Post-quantum key exchange – a new hope

Erdem Alkim[1], Léo Ducas[2], Thomas Pöppelmann[3], and Peter Schwabe[4] ⋆

[1] Department of Mathemathics, Ege University, Turkey
erdemalkim@gmail.com
[2] Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands
leo.ducas@cwi.nl
[3] Infineon Technologies AG, Munich, Germany
thomas.poeppelmann@infineon.com
[4] Digital Security Group, Radboud University, The Netherlands
peter@cryptojedi.org

Abstract. Earlier in 2015, Bos, Costello, Naehrig, and Stebila (IEEE Security & Privacy 2015) proposed an instantiation of Peikert's ring-learning-with-errors (Ring-LWE) based 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 8 in a portable C implementation and by more than a factor of 20 in an optimized implementation targeting current Intel CPUs. These speedups are achieved with comprehensive protection against timing attacks.

Keywords. Post-quantum key exchange, Ring-LWE, high-speed software, vectorization.

1 Introduction

The last decade in cryptography has seen the birth of numerous constructions of cryptosystems based on lattice problems, achieving functionalities that were

⋆ 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: 2015-11-10

previously unreachable (e.g., fully homomorphic cryptography [38]). 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* [2, 67], 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 in an open process.

For encryption and signatures, several competitive schemes have been proposed; examples are NTRU encryption [48, 77], Ring-LWE encryption [61] as well as the signature schemes BLISS [31], PASS [47] 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 [70], and 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[5], which has also made various other implementations [29, 72] slower and more complex than they would have to be.

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 [70]. 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 pre-supposed) 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.

[5] 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.

– 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 **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 in the public domain. It is available online at https://cryptojedi.org/crypto/#newhope and https://github.com/tpoeppelmann/newhope.

Organization of this paper. Section 2 first describes previous proposals for lattice-based key exchange and then introduces our proposal. Sections 3–6 discuss security properties of our proposal; Section 7 gives details about the message encoding and implementation techniques; and finally Section 8 presents performance results for our implementation.

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 $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ as the ring of integer polynomials modulo X^n+1. By $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n+1)$ 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 $\mathcal{R}$, then $x \xleftarrow{\$} \chi$ means the sampling of $x \in \mathcal{R}$ according to χ. When we write $\mathbf{a} \xleftarrow{\$} \mathcal{R}_q$ this means that all coefficients of **a** are chosen uniformly random from $\mathbb{Z}_q$. For a probabilistic algorithm $\mathcal{A}$ we denote by $y \xleftarrow{\$} \mathcal{A}$ that the output of $\mathcal{A}$ is assigned to y and that $\mathcal{A}$ 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 [70] 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 [60] but due to a new error-reconciliation mechanism, one $\mathcal{R}_q$ component of the ciphertext can be replaced by a more compact element in $\mathcal{R}_2$. This efficiency gain is possible due to the observation that its 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 $\mathbf{us} = \mathbf{ass}' + \mathbf{e}'\mathbf{s}$ 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 [70] 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 [70] was phrased as a Diffie-Hellman related 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 [57] with distinguishing advantage less than 2^{-128} and $2^{81.9}$ bits 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 [78] on top of existing elliptic-curve infrastructure [41].

2.3 Our proposal

In this section we detail our proposal and modifications of Peikert's protocol. 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 [33, 80]) 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, 70] all polynomials except for $\mathbf{r} \in \mathbb{Z}_4^n$ 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[6]. As polynomial arithmetic is fast and also scales better (doubling n roughly

[6] An acceptable security level could probably also be achieved with $n = 512$ given today's state of the art in cryptanalysis (see `scripts/PQsec.py`) . However, in order to be able to claim long term security properties we opted for a more conservative choice, anticipating further advances in cryptanalysis and cryptanalytic hardware or software.

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 [29, 58, 75] and BLISS signatures [31, 71]; thus sharing of some code (or hardware modules) between our proposal and an implementation of BLISS would be possible.

Changed noise distribution and improved 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 $k = 12$ (see Section 3). 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 4). 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 $k = 12$ this gives a standard deviation of $\varsigma = \sqrt{12/2}$. Contrary to [18, 70] 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 4). 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. With an error probability provably less than 2^{-110} this event is still negligible but not so unlikely that we have weakened our scheme. An additional benefit of this method over approaches like digital error-correcting codes is that it is very simple and can be implemented in constant time using only integer arithmetic—which is important on constrained devices without a floating-point unit.

Short-term public parameters and caching. At last, we do 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 6) in the flavor of the "Logjam" DLP attack [1].

If efficiency is a concern, the public parameter $\mathbf{a}$ chosen by the server may still be cached for some time (say two hours) on the server side without substantially affecting the security of our scheme. The saving would be quite significant,

especially in our AVX2 implementation where the generation of $\mathbf{a}$ costs roughly 44% of the cycles on the server side.

However, the secrets must never be cached. Note that 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 [22], Microsoft's SChannel library caches ephemeral keys for 2 hours. We remark that for the lattice-based key exchange described in [70], 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.

Parameters: $q = 12289 < 2^{14},\ n = 1024$ Error distribution: ψ_{12}		
Alice (server)		**Bob (client)**
$seed \xleftarrow{\$} \{0,1\}^{256}$		
$\mathbf{a} \leftarrow \mathsf{Parse}(\mathsf{SHAKE\text{-}128}(seed))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \psi_{12}^n$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \psi_{12}^n$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, seed)}$	$\mathbf{a} \leftarrow \mathsf{Parse}(\mathsf{SHAKE\text{-}128}(seed))$
		$\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{bs}' + \mathbf{e}''$
$\mathbf{v}' \leftarrow \mathbf{us}$	$\xleftarrow{(\mathbf{u}, \mathbf{r})}$	$\mathbf{r} \xleftarrow{\$} \mathsf{HelpRec}(\mathbf{v})$
$\nu \leftarrow \mathsf{Rec}(\mathbf{v}', \mathbf{r})$		$\nu \leftarrow \mathsf{Rec}(\mathbf{v}, \mathbf{r})$
$\mu \leftarrow \mathsf{SHA3\text{-}256}(\nu)$		$\mu \leftarrow \mathsf{SHA3\text{-}256}(\nu)$

Protocol 2: Our Scheme. For the definitions of HelpRec and Rec see Section 4. For the definition of encodings and the definition of Parse see Section 7.

3 Choice of the error distribution

On non-Gaussian errors. In works like [18, 29, 75], 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.

Indeed, we recall that the original worst-case to average-case reductions for LWE [73] and Ring-LWE [61] 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 crucial for signatures [59] and lattice trapdoors [39], to provide zero-knowledgeness. As for LWE, we do not know any attacks exploiting non-Gaussianity, and several works have established its hardness for other distributions, like uniform in a small interval [30, 63], but such proofs do not carry over to Ring-LWE. Below, we will provide a simple ad-hoc reduction for our application, based on Rényi divergence, similarly to [7].

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, 74] 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 $\sigma = \sqrt{k/2} = \sqrt{6}$, that is the distribution of $\lfloor \sqrt{6} \cdot x \rceil$ where x follows the standard normal distribution.

As idealized protocol we denote Protocol 2 where the distribution ψ_{12} is replaced by ξ.

A simple script (`scripts/Renyi.py`) computes $R_9(\psi_{12}\|\xi) \approx 1.00103$. Yet because $5n = 5120$ samples are used per instance of the protocol, we need to consider the divergence $R_9(P\|Q) = R_9(\psi_{12}, \xi)^{5n} \leq 195$ where $P = \psi_{12}^{5n}$ and $Q = \xi^{5n}$. We conclude as follows.

Lemma 1. *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 the idealized protocol with probability at least*

$$q \geq p^{9/8}/195.$$

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]).

Simple implementation. We remark that sampling from the centered binomial distribution ψ_{12} 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 (c.f. to the issues caused by the table-based approach used in [18]).

4 Improved error-recovery mechanism

The algorithms described in this section are provided in a python script for testing purposes (`scripts/Rec.py`), in addition to the fast C and AVX implementations.

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

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

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

4.1 Low-density encoding

In most of the literature, Ring-LWE encryption allows to encrypt 1 bit per coordinate of the ciphertext. It is also well known how to increase the encryption space allowing more than 1 bit per coordinate by having a larger modulus-to-error ratio (and therefore decreasing the security for a fixed dimension n). Nevertheless, when it comes to exchanging a symmetric key (of, say, 256 bits), we end up having a message space larger than necessary.

In [72] 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 probability of decryption failures. Below we propose a generalization of this technique in dimension 4 together with a geometric interpretation and a rigorous analysis.

The lattice D_4. One may construct the lattice D_4 as two shifted copies of $\mathbb{Z}^4$ using a glue vector $\mathbf{g}$:

$$D_4 = \mathbb{Z}^4 \cup \mathbf{g} + \mathbb{Z}^4 \text{ where } \mathbf{g}^t = \left(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right).$$

We recall that D_4 provides the densest lattice sphere packing in dimension 4 [26]. The Voronoi cell $\mathcal{V}$ of D_4 is the icositetrachoron [56] (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 D_4 should therefore be $e \in \mathcal{V}$, 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 $\mathcal{V}$ is the intersection of an ℓ_1-ball (an hexadecachoron) and an ℓ_∞-ball (a tessaract).

As our basis for 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 D_4 with a glue vector gives a simple and efficient algorithm for finding closest vectors in 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 $\mathsf{CVP}_{D_4}(\mathbf{x} \in \mathbb{R}^4)$

Ensure: An integer vector $\mathbf{y}$ such that $\mathbf{By}$ is the closest vector to $\mathbf{x}$
1: $\mathbf{v}_0 \leftarrow \lfloor \mathbf{x} \rceil$
2: $\mathbf{v}_1 \leftarrow \lfloor \mathbf{x} - \mathbf{g} \rceil$
3: $k \leftarrow (\|\mathbf{x} - \mathbf{v}_0\|_1 < 1)?1 : 0$
4: $(v_0, v_1, v_2, v_3)^t \leftarrow \mathbf{v}_k$
5: **return** $(v_0, v_1, v_2, k)^t + v_3 \cdot (-1, -1, -1, 2)^t$

Decoding in $D_4/\mathbb{Z}^4$. Because $\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2$ and $2\mathbf{g}$ belong to $\mathbb{Z}^4$, a vector in $D_4/\mathbb{Z}^4$ 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 $D_4/\mathbb{Z}^4$. The encoding is given by $\mathsf{LDEncode}(k \in \{0, 1\}) = k\mathbf{g}$ and the decoding is given below.

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

Ensure: An integer vector $\mathbf{y}$ such that $\mathbf{By}$ is the closest vector to $\mathbf{x}$
1: $\mathbf{v} = \mathbf{x}/q - \lfloor \mathbf{x}/q \rceil$
2: **return** 1 if $\|\mathbf{v}\|_1 \leq 1$ and 0 otherwise

When we want to decode to $D_4/\mathbb{Z}^4$ rather than to D_4, the 8 inequalities given by type-A vectors are irrelevant since those vectors belong to $\mathbb{Z}^4$. It follows that:

Lemma 2. *For any $k \in \{0, 1\}$ and any $\mathbf{e} \in \mathbb{R}^4$ such that $\|\mathbf{e}\|_1 < 1$, we have $\mathsf{CVP}_{D_4/\mathbb{Z}^4}(k\mathbf{g} + \mathbf{e}) = k$.*

4.2 Reconciliation

We define the following r-bit reconciliation function:

$$\mathsf{HelpRec}(\mathbf{x}) = \mathsf{CVP}_{D_4}\left(2^r\left(\frac{1}{q}\mathbf{x} + b\mathbf{g}\right)\right) - b\cdot(0,0,0,2^{r-1}) \bmod 2^r,$$

where b is a uniformly random chosen bit. This random bit is equivalent to the "doubling" trick of Peikert [70].

Lemma 3. *For any $\mathbf{x} \in \mathbb{Z}_q^4$, set $\mathbf{r} = \mathsf{HelpRec}(\mathbf{x}) \in \mathbb{Z}_{2^r}^4$. Then, $\mathbf{x} - \frac{q}{2^r}\mathbf{Br}$ is close to a point of $qD_4/q\mathbb{Z}^4$, precisely, for $\alpha = q/2^r$:*

$$\mathbf{x} - \alpha\mathbf{Br} \in \alpha\mathcal{V} \cup q\mathbf{g} + \alpha\mathcal{V} \mod q.$$

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

It remains to define $\mathsf{Rec}(\mathbf{x},\mathbf{r}) = \mathsf{LDDecode}(\mathbf{x} - \frac{q}{2^r}\mathbf{Br})$ to describe a 1-bit-out-of-4-dimensions reconciliation protocol (Protocol 3). 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 3: Reconciliation protocol in $qD_4/q\mathbb{Z}^4$.

Lemma 4. *If $\|\mathbf{x} - \mathbf{x}'\|_1 < (1 - 1/2^r)\cdot q$, then in the above protocol $k = k'$.*

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.

4.3 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 [79]. We recall that the moment-generating function of a real random variable $\mathcal{X}$ is defined as follows:

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

We extend the definition to distributions over $\mathbb{R}$: $M_\phi := M_\mathcal{X}$ where $\mathcal{X} \leftarrow \phi$. 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 $\mathcal{X}$ and $\mathcal{Y}$ are independent, then the moment-generating functions verify the identity $M_{\mathcal{X}+\mathcal{Y}}(t) = M_\mathcal{X}(t) \cdot M_\mathcal{Y}(t)$.

Theorem 1 (Chernoff-Cramer inequality). *Let ϕ be a distribution over $\mathbb{R}$ and let $\mathcal{X}_1 \dots \mathcal{X}_n$ 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*

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

Definition 1. *A centered distribution ϕ over $\mathbb{R}$ is said to be σ-subgaussian if its moment-generating function verifies $\mathbb{E}_{\mathcal{X}\leftarrow\phi}[\exp(t\mathcal{X})] \leq \exp(2t^2\sigma^2)$.*

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

Lemma 5 (Adapted from [79]). *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 $k = 12$) is $\sqrt{6}$-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 $d'_i \in \mathcal{S}$ form the decomposition of d described in Section 4.1. 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 $\mathbf{a}, \mathbf{b} \in \mathcal{R}$, one may rewrite $(\mathbf{ab})_i \in \mathcal{S}$

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

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

$$\|(\mathbf{es}' - \mathbf{e}'\mathbf{s})_i\|_1 = \max_y \sum_{j=0}^{255} \pm \left(\langle e_i, s_{i-j} y \rangle + \langle e'_i, s'_{i-j} y \rangle\right) \tag{1}$$

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

Lemma 6. *For each $y \in \mathcal{S}$ with ± 1 coefficients, and if $s, s' \in \mathcal{S}$ are drawn with independent coefficients from ψ_{12}, then, except with probability 2^{-111}, the vector $\mathbf{v} = (s_0 y, \dots, s_{255} y, s'_0 y, \dots, s'_{255} y) \in \mathbb{Z}^{2048}$ verifies $\|\mathbf{v}\|_2^2 \leq 77000$.*

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_{12}^4. Because the support of ψ_{12}^4 is reasonably small (of size $23^4 \approx 2^{18}$), we numerically compute the distribution $\varphi_y : \|sy\|_2^2$ where $s \leftarrow \psi_{12}^4$ for each y (see `scripts/Lem6Cor1.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 $\mu = \mathbb{E}_{\mathcal{X} \leftarrow \varphi_y}[\mathcal{X}] (= 96)$ 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 $n = 512$, $n\mu + \beta = 77000$ and $t = 0.0055$, and obtain the desired inequality for each given y, except with probability at most $2^{-115.06}$. We conclude by union-bound over the 2^4 choices of y.

Corollary 1. *For $\mathbf{se}, \mathbf{e}', \mathbf{e}'', \mathbf{s}, \mathbf{s}' \in \mathcal{R}$ drawn with independent coefficients according to ψ_{12}, except with probability at most 2^{-110} we have simultaneously for all $i \leq 255$ that*

$$\|(\mathbf{e}'\mathbf{s} + \mathbf{e}\mathbf{s}' + \mathbf{e}'')_i\|_1 \leq 9021 < \lfloor 3q/4 \rfloor = 9216.$$

Proof. First, we know that $\|(\mathbf{e}'')_i\|_1 \leq 4 \cdot 12 = 48$ for all i's, because the support of ψ is $[-12, 12]$. 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 1, 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_8. By Lemma 6 $\|\mathbf{v}_{0,y}\|^2 \leq 77000$ for all y with probability less that 2^{-111}. Because $\mathbf{v}_{i,y}$ is equal to $\mathbf{v}_{0,y}$ up to a signed permutation of the coefficient, we have $\|\mathbf{v}_{i,y}\| \leq 77000$ for all i, y.

Now, for each i, y, we apply the tail bound of Lemma 5 with $\tau = 13.2$ knowing that ψ_{12} is σ-subgaussian for $\sigma = \sqrt{12/2}$, and we obtain, except with probability at most $2 \cdot 2^{-125.6}$ that

$$|\langle \mathbf{v}_{i,y}, \mathbf{e} \rangle| \leq 8973.$$

By union bound, we conclude that the claimed inequality holds except with probability less than $2^{-111} + 2 \cdot 8 \cdot 256 \cdot 2^{-125} \leq 2^{-110}$.

5 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 [29, 42, 58, 72], 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.

Better safe than sorry. With all our improvements, it would be possible to build a scheme with $n = 512$ (and $k = 8$, $q = 12289$) and to obtain security similar to the one of [18], and therefore further improve efficiency. Nevertheless, as history showed us with RSA-512 [27], the standardization and deployment of a scheme awakens further cryptanalytic effort. In particular, our parameter set could withstand a dimension-halving attack in the line of [36, Sec 8.8.1] based on the Gentry-Szydlo algorithm [40, 55]. Note that so far, such attacks are only known for principal ideal lattices and there are serious obstructions to extend them to Ring-LWE, but such precaution seems reasonable for long-term security.

If efficiency dictates to downgrade the scheme to $n = 512$, we strongly advise that the option $n = 1024$ should also be included, easing a potential transition.

5.1 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 [24, 76] attacks, usually referred to as primal and dual attacks that we will briefly remind below.

The algorithm BKZ proceeds by reducing a lattice basis using an SVP oracle in a smaller dimension b. It is known [46] 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. 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).

5.2 Enumeration versus quantum sieve

Typical implementations [20, 24, 35] 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 $n \approx 130$. 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, 66], improving the heuristic complexity from

$(4/3)^{n+o(n)} \approx 2^{0.415n}$ down to $\sqrt{3/2}^{n+o(n)} \approx 2^{0.292n}$ (see [9, 50]). 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 algorithm have been shown [51] to benefit from Grover's quantum search algorithm, bringing the complexity down to $2^{0.268n}$. It is unclear if further improvements are to be expected. Yet because all those algorithm require to classically build lists of size $\sqrt{4/3}^{n+o(n)} \approx 2^{0.2075n}$, it is very plausible that the best quantum SVP algorithm would run in time greater than $2^{0.2075n}$.

Irrelevance of enumeration for our analysis. In [24], 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. While it is unclear if enumeration is easy to accelerate using quantum algorithms, a nested Grover's search approach [21] would at best square root those running times, therefore going from 2^{20} to 2^{51} as the dimension increases from 100 to 190.

On the other hand, our best-known attack bound $2^{0.268n}$ gives a cost of 2^{51} in dimension 190, and the best plausible attack bound $2^{0.2075n} \approx 2^{39}$. 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.

5.3 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 n in our case, but we numerically verified that the choice $m = n$ was optimal.

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, 23]. 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

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

5.4 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 $(\mathbf{x}, \mathbf{y})$

Quantum core hardness of the BCNS proposal [18]: small-secret LWE with modulus $q \approx 2^{32}$, dimension $n = 1024$, error standard deviation $\varsigma = 3.192$, and $m = n$ samples.

Attack	BKZ block dim. b	$\log_2$(BKC)	$\log_2$(BPC)
Primal	294	79	62
Dual ($\epsilon = 2^{-128}$)	228	62	48
Dual ($\epsilon = 1/2$)	330	89	69

Quantum core hardness of our proposal: small-secret LWE with modulus $q = 12289$, dimension $n = 1024$, error standard deviation $\varsigma = \sqrt{6}$ and $m = n$ samples.

Attack	BKZ block dim. b	$\log_2$(BKC)	$\log_2$(BPC)
Primal	934	251	194
Dual ($\epsilon = 2^{-128}$)	691	186	144
Dual ($\epsilon = 1/2$)	1498	402	311

Table 1: Core hardness of the proposal in [18] and of our proposal. BKC is the *Best Known (Quantum) Cost* for solving SVP in dimension b, which is $2^{0.268b}$ (see [51]). BPC is the *Best Plausible (Quantum) Cost*, that is $2^{0.2075b}$ as argued above. 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.

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 distribution have statistical distance about $\epsilon = \exp(-\pi\tau^2)$ 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

$$-\pi\tau^2 \geq \ln \epsilon. \tag{3}$$

In the context of key exchange, it is unclear that we want to consider this attack for very small advantages like $\epsilon = 2^{-128}$. Indeed, only a distinguisher with large advantage (say $\epsilon = 1/2$) significantly decreases the key space. We will therefore give the cost for both those extreme cases.

5.5 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 secu-

rity in our proposal. Yet, because of the numerous pessimistic assumption made in our analysis, we do not claim any quantum attacks reaching those bounds.

6 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 [48, 77], a dishonest authority may choose mildly small $\mathbf{f}, \mathbf{g}$ such that $\mathbf{f} = \mathbf{g} = 1 \bmod p$ for some prime $p \geq 33$ and set $\mathbf{a} = \mathbf{g}\mathbf{f}^{-1} \bmod q$. Then, given $(\mathbf{a}, \mathbf{b} = \mathbf{a}\mathbf{s} + \mathbf{e})$, the attacker can compute $\mathbf{b}\mathbf{f} = \mathbf{a}\mathbf{f}\mathbf{s} + \mathbf{f}\mathbf{e} = \mathbf{g}\mathbf{s} + \mathbf{f}\mathbf{e} \mod q$, and, because $\mathbf{g}, \mathbf{s}, \mathbf{f}, \mathbf{e}$ are small enough, compute $\mathbf{g}\mathbf{s} + \mathbf{f}\mathbf{e}$ 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 8, their sums are in $[-16, 16]$: knowing them modulo $p \geq 33$ is knowing them in $\mathbb{Z}$. It now only remains to compute $(\mathbf{b} - \mathbf{t}) \cdot (\mathbf{a} - 1)^{-1} = (\mathbf{a}\mathbf{s} - \mathbf{s}) \cdot (\mathbf{a} - 1)^{-1} = \mathbf{s} \bmod q$ 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] or at least lower the trust into the generated parameters, and, whenever possible should be avoided.

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 $\Lambda = \{(a, 1)x + (q, 0)y | x, y \in \mathcal{R}\}$, 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 attack 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 generating a fresh $\mathbf{a}$ at each instance of the protocol (as we propose). As mentioned in Section 2.3, if it in practice turns out to be too expensive to generate $\mathbf{a}$ for every connection, it is also possible to cache $\mathbf{a}$ on the server side[7] for, say a few hours without significantly weakening the

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

protection against all-for-the-price-of-one attacks. Additionally, the performance impact of generating $\mathbf{a}$ is reduced by sampling $\mathbf{a}$ uniformly directly in NTT format (recalling that the NTT is one-to-one map), and by transferring only a short 256-bit seed for $\mathbf{a}$ (see Section 7).

A subtle question is to choose an appropriate primitive to generate a "random-looking" polynomial $\mathbf{a}$ 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 [34] that such 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 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 [69], 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 $\mathbf{a}$, 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.

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 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 $\mathcal{R}_q$). *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 [29, 42, 58, 72]. For some background on the NTT and the description of fast software implementations we refer

to [45, 62]. 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 $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathcal{R}$. An NTT targeting ideal lattices defined in $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$ 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 $\gamma = \sqrt{\omega}$ 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.

We define $\mathsf{NTT}(\mathbf{g})$ for $\mathbf{g} \in \mathcal{R}_q$ with $\omega = 49$ and $\gamma = 7$ (note that $\gamma^2 = \omega$). Let rev_{10} be the function that reverses the bits in the binary representation of a 10-bit integer. Let

$$\begin{aligned}\overline{\mathbf{g}} = (\overline{g}_0, \ldots, \overline{g}_{1023}) &= (g_{\mathsf{rev}_{10}(0)}, \gamma g_{\mathsf{rev}_{10}(1)}, \gamma^2 g_{\mathsf{rev}_{10}(2)}, \ldots, \gamma^{1023} g_{\mathsf{rev}_{10}(1023)}) \\ &= (g_0, \gamma g_{512}, \gamma^2 g_{256}, \ldots, \gamma^{1023} g_{1023}).\end{aligned}$$

Then

$$\mathsf{NTT}(\mathbf{g}) = \sum_{i=0}^{1023} G_i X^i, \text{ with}$$
$$G_i = \sum_{j=0}^{1023} \overline{g}_j \omega^{ij}.$$

The function NTT^{-1} is the inverse of the function NTT where ω is replaced by $\omega^{-1} = 49^{-1} \mod q = 1254$ and where every the result is multiplied by the scalar n^{-1}.

Definition of Parse. The public parameter $\mathbf{a}$ is generated from a 256-bit seed through the extendable-output function SHAKE-128 [69, 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 KB. 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 $\mathbf{a}$ is considered to be in NTT domain. This is possible because the NTT transforms uniform noise to uniform noise.

The message format of $(\mathbf{b}, \boldsymbol{seed})$ 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 [72,75]). Polynomials are encoded as an array of 1024 little-endian 16-bit unsigned integers

in $\{0, \ldots, q-1\}$. Each of these integers encodes one coefficient, starting with the constant term, followed by the coefficient of X^1 and so on. This leaves two bits unused in every second byte, because q has only 14 bits. These bits are used to store *seed* (in the unused upper bits of the first 128 coefficients of $\mathbf{b}$) and $\mathbf{r}$ (in the unused upper bits of the coefficients of $\mathbf{u}$). Note that $\mathbf{r}$ is an array of 1024 2-bit integers and thus makes use of all the available space in the upper bits. 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 4.

Parameters: $q = 12289 < 2^{14},\ n = 1024$ Error distribution: ψ_{12}		
Alice (server)		**Bob (client)**
$seed \xleftarrow{\$} \{0, \ldots, 255\}^{32}$ $\mathsf{Parse}(\mathsf{SHAKE\text{-}128}(seed))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \psi_{12}^n$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \psi_{12}^n$
$\mathbf{b} \leftarrow \mathbf{a} \circ \mathsf{NTT}(\mathbf{s}) + \mathsf{NTT}(\mathbf{e})$	$\xrightarrow[2048\text{Bytes}]{m_a = \mathsf{encodeA}(\mathbf{b}, seed)}$	$(\mathbf{b}, seed) \leftarrow \mathsf{decodeA}(m_a)$ $\mathbf{a} \leftarrow \mathsf{Parse}(\mathsf{SHAKE\text{-}128}(seed))$ $\mathbf{t} \leftarrow \mathsf{NTT}(\mathbf{s}')$ $\mathbf{u} \leftarrow \mathbf{a} \circ \mathbf{t} + \mathsf{NTT}(\mathbf{e}')$ $\mathbf{v} \leftarrow \mathsf{NTT}^{-1}(\mathbf{b} \circ \mathbf{t} + \mathsf{NTT}(\mathbf{e}''))$
$(\mathbf{u}, \mathbf{r}) \leftarrow \mathsf{decodeB}(m_b)$ $\mathbf{v}' \leftarrow \mathsf{NTT}^{-1}(\mathbf{u} \circ \mathbf{s})$ $\nu \leftarrow \mathsf{Rec}(\mathbf{v}', \mathbf{r})$ $\mu \leftarrow \mathsf{SHA3\text{-}256}(\nu)$	$\xleftarrow[2048\ \text{Bytes}]{m_b = \mathsf{encodeB}(\mathbf{u}, \mathbf{r})}$	$\mathbf{r} \xleftarrow{\$} \mathsf{HelpRec}(\mathbf{v})$ $\nu \leftarrow \mathsf{Rec}(\mathbf{v}, \mathbf{r})$ $\mu \leftarrow \mathsf{SHA3\text{-}256}(\nu)$

Protocol 4: Our proposed protocol including NTT and NTT^{-1} computations and sizes of exchanged messages; $\circ$ denotes pointwise multiplication.

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 4 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 [25,37]. 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. Note that 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 [65] 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 $g\omega \pmod q$ 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. We will establish in the following 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 montgomery_reduce(uint32_t a)
{
  uint32_t u;

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

(b) Short Barrett reduction.

```
uint16_t barrett_reduce(uint16_t a)
{
  uint32_t u;

  u = ((uint32_t) a * 5) >> 16;
  u *= 12289;
  a -= u;
  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] = barrett_reduce(t + a[j+d]);
  a[j+d] = montgomery_reduce(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 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 12-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 [52, 53].

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, 15, 43]) 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 [45]. 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 [45]. One computation of a dimension-1024 NTT takes 10 984 cycles, unlike the numbers in [45] this does include multiplication by the powers of γ and unlike the numbers in [45], 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 [16]. 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 [16] 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 3 569 cycles, the cost for Rec is only 2 796 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)

that was running at 3491.444 MHz with Turbo Boost and Hyperthreading disabled for a fair comparison. We compiled our software with gcc-4.9.2 and flags `-O3 -fomit-frame-pointer -msse2avx -march=corei7-avx`. 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.

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 $\mathbf{a}$		57 134[a]	57 546[a]
		(56 957)[a]	(57 329)[a]
NTT		55 532	10 948
NTT^{-1}		59 220[b]	11 896[b]
Sampling of a noise polynomial		31 912[c]	4 760[c]
HelpRec		14 368	3 596
Rec		10 024	2 796
Key generation (server)	≈ 2 477 958	265 968	107 534
		(265 933)	(107 385)
Key gen + shared key (client)	≈ 3 995 977	380 676	126 236
		(380 936)	(126 336)
Shared key (server)	≈ 481 937	82 312	22 104

[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

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 [44]. Also OpenSSL includes fast software for `nistp256` ECDH by Käsper and Langley. This implementation has to be enabled using the `enable-ec_nistp_64_gcc_128` flag; we assume that the authors of [18] omitted this. Compared to BCNS, our C implementation is more than 8 times faster and our AVX implementation even achieves a speedup factor of more than 20. At this performance it is in the same ballpark as state-of-the-art ECDH software, even when TLS switches [54] to faster 128-bit secure ECDH key exchange based on Curve25519 [10].

When looking at the BCNS proposal, one reason for our performance advantage is certainly the switch to the binomial error distribution. In [18] the inversion method [32] is used to sample from a discrete Gaussian distribution with a high precision, which basically requires a search operation in a precomputed lookup table. In BCNS this lookup table has 52 entries of 192 bits representing integer values to achieve a statistics distance of less than 2^{-128}. Sampling of one coefficient requires 192 random bits that were generated using the AES so that one polynomial consisting of 1024 coefficients requires 296 608 bits of random data and the constant time linear search takes 1 042 700 cycles. Our C implementation just requires 21 068 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 [68]. An advantage of Nussbaumer's algorithm compared to the NTT is that it allows to choose a non-prime modulus. However, the implementation in [18] still only achieves lower performance of 342 800 cycles for a constant-time multiplication. Nevertheless, it appears to be an interesting future work to investigate optimization and vectorization of Nussbaumer's algorithm. Additionally, the authors of [18] did not perform pre-transformation of constants (e.g., $\mathbf{a}$) or transmission of coefficients in FFT/Nussbaumer representation. Note also that the on-the-fly generation of $\mathbf{a}$, which is not implemented in BCNS, results in a certain overhead in our implementation and caching of $\mathbf{a}$ would lead to an even better (server-side) performance in our implementation.

Comparison with an authenticated key exchange. The most recent proposal of an authenticated key exchange based on ideal lattices is given by Zhang, Zhang, Ding, Snook, and Dagdelen in [80]. Their proposal is based on the HMQV protocol being adapted to Ring-LWE under the observation that Ring-LWE supports kind of "approximate" multiplication of group elements where only small noise is not recoverable. The security proof relies on the random oracle model and the Fiat-Shamir transform; due to some uncertainty regarding rewinding transforms like Fiat-Shamir in the random oracle model [28], no quantum resistance is claimed. In [80] several parameter sets for a one-pass and a two-pass variant of the protocol are proposed. For their low-security two-pass protocol (75–80 bits) they choose dimension $n = 1024$ and modulus q in the range of 45–47 bits and for high security (210–230 bits) they choose $n = 2048$ and q in the range of 47–50 bits. For the same dimension n the one-pass instantiation

provides higher security and requires moduli in the range of $30 - 33$ bits. The proof-of-concept implementation uses the NTL library to implement FFT-based polynomial arithmetic and thus cannot be compared in a fair manner to an optimized C or assembly implementation. Gaussian sampling is also realized using the inversion method. The probability of a decryption error is about 2^{-87}. On a 2.83 GHz Intel Core 2 Quad CPU the key generation requires between 14.26 ms and 49.77 ms, the response between 19 ms and 60.31 ms and the finalization step 4.35 ms to 9.59 ms. This is more than an order of magnitude slower than the unauthenticated key exchange presented by [18] and more than two orders of magnitude slower than our proposal.

References

1. David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella Béguelin, and Paul Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *CCS '15 Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 5–17. ACM, 2015. https://weakdh.org/. 6, 17
2. National Security Agency. NSA suite B cryptography. https://www.nsa.gov/ia/programs/suiteb_cryptography/, Updated on August 19, 2015. 2
3. Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. IACR Cryptology ePrint Archive report 2015/046, 2015. http://eprint.iacr.org/2015/046/. 14, 15
4. Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In Luca Aceto, Monika Henzingeri, and Jiří Sgall, editors, *Automata, Languages and Programming*, volume 6755 of *LNCS*, pages 403–415. Springer, 2011. https://www.cs.duke.edu/~rongge/LPSN.pdf. 9, 14
5. László Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986. http://www.csie.nuk.edu.tw/~cychen/Lattices/On%20lovasz%20lattice%20reduction%20and%20the%20nearest%20lattice%20point%20problem.pdf. 17
6. Shi Bai and Steven D. Galbraith. An improved compression technique for signatures based on learning with errors. In Josh Benaloh, editor, *Topics in Cryptology – CT-RSA 2014*, volume 8366 of *LNCS*, pages 28–47. Springer, 2014. https://eprint.iacr.org/2013/838/. 2
7. Shi Bai, Adeline Langlois, Tancrède Lepoint, Damien Stehlé, and Ron Steinfeld. Improved security proofs in lattice-based cryptography: using the Rényi divergence rather than the statistical distance. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015*, LNCS. Springer, 2015 (to appear). http://eprint.iacr.org/2015/483/. 8
8. Paul Barrett. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 311–323. Springer-Verlag Berlin Heidelberg, 1987. 21
9. Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. 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. SIAM, 2016 (to appear). 15

10. Daniel J. Bernstein. Curve25519: new Diffie-Hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography – PKC 2006*, volume 3958 of *LNCS*, pages 207–228. Springer, 2006. http://cr.yp.to/papers.html#curve25519. 25
11. Daniel J. Bernstein. 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. 22
12. Daniel J. Bernstein, Tung Chou, Chitchanok Chuengsatiansup, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, and Christine van Vredendaal. 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/. 6, 17
13. Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Peter Schwabe. Kummer strikes back: new DH speed records. In Tetsu Iwata and Palash Sarkar, editors, *Advances in Cryptology – EUROCRYPT 2015*, volume 8873 of *LNCS*, pages 317–337. Springer, 2014. full version: http://cryptojedi.org/papers/#kummer. 23
14. Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. SPHINCS: practical stateless hash-based signatures. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 368–397. Springer, 2015. https://cryptojedi.org/papers/#sphincs. 5
15. Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Peter Schwabe, and Zooko Wilcox-O'Hearn. SPHINCS: practical stateless hash-based signatures. In Marc Fischlin and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 368–397. Springer, 2015. http://cryptojedi.org/papers/#sphincs. 23
16. Daniel J. Bernstein and Tanja Lange. eBACS: ECRYPT benchmarking of cryptographic systems. http://bench.cr.yp.to (accessed 2015-10-07). 23
17. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 313–314. Springer, 2013. 18
18. Joppe W. Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In *2015 IEEE Symposium on Security and Privacy*, pages 553–570, 2015. http://eprint.iacr.org/2014/599. 2, 3, 4, 5, 6, 7, 9, 13, 14, 16, 17, 24, 25, 26
19. Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 575–584. ACM, 2013. http://arxiv.org/pdf/1306.0281. 7
20. David Cadé, Xavier Pujol, and Damien Stehlé. fplll 4.0.4, 2013. https://github.com/dstehle/fplll (accessed 2015-10-13). 14
21. Nicolas J. Cerf, Lov K. Grover, and Colin P. Williams. Nested quantum search and structured problems. *Physical Review A*, 61(3):032303, 2000. 15
22. Stephen Checkoway, Matthew Fredrikson, Ruben Niederhagen, Adam Everspaugh, Matthew Green, Tanja Lange, Thomas Ristenpart, Daniel J. Bernstein, Jake Maskiewicz, and Hovav Shacham. 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. 7
23. Yuanmi Chen. Lattice reduction and concrete security of fully homomorphic encryption. 2013. Ph.D Thesis, available at http://www.di.ens.fr/~ychen/research/these.pdf. 15
24. Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 1–20. Springer, 2011. http://www.iacr.org/archive/asiacrypt2011/70730001/70730001.pdf. 14, 15
25. Eleanor Chu and Alan George. *Inside the FFT Black Box Serial and Parallel Fast Fourier Transform Algorithms*. CRC Press, Boca Raton, FL, USA, 2000. 21
26. Jon Conway and Neil J. A. Sloane. *Sphere packings, lattices and groups*, volume 290 of *Grundlehren der mathematischen Wissenschaften*. Springer Science & Business Media, 3 edition, 1999. 9
27. James Cowie, Bruce Dodson, R. Marije Elkenbracht-Huizing, Arjen K. Lenstra, Peter L. Montgomery, and Joerg Zayer. A world wide number field sieve factoring record: on to 512 bits. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology – ASIACRYPT'96*, volume 1163 of *LNCS*, pages 382–394. Springer, 1996. http://oai.cwi.nl/oai/asset/1940/1940A.pdf. 14
28. Özgür Dagdelen, Marc Fischlin, and Tommaso Gagliardoni. The Fiat-Shamir transformation in a quantum world. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013*, volume 8270 of *LNCS*, pages 62–81. Springer, 2013. https://eprint.iacr.org/2013/245/. 25
29. Ruan de Clercq, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. Efficient software implementation of ring-LWE encryption. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2015*, pages 339–344. EDA Consortium, 2015. http://eprint.iacr.org/2014/725. 2, 6, 7, 13, 18
30. Nico Döttling and Jörn Müller-Quade. Lossy codes and a new variant of the learning-with-errors problem. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 18–34. Springer, 2013. http://www.iacr.org/archive/eurocrypt2013/78810018/78810018.pdf. 8
31. Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal Gaussians. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, volume 8042 of *LNCS*, pages 40–56. Springer, 2013. https://eprint.iacr.org/2013/383/. 2, 6
32. Nagarjun C. Dwarakanath and Steven D. Galbraith. Sampling from discrete Gaussians for lattice-based cryptography on a constrained device. *Applicable Algebra in Engineering, Communication and Computing*, 25(3):159–180, 2014. https://www.math.auckland.ac.nz/~sgal018/gen-gaussians.pdf. 25
33. Atsushi Fujioka, Koutarou Suzuki, Keita Xagawa, and Kazuki Yoneyama. Practical and post-quantum authenticated key exchange from one-way secure key encapsulation mechanism. In Kefei Chen, Qi Xie, Weidong Qiu, Ninghui Li, and Wen-Guey Tzeng, editors, *Symposium on Information, Computer and Communications Security, ASIA CCS 2013*, pages 83–94. ACM, 2013. 5
34. Steven D. Galbraith. Space-efficient variants of cryptosystems based on learning with errors, 2013. https://www.math.auckland.ac.nz/~sgal018/compact-LWE.pdf. 18
35. Nicolas Gama, Phong Q Nguyen, and Oded Regev. Lattice enumeration using extreme pruning. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT*

2010, volume 6110 of *LNCS*, pages 257–278. Springer, 2010. http://www.iacr.org/archive/eurocrypt2010/66320257/66320257.pdf. 14

36. Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *Eurocrypt*, volume 7881, pages 1–17. Springer, 2013. 14
37. W. M. Gentleman and G. Sande. Fast Fourier transforms: for fun and profit. In *Fall Joint Computer Conference*, volume 29 of *AFIPS Proceedings*, pages 563–578, 1966. http://cis.rit.edu/class/simg716/FFT_Fun_Profit.pdf. 21
38. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC '09 Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178. ACM, 2009. https://www.cs.cmu.edu/~odonnell/hits09/gentry-homomorphic-encryption.pdf. 2
39. Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC '08 Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 197–206. ACM, 2008. https://eprint.iacr.org/2007/432/. 8
40. Craig Gentry and Mike Szydlo. Cryptanalysis of the revised ntru signature scheme. In *Advances in Cryptology—EUROCRYPT 2002*, pages 299–320. Springer, 2002. 14
41. Satrajit Ghosh and Aniket Kate. 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. 5
42. Norman Göttert, Thomas Feller, Michael Schneider, Johannes A. Buchmann, and Sorin A. Huss. On the design of hardware building blocks for modern lattice-based encryption schemes. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems - CHES 2012*, volume 7428 of *LNCS*, pages 512–529. Springer, 2012. http://www.iacr.org/archive/ches2012/74280511/74280511.pdf. 13, 18
43. Shay Gueron. 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. 23
44. Shay Gueron and Vlad Krasnov. Fast prime field elliptic-curve cryptography with 256-bit primes. *Journal of Cryptographic Engineering*, 5(2):141–151, 2014. https://eprint.iacr.org/2013/816/. 25
45. Tim Güneysu, Tobias Oder, Thomas Pöppelmann, and Peter Schwabe. Software speed records for lattice-based signatures. In Philippe Gaborit, editor, *Post-Quantum Cryptography*, volume 7932 of *LNCS*, pages 67–82. Springer, 2013. http://cryptojedi.org/papers/#lattisigns. 19, 23
46. Guillaume Hanrot, Xavier Pujol, and Damien Stehléé. Terminating BKZ. IACR Cryptology ePrint Archive report 2011/198, 2011. http://eprint.iacr.org/2011/198/. 14
47. Jeff Hoffstein, Jill Pipher, John M. Schanck, Joseph H. Silverman, and William Whyte. Practical signatures from the partial Fourier recovery problem. In Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay, editors, *Applied Cryptography and Network Security*, volume 8479 of *LNCS*, pages 476–493. Springer, 2014. https://eprint.iacr.org/2013/757/. 2
48. Jeffrey Hoffstein, Jull Pipher, and Joseph H. Silverman. NTRU: a ring-based public key cryptosystem. In Joe P. Buhler, editor, *Algorithmic number theory*, volume 1423 of *LNCS*, pages 267–288. Springer, 1998. https://www.securityinnovation.com/uploads/Crypto/ANTS97.ps.gz. 2, 17

49. Paul Kirchner and Pierre-Alain Fouque. An improved BKW algorithm for LWE with applications to cryptography and lattices. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, volume 9215 of *LNCS*, pages 43–62. Springer, 2015. https://eprint.iacr.org/2015/552/. 9, 14
50. Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In Rosiario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, volume 9216 of *LNCS*, pages 3–22. Springer, 2015. https://eprint.iacr.org/2014/744/. 15
51. Thijs Laarhoven, Michele Mosca, and Joop van de Pol. Finding shortest lattice vectors faster using quantum search. *Designs, Codes and Cryptography*, 77(2):375–400, 2015. https://eprint.iacr.org/2014/907/. 15, 16
52. Adam Langley. TLS symmetric crypto. Blog post on imperialviolet.org, 2014. https://www.imperialviolet.org/2014/02/27/tlssymmetriccrypto.html (accessed 2015-10-07). 22
53. Adam Langley and Wan-Teh Chang. ChaCha20 and Poly1305 based cipher suites for TLS: Internet draft. https://tools.ietf.org/html/draft-agl-tls-chacha20poly1305-04 (accessed 2015-02-01). 22
54. Adam Langley, Michael Hamburg, and Sean Turner. Elliptic curves for security. draft-irtf-cfrg-curves-11, 2015. https://tools.ietf.org/html/draft-irtf-cfrg-curves-11. 25
55. HW Lenstra and A Silverberg. Revisiting the gentry-szydlo algorithm. In *Advances in Cryptology–CRYPTO 2014*, pages 280–296. Springer, 2014. 14
56. Jos Leys, Étienne Ghys, and Auélien Alvarez. Dimensions, 2010. http://www.dimensions-math.org/ (accessed 2015-10-19). 9
57. Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In Aggelos Kiayias, editor, *Topics in Cryptology - CT-RSA 2011*, volume 6558 of *LNCS*, pages 319–339. Springer, 2011. https://eprint.iacr.org/2010/613/. 5
58. Zhe Liu, Hwajeong Seo, Sujoy Sinha Roy, Johann Großschädl, Howon Kim, and Ingrid Verbauwhede. Efficient Ring-LWE encryption on 8-bit AVR processors. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems - CHES 2015*, volume 9293 of *LNCS*, pages 663–682. Springer, 2015. https://eprint.iacr.org/2015/410/. 6, 13, 18
59. Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 738–755. Springer, 2012. https://eprint.iacr.org/2011/537/. 8
60. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, 2010. http://www.di.ens.fr/~lyubash/papers/ringLWE.pdf. 4
61. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *Journal of the ACM (JACM)*, 60(6):43:1–43:35, 2013. http://www.cims.nyu.edu/~regev/papers/ideal-lwe.pdf. 2, 7
62. Carlos Aguilar Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. XPIRe: Private information retrieval for everyone. IACR Cryptology ePrint Archive report 2014/1025, 2014. http://eprint.iacr.org/2014/1025. 19
63. Daniele Micciancio and Chris Peikert. Hardness of SIS and LWE with small parameters. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, volume 8042 of *LNCS*, pages 21–39. Springer, 2013. https://eprint.iacr.org/2013/069/. 8

64. Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *SODA '10 Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1468–1480. SIAM, 2010. http://dl.acm.org/citation.cfm?id=1873720. 14
65. Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985. http://www.ams.org/journals/mcom/1985-44-170/S0025-5718-1985-0777282-X/S0025-5718-1985-0777282-X.pdf. 21
66. Phong Q. Nguyen and Thomas Vidick. Sieve algorithms for the shortest vector problem are practical. *Journal of Mathematical Cryptology*, 2(2):181–207, 2008. ftp://ftp.di.ens.fr/pub/users/pnguyen/JoMC08.pdf. 14
67. NIST. Workshop on cybersecurity in a post-quantum world, 2015. http://www.nist.gov/itl/csd/ct/post-quantum-crypto-workshop-2015.cfm. 2
68. Henri J. Nussbaumer. Fast polynomial transform algorithms for digital convolution. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 28(2):205–215, 1980. 25
69. 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. 18, 19
70. Chris Peikert. Lattice cryptography for the Internet. In Michele Mosca, editor, *Post-Quantum Cryptography*, volume 8772 of *LNCS*, pages 197–219. Springer, 2014. http://web.eecs.umich.edu/~cpeikert/pubs/suite.pdf. 2, 3, 4, 5, 6, 7, 11
71. Thomas Pöppelmann, Léo Ducas, and Tim Güneysu. Enhanced lattice-based signatures on reconfigurable hardware. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems – CHES 2014*, volume 8731 of *LNCS*, pages 353–370. Springer, 2014. https://eprint.iacr.org/2014/254/. 6
72. Thomas Pöppelmann and Tim Güneysu. Towards practical lattice-based public-key encryption on reconfigurable hardware. In Tanja Lange, Kristin Lauter, and Petr Lisoněk, editors, *Selected Areas in Cryptography – SAC 2013*, volume 8282 of *LNCS*, pages 68–85. Springer, 2013. https://www.ei.rub.de/media/sh/veroeffentlichungen/2013/08/14/lwe_encrypt.pdf. 2, 9, 13, 18, 19
73. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 56(6):34, 2009. http://www.cims.nyu.edu/~regev/papers/qcrypto.pdf. 7
74. Alfréd Rényi. On measures of entropy and information. In *Fourth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 547–561, 1961. http://projecteuclid.org/euclid.bsmsp/1200512181. 8
75. Sujoy Sinha Roy, Frederik Vercauteren, Nele Mentens, Donald Donglong Chen, and Ingrid Verbauwhede. Compact Ring-LWE cryptoprocessor. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems – CHES 2014*, volume 8731 of *LNCS*, pages 371–391. Springer, 2014. https://eprint.iacr.org/2013/866/. 6, 7, 19
76. Claus-Peter Schnorr and Martin Euchner. Lattice basis reduction: improved practical algorithms and solving subset sum problems. *Mathematical programming*, 66(1-3):181–199, 1994. http://www.csie.nuk.edu.tw/~cychen/Lattices/Lattice%20Basis%20Reduction_%20Improved%20Practical%20Algorithms%20and%20Solving%20Subset%20Sum%20Problems.pdf. 14
77. Damien Stehlé and Ron Steinfeld. Making NTRU as secure as worst-case problems over ideal lattices. In Kenneth G. Paterson, editor, *Advances in Cryptology –*

EUROCRYPT 2011, volume 6632 of *LNCS*, pages 27–47. Springer, 2011. http://www.iacr.org/archive/eurocrypt2011/66320027/66320027.pdf. 2, 17

78. Tor project: Anonymity online. https://www.torproject.org/. 5
79. Roman Vershynin. Introduction to the non-asymptotic analysis of random matrices. In *Compressed sensing*, pages 210–268. Cambridge University Press, 2012. http://www-personal.umich.edu/~romanv/papers/non-asymptotic-rmt-plain.pdf. 11, 12
80. Jiang Zhang, Zhenfeng Zhang, Jintai Ding, Michael Snook, and Özgür Dagdelen. Authenticated key exchange from ideal lattices. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, volume 9057 of *LNCS*, pages 719–751. Springer, 2015. https://eprint.iacr.org/2014/589/. 5, 25