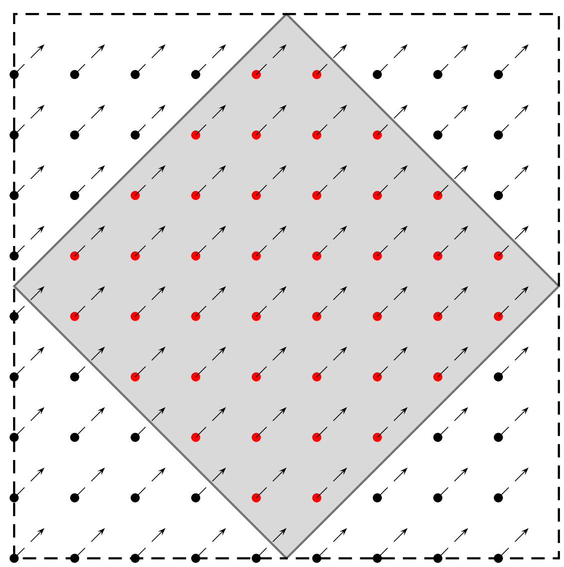

Figure 3: Possible values of a vector in $\mathbb{Z}_9 \times \mathbb{Z}_9$, their mapping to a zero or one bit, and the effect of blurring.