HILA5: On Reliability, Reconciliation, and Error Correction for Ring-LWE Encryption

Markku-Juhani O. Saarinen[⋆]

Helsinki, Finland
`mjos@iki.fi`

Abstract. We describe a new reconciliation method for Ring-LWE that has a significantly smaller failure rate than previous proposals while reducing ciphertext size and the amount of randomness required. It is based on a simple, deterministic variant of Peikert's reconciliation that works with our new "safe bits" selection and constant-time error correction techniques. The new method does not need randomized smoothing to achieve non-biased secrets. When used with the very efficient "New Hope" Ring-LWE parametrization we achieve a decryption failure rate well below 2^{-128} (compared to 2^{-60} of the original), making the scheme suitable for public key encryption in addition to key exchange protocols; the reconciliation approach saves about 40% in ciphertext size when compared to the common LP11 Ring-LWE encryption scheme. We perform a combinatorial failure analysis using full probability convolutions, leading to a precise understanding of decryption failure conditions on bit level. Even with additional implementation security and safety measures the new scheme is still essentially as fast as the New Hope but has slightly shorter messages. The new techniques have been instantiated and implemented as a Key Encapsulation Mechanism (KEM) and public key encryption scheme designed to meet the requirements of NIST's Post-Quantum Cryptography effort at very high security level.

Keywords: Ring-LWE, Reconciliation, Post-Quantum Encryption, New Hope.

1 Introduction

Some classes of encrypted data must remain confidential for a long period of time – often at least few decades in national security applications. Therefore high-security cryptography should be resistant to attacks even with projected future technologies. As there are no physical or theoretical barriers preventing progressive development of quantum computing technologies capable of breaking current RSA- and Elliptic Curve based cryptographic standards (using polynomial-time quantum algorithms already known [36,42]), a need for such quantum-resistant algorithms in national security applications has been identified [32].

In December 2016 NIST issued a standardization call for quantum-resistant public key algorithms, together with requirements and evaluation criteria [31].

[⋆] Most of this work was performed while the author was with DARKMATTER, UAE.

This has made "Post-Quantum Cryptography" (PQC) central to cryptographic engineers who must now design concrete proposals for standardization. Practical issues such as performance, reliability, message and key sizes, implementation and side-channel security, and compatibility with existing and anticipated applications, protocols, and standards are as relevant as mere theoretical security and asymptotic feasibility when evaluating these proposals.

Ring-LWE lattice primitives offer some of the best performance and key size characteristics among quantum-resistant candidates [15]. These algorithms rely on "random noise" for security and always have some risk of decryption failure. This reliability issue can pose problems when used in non-interactive applications which are not designed to tolerate errors. The issue of decryption failure can be addressed via reconciliation methods, which is the focus of present work.

Structure of this paper and our contributions. Section 2 provides a practical introduction to Ring-LWE Key Exchange and prior work on reconciliation. Section 3 introduces our new reconciliation techniques, together with detailed analysis. Section 4 discusses design, analysis, and implementation of XE5, a simple constant-time error correction code suitable for Ring-LWE. Section 5 contains the specification and implementation benchmarks for our instantiation HILA5, designed to meet the NIST PQC criteria at high security level. We conclude in Section 6. Additional algorithmic listings are provided in Appendix A.

2 Ring-LWE Key Exchange and Key Encapsulation

Notation and Basic Properties. Reduction $x \mod q$ puts a number in non-negative range $0 \leq x < q$. We write the rounding function as $\lfloor x \rceil = \lfloor x + \frac{1}{2} \rfloor$.

Let $\mathcal{R}$ be a ring with elements $\mathbf{v} \in \mathbb{Z}_q^n$. Its coefficients $v_i \in [0, q-1]$ $(0 \leq i < n)$ can be interpreted as a polynomial via $v(x) = \sum_{i=0}^{n-1} v_i x^i$, or as a zero-indexed vector. Addition, subtraction, and scaling (scalar multiplication with c) follow the basic rules for polynomials or vectors with coefficients in $\mathbb{Z}_q$.

For multiplication in $\mathcal{R}$ we use cyclotomic polynomial basis $\mathbb{Z}_q[x]/(x^n+1)$. Products are reduced modulo q and x^n+1 and results are bound by degree $n-1$ since $x^n \equiv q-1$ in $\mathcal{R}$. We may write a direct wrap-around multiplication rule:

$$\mathbf{h} = \mathbf{f} * \mathbf{g} \mod (x^n+1) \iff h_i = \sum_{j=0}^{i} f_j g_{(i-j)} - \sum_{j=i+1}^{n-1} f_j g_{(n+i-j)}. \quad (1)$$

Algorithmically the multiplication rule of Equation 1 requires $O(n^2)$ elementary operations. However, there is an $O(n \log n)$ method using the Number Theoretic Transform (NTT), originally from Nussbaumer [33]. For efficient NTT implementation n should be a power of two and q a small prime, with $2n \mid q-1$.

Definition 1 (Informal). *With all distributions and computations in ring $\mathcal{R}$, let $\mathbf{s}, \mathbf{e}$ be elements randomly chosen from some non-uniform distribution χ, and $\mathbf{g}$ be a uniformly random public value. Determining $\mathbf{s}$ from $(\mathbf{g}, \mathbf{g}*\mathbf{s}+\mathbf{e})$ in ring $\mathcal{R}$ is the (Normal Form Search) Ring Learning With Errors (RLWE$_{\mathcal{R},\chi}$) problem.*

Typically, χ is chosen so that each coefficient is a Discrete Gaussian or from some other "Bell-Shaped" distribution that is relatively tightly concentrated around zero. The hardness of the problem is a function of n, q, and χ. [1]

2.1 Noisy Diffie-Hellman in a Ring

A key exchange method analogous to Diffie-Hellman can be constructed in $\mathcal{R}$ in a straightforward manner, as first described in [1,34]. Let $\mathbf{g} \xleftarrow{\$} \mathcal{R}$ be a uniformly random common parameter ("generator"), and χ a non-uniform distribution.

Alice		**Bob**
$\mathbf{a} \xleftarrow{\$} \chi$	*private keys*	$\mathbf{b} \xleftarrow{\$} \chi$
$\mathbf{e} \xleftarrow{\$} \chi$	*noise*	$\mathbf{e}' \xleftarrow{\$} \chi$
$\mathbf{A} = \mathbf{g} * \mathbf{a} + \mathbf{e}$	*public keys*	$\mathbf{B} = \mathbf{g} * \mathbf{b} + \mathbf{e}'$
	$\xrightarrow{\mathbf{A}}$	
	$\xleftarrow{\mathbf{B}}$	
$\mathbf{x} = \mathbf{B} * \mathbf{a}$	*shared secret*	$\mathbf{y} = \mathbf{A} * \mathbf{b}$

We see that that the way messages $\mathbf{A}, \mathbf{B}$ are generated makes the security of the scheme equivalent to Definition 1. This commutative scheme "almost" works like Diffie-Hellman because the shared secrets only approximately agree; $\mathbf{x} \approx \mathbf{y}$. Since the ring $\mathcal{R}$ is commutative, substituting $\mathbf{A}$ and $\mathbf{B}$ gives

$$\mathbf{x} = (\mathbf{g} * \mathbf{b} + \mathbf{e}') * \mathbf{a} = \mathbf{g} * \mathbf{a} * \mathbf{b} + \mathbf{e}' * \mathbf{a} \tag{2}$$

$$\mathbf{y} = (\mathbf{g} * \mathbf{a} + \mathbf{e}) * \mathbf{b} = \mathbf{g} * \mathbf{a} * \mathbf{b} + \mathbf{e} * \mathbf{b}. \tag{3}$$

The distance Δ therefore consists only of products of "noise" parameters:

$$\Delta = \mathbf{x} - \mathbf{y} = \mathbf{e}' * \mathbf{a} - \mathbf{e} * \mathbf{b}. \tag{4}$$

We observe that each of $\{\mathbf{a}, \mathbf{b}, \mathbf{e}, \mathbf{e}'\}$ in Δ are picked independently from χ, which should be relatively "small' and zero-centered. The coefficients of both $\mathbf{x}$ and $\mathbf{y}$ are dominated by common, uniformly distributed factor $\mathbf{g} * \mathbf{a} * \mathbf{b} \approx \mathbf{x} \approx \mathbf{y}$. Up to n shared bits can be decoded from coefficients of $\mathbf{x}$ and $\mathbf{y}$ by a simple binary classifier such as $\lfloor \frac{2x_i}{q} \rceil \approx \lfloor \frac{2y_i}{q} \rceil$. This type of generation will generate some disagreeing bits due to error Δ, however. Furthermore, the output of the classifier is slightly biased when q is odd. This is why additional steps are required.

[1] **References and Notes on RLWE.** The Learning With Errors (LWE) problem in cryptography originates with Regev [37] who showed its connection to fundamental lattice problems in a quantum setting. Regev also showed equivalence of search and decision variants [38]. These ideas were extended to ring setting (RLWE) starting with [28]. The connection between a uniform secret $\mathbf{s}$ and a secret chosen from χ is provided by Applebaum et al. [7] for LWE case, and for the ring setting in [29]. Due to these reductions, the informal problem of Definition 1 can be understood to describe "RLWE". Best known methods for solving the problem expand an RLWE instance to the general (lattice) LWE, and therefore RLWE falls under "lattice cryptography" umbrella. For a recent review of its concrete hardness, see [2].

2.2 Reconciliation

Let $\mathbf{x} \approx \mathbf{y}$ be two vectors in $\mathbb{Z}_q^n$ with a relatively small difference in each coefficient; the distribution of the distance $\delta_i = x_i - y_i$ is strongly centered around zero. In reconciliation, we wish the holders of $\mathbf{x}$ and $\mathbf{y}$ (Alice and Bob, respectively) to be able to arrive at exactly the same shared secret (key) $\mathbf{k}$ with a small amount of communication $\mathbf{c}$. However, single-message reconciliation can also be described simply as a part of an encryption algorithm (not a protocol). [2]

Peikert's Reconciliation and BCNS Instantiation. In Peikert's reconciliation for odd modulus [35], Bob first generates a randomization vector $\mathbf{r}$ such that each $r_i \in \{0, \pm 1\}$ is uniform modulo two. Bob can then determine the public reconciliation $\mathbf{c}$ and shared secret $\mathbf{k}$ via

$$c_i = \left\lfloor \frac{2(2y_i - r_i)}{q} \right\rfloor \bmod 2 \qquad k_i = \left\lfloor \frac{2y_i - r_i}{q} \right\rceil \bmod 2. \tag{5}$$

We define disjoint helper sets $I_0 = [0, \lfloor \frac{q}{2} \rceil]$ and $I_1 = [-\lfloor \frac{q}{2} \rfloor, -1]$ and $E = [-\frac{q}{4}, \frac{q}{4})$. Alice uses $\mathbf{x}$ to arrive at the shared secret $\mathbf{k}' = \mathbf{k}$ via

$$k_i' = \begin{cases} 0, \text{ if } 2x_i \in I_{c_i} + E \mod 2q \\ 1, \text{otherwise.} \end{cases} \tag{6}$$

This mechanism is illustrated in Figure 1. Peikert's reconciliation was adopted for the Internet-oriented "BCNS" instantiation [13], which has a vanishingly small failure probability; $Pr(\mathbf{k}' \neq \mathbf{k}) < 2^{-16384}$.

New Hope Variants. "New Hope" is a prominent, more recent instantiation of Peikert's key exchange scheme [4]. New Hope is parametrized at $n = 1024$, yet produces a 256-bit secret key k. This allowed the designers to develop a relatively complex reconciliation mechanism that uses $\frac{1024}{256} = 4$ coefficients of $\mathbf{x}$ and $2 * 4 = 8$ bits of reconciliation information to reach $< 2^{-60}$ failure rate.

In a follow-up paper [3] the New Hope authors let Bob unilaterally choose the secret key, and significantly simplified their approach. This version also uses four coefficients, but requires $3 * 4 = 12$ bits of reconciliation (or "ciphertext") information per key bit. The total failure probability is the same $< 2^{-60}$.

Security Level and Failure Probability. Note that despite having a higher failure probability, the security level of New Hope (Section 2.2) is higher than that of BCNS (Section 2.2). Security of RLWE is closely related to the entropy

[2] **References and Notes on Reconciliation.** The term "reconciliation" comes from Quantum Cryptography. Standard Quantum Key Distribution (QKD) protocols such as BB84 [9] result in approximately agreeing shared secrets, which must be reconciled over a public channel with the help of classical information theory and cryptography [10,14]. Ding et al. describe functionally similar (but mathematically very different) "Robust Extractors" in later versions of [20] and patent application [19].

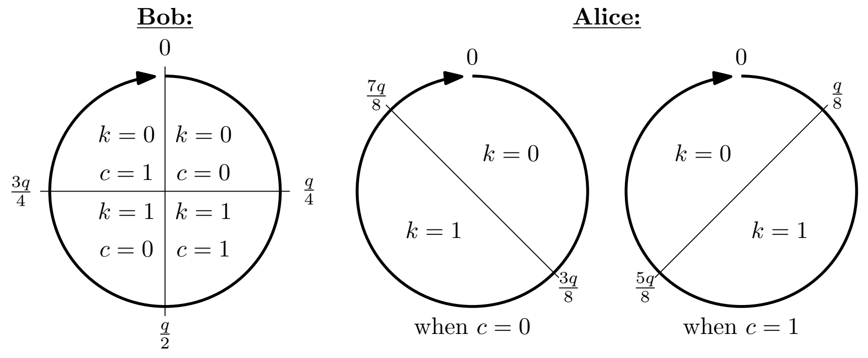

Fig. 1. Simplified view of Peikert's original reconciliation mechanism [35], ignoring randomized rounding. Alice and Bob have points $x \approx y \in \mathbb{Z}_q$ that are close to each other. Bob uses y to choose k and c as shown on left, and transmits c to Alice. Alice can use x, c to always arrive at the same shared bit k' if $|x - y| < \frac{q}{8}$, as shown on right. Without randomized smoothing the two halves $k = 0$ and $k = 1$ have an area of unequal size (when q is an odd prime) and the resulting key will be slightly biased.

and deviation of noise distribution χ in relation to modulus q. Higher noise ratio increases security against attacks, but also increases failure probability [2]. This is a fundamental trade-off in all Ring-LWE schemes.

2.3 Formalization as a KEM

Following the NIST call [31] and Peikert [35], such a scheme can be formalized as a Key Encapsulation Mechanism (KEM), which consists of three algorithms:

- $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}()$. Generate a public key PK and a secret key SK (pair).
- $(\mathsf{CT}, \mathsf{K}) \leftarrow \mathsf{Encaps}(\mathsf{PK})$. Encapsulate a (random) key K in ciphertext CT.
- $\mathsf{K} \leftarrow \mathsf{Decaps}(\mathsf{SK}, \mathsf{CT})$. Decapsulate shared key K from CT with SK.

In this model, reconciliation data is a part of ciphertext produced by Encaps. The three KEM algorithms constitute a natural single-roundtrip key exchange:

Alice		**Bob**
$(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}()$	$\xrightarrow{\mathsf{PK}}$	
	$\xleftarrow{\mathsf{CT}}$	$(\mathsf{CT}, \mathsf{K}) \leftarrow \mathsf{Encaps}(\mathsf{PK})$
$\mathsf{K} \leftarrow \mathsf{Decaps}(\mathsf{SK}, \mathsf{CT})$		

Even though a KEM cannot encrypt per se, a hybrid set-up that uses a KEM to determine random shared keys for message payload confidentiality (symmetric encryption) and integrity (via a message authentication code) is usually preferable to using asymmetric encryption directly on payload [17].

NIST requires at least IND-CPA [8] security from such a scheme. For a KEM without "plaintext", this essentially means that valid $(\mathsf{PK}, \mathsf{CT}, \mathsf{K})$ triplets are computationally indistinguishable from $(\mathsf{PK}, \mathsf{CT}, \mathsf{K}')$, where K' is random.

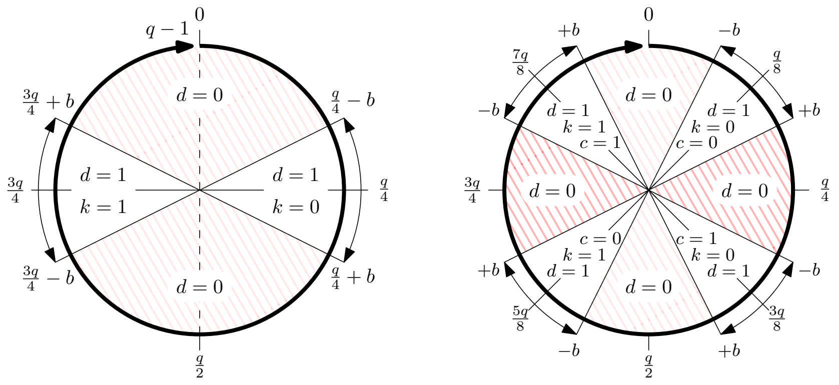

Fig. 2. We use $k = \lfloor \frac{2y}{2} \rfloor$ ($k = 1$ on left half) instead of signed rounding $k = \lfloor \frac{2y}{2} + \epsilon \rceil$ ($k = 1$ in lower half) of Peikert (Figure 1). Illustration on the left gives intuition for the simple key bit selection and SafeBits without reconciliation. Bob uses window parameter b to select "safe" bits $d = 1$ which are farthest away from the negative ($k = 1$) / positive ($k = 0$) threshold. The bit selection d is sent to Alice, who then chooses the same bits as part of the shared secret k'. On right, safe bit selection when reconciliation bits c are used; this doubles the SafeBits "area". Each section constitutes a fraction $\frac{2b+1}{q}$, so bits are unbiased. However the number of shared bits is not constant.

3 New Reconciliation Method

We define a simpler, deterministic key and reconciliation bit generation rule from Bob's share $\mathbf{y}$ to be

$$k_i = \left\lfloor \frac{2y_i}{q} \right\rfloor \quad \text{and} \quad c_i = \left\lfloor \frac{4y_i}{q} \right\rfloor \bmod 2. \tag{7}$$

Input y_i can be assumed to be uniform in range $[0, q-1]$. If taken in this plain form, the generator is slightly biased towards zero, since the interval for $k_i = 0$, $[0, \lfloor \frac{q}{2} \rfloor]$ is 1 larger than the interval $[\lceil \frac{q}{2} \rceil, q-1]$ for $k_i = 1$ when q is odd.

Intuition: Selecting safe bits (without reconciliation). Let's assume that we don't need all n bits given by the ring dimension. There is a straight-forward strategy for Bob to select m indexes in $\mathbf{y}$ that are most likely to agree. These safe coefficients are those that are closest to center points of $k = 0$ and $k = 1$ ranges, which in this case are $\frac{q}{4}$ and $\frac{3q}{4}$, respectively. Bob may choose a boundary window b, which defines shared bits to be used, and then communicate his binary selection vector $\mathbf{d}$ to Alice:

$$d_i = \begin{cases} 1 \text{ if } y_i \in \left[\lfloor \frac{q}{4} \rceil - b, \lfloor \frac{q}{4} \rceil + b\right] \quad \text{or} \quad y_i \in \left[\lfloor \frac{3q}{4} \rceil - b, \lfloor \frac{3q}{4} \rceil + b\right] \\ 0 \text{ otherwise.} \end{cases} \tag{8}$$

This simple case is illustrated on left side of Figure 2.

Since $\mathbf{y}$ is uniform in $\mathbb{Z}_q^n$, the Hamming weight of $\mathbf{d} = \mathsf{SafeBits}(\mathbf{y})$ satisfies $\mathsf{Wt}(\mathbf{d}) = \sum_{i=1}^{n-1} d_i \approx \frac{4b+2}{q} n$. Note that if not enough bits for the required payload can be obtained with bound b, Bob should re-randomize $\mathbf{y}$ rather than raising b as that can have an unexpected effect on failure rate. If there are too many selection bits for desired payload, one can just ignore them.

Importantly, both partitions are of equal size $2b+1$ and therefore k is unbiased if there are no bit failures. If Alice also uses the simple rule $k'_i = \lfloor \frac{2x_i}{q} \rceil$ to derive key bits (without c_i), the distance between shares must be at least $|x_i - y_i| > \frac{q}{4} - b$ for a bit error to occur.

3.1 Even safer bits via Peikert's reconciliation

Let Bob use Equation 7 to determine his private key bits k_i and reconciliation bits c_i. Bob also uses a new $\mathbf{d} = \mathsf{SafeBits}(\mathbf{y}, b)$ function that accounts for Peikert-style reconciliation via

$$d_i = \begin{cases} 1 \text{ if } |(y_i \bmod \lfloor \frac{q}{4} \rceil) - \lfloor \frac{q}{8} \rfloor| \le b \\ 0 \text{ otherwise.} \end{cases} \tag{9}$$

Note that there are now four "safe zones" (Figure 2, right side). Bob sends his bit selection vector $\mathbf{d}$ to Alice, along with reconciliation bits c_i at selected positions with $d_i = 1$. Alice can then get corresponding k'_i using c_i via

$$k'_i = \left\lfloor \frac{2}{q} \left(x_i - c_i \left\lfloor \frac{q}{4} \right\rceil + \left\lfloor \frac{q}{8} \right\rceil \mod q \right) \right\rfloor. \tag{10}$$

Both parties derive a final key of length $m \le \mathsf{Wt}(d)$ bits by concatenating the selected bits. Since $\mathbf{y}$ is uniform, each partition is still of size $2b+1$, and the expected weight is now $\mathsf{Wt}(\mathbf{d}) = \sum_{i=1}^{n-1} d_i \approx \frac{8b+4}{q} n$, allowing the selection to be made essentially twice as tight while producing unbiased output.

Note that when selection mechanism is used, one needs to "pack" keys to payload size m by removing k_i and k'_i at positions where $d_i = 0$. Algorithms 3 and 4 in Appendix A implement Equations 9 and 10 with packing.

3.2 Instantiation and Failure Analysis

We adopt the well-analyzed and optimized external ring parameters ($q = 12289$, $n = 1024$, and $\chi = \Psi_{16}$) from New Hope [3,4] in our instantiation.

Definition 2. *Let Ψ_k be a binomial distribution source*

$$\Psi_k = \sum_{i=0}^{k} b_i - b'_i \quad \textit{where} \quad b_i, b'_i \xleftarrow{\$} \{0,1\}. \tag{11}$$

For random variable X from Ψ_k we have $P(X = i) = 2^{-2k}\binom{2k}{k+i}$. Furthermore, Ψ_k^n is a source of $\mathcal{R}$ elements where each one of n coefficients is independently chosen from Ψ_k. Since scheme is uses $k = 16$, a typical sampler implementation just computes the Hamming weight of a 32-bit random word and subtracts 16.

Lemma 1. *Let $\varepsilon, \varepsilon'$ be vectors of length $2n$ from Ψ_k^{2n}. Individual coefficients $\delta = \Delta_i$ of distance Equation 4 will have distribution equivalent to*

$$\delta = \sum_{i=1}^{2n} \varepsilon_i \varepsilon'_i. \tag{12}$$

Proof. When we investigate the multiplication rule of Equation 1, we see that each coefficient of independent polynomials $\{\mathbf{a}, \mathbf{b}, \mathbf{e}, \mathbf{e}'\}$ (or its inverse) in Δ is used in computation of each $\Delta_i = \delta$ exactly once. One may equivalently pick coefficients of $\varepsilon, \varepsilon'$ from $\{\pm\mathbf{e}, \pm\mathbf{e}', \pm\mathbf{s}_A, \pm\mathbf{s}_B\}$, without repetition. Therefore coefficients of $\varepsilon_i, \varepsilon'_i$ are independent and have distribution Ψ_k. □

Independence Assumption. Even though all of the variables in the sum of individual element $\delta = \Delta_i$ are independent in Equation 12, they are reused in other sums for $\Delta_j, i \neq j$. Therefore, while the average-case distribution of each one of the n coefficients of Δ is the same and precisely analyzable, they are not fully independent. In this work we perform error analysis on a single coefficient and then simply expand it to the whole vector. This independence assumption is analogous to our extension of LWE security properties to Ring-LWE with more structure and less independent variables.

The assumption is supported by our strictly bound error distribution Ψ_k (when using discrete Gaussian distributions, which are infinite up to a tail bound, a few highly anomalous values would be more likely to cause multiple errors) and the structure of convolutions of signed random vectors (Equation 1). Our error estimate has a significant safety margin, however.

Estimation via Central Limit Theorem. The distribution of the product from two random variables from Ψ_k in Equation 12 is no longer binomial. Clearly its range is $[-k^2, k^2]$, but not all values are possible; for example, primes $p > k$ cannot occur in the product. However, it is easy to verify that the product is zero-centered and its standard deviation is exactly

$$\sigma = \sqrt{\sum_{i=-k}^{k} \sum_{j=-k}^{k} \frac{\binom{2k}{k+i}\binom{2k}{k+j}}{2^{4k}} (ij)^2} = \frac{k}{2}. \tag{13}$$

Hence, we may estimate δ of Equation 12 using the Central Limit Theorem as a Gaussian distribution with deviation

$$\sigma = \frac{k}{2}\sqrt{2n} \tag{14}$$

With our parameter selection this yields $\sigma \approx 362.0386$ (variance $\sigma^2 = 2^{17}$). Figure 3 illustrates this error distribution.

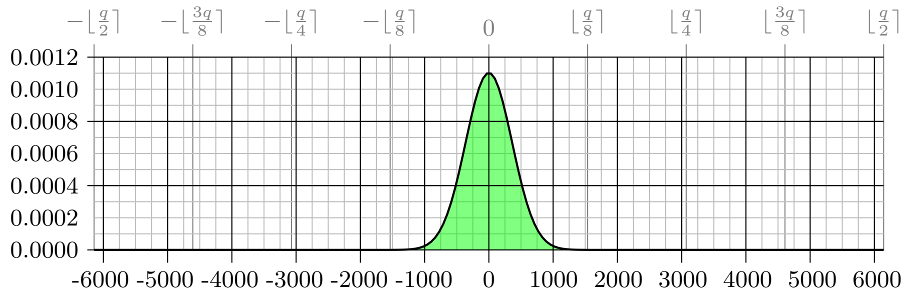

Fig. 3. The error distribution E of $\delta = x_i - y_i$ (which we compute with high precision) is bell-shaped with variance $\sigma^2 = 2^{17}$. Its statistical distance to corresponding discrete Gaussian (with same σ) is $\approx 2^{-12.6}$, which has a significant effect on the bit failure rate. This is why we compute the discrete distributions numerically.

More precise computation via convolutions. The distribution of $X = \varepsilon_i \varepsilon_i'$ in Equation 12 is far from being "Bell-shaped" – its (total variation) statistical distance to a discrete Gaussian (with the same $\sigma = 8$) is ≈ 0.307988.

We observe that since our domain $\mathbb{Z}_q$ is finite, we may always perform full convolutions between statistical distributions of independent random variables X and Y to arrive at the distribution of $X+Y$. The distributions can be represented as vectors of q real numbers (which are non-negative and add up to 1).

In order to get the exact shape of the error distribution we start with X, which is a "square" of Ψ_{16} and can be computed via binomial coefficients, as is done in Equation 13. The error distribution (Equation 12) is a sum $X + X + \cdots + X$ of $2n$ independent variables from that distribution. Using the convolution summing rule we can create a general "scalar multiplication algorithm" (analogous to square-and-multiply exponentiation) to quickly arrive at $E = 2048 \times X$.

We implemented finite distribution evaluation arithmetic in 256-bit floating point precision using the GNU MPFR library[3]. From these computations we know that the statistical distance of E to a discrete Gaussian with (same) $\sigma^2 = 2^{17}$ is approximately 0.0001603 or $2^{-12.6}$.

Proposition 1. *Bit selection mechanism of Section 3.1 yields unbiased shared secret bits $k = k'$ if $\mathbf{y}$ is uniform. Discrete failure rate for individual bits $k \neq k'$ can be computed with high precision in our instance.*

Proof. Consider Bob's k value from in Equation 7, Bob's c and Alice's k' from Equation 10, and the four equiv-probable SafeBits ranges in Equation 9. With our $q = 12289$ instantiation the four possible $k \neq k'$ error conditions are:

[3] The GNU MPFR is a widely available, free C library for multiple-precision floating-point computations with correct rounding: `http://www.mpfr.org/`

Failure Case	**Bob's y_i range for Y**	**Alice's Failing x_i**
$k = 0, c = 0, k' = 1$	$[1536 - b, 1536 + b]$	$[4609, 10752]$
$k = 0, c = 1, k' = 1$	$[4608 - b, 4608 + b]$	$[0, 1535] \cup [7681, 12288]$
$k = 1, c = 0, k' = 0$	$[7680 - b, 7680 + b]$	$[0, 4608] \cup [10753, 12288]$
$k = 1, c = 1, k' = 0$	$[10752 - b, 10752 + b]$	$[1536, 7680]$

We examine each case separately (See Figure 2). Since the four non-overlapping y_i ranges are of the same size $2b+1$ and together constitute all selectable points $d_i = 1$ (Equation 9), the distribution of $k = k'$ is uniform. Furthermore, bit fail probability $k \neq k'$ is the average of these four cases. For each case, compute distribution Y which is uniform in the range of y_i. Then convolute it with error distribution to obtain $X = Y + E$, the distribution of x_i. The probability of failure is the sum of probabilities in X in the corresponding x_i failure range. $\square$

Parameter Selection for Instantiation. As can be seen in Figure 4, the relationship between window size b and bit failure rate is almost exponential.

Some representative window sizes and payloads are given in Table 1, which also puts our selection $b = 799$ in context. Five-error correction (Section 4) lowers the message failure probability to roughly $(2^{-27})^5 \approx 2^{-135}$ or even lower as 99% of six-bit errors are also corrected. We therefore meet the 2^{-128} message failure requirement with some safety margin.

Table 1. Potential window b sizes for safe bit selection (Equation 9) for different payload sizes. We target a payload of 496 bits, of which 256 are actual key bits and 240 bits are used to encrypt a five-error correcting code from XE5.

Payload bits[*] $m \approx r \times n$	**Selection Window** b	**Selection Ratio** $r = \frac{4(2b+1)}{q}$	**Bit fail Probability** p	**Payload Failure** $1 - (1-p)^m$
128	191	0.124664	$2^{-51.4715}$	$2^{-44.4715}$
256	383	0.249654	$2^{-46.5521}$	$2^{-38.5521}$
384	575	0.374644	$2^{-41.5811}$	$2^{-32.9962}$
496[†]	799	0.520465	$2^{-36.0359}$	$2^{-27.0818}$
512	767	0.499634	$2^{-36.8063}$	$2^{-27.8063}$
768	1151	0.749613	$2^{-28.1151}$	$2^{-18.5302}$
1024	1535	0.999593	$2^{-20.7259}$	$2^{-10.7263}$

[*] This is the minimum number of payload bits you get with 50% probability. The actual number is binomially distributed with density $f(k) = \binom{n}{k} r^k (1-r)^{n-k}$. Probability of at least m bits is therefore $\sum_{k=m}^{n} f(k)$.
[†] The payload could be 533 bits with 50% probability. We get 496 bits with 99% probability – this safety margin was chosen to minimize repetition rate (to $\approx \frac{1}{100}$).

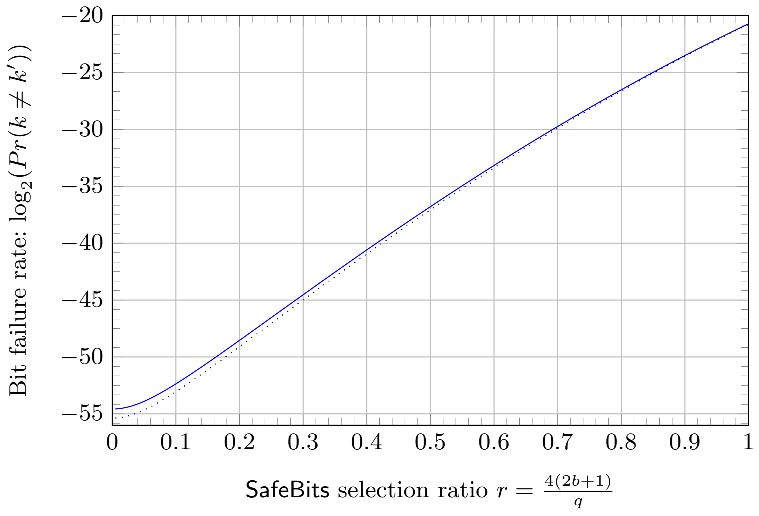

Fig. 4. Relationship between individual bit failure rate and the selection window b. Dotted line is the rate derived from Gaussian approximation – it's up to $2\times$ lower.

4 Constant-Time Error Correction

We note that in our application the error correction mechanism operates on secret data. As with all other components of the scheme it is highly desirable that decoding can be implemented with an algorithm that requires constant processing time regardless of number of errors present. We are not aware of satisfactory constant-time decoding algorithms for BCH, Reed-Solomon, or other standard block multiple-error correcting codes [30,46].

We chose to design a linear block code specifically for our application. The design methodology is general, and a similar approach was used by the Author in the TRUNC8 Ring-LWE lightweight authentication scheme [41]. However, that work did not provide a detailed justification for the error correction code.

Definition 3. XE5 *has a block size of 496 bits, out of which 256 bits are payload bits* $\mathbf{p} = (p_0, p_1, \cdots, p_{255})$ *and 240 provide redundancy* $\mathbf{r}$. *Redundancy is divided into ten subcodewords* $r_0, r_1, \cdots, r_9$ *of varying bit length* $|r_i| = L_i$ *with*

$$(L_0, L_1, \cdots, L_9) = (16, 16, 17, 31, 19, 29, 23, 25, 27, 37). \tag{15}$$

Bits in each r_i *are indexed* $r_{(i,0)}, r_{(i,1)}, \cdots, r_{(i,L_i-1)}$. *Each bit* $k \in [0,\ L_0 - 1]$ *in first subcodeword* r_0 *satisfies the parity equation*

$$r_{0,k} = \sum_{j=0}^{15} p_{(16k+j)} \pmod 2 \tag{16}$$

and bits in $r_1, r_2, \cdots, r_9$ satisfy the parity congruence

$$r_{i,k} = \sum_{j-k \,\mid\, L_i} p_j \pmod 2. \tag{17}$$

We see that $r_{0,k}$ in Equation 16 is the parity of $k+1$:th block of 16 bits, while the $r_{i,k}$ in Equation 17 is parity of all p_j at congruent positions $j \equiv k \pmod{L_i}$.

Definition 4. *For each payload bit position p_i we can assign corresponding integer "weight" $w_i \in [0, 10]$ as a sum*

$$w_i = r_{(0,\lfloor i/16 \rfloor)} + \sum_{j=1}^{9} r_{(j, i \bmod L_j)}. \tag{18}$$

Lemma 2. *If message payload* $\mathbf{p}$ *only has a single nonzero bit p_e, then $w_e = 10$ and $w_i \leq 1$ for all $i \neq e$.*

Proof. Since each $L_i \geq \sqrt{|\mathbf{p}|}$ and all $L_{i \geq 1}$ are coprime (each is a prime power) it follows from the Chinese Remainder Theorem that any nonzero $i \neq j$ pair can satisfy both $r_{i, a \bmod L_i} = 1$ and $r_{j, a \bmod L_j} = 1$ only at $a = e$. Similar argument can be made for pairing $r_{0,a}$ with $r_{i \geq 1}$. Since the residues can be true pairwise only at e, weight w_a cannot be 2 or above when $a \neq e$. The $w_e = 10$ case follows directly from the Definition 3. □

Definition 5. *Given XE5 input block $\mathbf{p} \mid \mathbf{r}$, we deliver a redundancy check $\mathbf{r}'$ from $\mathbf{p}$ via Equations 16 and 17. Furthermore we have distance $\mathbf{r}^{\Delta} = \mathbf{r} \oplus \mathbf{r}'$. Payload distance weight vector $\mathbf{w}^{\Delta}$ is derived from $\mathbf{r}^{\Delta}$ via Equation 18.*

Since the code is entirely linear, Lemma 2 implies a direct way to correct a single error in $\mathbf{p}$ using Definition 5 – just flip bit p_x at position x where $w_x^{\Delta} = 10$. In fact any two redundancy subcodewords r_i and r_j would be sufficient to correct a single error in the payload; it's where $w_i^{\Delta} \geq 2$. It's easy to see if the single error would be in the redundancy part (r_i or r_j) instead of the payload, this is not an issue since in that case $w_x^{\Delta} \leq 1$ for all x. This type of reasoning leads to our main error correction strategy that is valid for up to five errors:

Theorem 1. *Let $\mathbf{b} \mid \mathbf{r}$ be an XE5 message block as in Definition 5. Changing each bit p_i when $w_i^{\Delta} \geq 6$ will correct a total of five bit errors in the block.*

Proof. We first note that if all five errors are in the redundancy part $\mathbf{r}$, then $w_i^{\Delta} \leq 5$ and no modifications in payload are done. If there are 4 errors in $\mathbf{r}$ and one in payload we still have $w_x^{\Delta} \geq 6$ at the payload error position p_x, etc. For each payload error p_x, each of ten subcodeword $\mathbf{r_i}$ will contribute one to weight w_x^{Δ} unless there is another congruent error p_y – i.e. we have $\lfloor x/16 \rfloor = \lfloor y/16 \rfloor$ for r_0 or $x \equiv y \pmod{L_i}$ for $r_{i \geq 1}$. Four errors cannot generate more than four such congruences (due to properties shown in the proof of Lemma 2), leaving fifth correctable via remaining six subcodewords ($w_i^{\Delta} \geq 6$). □

In order to verify the correctness of our implementation, we also performed a full exhaustive test (search space $\sum_{i=0}^{5} \frac{496!}{i!(496-i)!} \approx 2^{37.8}$). Experimentally XE5 corrects 99.4% of random 6-bit errors and 97.0% of random 7-bit errors.

Efficient constant-time implementation. The code generation and error correcting schemes can be implemented in bit-sliced fashion, without conditional clauses or table-lookups on secret data. Algorithm listings 5 and 6 in Appendix A provide an implementation and demonstrate the basic techniques for this.

The block is encoded simply as a 496-bit concatenation $\mathbf{p} \mid \mathbf{r}$. The reason for the ordering of L_i in Equation 15 is so that they can be packed into byte boundaries: $17 + 31 = 48$, $19 + 29 = 48$, $23 + 25 = 48$ and $27 + 37 = 64$.

5 Instantiation and Implementation

Our instantiation – codenamed HILA5[4] – shares core Ring-LWE parameters with various "New Hope" variants, but uses an entirely different error management strategy. Algorithm 1 contains a pseudocode overview of the entire HILA5 Key Encapsulation Mechanism, using a number of auxiliary primitives and functions.

Notation and auxiliary functions. We represent elements of $\mathcal{R}$ in two different domains; the normal polynomial representation $\mathbf{v}$ and Number Theoretic Transform representation $\hat{\mathbf{v}}$. Convolution (polynomial multiplication) in the NTT domain is a linear-complexity operation, written $\hat{\mathbf{x}} \circledast \hat{\mathbf{y}}$. Addition and subtraction work as in normal representation. The transform and its inverse are denoted $\mathsf{NTT}(\mathbf{v}) = \hat{\mathbf{v}}$ and $\mathsf{NTT}^{-1}(\hat{\mathbf{v}}) = \mathbf{v}$, respectively. The transform algorithm is adopted from Longa and Naehrig [27], and not detailed here.

The hash $h(x)$ is SHA3-256 [23]. Function $\mathsf{Parse}()$ (Algorithm 2) deterministically samples a uniform $\hat{\mathbf{g}} \in \mathcal{R}$ based on arbitrary seed s using SHA3's XOF mode SHAKE-256 [23]. While New Hope uses the slightly faster SHAKE-128 for this purpose, we consistently use SHAKE-256 or SHA3-256 in all parts of HILA5. For sampling modulo q we use the $5q$ trick suggested by Gueron and Schlieker in [24]. Binomial distribution values Ψ_{16} can be computed directly from 32 random bits per Definition 2.

Bob's reconciliation function $\mathsf{SafeBits}()$ (Algorithm 3) captures Equations 7 and 9 from Section 3. Conversely Alice's reconciliation function $\mathsf{Select}()$ (Algorithm 4) captures Equation 10.

The XE5 error correction functions $\mathbf{r} = \mathsf{XE5\_Cod}(\mathbf{p})$ and $\mathbf{p}' = \mathsf{XE5\_Fix}(\mathbf{r} \oplus \mathbf{r}') \oplus \mathbf{p}$ are defined in Section 4 and Algorithms 5 and 6. Here we have "error key" $\mathbf{k} = \mathbf{p} \mid \mathbf{r}$ with the payload key $\mathbf{p} \in \{0,1\}^{256}$ and redundancy $\mathbf{r} \in \{0,1\}^{240}$.

Encoding – shorter messages Ring elements, whether or not in NTT domain, are encoded into $|\mathcal{R}| = \lceil \log_2 q \rceil n$ bits $= 1,792$ bytes. This is the private key size. Alice's public key PK with a 256-bit seed s and $\hat{\mathsf{A}}$ is $1,824$ bytes. Ciphertext CT is $|\mathcal{R}| + n + m + |\mathbf{r}|$ bits or $2,012$ bytes; 36 bytes less than New Hope [4], 196 bytes less than the variant of [3], and $1,572$ bytes less than LP11 [26].

[4] *Hila* is Finnish for a lattice. HILA5 – especially when written as "Hila V" – also refers to *hilavitkutin*, a nonsensical placeholder name usually meaning an unidentified, incomprehensibly complicated apparatus or gizmo.

Algorithm 1 The HILA5 KEM Components and (key exchange) protocol flow.

```
                         Alice                                                  Bob
  (PK, SK) ← KeyGen()
    $\mathbf{s} \xleftarrow{\$} \{0,1\}^{256}$                    Public random seed.
    $\hat{\mathbf{g}} \leftarrow \mathsf{Parse}(s)$                       Expand to "generator" in NTT domain.
    $\mathbf{a} \xleftarrow{\$} \psi_{16}^{n}$                           Randomize Alice's secret key.
    $\hat{\mathbf{a}} \leftarrow \mathsf{NTT}(\mathbf{a})$                          Transform it.
    $\mathbf{e} \xleftarrow{\$} \psi_{16}^{n}$                           Generate masking noise.
    $\hat{\mathbf{A}} \leftarrow \hat{\mathbf{g}} \circledast \hat{\mathbf{a}} + \mathsf{NTT}(\mathbf{e})$            Compute Alice's public key in NTT domain.
    $\rightarrow$ Send $\mathsf{PK} = \mathbf{s} \mid \hat{\mathbf{A}}$                  $\xrightarrow{\mathsf{PK}}$
    $\downarrow$ Keep $\mathsf{SK} = \hat{\mathbf{a}}$ and $h(\mathsf{PK})$.                          (CT, K) ← Encaps(PK)
            Randomize Bob's ephemeral secret key.    $\mathbf{b} \xleftarrow{\$} \psi_{16}^{n}$
                                    Transform it.    $\hat{\mathbf{b}} \leftarrow \mathsf{NTT}(\mathbf{b})$
                 Bob's version of shared secret.    $\mathbf{y} \leftarrow \mathsf{NTT}^{-1}(\hat{\mathbf{A}} \circledast \hat{\mathbf{b}})$
           Get payload and reconciliation values.    $(\mathbf{d}, \mathbf{k}, \mathbf{c}) \leftarrow \mathsf{SafeBits}(\mathbf{y})$
      (Fail hard after more than a dozen restarts.)    If $\mathbf{k} = \mathsf{FAIL}$ restart Encaps()
     Split to payload and redundancy "keystream".    $\mathbf{p} \mid \mathbf{z} = \mathbf{k}$
              Error correction code, encrypt it.    $\mathbf{r} \leftarrow \mathsf{XE5\_Cod}(\mathbf{p}) \oplus \mathbf{z}$
             Get "generator" from Alice's seed.    $\hat{\mathbf{g}} \leftarrow \mathsf{Parse}(s)$
                       Generate masking noise.    $\mathbf{e}' \xleftarrow{\$} \psi_{16}^{n}$
            Compute Bob's one-time public value.    $\hat{\mathbf{B}} \leftarrow \hat{\mathbf{g}} \circledast \hat{\mathbf{b}} + \mathsf{NTT}(\mathbf{e}')$
                                         $\xleftarrow{\mathsf{CT}}$    $\leftarrow$ Send $\mathsf{CT} = \hat{\mathbf{B}} \mid \mathbf{d} \mid \mathbf{c} \mid \mathbf{r}$
Hash the shared secret. V is a version identifier.    $\downarrow \mathsf{K} = h(\ \mathsf{V} \mid h(\mathsf{PK}) \mid h(\mathsf{CT}) \mid \mathbf{p}\ )$
  K ← Decaps(SK, CT)
    $\mathbf{x} \leftarrow \mathsf{NTT}^{-1}(\hat{\mathbf{B}} \circledast \hat{\mathbf{a}})$                 Alice's version of the shared secret.
    $\mathbf{k}' \leftarrow \mathsf{Select}(\mathbf{x}, \mathbf{d}, \mathbf{c})$                 Get payload with the help of reconciliation.
    $\mathbf{p}' \mid \mathbf{z}' = \mathbf{k}'$                          Split to payload and redundancy "keystream".
    $\mathbf{r}' \leftarrow \mathsf{XE5\_Cod}(\mathbf{p}')$                     Get error correction code from Alice's version.
    $\mathbf{p}'' \leftarrow \mathsf{XE5\_Fix}(\mathbf{r} \oplus \mathbf{z}' \oplus \mathbf{r}') \oplus \mathbf{p}'$    Decrypt and apply Bob's error correction.
    $\downarrow \mathsf{K}' = h(\ \mathsf{V} \mid h(\mathsf{PK}) \mid h(\mathsf{CT}) \mid \mathbf{p}''\ )$ Upon success shared secret K = K′.
```

5.1 Encryption: From noisy Diffie-Hellman to noisy ElGamal

Modification of the scheme for public-key encryption is straightforward. Compared to the more usual "LP11" Ring-LWE Public Key Encryption construction [26] our reconciliation approach saves about 44 % in ciphertext size.

For minimal ciphertext expansion with only passive security, one may replace SHA3 at the end of Encaps() and Decaps() with SHAKE-256 and use the output K as keystream to XOR with plaintext to produce ciphertext or vice versa.

However, for active security we suggest that K is used as keying material for an AEAD (Authenticated Encryption with Associated Data) [39] scheme such as AES256-GCM [21,22] or Keyak [11] in order to protect message integrity. See Section 5 of [35] for details of the formal security argument.

5.2 Security

In Algorithm 1 the error correction data $\mathbf{r}$ is transmitted encrypted with shared secret bits $\mathbf{z}$, and therefore does not leak entropy about the actual key data $\mathbf{p}$, also derived from the shared secret. Shared secret bits are unbiased. The shared key K also includes plaintext PT and ciphertext CT in the final hash to protect against a class of active attacks.

Our reconciliation mechanism has no effect on the security against (quantum) lattice attacks, so estimates in [4] are applicable (2^{255} quantum security, with 2^{199} attacks plausible). Pre-image security is expected from SHA3 and SHAKE-256 in HILA5. Breaking the construction via these algorithms, if possible, would require approximately 2^{166} logical-qubit-cycles [6,18,45].

This leads us to claim that the HILA5 meets NIST's "Category 5" post-quantum security requirement ([31], Section 4.A.5): Compromising key K in a passive attack requires computational resources comparable to or greater than those required for key search on a block cipher with a 256-bit key (e.g. AES 256). The scheme can also be made secure against active attacks with an appropriate AEAD mechanism, as discussed in Section 5.1.

The scheme has been designed from ground-up to be resistant against timing and side-channel attacks. The sampler Ψ_{16} is constant-time, as is our error correction code XE5. Ring arithmetic can also be implemented in constant time, but leakage can be further minimized via blinding [40] (Section 6).

5.3 Performance

Our main contribution, a new reconciliation mechanism, has a minor effect on performance of the scheme, but a significant impact on failure probability.

We chose to recycle "New Hope" NTT (n, q) and sampler (q, Ψ_{16}) parameters as they have been extensively vetted for security against lattice attacks and originally selected for performance. A significant effort has subsequently been dedicated (by several research groups) for the optimization of NTT and Sampler components. There already exists a number of permissively licensed open source implementations and a body of publications detailing specific optimizations or these particular NTT and sampler parameters.

There are at least two very fast AVX2 Intel optimized versions of the NTT core and Ψ_{16} sampler – the original [4] and one by Longa and Naehrig [27]. Further sampler optimizations have been suggested in [24]. Implementations have also been reported for ARM Cortex-M microcontrollers [5], ARM NEON SIMD instruction set [44], and for FPGA hardware [25].

New Hope has also been integrated in TLS stacks and cryptographic toolkits in 2016-17 by Google (BoringSSL), the Open Quantum Safe project, Microsoft (MS Lattice Library), ISARA Corporation, and possibly others.

Our prototype implementation was integrated into a branch of the Open Quantum Safe (OQS) framework[5] where it was benchmarked against other

[5] OQS: `https://openquantumsafe.org/` HILA5 patch: `https://mjos.fi/hila5`

Table 2. Performance of HILA5 within the Open Quantum Safe test bench C implementations [43]. The slight (under 4%) performance difference to New Hope is principally due to our use of error correction and SHAKE-256. Testing was performed on an Ubuntu 17.04 workstation with Core i7-6700 @ 3.40 GHz. For reference and scale we are also including RSA numbers with OpenSSL 1.0.2 (system default) on this target. A single Elliptic Curve DH operation requires $45.4\mu s$ for the NIST P-256 curve (highly optimized implementation), and $331.7\mu s$ for NIST P-521. Full source code of our implementation is available at `https://mjos.fi/hila5/`

	Init	**Public**	**Private**	**Key Ex.**	**Data**
Scheme	KeyGen()	Encaps()	Decaps()	Total	Tot. xfer
RLWE New Hope [4]	$60.7\mu s$	$92.3\mu s$	$16.2\mu s$	$169.2\mu s$	3,872 B
RLWE Hila5 [This work]	$68.7\mu s$	$89.9\mu s$	$16.9\mu s$	$175.4\mu s$	3,836 B
RLWE BCNS15 [13]	$951.6\mu s$	$1546\mu s$	$196.9\mu s$	$2.694ms$	8,320 B
LWE Frodo [12]	$2.839ms$	$3.144ms$	$84.9\mu s$	$6.068ms$	22,568 B
SIDH CLN16 [16]	$10.3ms$	$22.9ms$	$9.853ms$	$43.1ms$	1,152 B
RSA-2048 [OpenSSL]	$60ms$	$15.9\mu s$	$559.9\mu s$	N/A	N/A
RSA-4096 [OpenSSL]	$400ms$	$55.7\mu s$	$3.687ms$	N/A	N/A

quantum-resistant KEM schemes [43]. Table 2 summarizes the performance of our implementation. It is essentially the same as New Hope C implementation variants, with slightly smaller message size.

6 Conclusions

With NIST's ongoing post-quantum standardization effort, the practical performance, implementation security, and reliability of Ring-LWE public key encryption and key exchange implementations have emerged as major research area.

We have described an improved general reconciliation scheme for Ring-LWE. Our SafeBits selection technique avoids randomized "blurring" of previous Peikert's, Ding's, and New Hope reconciliation schemes to achieve unbiased secret bits, therefore needing less randomness. We have given detailed, precise arguments for its effectiveness.

The failure probability can also be addressed using error correcting codes. For this purpose we described a class of linear forward-error correcting block codes that can be implemented without branches or table lookups on secret data, guarding against side-channel attacks.

We instantiate the new techniques in "HILA5" with well-studied and efficient "New Hope" Ring-LWE parameters. The new reconciliation methods are shown to have minimal negative performance impact, while significantly improving the failure probability. The failure probability, which is shown to be under 2^{-128}, allows the KEM to be used for actively secure public key encryption in addition to interactive key exchange protocols. Furthermore the message sizes are shorter than with previous proposals, especially when used for public key encryption.

We claim that the HILA5 instantiation meets "Category 5" NIST PQC security requirements as a KEM and public key encryption scheme. Furthermore, it has been explicitly designed to be robust against side-channel attacks.

Acknowledgements. The author wishes to thank the DARKMATTER Crypto Team and Dr. Najwa Aaraj for providing feedback and supporting this research.

References

1. Carlos Aguilar, Philippe Gaborit, Patrick Lacharme, Julien Schrek, and Gilles Zémor. Noisy Diffie-Hellman protocols, May 2010. Talk given by Philippe Gaborit at PQCrypto 2010 "Recent Results" session. URL: `https://pqc2010.cased.de/rr/03.pdf`.
2. Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, October 2015. URL: `https://eprint.iacr.org/2015/046`, `doi:10.1515/jmc-2015-0016`.
3. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Newhope without reconciliation. IACR ePrint 2016/1157, December 2016. URL: `https://eprint.iacr.org/2016/1157`.
4. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange – A new hope. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 16*, pages 327–343. USENIX Association, August 2016. Full version available as `https://eprint.iacr.org/2015/1092`. URL: `https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_alkim.pdf`.
5. Erdem Alkim, Philipp Jakubeit, and Peter Schwabe. A new hope on ARM Cortex-M. IACR ePrint 2016/758, 2016. URL: `https://eprint.iacr.org/2016/758`.
6. Matthew Amy, Olivia Di Matteo, Vlad Gheorghiu, Michele Mosca, Alex Parent, and John Schanck. Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3. IACR ePrint 2016/992, 2016. To appear in Proc. SAC 2016. URL: `http://eprint.iacr.org/2016/992`.
7. Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 595–618. Springer, 2009. `doi:10.1007/978-3-642-03356-8_35`.
8. Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In Hugo Krawczyk, editor, *CRYPTO 1998*, volume 1462 of *LNCS*, pages 26–45. Springer, 1998. URL: `https://www.di.ens.fr/~pointche/Documents/Papers/1998_crypto.pdf`, `doi:10.1007/BFb0055718`.
9. Charles H. Bennett and Gilles Brassard. Quantum cryptography: Public key distribution and coin tossing. In *Proceedings of IEEE International Conference on Computers, Systems and Signal Processing,*, pages 175–179. IEEE, December 1984. URL: `http://researcher.watson.ibm.com/researcher/files/us-bennetc/BB84highest.pdf`.
10. Charles H. Bennett, Gilles Brassard, and Jean-Marc Robert. Privacy amplification by public discussion. *Siam Journal on Computing*, 17(2):210–229, April 1988. `doi:10.1137/0217014`.

11. Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Caesar submission: Keyak v2, September 2016. CAESAR Candidate Specification. URL: http://keyak.noekeon.org/.
12. Joppe Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from LWE. In *ACM CCS 2016*, pages 1006–1018. ACM, October 2016. Full version available as IACR ePrint 2016/659. URL: https://eprint.iacr.org/2016/659, doi:10.1145/2976749.2978425.
13. 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 *IEEE S & P 2015*, pages 553–570. IEEE Computer Society, 2015. Extended version available as IACR ePrint 2014/599. URL: https://eprint.iacr.org/2014/599, doi:10.1109/SP.2015.40.
14. Gilles Brassard and Louis Salvail. Secret-key reconciliation by public discussion. In Tor Helleseth, editor, *EUROCRYPT 1993*, volume 765 of *LNCS*, pages 410–423. Springer, 1993. doi:10.1007/3-540-48285-7_35.
15. Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. Report on post-quantum cryptography. NISTIR 8105, April 2016. URL: http://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8105.pdf, doi:10.6028/NIST.IR.8105.
16. Craig Costello, Patrick Longa, and Michael Naehrig. Efficient algorithms for supersingular isogeny Diffie-Hellman. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016*, volume 9814 of *LNCS*, pages 572–601. Springer, 2016. URL: https://eprint.iacr.org/2016/413, doi:10.1007/978-3-662-53018-4_21.
17. Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003. URL: http://www.shoup.net/papers/cca2.pdf, doi:10.1137/S0097539702403773.
18. Jan Czajkowski, Leon Groot Bruinderink, Andreas Hülsing, and Christian Schaffner. Quantum preimage, 2nd-preimage, and collision resistance of SHA3. IACR ePrint 2017/302, 2017. URL: https://eprint.iacr.org/2017/302.
19. Jintai Ding. Improvements on cryptographic systems using pairing with errors, June 2015. Application PCT/CN2015/080697. URL: https://patents.google.com/patent/WO2015184991A1/en.
20. Jintai Ding, Xiang Xie, and Xiaodong Lin. A simple provably secure key exchange scheme based on the learning with errors problem. IACR ePrint 2012/688, 2012. URL: https://eprint.iacr.org/2012/688.
21. Morris Dworkin. Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC. NIST Special Publication 800-38D, November 2007. URL: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38D.pdf, doi:10.6028/NIST.SP.800-38D.
22. FIPS. Specification for the Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, November 2001. URL: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.
23. FIPS. SHA-3 standard: Permutation-based hash and extendable-output functions. Federal Information Processing Standards Publication 202, August 2015. doi:10.6028/NIST.FIPS.202.
24. Shay Gueron and Fabian Schlieker. Speeding up R-LWE post-quantum key exchange. IACR ePrint 2016/467, 2016. URL: https://eprint.iacr.org/2016/467.

25. Po-Chun Kuo, Wen-Ding Li, Yu-Wei Chen, Yuan-Che Hsu, Bo-Yuan Peng, Chen-Mou Cheng, and Bo-Yin Yang. Post-quantum key exchange on FPGAs. IACR ePrint 2017/690, 2017. URL: `https://eprint.iacr.org/2017/690`.
26. Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In Aggelos Kiayias, editor, *CT-RSA 2011*, volume 6558 of *LNCS*, pages 319–339. Springer, 2011. `doi:10.1007/978-3-642-19074-2_21`.
27. Patrick Longa and Michael Naehrig. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In Sara Foresti and Giuseppe Persiano, editors, *CANS 2016*, volume 10052 of *LNCS*, pages 124–139. Springer, 2016. URL: `https://eprint.iacr.org/2016/504`, `doi:10.1007/978-3-319-48965-0_8`.
28. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, 2010. `doi:10.1007/978-3-642-13190-5_1`.
29. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-LWE cryptography. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 35–54. Springer, 2013. Full version available as IACR ePrint 2013/293. URL: `https://eprint.iacr.org/2013/293`, `doi:10.1007/978-3-642-38348-9_3`.
30. F. Jessie MacWilliams and Neil J.A. Sloane. *The theory of error-correcting codes*. North-Holland, 1977.
31. NIST. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process. Official Call for Proposals, National Institute for Standards and Technology, December 2016. URL: `http://csrc.nist.gov/groups/ST/post-quantum-crypto/documents/call-for-proposals-final-dec-2016.pdf`.
32. NSA/CSS. Information assurance directorate: Commercial national security algorithm suite and quantum computing FAQ, January 2016. URL: `https://www.iad.gov/iad/library/ia-guidance/ia-solutions-for-classified/algorithm-guidance/cnsa-suite-and-quantum-computing-faq.cfm`.
33. Henri J. Nussbaumer. Fast polynomial transform algorithms for digital convolution. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 28:205–215, 1980. `doi:10.1109/TASSP.1980.1163372`.
34. Chris Peikert. Some recent progress in lattice-based cryptography, March 2009. Invited Talk given at TCC 2009. URL: `http://www.cc.gatech.edu/fac/cpeikert/pubs/slides-tcc09.pdf`, `doi:10.1007/978-3-642-00457-5_5`.
35. Chris Peikert. Lattice cryptography for the internet. In Michele Mosca, editor, *PQCrypto 2014*, volume 8772 of *LNCS*, pages 197–219. Springer, 2014. URL: `https://eprint.iacr.org/2014/070`, `doi:10.1007/978-3-319-11659-4_12`.
36. John Proos and Christof Zalka. Shor's discrete logarithm quantum algorithm for elliptic curves. *Quantum Information & Computation*, 3(4):317–344, July 2003. Updated version available on arXiv. URL: `https://arxiv.org/abs/quant-ph/9508027`.
37. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC '05*, pages 84–93. ACM, May 2005. `doi:10.1145/1060590.1060603`.
38. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 56(6):34:1–34:40, September 2009. `doi:10.1145/1568318.1568324`.
39. Phillip Rogaway. Authenticated-encryption with associated-data. In *ACM CCS 2002*, pages 98–107. ACM Press, 2002. URL: `http://web.cs.ucdavis.edu/~rogaway/papers/ad.pdf`, `doi:10.1145/586110.586125`.

40. Markku-Juhani O. Saarinen. Arithmetic coding and blinding countermeasures for lattice signatures. *Journal of Cryptographic Engineering*, 2017. To appear. URL: `http://rdcu.be/oHun`, `doi:10.1007/s13389-017-0149-6`.
41. Markku-Juhani O. Saarinen. Ring-LWE ciphertext compression and error correction: Tools for lightweight post-quantum cryptography. In *Proceedings of the 3rd ACM International Workshop on IoT Privacy, Trust, and Security*, IoTPTS '17, pages 15–22. ACM, April 2017. `doi:10.1145/3055245.3055254`.
42. Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proc. FOCS '94*, pages 124–134. IEEE, 1994. Updated version available on arXiv. URL: `https://arxiv.org/abs/quant-ph/9508027`, `doi:10.1109/SFCS.1994.365700`.
43. Douglas Stebila and Michele Mosca. Post-quantum key exchange for the internet and the open quantum safe project. IACR ePrint 2016/1017, 2016. Based on the Stafford Tavares Invited Lecture at Selected Areas in Cryptography (SAC) 2016 by D. Stebila. URL: `https://eprint.iacr.org/2016/1017`.
44. Silvan Streit and Fabrizio De Santis. Post-quantum key exchange on ARMv8-A – a new hope for NEON made simple. IACR ePrint 2017/388, 2017. URL: `https://eprint.iacr.org/2017/388`.
45. Dominique Unruh. Collapsing sponges: Post-quantum security of the sponge construction. IACR ePrint 2017/282, 2017. URL: `https://eprint.iacr.org/2017/282`.
46. Jacobus H. van Lint. *Introduction to Coding Theory*, volume 86 of *Graduate Texts in Mathematics*. Springer, 3rd edition, 1999. `doi:10.1007/978-3-642-58575-3`.

A Algorithmic Definitions

Algorithm 2 Parse(s): Deterministic sampling in ring $\mathcal{R}$ based on seed s.

Input: Seed value s.

```
1: $z \leftarrow \mathsf{SHAKE}-256(s)$                 Absorb the seed s into Keccak state.
2: for $i = 0, 1, \ldots n-1$ do
3:     repeat
4:         $t \leftarrow$ next 16 bits from $z$        z represents the (endless) output of XOF.
5:     until $t < 5q$                                  Acceptance rate is $\frac{5q}{2^{16}} \approx 93.76\%$.
6:     $\hat{g}_i \leftarrow t$                        No further transformation needed.
7: end for
```

Output: A ring element $\hat{\mathbf{g}}$ which is understood to be in NTT domain.

Algorithm 3 SafeBits($\mathbf{y}$): Determine Bob's key bit, reconciliation, and payload. HILA5 has $n = 1024$, $q = 12289$, selection bound $b = 799$, and payload $m = 496$.

Input: Bob's share $\mathbf{y} \in \mathcal{R}$.

```
 1: $j \leftarrow 0$, $\mathbf{d} \leftarrow 0^n$, $\mathbf{k} \leftarrow 0^m$, $\mathbf{c} \leftarrow 0^m$      Initialize.
 2: for $i = 0, 1, \ldots n-1$ do
 3:     $t \leftarrow y_i \bmod \lfloor \frac{q}{4} \rfloor$                     Position within the quadrant.
 4:     if $t \in \left[ \lfloor \frac{q}{8} \rfloor - b, \lfloor \frac{q}{8} \rfloor + b \right]$ then
 5:         $d_i \leftarrow 1$                                Mark selection bit.
 6:         $k_j \leftarrow \lfloor 2y_i/q \rfloor$                      Key bit (really just bound comparisons).
 7:         $c_j \leftarrow \lfloor 4y_i/q \rfloor \bmod 2$               Reconciliation bit (also just bounds).
 8:         $j \leftarrow j + 1$
 9:         if $j = m$ then
10:             return $(\mathbf{d}, \mathbf{k}, \mathbf{c})$               We have enough bits, done.
11:         end if
12:     end if
13: end for
14: return FAIL                            $j < m$: not enough bits ($< 1\%$ probability).
```

Output: Either three binary vectors $\mathbf{d} \in \{0,1\}^n$, $\mathbf{k} \in \{0,1\}^m$, $\mathbf{c} \in \{0,1\}^m$ or FAIL.

Algorithm 4 Select($\mathbf{x}, \mathbf{d}, \mathbf{c}$): Determine Alice's key bits.

Input: Alice's share $\mathbf{x} \in \mathcal{R}$.

Input: Bob's reconciliation vectors $\mathbf{d} \in \{0,1\}^n$ and $\mathbf{c} \in \{0,1\}^m$.

```
 1: $j \leftarrow 0$, $\mathbf{k} \leftarrow 0^m$                       Initialize.
 2: for $i = 0, 1, \ldots n-1$ do
 3:     if $d_i = 1$ then
 4:         if $c_j = 1$ then
 5:             $t \leftarrow x_i - \lfloor \frac{q}{8} \rceil$               Reconciliation $45^\circ$ anticlockwise.
 6:         else
 7:             $t \leftarrow x_i + \lfloor \frac{q}{8} \rceil$               Reconciliation $45^\circ$ clockwise.
 8:         end if
 9:         $k_j = \left\lfloor \frac{2}{q}(t \mod q) \right\rceil$          Really a conditional.
10:         $j \leftarrow j + 1$
11:         if $j = m$ then
12:             return $\mathbf{k}$                        Done.
13:         end if
14:     end if
15: end for
16: return FAIL                            $j < m$: not enough bits
```

Output: Either key bits $\mathbf{k} \in \{0,1\}^m$ or FAIL.

Algorithm 5 Constant time bit-slicing techniques for $\mathbf{r} = \mathsf{XE5\_Cod}(\mathbf{p})$.

```
// Field      subcodeword: r0  r1  r2  r3  r4  r5  r6  r7  r8  r9 (end)
// lengths.    bit offset:  0  16  32  49  80  99  128 151 176 203 240
const int xe5_len[10] = { 16, 16, 17, 31, 19, 29, 23, 25, 27, 37 };

// Compute redundancy r[] (XOR over original) from payload p[]

void xe5_cod(uint64_t r[4], const uint64_t p[4])
{
    int i, j, l;
    uint64_t x, t, ri[10];

    for (i = 0; i < 10; i++)                // initialize
        ri[i] = 0;

    for (i = 3; i >= 0; i--) {              // four words
        x = p[i];                           // payload
        for (j = 1; j < 10; j++) {
            l = xe5_len[j];                 // length
            t = (ri[j] << (64 % l));        // rotate
            t ^= x;                         // payload
            if (l < 32)                     // extra fold
                t ^= t >> (2 * l);
            t ^= t >> l;                    // fold
            ri[j] = t & ((1lu << l) - 1);   // mask
        }
        x ^= x >> 8;                        // parity of 16
        x ^= x >> 4;
        x ^= x >> 2;
        x ^= x >> 1;
        x &= 0x0001000100010001;            // four parallel
        x ^= (x >> (16 - 1)) ^ (x >> (32 - 2)) ^ (x >> (48 - 3));
        ri[0] |= (x & 0xF) << (4 * i);
    }

    // pack coefficients into 240 bits (note output the XOR)
    r[0] ^= ri[0] ^ (ri[1] << 16) ^ (ri[2] << 32) ^ (ri[3] << 49);
    r[1] ^= (ri[3] >> 15) ^ (ri[4] << 16) ^ (ri[5] << 35);
    r[2] ^= ri[6] ^ (ri[7] << 23) ^ (ri[8] << 48);
    r[3] ^= (ri[8] >> 16) ^ (ri[9] << 11);
}
```

Algorithm 6 Constant time bit-slicing techniques for $\mathbf{p} = \mathsf{XE5\_Fix}(\mathbf{r} \oplus \mathbf{r}') \oplus \mathbf{p}'$.

```
// Fix errors in p[] using redundancy in r[]

void xe5_fix(uint64_t p[4], const uint64_t r[4])
{
    int i, j, k, l;
    uint64_t x, t, ri[10];

    ri[0] = r[0];                               // unpack
    ri[1] = r[0] >> 16;
    ri[2] = r[0] >> 32;
    ri[3] = (r[0] >> 49) ^ (r[1] << 15);
    ri[4] = r[1] >> 16;
    ri[5] = r[1] >> 35;
    ri[6] = r[2];
    ri[7] = r[2] >> 23;
    ri[8] = (r[2] >> 48) ^ (r[3] << 16);
    ri[9] = r[3] >> 11;

    for (i = 0; i < 4; i++) {                  // four words
        for (j = 1; j < 10; j++) {
            l = xe5_len[j];                    // length
            x = ri[j] & ((1lu << l) - 1);      // mask
            x |= x << l;                       // expand
            if (l < 32)                        // extra unfold
                x |= (x << (2 * l));
            ri[j] = x;                         // store it
        }
        x = (ri[0] >> (4 * i)) & 0xF;          // parity mask for ri[0]
        x ^= (x << (16 - 1)) ^ (x << (32 - 2)) ^ (x << (48 - 3));
        x  = 0x0100010001000100 - (x & 0x0001000100010001);
        x &= 0x00FF00FF00FF00FF;
        x |= x << 8;

        for (j = 0; j < 4; j++) {              // threshold sum
            t = (x >> j) & 0x1111111111111111;
            for (k = 1; k < 10; k++)
                t += (ri[k] >> j) & 0x1111111111111111;
            // threshold 6 -- add 2 to weight and take bit number 3
            t = ((t + 0x2222222222222222) >> 3) & 0x1111111111111111;
            p[i] ^= t << j;                    // fix bits
        }
        if (i < 3) {                           // rotate if not last
            for (j = 1; j < 10; j++)
                ri[j] >>= 64 % xe5_len[j];
        }
    }
}
```