KAIBURR: ML-KEM at (much) higher security

Swechchha Adhikari[1], Vadim Lyubashevsky[2] and Peter Schwabe[3,4]

[1] Brigham Young University, Provo, USA, mail@swechchhaa.com
[2] IBM Research, Zurich, Switzerland, vad@zurich.ibm.com
[3] Max Planck Institute for Security and Privacy, Bochum, Germany, peter@cryptojedi.org
[4] Radboud University, Nijmegen, The Netherlands

Abstract. With deployment of standardized post-quantum cryptography well on the way, discussions continue about concrete choices of schemes and parameter sets. For example, while NIST standardized structured-lattice-based ML-KEM, European agencies like BSI and ANSSI favor unstructured-lattice-based FrodoKEM. Another discussion revolves around different approaches to hybrid deployments – either combining post-quantum schemes with "traditional" elliptic-curve cryptography (or RSA) or combining multiple different post-quantum schemes. The typical question for systems engineers to ask is what the "best" or "most conservative" approach is within a certain performance budget.

In this paper we add a data point to this discussion by investigating how to scale ML-KEM to (much) higher security levels of the underlying MLWE problem. We first observe that increasing security simply by increasing the module rank k quickly reaches a limit imposed by decryption failure probability. We then propose a generalization of the noise distribution, which enables scaling to much larger module rank and MLWE hardness, while preserving the essence of ML-KEM, most notably arithmetic in the ring $\mathbb{Z}_{3329}[X]/(X^{256}+1)$ and a simple and constant-time secret / noise generation.

We select three concrete parameter sets of this high-security variant of ML-KEM, which we call KAIBURR. We present benchmarking results for optimized implementations of KAIBURR on 64-bit Intel CPUs and on the Asymmetric Cryptography Coprocessor (ACC) of the Pavona open-source silicon framework. Based on these benchmarks, we argue that, for example, a hybrid of FrodoKEM-976 and KAIBURR-1792 not only offers better performance than FrodoKEM-1344, it also offers protection against a broader range of hypothetical future attacks against (M)LWE.

Keywords: (M)LWE hardness, CRYSTALS-KYBER, ML-KEM, lattice-based KEMs

1 Introduction

After a multi-year, multi-round public competition, NIST published its first post-quantum standards FIPS-203 [Nat24b], FIPS-204 [Nat24a], and FIPS-205 [Nat24c] in 2024, specifying ML-KEM, ML-DSA, and SLH-DSA, respectively, with FN-DSA to follow in FIPS-206 [Per25]. In 2025, NIST selected an additional KEM, HQC, for standardization in the fourth round [ABC+25]. Unlike ML-KEM, which is based on Module Learning With Errors (MLWE), HQC relies on the Quasi-Cyclic Syndrome Decoding (QCSD) problem [HQC25], providing diversity beyond lattice-based assumptions. Among lattice-based designs, the Round 3 KEMs [Nat20] span a spectrum of algebraic structure, from unstructured LWE (FrodoKEM [BCD+16]), through cyclotomic modules (KYBER [BDK+18] and Saber [DKRV18]), to prime-degree number fields (NTRU LPRime [BCLvV17]), reflecting a trade-off between efficiency and conservatism. ML-KEM's module structure gives small keys and fast NTT arithmetic, but is also seen as a potential attack surface, which some

other designs avoid; FrodoKEM, for example, rests on plain LWE, which is the reason that it was recommended by BSI, ENISA, and NLNCSA [The26] and recently standardized by ISO [ISO26].

Structure, margin, and diversity. The efficiency-vs.-conservatism framing bundles together two notions of conservatism that concern different classes of attacks. The first arises from the choice of the hardness assumption: FrodoKEM is based on unstructured LWE to avoid a hypothetical attack that would exploit the algebraic structure of ring/module LWE but not apply directly to plain LWE. The second is the security margin a scheme carries above its target level, which protects against advances in the best known attacks, such as improved lattice reduction, dual attacks, or sharper cost models. Such advances (e.g. hypothetical major advances of [GJ21, MAT22, CMST25, GHH26]) would affect the concrete security of every lattice scheme at its current parameters, including FrodoKEM. A third, orthogonal, hedge is to change the hardness assumption, as HQC does by relying on QCSD instead of LWE. This provides diversity, and at first glance, some additional assurance since a break of LWE could still leave HQC secure. However, it is not entirely certain that the two hardness assumptions are cryptanalytically independent. Recent work has shown connections between lattice and code-based cryptanalysis, including adaptations of lattice-sieving techniques to codes [DEEK24]. The quasi-cyclic structure of HQC has also been noted to have important parallels with the ring and module structures used in structured lattice schemes [NIS25].

Combining schemes. Since margin comes at a growing cost, and it is impossible to predict which of the (hypothetical) breakthroughs above is most likely, a natural response is also to combine schemes. Hybrid key exchange, which combines two KEMs so that an attacker has to break both [GHP18, BBF+19], is already standard practice in the post-quantum transition, where it typically pairs a post-quantum KEM with a classical one, as in the X25519+ML-KEM combinations now deployed in TLS [KKWS26]. There, the classical component is a safety net, as post-quantum schemes are younger and have received less cryptanalytic scrutiny than their classical counterparts. Against a quantum adversary, however, the classical component offers no protection, so for long-term confidentiality the combination is only as strong as its post-quantum half. This same principle extends naturally to combining two post-quantum KEMs, each covering a different failure mode.

Beyond FrodoKEM-1344. Of the standalone schemes available today, FrodoKEM-1344 comes closest to covering both the structure and margin hedges, as it avoids algebraic structure entirely and carries the largest security estimate among the standardized lattice KEMs. It is also by far the most expensive, with public keys and ciphertexts of over 21 KB each, more than an order of magnitude larger than those of ML-KEM-1024 [BCD+16, Nat24b]. This cost reflects the fact that structure determines the *price* of buying margin. At matched security, an unstructured scheme already needs roughly an order of magnitude more bandwidth and considerably more computation than a module scheme [BCD+16, BDK+18]. Within FrodoKEM's communication budget, a structured scheme can therefore afford far larger parameters, and, combined in a hybrid with a smaller FrodoKEM parameter set, potentially cover both hedges more cheaply than FrodoKEM-1344 alone.

Scaling ML-KEM. The natural way to obtain such a module scheme is to scale ML-KEM itself, whose parameter sets already differ mainly in their module rank k, so it allows lattice security to be adjusted by changing k while reusing the same ring arithmetic [BDK+18]. Recent work has explored the analogous high-margin regime for unstructured LWE in

the form of KyFrog [MB25], which increases the LWE dimension and adjusts the noise distribution to reach substantially higher security estimates at the cost of much larger ciphertexts. What has received less attention is how far module rank can be pushed before the correctness limit discussed above takes over, and what it costs to push past it. We therefore ask:

> *Can a scaled ML-KEM remain correct and efficient within FrodoKEM's tightest communication budget, and, combined with FrodoKEM in a hybrid, provide a better hedge than FrodoKEM-1344?*

Our approach. We argue for both parts affirmatively through KAIBURR[1], which increases the module rank k in ML-KEM, replaces its centered binomial distribution with a generalized one-parameter family that remains easy to sample, and removes ciphertext compression. The ring, NTT arithmetic, symmetric primitives, and FO transform are unchanged, so existing optimized ML-KEM software and hardware can largely be reused. We instantiate KAIBURR up to lattice dimension 6144, reaching 1717 bits of classical Core-SVP security within FrodoKEM-640's communication budget, and show that a hybrid of KAIBURR with FrodoKEM hedges against both structural attacks and improvements to lattice attacks at lower cost than FrodoKEM-1344.

Our contributions. We make several contributions in this work, summarized below.

- *A generalized noise distribution that lifts the correctness limit.* Raising the module rank k alone, with KYBER's noise and modulus fixed, keeps decryption failure below 2^{-128} only up to $k = 7$ (Section 3). We therefore introduce $f(t)$, a one-parameter family that generalizes KYBER's centered binomial distribution and retains simple constant-time sampling. Tuning t against k keeps decryption failure below 2^{-128} as k grows (Section 4).
- *Concrete parameter sets with large margins within FrodoKEM-640's communication budget.* We define three parameter sets, KAIBURR-1792, -4608, and -6144, at $k = 7, 18, 24$, whose classical/quantum Core-SVP security climbs from 479/434 to 1717/1557 bits, while their public keys and ciphertexts remain smaller than even those of FrodoKEM-640, the instantiation of FrodoKEM with around 160 bits of claimed security (Section 4).
- *A formally verified bound on decryption failure.* Building on the EasyCrypt framework of [BKL+25], we show that removing ciphertext compression makes this bound provable, closing the gap between provable and heuristic bounds that compression causes for ML-KEM (Section 5).
- *Implementations across three targets.* Since KAIBURR keeps ML-KEM's ring, NTT arithmetic, and symmetric primitives, existing implementation techniques carry over. We implement all three parameter sets in C, in Jasmin with machine-checked constant-time verification, and as an ACC variant targeting the Pavona hardware root of trust, and describe the engineering consequences of scaling k and removing ciphertext compression (Section 6).
- *Benchmarks against FrodoKEM and HQC.* Across three Intel microarchitectures, our C and Jasmin AVX2 implementations run substantially faster than FrodoKEM-640 and hence also the larger FrodoKEM parameter sets. KAIBURR-1792 outperforms HQC-5 in every operation. Against HQC-1/3/5 at matching indices, HQC is faster in key generation, and in encapsulation for the two larger parameter sets, while decapsulation, HQC's most expensive operation, favors KAIBURR (Section 7).

[1] https://starwars.fandom.com/wiki/Kaiburr_crystal

- *KAIBURR–FrodoKEM hybrids instead of high-security FrodoKEM parameter sets.* Our results call into question the use of FrodoKEM's larger parameter sets. We show that, combined with FrodoKEM-640 or FrodoKEM-976, KAIBURR is faster than the next-larger FrodoKEM parameter set. For example, the hybrid of KAIBURR-1792 and FrodoKEM-976 is smaller and about 1.7× faster than FrodoKEM-1344 and has a much higher lattice attack cost ($2^{467.3}$ vs. $2^{281.7}$) (Section 8).

Use of generative AI. The EasyCrypt proofs for the correctness bounds explained in Section 3 and then in Section 5 were adapted from [BKL+25] to KAIBURR with the help of AI (mainly Claude Opus 5.5). Some of the tests discussed in Section 6.4 (specifically those not directly inherited from upstream) were also generated using Opus 5.5.

Access to artifacts. All software, including parameter analysis scripts, implementation source code, and other artifacts, is publicly available on GitHub.

2 Preliminaries

ML-KEM[2] [Nat24b] is built in two layers. At its core is an IND-CPA-secure public-key encryption scheme, K-PKE, whose security reduces to module-learning-with-errors (MLWE). ML-KEM uses a variant of the Fujisaki–Okamoto (FO) transform to lift K-PKE to an IND-CCA2-secure KEM where decapsulation decrypts, re-encrypts the recovered message with the derived randomness, and returns a pseudorandom key if the result differs from the received ciphertext. In this work, we leave the FO transform and the symmetric primitives unchanged, so all of our changes are confined to K-PKE, which we briefly review in this section. We refer to [Nat24b] for the full specification.

Ring and module setup. Let $n = 256$ and $q = 3329$, and let $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$ be KYBER's base ring, with coefficients centered in $(-q/2, q/2]$. The modulus satisfies $n \mid (q-1)$, which enables the number-theoretic transform (NTT) used for multiplication in $\mathcal{R}_q$. A module rank k determines the dimensions of the public matrix $\mathbf{A} \in \mathcal{R}_q^{k\times k}$ and of the secret and error vectors in $\mathcal{R}_q^k$. $k = 2, 3, 4$ give ML-KEM-512, -768, and -1024, respectively [Nat24b].

K-PKE. Let χ be a noise distribution over $\mathcal{R}_q$ with small coefficients, instantiated below.

- *Key generation:* expand a public 32-byte seed ρ into $\mathbf{A} \in \mathcal{R}_q^{k\times k}$ using SHAKE-128, directly in the NTT domain. Sample $\mathbf{s}, \mathbf{e} \leftarrow \chi^k$ and set $\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e}$. The public key is $(\mathbf{t}, \rho)$ and the secret key is $\mathbf{s}$.

- *Encryption* of a message $m \in \{0,1\}^n$, viewed as a polynomial with binary coefficients: sample $\mathbf{r} \leftarrow \chi^k$, $\mathbf{e}_1 \leftarrow \chi^k$, and $e_2 \leftarrow \chi$, and compute $\mathbf{u} = \mathbf{A}^T\mathbf{r} + \mathbf{e}_1$ and $v = \mathbf{t}^T\mathbf{r} + e_2 + \lceil q/2 \rfloor \cdot m$.

- *Decryption:* compute $v - \mathbf{s}^T\mathbf{u}$ and decode each coefficient to 1 if it is closer to $\lceil q/2 \rfloor$ than to 0, and to 0 otherwise.

Expanding the decryption step gives $v - \mathbf{s}^T\mathbf{u} = \lceil q/2 \rfloor \cdot m + \tilde{n}$ with noise term

$$\tilde{n} = \mathbf{e}^T\mathbf{r} - \mathbf{s}^T\mathbf{e}_1 + e_2.$$

[2] We use KYBER and ML-KEM interchangeably, except when separating the NIST submission from the standardized version.

Centered binomial noise. In ML-KEM, χ samples each coefficient from the centered binomial distribution CBD_η, which draws 2η uniform bits $a_1, \ldots, a_\eta, b_1, \ldots, b_\eta$ and outputs $\sum_{i=1}^{\eta} a_i - \sum_{i=1}^{\eta} b_i$, a value in $\{-\eta, \ldots, \eta\}$ with variance $\eta/2$. ML-KEM uses η_1 for $\mathbf{s}, \mathbf{e}$, and $\mathbf{r}$, and η_2 for $\mathbf{e}_1$ and e_2; ML-KEM-768 and -1024 use $\eta_1 = \eta_2 = 2$, and ML-KEM-512 uses $\eta_1 = 3$ and $\eta_2 = 2$. CBD_η is especially desirable because it can be sampled from raw PRF output in constant time without rejection, using only bit-counting and a subtraction. KAIBURR replaces it with the noise family of Section 4.

Decryption failure. Decryption is correct as long as every coefficient of $\tilde{n}$ is smaller than $q/4$ in absolute value. We refer to the decryption-failure probability δ as the probability, over the randomness of key generation and encryption, that at least one of the n coefficients of $\tilde{n}$ exceeds this threshold[3]. To compute δ, we adapt the KYBER team's scripts [CRY]. Each coefficient of $\mathbf{e}^T\mathbf{r}$ and of $\mathbf{s}^T\mathbf{e}_1$ is a signed sum of kn products of two independent noise coefficients. The scripts compute the distribution of a single coefficient of $\tilde{n}$ by convolution, take its tail mass beyond $q/4$, and multiply by n (a union bound over the coefficients).

Ciphertext compression. Instead of transmitting $\mathbf{u}$ and v at full 12-bit precision, ML-KEM compresses them by rounding each coefficient to d_u and d_v bits, respectively, with $(d_u, d_v) = (10, 4)$ for ML-KEM-512 and -768 and $(11, 5)$ for ML-KEM-1024. This reduces the ciphertext from $384(k+1)$ to $32(d_u k + d_v)$ bytes, e.g. from 1536 to 1088 bytes for ML-KEM-768. The receiver decompresses to $\mathbf{u} + \mathbf{c}_u$ and $v + c_v$, where $\mathbf{c}_u$ and c_v are the rounding errors, so the noise term becomes

$$\tilde{n} = \mathbf{e}^T\mathbf{r} - \mathbf{s}^T\mathbf{e}_1 - \mathbf{s}^T\mathbf{c}_u + e_2 + c_v,$$

trading a higher δ for a smaller ciphertext. Throughout this work, "ciphertext compression" refers to this rounding of $\mathbf{u}$ and v.

Module-LWE. The decisional Module Learning With Errors (MLWE) problem [LS15] over $\mathcal{R}_q$ asks an adversary to distinguish $(\mathbf{A}, \mathbf{A}\mathbf{s} + \mathbf{e})$, where $\mathbf{A} \leftarrow \mathcal{R}_q^{\ell \times k}$ is uniform and $\mathbf{s} \leftarrow \chi^k$, $\mathbf{e} \leftarrow \chi^\ell$, from a uniformly random pair in $\mathcal{R}_q^{\ell \times k} \times \mathcal{R}_q^{\ell}$. The IND-CPA security of K-PKE reduces to MLWE with χ. The public key is an instance with $\ell = k$, and, with $\mathbf{t}$ replaced by a uniform vector, the ciphertext is an instance with $\ell = k + 1$. Increasing k increases the dimension kn of the underlying lattice problem and is the main lever KYBER uses to raise its claimed security level [ABD+17].

Core-SVP hardness estimates. We estimate MLWE hardness with the standard Core-SVP methodology [ADPS16] used throughout the KYBER submission documents and implemented in the KYBER team's public estimator scripts [CRY]. The adversary runs BKZ with block size b in either the primal or the dual attack, and the cost of BKZ-b is approximated by a single SVP call in dimension b, estimated as 2^{cb} operations with $c \approx 0.292$ for classical sieving [BDGL16] and $c \approx 0.265$ for quantum sieving [Laa16]. For each parameter set, the scripts search over b and the number of samples ℓ for the cheapest successful attack, primal or dual, and report its $\log_2$ cost as the *classical* and *quantum* Core-SVP security level. They depend on the noise distribution only through its standard deviation. For reference, the same scripts give classical/quantum levels of 117/106, 181/164, and 253/230 for ML-KEM-512, -768, and -1024, respectively.

[3] It is well established that once an attacker finds a single failure, they can use it as a compass to drastically accelerate finding more failures. This type of attack is known as "directional failure-boosting" [DRV20].

Lattice estimator. We cross-check the numbers from the KYBER Core-SVP scripts with the lattice estimator [APS15], which aims to provide tighter estimates and evaluates a wider range of attacks[4]. Its full estimate costs lattice reduction with the MATZOV model [MAT22] by default, rather than with Core-SVP. We model each KAIBURR parameter set as an LWE instance of dimension $256k$ with $256k$ samples, $q = 3329$, and secret and error coefficients of standard deviation $\sigma(t)$ (Section 4). For FrodoKEM, we use the estimator's built-in parameter sets. We keep the estimator's defaults with one exception, explained below.

Raising the block-size limit. By default, the estimator caps the BKZ block size at 1754, where the Core-SVP cost reaches 2^{512}, and reports an infinite cost for any attack beyond it, so every attack on the two largest KAIBURR parameter sets is infinite. We raise the cap to 16384, together with the same hard-coded limit in the primal hybrid and BDD attacks; these are our only changes to the estimator.

3 Warmup: Simply increasing the module rank

The main mechanism to increase the security margin in KYBER is to raise the module rank k [ABD+17]. Fixing the ML-KEM base ring, noise distribution, and modulus, and varying only k, we compute the decryption-failure probability. Throughout we use the centered binomial noise CBD_2 (the noise of ML-KEM-768 and -1024; recall that ML-KEM-512 uses $\eta_1 = 3$). The ranks $k = 3$ and 4 recover the uncompressed analogues of the standardized sets ML-KEM-768 and -1024, while $k \geq 5$ is an extrapolation.

We will see that increasing k also increases the decryption-failure probability. This is a problem for IND-CCA security, where the failure probability shows up as part of the attacker's advantage in the security proofs of (variants of) the FO transform; for details see [HHK17]. Throughout the paper we require this failure probability δ to be smaller than 2^{-128}; this is also in line with the standardized parameter sets of ML-KEM, which reach the highest value for ML-KEM-512 at $\delta = 2^{-138.8}$ [Nat24b, Table 1]. We initially determine the failure probability using an adapted version of the Python scripts provided by the Kyber team [CRY]. Furthermore, we adapt the EasyCrypt proofs of the tight failure-probability bounds for FrodoKEM presented in [BKL+25]. Note that the bounds for ML-KEM in [BKL+25] are either non-tight or heuristic. For KAIBURR we are able to prove tight non-heuristic bounds because KAIBURR does not use any ciphertext compression. We discuss this in more detail in Section 5.

The results of our analysis are shown in Table 1; we see that our failure-probability upper bound of 2^{-128} holds at $k = 7$ ($2^{-133.7}$ by the scripts, 2^{-133} machine-checked) but fails at $k = 8$ ($2^{-116.8}$ and 2^{-116}, respectively). This demonstrates that raising k alone cannot buy much higher security. With KYBER's ciphertext compression, the rounding of $\mathbf{u}$ and v would add further noise, so the correctness limit on k would be reached even earlier.

4 Revisiting the noise distribution

Even though the resulting public key and ciphertext sizes at $k = 7$ remain relatively small, increasing k further is not possible without affecting correctness. Instead, one could reduce the failure probability at large k by increasing the modulus q. We note that all variants of KYBER fix $q = 3329$, which is explicitly justified in the submission documentation to NIST. The modulus $q = 3329$ is a suitable small prime satisfying the NTT-enabling condition $n \mid (q-1)$, notably weaker than the $2n \mid (q-1)$ one might expect, which KYBER

[4] Forked from `https://github.com/malb/lattice-estimator`, commit `53da598`.

Table 1: Decryption-failure probability and public-key and ciphertext sizes as a function of module rank k, at fixed ML-KEM noise and modulus, without ciphertext compression. Ranks $k = 3, 4$ correspond to uncompressed ML-KEM-768 and -1024. *Script*: estimate from the KYBER team's scripts. *Machine-checked*: upper bound on the adversarial failure probability proved in EasyCrypt (Section 5), with the final tail probability computed by OCaml code that rounds upward.

	Failure probability			
k	Script	Machine-checked	pk (bytes)	ct (bytes)
3	$2^{-299.6}$	2^{-300}	1184	1536
4	$2^{-231.1}$	2^{-230}	1568	1920
5	$2^{-186.5}$	2^{-186}	1952	2304
6	$2^{-155.9}$	2^{-155}	2336	2688
7	$2^{-133.7}$	2^{-133}	2720	3072
8	$2^{-116.8}$	2^{-116}	3104	3456

deliberately does not require [ABD+19]. It was reduced from the round-1 value 7681 to recover the bandwidth spent by dropping public-key compression and co-tuned to balance ciphertext size, failure probability, and security.

Since ML-KEM's standardization, highly optimized and high-assurance implementations have emerged across software and hardware platforms [PQ 26, ABB+23, AOP+25]. Changing q would mean not being able to reuse any existing optimized routines in software or hardware, which sacrifices an important engineering advantage and results in a different scheme altogether. We choose not to change q for this reason.

This is not to say that k is the only parameter that can be modified. We focus on changes that preserve the *essence* of KYBER: the ring $\mathbb{Z}_q[X]/(X^{256}+1)$, the NTT implementation and its constants, and the existing analysis framework. The noise, however, only affects the secret and error terms and does not interact with the arithmetic structure or the NTT, and so it can be replaced without altering the core implementation. The noise affects only the MLWE hardness estimates and the decryption-failure probability, both of which can be recomputed through the same scripts.

To scale past $k = 7$, we need a noise knob that pushes the failure-driving part of the error back down as k grows, while preserving KYBER's *essence*. Equally important, it should retain the cheap, branchless, constant-time sampling that motivates KYBER's use of a centered binomial in the first place. This requirement rules out simply using a discrete Gaussian, as in FrodoKEM.

4.1 A parameterized noise distribution

We introduce a one-parameter family $f(t)$ on $\{-2, -1, 0, 1, 2\}$, the same support as CBD_2, defined by

$$\Pr[\pm 2] = 2^{-t}, \qquad \Pr[\pm 1] = \tfrac{1}{4}, \qquad \Pr[0] = \tfrac{1}{2} - 2^{-(t-1)}.$$

The mass at ± 1 is fixed. The single parameter t acts only on the ± 2 tail, thinning it geometrically. A sample is drawn from uniform bits $a_1, \ldots, a_t$ as

$$x = (2a_t - 1)\Big(1 - a_1 + 2\textstyle\prod_{i=1}^{t-1} a_i\Big)$$

(Algorithm 1), which is branchless and therefore samples in constant time; we machine-check constant-time (CT) for all our Jasmin builds [ABB+17], and speculative constant-time (SCT) for the AVX2 builds (see Section 6.4). The standard deviation $\sigma(t) = \sqrt{0.5 + 2^{3-t}}$ decreases from 1 at $t = 4$ toward $\sqrt{0.5}$ as t grows.

Algorithm 1: Sample-$f(t)$

Input: t uniform bits
Output: $x \in \{-2, -1, 0, 1, 2\}$
Sample $a_1, \ldots, a_t \in \{0, 1\}$
$b \leftarrow \prod_{i=1}^{t-1} a_i$
$x \leftarrow (2a_t - 1)\,(1 - a_1 + 2b)$
return x

We note that $f(t)$ generalizes KYBER's own noise distribution. At $t = 4$, the probabilities become

$$\Pr[\pm 2] = \frac{1}{16}, \qquad \Pr[\pm 1] = \frac{1}{4}, \qquad \Pr[0] = \frac{3}{8},$$

which matches CBD_2, the noise distribution used by ML-KEM-768 and -1024. As $t \to \infty$, the ± 2 mass vanishes and $f(t)$ approaches CBD_1.

4.2 Finer control over the noise

A key property of $f(t)$ is that increasing t reduces the probability of the ± 2 terms exponentially while leaving the ± 1 mass unchanged. This gives finer control over the noise than the standard centered binomial choices. In particular, CBD_1 and CBD_2 have standard deviations $\sqrt{0.5}$ and 1, respectively, with no centered binomial parameter providing an intermediate distribution.

This finer control is useful as the module rank k increases. Rather than moving directly from CBD_2 to CBD_1, we can increase t as k grows, reducing the ± 2 contribution while keeping the ± 1 contribution fixed.

4.3 Selecting t

Since t is the number of uniform bits per sample, vectorization-friendly values are natural, especially for AVX2 implementations (see Section 6). As shown above, $t = 4$ recovers the CBD_2 distribution, so it serves as our anchor. Notice that $t = 8$ consumes exactly one PRF byte per coefficient, meaning it admits one-byte-per-coefficient SIMD packing without cross-byte bit-splitting. Our third choice, $t = 6$, sits in the middle and does not share the byte-alignment of $t = 4$ or $t = 8$. It is nonetheless interesting relative to $t = 8$, for two reasons. First, it retains more of the ± 2 tail, giving a higher standard deviation and more security per dimension. Second, it spends fewer PRF bytes per coefficient (0.75 vs. 1), which is relevant on platforms *without* Keccak acceleration. For each choice of t, we pick the highest k satisfying the failure probability bound of 2^{-128}, staying within the communication budget of FrodoKEM-640.

We use FrodoKEM-640 as our baseline for parameter selection and comparison throughout the paper because it is the smallest FrodoKEM parameter set (NIST Level 1) and hence imposes the tightest communication budget of 9,616 public-key bytes and 9,720 ciphertext bytes. This gives us a restrictive baseline for asking how much security margin KAIBURR can achieve within the communication cost of an unstructured lattice-based KEM.

Table 2 lists, for every t from 2 to 8, the largest k that satisfies both constraints. The three parameter sets we carry forward are $f(4)$, $f(6)$, and $f(8)$, at $k = 7$, 18, and 24 respectively, which follow from these choices.

Table 2: Largest module rank k for each noise parameter t under the 2^{-128} failure bound and FrodoKEM-640's communication budget. Each name gives the lattice dimension $256k$. Shaded rows are the selected KAIBURR parameter sets. For $t = 8$, $k = 25$ would give a 9,632-byte public key, just over the budget. Classical/quantum security (Core-SVP) and failure probability computed with the KYBER team's scripts; other parameters as in ML-KEM, compression removed.

Scheme	k	noise	failure	class.	quant.	pk (B)	ct (B)
KAIBURR-256	1	$f(2)$	$2^{-159.1}$	52	47	416	768
KAIBURR-768	3	$f(3)$	$2^{-139.3}$	191	174	1184	1536
KAIBURR-1792	7	$f(4)$	$2^{-133.7}$	479	434	2720	3072
KAIBURR-3328	13	$f(5)$	$2^{-128.4}$	918	832	5024	5376
KAIBURR-4608	18	$f(6)$	$2^{-134.1}$	1281	1162	6944	7296
KAIBURR-5888	23	$f(7)$	$2^{-129.8}$	1650	1497	8864	9216
KAIBURR-6144	24	$f(8)$	$2^{-139.9}$	1717	1557	9248	9600

To get tighter estimates, we also run all eight attacks of the lattice estimator (Section 2) on the three selected parameter sets. Table 3 summarizes the results. For KAIBURR-1792, the cheapest attack is the dual-hybrid attack at $2^{467.3}$, below the 479 bits from the KYBER scripts, which do not model hybrid attacks. For KAIBURR-4608 and KAIBURR-6144, the cheapest attack is BDD, at $2^{1268.9}$ and $2^{1694.3}$, which is 12 and 23 bits below their Core-SVP estimates.

Table 3: Lattice estimator results ($\log_2$ of the cost) for all eight attacks, with the block-size limit raised to 16384. The cheapest attack for each parameter set is in bold. Entries marked – are attacks for which the estimator crashed and returned no cost.

Attack	KAIBURR-1792	KAIBURR-4608	KAIBURR-6144
uSVP	494.0	1275.5	1701.0
BDD	488.7	**1268.9**	**1694.3**
BDD hybrid	488.7	1268.9	1694.3
BDD MITM hybrid	883.0	2411.1	3265.2
dual	519.8	1400.8	–
dual hybrid	**467.3**	1458.3	2120.4
coded BKW	–	–	–
Arora–Ge (GB)	∞	∞	∞

4.4 Sizes

Table 4 places the three KAIBURR parameter sets alongside all FrodoKEM and HQC parameter sets. KAIBURR-1792 is smaller in combined pk+ct size than HQC-1 (5,792 vs. 6,674 bytes), and less than a third the size of HQC-5 (21,658 bytes).

4.5 The ceiling of the family

The family $f(t)$ is not unbounded. The lightest noise it can produce is its limit $f(\infty)$, with $\Pr[\pm 1] = \frac{1}{4}$, $\Pr[0] = \frac{1}{2}$, $\Pr[\pm 2] = 0$, and minimal variance $\sigma^2 = 0.5$. At $n = 256$ and $q = 3329$, the failure probability crosses 2^{-128} at $k = 30$, so $k = 29$ is the largest rank

[5]The security categories for HQC are as claimed by the design team against NIST's category 1/3/5 targets. They are not directly comparable to either lattice figure, as HQC is a code-based scheme.

Table 4: Public-key and ciphertext sizes across KAIBURR, FrodoKEM, and HQC. KAIBURR's classical/quantum Core-SVP security is as computed with the KYBER scripts (Table 2); Core-SVP is not computed for FrodoKEM (–). The lattice estimator column gives the $\log_2$ cost of the cheapest attack found by the lattice estimator for KAIBURR and FrodoKEM, both evaluated with max block size set to 16384.

Scheme	Core-SVP (class./quant.)	Lattice est.	pk (bytes)	ct (bytes)	pk+ct (bytes)
KAIBURR-1792	479 / 434	467.3	2720	3072	5792
KAIBURR-4608	1281 / 1162	1268.9	6944	7296	14240
KAIBURR-6144	1717 / 1557	1694.3	9248	9600	18848
FrodoKEM-640	–	162.9	9616	9720	19336
FrodoKEM-976	–	225.2	15632	15744	31376
FrodoKEM-1344	–	281.7	21520	21632	43152
HQC-1	Category 1[5]		2241	4433	6674
HQC-3	Category 3		4514	8978	13492
HQC-5	Category 5		7237	14421	21658

that $f(t)$ can support (Table 5). Reaching higher security levels would require another change, such as changing the noise family or modifying q or n.

Table 5: Failure probability under the limiting distribution $f(\infty)$ (minimum variance $\sigma^2 = 0.5$) as k increases, uncompressed. The 2^{-128} bound is first violated at $k = 30$, so $k = 29$ is the ceiling of the $f(t)$ family at $n = 256$, $q = 3329$.

k	Failure probability	pk (bytes)	ct (bytes)
24	$2^{-158.7}$	9248	9600
26	$2^{-146.2}$	10016	10368
28	$2^{-135.5}$	10784	11136
29	$2^{-130.6}$	11168	11520
30	$2^{-126.1}$	11552	11904

5 Formally verified bounds for Kaiburr

In this section, we show that removing compression turns the decryption-failure probability from a heuristic estimate into a formally verified bound[6].

Why compression is the obstruction. In the noise expression for ML-KEM, $\tilde{n} = \mathbf{e}^T\mathbf{r} - \mathbf{s}^T\mathbf{e}_1 - \mathbf{s}^T\mathbf{c}_u + e_2 + c_v$ (Section 2), every term except the rounding errors $\mathbf{c}_u$ and c_v has a coefficient distribution that can be computed exactly. The rounding errors depend on the ciphertext $(\mathbf{u}, v)$, which is correlated with the other noise, so the ML-KEM analysis assumes $\mathbf{u}$ and v are uniform and independent [BKL+25]. MLWE cannot justify this, because the correctness adversary holds the secret key, and deciding whether decryption failed requires that key and all the noise. The provable alternative of [BKL+25] bounds c_v by its worst case and uses MLWE only for a separately thresholded $\mathbf{s}^T\mathbf{c}_u$ term. Because

[6]Removing compression also removes a step that is costly to mask in ML-KEM decapsulation, which existing work avoids with a comparison technique [BGR+21]. We do not study masking for KAIBURR further in this paper.

this ignores cancellation between the two parts, the provable bound for ML-KEM-768 is 2^{-80} (plus an MLWE term), against 2^{-164} under the heuristic, a gap of about 84 bits. FrodoKEM does not compress ciphertexts and has no such gap.

What we prove. We adapt the EasyCrypt framework of [BKL+25], including its theory of distributions over the ring and its OCaml implementation for computing a sound upper bound on the final tail probability, to KAIBURR. For each parameter set, we prove that for every adversary that receives both the public and the secret key and chooses the message, decryption fails with probability at most δ, the bound in Table 6. This is the notion of correctness used in the Fujisaki–Okamoto bound. Since the rounding terms vanish, the noise collapses to $\tilde{n} = \mathbf{e}^T\mathbf{r} - \mathbf{s}^T\mathbf{e}_1 + e_2$, which does not depend on the message chosen by the adversary. Since $f(t)$ is non-trivial and symmetric around 0, every coefficient is then identically distributed, and a union bound reduces the failure probability to 256 times the tail of a single coefficient, with no additional MLWE term. As in [BKL+25], the proof is in the random-oracle model, and the tail is computed outside EasyCrypt by OCaml code that mirrors the EasyCrypt expression and rounds upward, which is the only step not checked in EasyCrypt. The same framework with the original CBD_2 gives the machine-checked bounds for the warmup in Table 1.

Table 6: Decryption-failure probability for each KAIBURR parameter set: the formally verified bound obtained in this work, and the estimate from the KYBER team's scripts (Table 2).

Scheme	k	Verified bound (this work)	KYBER-script estimate
KAIBURR-1792	7	2^{-133}	$2^{-133.7}$
KAIBURR-4608	18	2^{-133}	$2^{-134.1}$
KAIBURR-6144	24	2^{-139}	$2^{-139.9}$

Results. Table 6 shows that our provable bound closely tracks the KYBER team's own estimate for every parameter set, so for KAIBURR there is no gap between a heuristic and a provable bound to close. The price KAIBURR pays for this is ciphertext size, which remains within our fixed communication budget.

6 Implementation

We present three implementations of KAIBURR, each forked from an existing ML-KEM implementation: a C/asm implementation[7], a Jasmin implementation with machine-checked constant-time verification[8]; and an assembly implementation for Pavona's Asymmetric Cryptography Coprocessor (ACC)[9]. As in upstream, the C and Jasmin code each contain a reference implementation and an implementation using AVX2 vector instructions.

The choice of optimized implementations targeting modern 64-bit Intel CPUs is motivated by the fact that such CPUs were declared by NIST as the reference platform for performance comparison in the PQC standardization project [Nat16]. The reason for targeting Pavona's ACC is to obtain benchmarks on a platform with hardware-accelerated Keccak. The ACC is the coprocessor for public-key cryptography in the Pavona open-source silicon project. To accelerate ML-KEM and ML-DSA, its post-quantum configuration adds a vector extension to its instruction set and a direct connection to the chip's

[7] Forked from KYBER, `https://github.com/pq-crystals/kyber`, commit `4768bd3`.
[8] Forked from `formosa-mlkem`, `https://github.com/formosa-crypto/formosa-mlkem`, commit `475b874`.
[9] Forked from `https://github.com/pavona/pavona`, commit `07ccb24`.

KMAC block [AOP+25, NP26], a hardware SHA-3 and SHAKE engine, which the ACC feeds and reads through special registers. It has 32 general-purpose registers of 32 bits and 32 wide data registers of 256 bits. Its vector instructions interpret a wide register as 16 lanes of 16 bits or 8 lanes of 32 bits, so one wide register holds 16 polynomial coefficients.

All implementations make the same three changes to ML-KEM:

1. the module rank k is scaled to $\{7, 18, 24\}$, yielding the parameter sets KAIBURR-1792, -4608, and -6144 (see Table 2);

2. ciphertext compression is removed, so both ciphertext polynomials are serialized at full 12-bit precision; and

3. the centered binomial sampler is replaced by a sampler for the noise family $f(t)$, with $t \in \{4, 6, 8\}$.

Everything else, including the ring, the modulus $q = 3329$, the NTT and its constants, the Fujisaki–Okamoto transform, and the symmetric primitives, is left unchanged.

These modifications nonetheless require more than just mechanical changes to the code. ML-KEM implementations assume $k \leq 4$ in several places, and the AVX2 packers assume a particular coefficient order. Most of the rewritten code follows from scaling k, which we describe first (Section 6.1). We then cover the noise sampler (Section 6.2), the removal of compression (Section 6.3), and what is checked about each implementation (Section 6.4).

6.1 Scaling k

Accumulator overflow. Larger values of k can overflow the matrix-vector product. In the NTT domain, each polynomial of the product is a sum of k pointwise products, which ML-KEM accumulates in an `int16_t` and reduces once at the end. In the C reference, the C/asm AVX2-optimized, and the Jasmin reference code, each output coefficient is the sum of two separately Montgomery-reduced products, so it is bounded by $2(q-1) = 6656$ rather than by q, and the sum of k terms fits only while $2kq < 2^{15}$, i.e. $k \leq 4$. On the other hand, the Jasmin AVX2 code accumulates both products at 32 bits and applies a single Montgomery reduction, so its terms are bounded by q and it is safe to $k \leq 9$; it also fails at $k = 18$ and $k = 24$.

The C and Jasmin implementations therefore Barrett-reduce the accumulator after every pointwise product, at a cost of one extra reduction per term[10]. ACC needs no such change since it accumulates with `bn.addvm`, a modular addition against the `MOD` register, which is set to $2q$ throughout the basemul. Its accumulator stays below $2q$ for every k, and the reduction is free.

Matrix generation. The AVX2 matrix sampler in ML-KEM hardcodes how the k^2 matrix entries are grouped into batches of four for the 4-way-parallel Keccak for each $k \leq 4$. We replace this with a generic loop; it samples four entries at a time with 4-way Keccak and handles any remaining entries with a scalar tail. Only $k = 7$ has a remainder, of one entry.

The same loop generates both $\mathbf{A}$ for key generation and its transpose $\mathbf{A}^T$ for encapsulation, by selecting the matching domain separators. The C code computes these separators from the row and column indices, while the Jasmin code reads them from a precomputed table. ACC generates one entry at a time and derives each separator from a running nonce, with separate handling for the transposed and non-transposed orders.

[10] Reducing after every term is more conservative than necessary, since a Barrett reduction leaves headroom for several further terms. We did not implement this optimization.

Memory. The C and Jasmin code store the full matrix $\mathbf{A}$ on the stack. Its size grows quadratically with k, from 24.5 KiB at $k = 7$ to 288 KiB at $k = 24$. This is not suitable for ACC, whose stock configuration only provides 32 KiB of data memory. That is less than the stack frame of a single decapsulation at $k = 24$, which is about 34 KiB even without storing the matrix, before the input ciphertext and secret key are counted.

We address this in two ways. First, the ACC implementation never stores $\mathbf{A}$. It generates one polynomial at a time, multiplies it into a running accumulator, and discards it. Second, we enlarge ACC's visible data memory from 31 KiB to 127 KiB in the hardware description, out of 128 KiB of physical DMEM including a 1 KiB scratch region. The ACC results in Section 7 are therefore for this modified hardware configuration.

6.2 The noise sampler

The noise family $f(t)$ maps t uniformly random bits $a_1, \ldots, a_t$, read least-significant first, to the coefficient

$$(2a_t - 1)\left(1 - a_1 + 2\prod_{i<t} a_i\right).$$

All our implementations replace the centered binomial sampler with this expression, and the PRF output grows to $t \cdot n/8$ bytes per polynomial. The five variants differ in how they compute two parts of the expression: the product term $\prod_{i<t} a_i$ and the sign $(2a_t - 1)$. Table 7 summarizes their choices.

Table 7: How each implementation computes the product term and applies the sign in $f(t)$.

Implementation	Product term	Sign	Word size
C ref	bit-by-bit AND chain	integer multiply	scalar
C AVX2	compare against a mask	conditional lane negate	8–32-bit lanes
Jasmin ref	carry propagation	conditional two's-complement	scalar
Jasmin AVX2	compare against a mask	conditional two's-complement	8–16-bit lanes
ACC	carry propagation	modular subtract + blend	16-bit lanes

The Jasmin reference code and the ACC assembly avoid the literal AND chain with a carry trick. Adding one to the low $t-1$ bits carries into bit $t-1$ exactly when all of those bits are set, so a right shift gives us the product. The AVX2 kernels instead isolate the same bits and apply a vector compare, which gives the product directly as an all-ones or all-zero mask. In every vectorized and assembly variant, we then apply the sign separately and without branching.

Extracting t-bit fields from the PRF output is simplest for $t = 8$ and $t = 4$, which align with bytes and nibbles, and hardest for $t = 6$, as explained in Section 4. Both AVX2 kernels use an extra shuffle or gather for this step. The C kernel also reads slightly beyond the end of the PRF output, so the buffer includes 32 bytes of trailing slack. ACC lacks a byte-shuffle instruction, so it instead uses a sequence of `bn.rshi` shifts to move the 6-bit fields across register boundaries.

Key generation needs $2k$ noise polynomials. The AVX2 code samples them in groups of four using the 4-way Keccak.

6.3 Removing compression

Kaiburr serializes both ciphertext polynomials $\mathbf{u}$ and v at full 12-bit precision, so the ciphertext grows to $(k + 1) \cdot 384$ bytes. The public key keeps its size of $384k + 32$ bytes, since ML-KEM never compresses it.

The reference code and ACC reuse their existing 12-bit serialization routine unchanged. The AVX2 code needs a new one. Its existing serialization routine is used only for the public key, which is stored in the permuted coefficient order of the AVX2 NTT, so it applies that permutation while packing. Ciphertext polynomials leave the inverse NTT in standard order. Both AVX2 forks (C/asm and Jasmin) therefore add a serialization routine and corresponding deserialization routine that work in standard order, without the permutation. This new routine emits 24 bytes per iteration as two 16-byte stores twelve bytes apart, so its final iteration would write past the end of the buffer. The C code therefore writes every iteration to a scratch buffer and copies 24 bytes out, and the Jasmin code handles the final iteration as a separate case.

6.4 Assurance

Jasmin. The Jasmin implementation is a fork of `formosa-mlkem` [ABB+23], whose EasyCrypt proofs establish functional correctness for ML-KEM. These proofs do not carry over to KAIBURR, since the sampler, module rank, and serialization all change. What we check is much narrower.

Every exported function carries a security-type annotation. We check all six builds with Jasmin's constant-time checker for constant-time (CT). The three AVX2 builds, whose exported functions carry `sct` annotations, are additionally checked for speculative constant-time (SCT) [SBG+23], or, more specifically, the absence of Spectre-v1-type leakage. As in upstream, the reference builds carry `ct` annotations and are checked for CT only.

Testing. The C implementation relies on the deterministic round-trip self-tests inherited from its upstream ML-KEM codebase. The Jasmin implementation runs the same kind of self-test, and additionally cross-checks its output against the C implementation across all six builds. The ACC implementation is validated against a Python model based on `kyber-py`[11], which subclasses `ML_KEM`, overrides the noise sampler, and disables ciphertext compression. For each parameter set, the model generates known-answer tests (KATs) for key generation, encapsulation, and decapsulation (including implicit rejection), as well as round-trip tests. Unlike the Jasmin code, the ACC implementation is not machine-checked for constant-time behavior, but also avoids secret-dependent branches. The ACC's focus on security rather than sheer performance generally makes it easier to protect against timing side channels, as it features no caches, no speculative execution, and fixed-latency memory accesses.

7 Benchmarks

7.1 Benchmarks on 64-bit Intel CPUs

We benchmark the C/asm and Jasmin implementations on three Intel microarchitectures: Skylake, Coffee Lake, and Raptor Lake, with hyper-threading and Turbo Boost disabled. For the hybrid Raptor Lake processor, we pin the benchmark to a performance core. All KAIBURR code is compiled using `GCC 12.2.0` with the flags `-O3`, `-march=native`, and `-fomit-frame-pointer`.

We compare the KAIBURR variants with FrodoKEM and HQC. We use the SHAKE-based variant of FrodoKEM as the primary comparison baseline, as both FrodoKEM-SHAKE and KAIBURR use SHAKE-based symmetric primitives. FrodoKEM-AES can take advantage of dedicated AES instructions on our Intel processors, which introduces

[11] https://github.com/GiacomoPope/kyber-py

an implementation-specific hardware advantage unrelated to the lattice-based KEM operations we are looking to compare[12]. We mainly compare against FrodoKEM-640, the smallest FrodoKEM parameter set, since it is already substantially more expensive than Kaiburr. On Coffee Lake, we also report FrodoKEM-976 and FrodoKEM-1344, which we use in the hybrid comparison of Section 8. For additional context, we also report ML-KEM-1024 performance on this microarchitecture.

All three KEMs are evaluated using a common benchmark driver in which we first perform 1000 untimed warm-up iterations, followed by 10001 timed iterations. We report the median execution time and interquartile range (Q_1–Q_3). This procedure is repeated three times, and we report the results from the run whose median is itself the median of the three runs, together with its corresponding quartiles. Each iteration verifies that the decapsulated shared secret matches the encapsulated shared secret.

Randomness comes from Linux `getrandom`. FrodoKEM and Kaiburr call it within `keypair` and `enc`, so that syscall is inside the measured region. HQC instead draws randomness from a SHAKE-256 PRNG that the caller must seed; we seed it once from `getrandom` before timing begins, so HQC's timings exclude the syscall[13].

The benchmark results are reported separately for each processor. Results for the Intel Core i7-6700K (Skylake), Intel Core i7-8700K (Coffee Lake), and Intel Core i9-13900K (Raptor Lake) are given in Table 8, Table 9, and Table 10, respectively.

Table 8: Benchmark results in cycles on an Intel Core i7-6700K. We report only the AVX2 implementations. The reference implementations are, on average, 545.7 percent slower for Kaiburr C and 637.8 percent slower for Kaiburr Jasmin, measured using the geometric mean. Values are reported as median (Q1–Q3).

KEM	Impl.	KeyGen	Enc	Dec
Kaiburr-1792	C/asm	156186 (155490–157822)	160062 (159576–161914)	171888 (171050–173356)
Kaiburr-1792	Jasmin	146528 (145948–147226)	148044 (147606–148788)	151990 (151322–152728)
Kaiburr-4608	C/asm	809842 (807208–813058)	817844 (815224–821062)	845458 (842838–848794)
Kaiburr-4608	Jasmin	780574 (779084–782266)	781302 (779932–782906)	796676 (795256–798396)
Kaiburr-6144	C/asm	1361664 (1358070–1366138)	1365370 (1361692–1369542)	1407632 (1403884–1411932)
Kaiburr-6144	Jasmin	1308772 (1306774–1311182)	1309238 (1307196–1312060)	1324338 (1322338–1326730)
FrodoKEM-640	[Fro]	3860180 (3859268–3868164)	4077768 (4076966–4087064)	4044176 (4043428–4053764)
HQC-1	[HQC]	88162 (88006–89002)	172786 (172618–173672)	405674 (405402–406028)
HQC-3	[HQC]	209986 (209758–210708)	407810 (407430–408654)	858366 (857904–859064)
HQC-5	[HQC]	433948 (433570–434640)	847696 (846974–848576)	1712396 (1711782–1713680)

[12]Additionally, we do not report a direct cycle-count comparison with KyFrog [MB25], another high-security KEM introduced in Section 1, since no public hand-optimized implementation of the scheme exists.

[13]Each implementation keeps the randomness handling of its upstream codebase; we chose not to modify HQC. This also *slightly* favors HQC in our comparison.

Table 9: Benchmark results in cycles on an Intel Core i7-8700K. We report only the AVX2 implementations. The reference implementations are, on average, 540.8 percent slower for KAIBURR C and 653.8 percent slower for KAIBURR Jasmin, measured using the geometric mean. Values are reported as median (Q1–Q3).

KEM	Impl.	KeyGen	Enc	Dec
ML-KEM-1024	C/asm	59973 (59693–60396)	62274 (62075–62548)	67846 (67282–68699)
ML-KEM-1024	Jasmin	54393 (54224–54701)	56092 (55947–56287)	59425 (58717–60114)
KAIBURR-1792	C/asm	156156 (155579–157778)	159719 (159235–161416)	173015 (172106–174398)
KAIBURR-1792	Jasmin	143857 (143402–144559)	144987 (144656–145831)	150435 (149704–151098)
KAIBURR-4608	C/asm	814685 (812084–817870)	821346 (818719–824533)	848608 (845755–852000)
KAIBURR-4608	Jasmin	762767 (761322–764429)	765475 (764033–767156)	781079 (779648–782776)
KAIBURR-6144	C/asm	1367405 (1363808–1371615)	1376530 (1372953–1380951)	1411130 (1407567–1415488)
KAIBURR-6144	Jasmin	1293036 (1291067–1295424)	1295674 (1293713–1298070)	1313175 (1311174–1315640)
FrodoKEM-640	[Fro]	3860446 (3857908–3864447)	4062824 (4061683–4069217)	4029412 (4027484–4035297)
FrodoKEM-976	[Fro]	8270912 (8266234–8274033)	8528877 (8523192–8530688)	8411864 (8402988–8419383)
FrodoKEM-1344	[Fro]	14508496 (14505193–14510927)	14790823 (14788919–14793792)	14614161 (14611158–14617148)
HQC-1	[HQC]	90229 (90070–90974)	172063 (171888–172892)	404274 (404018–404596)
HQC-3	[HQC]	215176 (214956–215857)	404687 (404324–405517)	857576 (857048–858323)
HQC-5	[HQC]	439786 (439458–440546)	838353 (837433–839669)	1698485 (1697851–1699766)

Our benchmarks show that KAIBURR is faster than FrodoKEM-640 across every **operation**, **parameter set**, **implementation**, and **machine**. KAIBURR-1792 is 21.5–28.0× faster, KAIBURR-4608 is 4.2–5.3× faster, and KAIBURR-6144 is 2.4–3.1× faster than FrodoKEM-640.

Which KAIBURR implementation is faster depends on the microarchitecture: Jasmin beats C/asm in every operation on Skylake and Coffee Lake (by 3.6–13.1%), while C/asm beats Jasmin in every operation on Raptor Lake (by 9.2–14.0%).

From KAIBURR-1792 ($k = 7$) to KAIBURR-4608 ($k = 18$) to KAIBURR-6144 ($k = 24$), C/asm KeyGen grows by ≈5.2× and ≈1.7× on all three machines, which lies between the growth of k (2.57×, 1.33×) and of k^2 (6.61×, 1.78×).

On Coffee Lake, KAIBURR-1792 takes 2.5–2.6× the cycles of ML-KEM-1024 in every operation, for both C/asm and Jasmin, which is expected. From $k = 4$ to $k = 7$, k grows by 1.75× and k^2 by 3.06×, and the ratio lies between the two, just as it does across KAIBURR parameter sets.

KAIBURR-1792 is faster than HQC-5 in every operation on every machine (C/asm: 64–90% fewer cycles). At matching indices (KAIBURR-1792/HQC-1, KAIBURR-4608/HQC-3, KAIBURR-6144/HQC-5), the comparison depends on the operation. With C/asm, HQC

Table 10: Benchmark results in cycles on an Intel Core i9-13900K. We report only the AVX2 implementations. The reference implementations are, on average, 498.2 percent slower for KAIBURR C and 557.8 percent slower for KAIBURR Jasmin, measured using the geometric mean. Values are reported as median (Q1–Q3).

KEM	Impl.	KeyGen	Enc	Dec
KAIBURR-1792	C/asm	121724 (120872–122872)	125204 (124310–126302)	134816 (134058–136082)
KAIBURR-1792	Jasmin	141364 (140882–142142)	143058 (142620–143842)	148502 (147944–149212)
KAIBURR-4608	C/asm	648064 (645678–650674)	654366 (651728–657234)	676840 (674446–679554)
KAIBURR-4608	Jasmin	747376 (745792–749120)	749472 (747868–751250)	764490 (762874–766280)
KAIBURR-6144	C/asm	1095102 (1091646–1098922)	1097334 (1093566–1101348)	1127588 (1124030–1131712)
KAIBURR-6144	Jasmin	1272644 (1270318–1275208)	1272480 (1270246–1274906)	1285796 (1283420–1288436)
FrodoKEM-640	[Fro]	3112706 (3107260–3124744)	3239826 (3233936–3249204)	3195408 (3187982–3208002)
HQC-1	[HQC]	72578 (71902–73220)	139896 (138680–141026)	312692 (310798–314622)
HQC-3	[HQC]	179494 (178058–180668)	332806 (329936–335788)	668334 (663984–672664)
HQC-5	[HQC]	377670 (376224–380158)	705038 (700890–709262)	1357572 (1350500–1364182)

KeyGen is 1.7–3.9× faster, and HQC Enc is 1.6–2.0× faster for the two larger parameter sets. On the other hand, decapsulation, which is HQC's most expensive operation, favors KAIBURR meaningfully: KAIBURR-1792 is 2.3× faster than HQC-1, KAIBURR-6144 is 1.2× faster than HQC-5, and KAIBURR-4608 is within 2% of HQC-3.

7.2 Benchmarks on Pavona's ACC

We additionally benchmark the ACC implementation on Pavona's cycle-accurate RTL simulator. The simulator is deterministic, so repeated runs on the same input are cycle-identical. We therefore run each operation once on a fixed input (all-zero randomness) rather than reporting a median and interquartile range. To obtain per-operation costs, we build the keypair–encapsulate–decapsulate round trip three times, terminating after KeyGen, after Enc, and after Dec, giving total cycle counts M_1, M_2, M_3; then KeyGen $= M_1$, Enc $= M_2 - M_1$, and Dec $= M_3 - M_2$. Results are shown in Table 11[14].

Table 12 breaks down the same round trip using the simulator's per-function and per-instruction execution statistics. Matrix generation (`poly_gen_matrix`), which expands the public seed into $\mathbf{A}$ by rejection sampling on SHAKE-128 output, takes 73–84% of all cycles. Almost none of this is spent waiting for the hardware SHAKE-128 core, since stalls on `bn.wsrr` stay at about 6% of the total for every parameter set. The cost is the software rejection-sampling loop itself, namely its mask/compare/select arithmetic (`bn.and`, `bn.cmp`, `bn.sel`) and the accept/reject branches (`bne`, `beq`).

[14]To the best of our knowledge, there is no publicly available implementation of FrodoKEM on ACC, so we omit this comparison.

Table 11: Benchmark results in cycles for the ACC implementation on Pavona's cycle-accurate RTL simulator. Values are exact cycle counts.

KEM	KeyGen	Enc	Dec
KAIBURR-1792	240816	239348	257474
KAIBURR-4608	1453490	1424342	1467109
KAIBURR-6144	2505793	2449864	2506070

Table 12: Share of total round-trip cycles in the ACC benchmark, from the simulator's per-function and per-instruction execution statistics. The four middle columns split `poly_gen_matrix` by instruction class. All values are percentages of each variant's total round-trip cycles, and cell shading is scaled to the largest value in the table.

		of which					
KEM	`poly_gen_matrix`	Rejection sampling	Branch overhead	Keccak wait	Memory latency	`basemul_acc`	Noise $f(t)$
KAIBURR-1792	72.6	60.6	6.2	5.5	0.3	8.1	6.3
KAIBURR-4608	80.8	67.6	6.8	6.0	0.4	9.2	4.0
KAIBURR-6144	83.6	69.9	7.1	6.2	0.4	9.6	2.1

8 Discussion

Our results show that with the proposed generalization of the noise distribution used in ML-KEM, we can increase the module rank of ML-KEM and thereby obtain a massive security margin to hedge against possible future improvements in solving (M)LWE. The largest parameter set we propose, KAIBURR-6144, reaches Core-SVP estimates of 1717 bits classical and 1557 bits quantum, while still outperforming the *smallest* parameter set of FrodoKEM in every regard: it has slightly smaller public keys and ciphertexts and is more than twice as fast. Also in comparison with HQC, KAIBURR is competitive in performance at a much larger security margin.

> *What does this mean for deployments of post-quantum cryptography that aim for extra protection against possible future advances in cryptanalysis and are willing to accept a significant performance decrease compared to, e.g., simply deploying ML-KEM-1024?*

Certainly, *some* deployments are constrained to standardized schemes and parameter sets, possibly even to specific combiners or non-hybrid deployment. In the following discussion, we ignore such constraints imposed by policy or certification requirements and instead focus purely on security and performance. We note, however, that the smallest parameter set of FrodoKEM, which inspires our performance limit for KAIBURR, is *not* standardized by ISO [San26].

General hybridization remarks. The first observation is that for any post-quantum KEM at a performance profile in the ballpark of FrodoKEM, a hybrid combination with classical ECDH should be a no-brainer. To support this statement, let us consider the performance of the NIST level-3 parameter set of FrodoKEM and of X25519 ECDH on the Intel Coffee Lake microarchitecture (cf. Table 9). Median cycles for FrodoKEM key generation are 8270912, for encapsulation 8528877, and for decapsulation 8411864. According to SUPERCOP [BL24], X25519 takes 27345 median cycles for key generation and 83438 cycles for shared-key computation on the same microarchitecture (specifically, on an Intel Xeon

E-2124), so adding those cycles to FrodoKEM costs less than 2% overhead (plus the cost of hashing incurred by the combiner). In terms of increase to public-key and ciphertext sizes the argument is even more obvious: adding 32 bytes to objects of more than 15 KB should not be noticeable in pretty much any application. Given the remarkably stable security track record of ECC, this hybrid is very likely to protect ciphertexts until the arrival of large universal quantum computers, even if lattice cryptanalysis makes unexpected progress to the extent that FrodoKEM ciphertexts are no longer secure.

A slightly less obvious recommendation is to always combine FrodoKEM (or any other post-quantum KEM at a similar performance profile) with at least a standardized parameter set of ML-KEM. As ML-KEM offers performance roughly comparable to (in fact, somewhat better than) that of X25519, the argument regarding computational overhead is similar to that for X25519. With regard to the increase in ciphertext and public-key size, the overhead becomes noticeable, but remains solidly at single-digit percentages. One might think that such a hybrid does not actually add much in terms of security, as a break of FrodoKEM is fairly likely to also affect ML-KEM. However, this is not true for software bugs and implementation attacks, which may well affect only one of the two schemes. This is particularly true as implementation security of post-quantum schemes is more complex and far less mature than for ECC. Hence, adding ML-KEM to FrodoKEM offers additional *post-quantum* security at a very low cost.

FrodoKEM vs. FrodoKEM+Kaiburr hybrids. Even more importantly, and this is the central message of this paper, our results call into question the use of high-security parameter sets of FrodoKEM: A combination of FrodoKEM-640 and Kaiburr-1792 offers significantly better performance than FrodoKEM-976 both in terms of cycle counts and in terms of ciphertext and public-key sizes. A combination of FrodoKEM-640 and Kaiburr-4608 is still considerably faster than FrodoKEM-976 and has only slightly larger ciphertext and public key.

Similarly, a combination of FrodoKEM-976 and Kaiburr-1792 offers much better performance than FrodoKEM-1344 both in terms of speed and size and a combination of FrodoKEM-976 and Kaiburr-4608 is still faster than FrodoKEM-1344 with public keys and ciphertexts only about 6% larger. In terms of security, such hybrids of FrodoKEM and Kaiburr hedge simultaneously against even immense improvements of attacks against lattice schemes in general (through the huge margin of Kaiburr) and against any improvements in attacks that only apply to structured lattices (through the use of FrodoKEM). An advantage of a higher-security FrodoKEM variant would come into play only if both structured lattices are completely broken and attacks against unstructured lattices improve significantly, but not by too much (i.e., within the slightly increased margin of the larger FrodoKEM parameters). Additionally, a combination of Kaiburr and FrodoKEM has the same benefits of some additional hedging against implementation attacks that we mention above in the context of hybridization with ML-KEM.

In fact, if one is willing to accept the speed of FrodoKEM-1344 and somewhat larger sizes, a hybrid of FrodoKEM-976, Kaiburr-1792, and HQC-3 is worth considering. On Coffee Lake, it is *still* 1.5–1.7× faster than FrodoKEM-1344 (8.6–9.4M vs. 14.5–14.8M cycles per operation), with public keys only 6% larger and ciphertexts 28% larger.

Future work. We see various directions of future work, mostly on the engineering and assurance side. For example, if one were to actually deploy Kaiburr in software, we would recommend extending the functional correctness proof of ML-KEM from [ABB+23, AOB+24, AAB+25] to the Jasmin implementations of Kaiburr. Deployments on Pavona's ACC will require significantly increased data memory, but if such a deployment were considered, one may want to consider also adding hardware acceleration for the rejection sampling to generate the matrix $\mathbf{A}$. This is by far the most expensive step in Kaiburr

on the ACC, accounting for between 73% and 84% of the total cycles for the different parameter sets. Finally, for this paper we refrained from actually implementing all the hybrid combinations, possibly with different choices for the combiner. We leave this as an open task with a concrete instantiation of schemes, parameter sets, and combiner for systems that consider using such a hybrid.

Acknowledgements

This work was supported by the Dutch Ministry of Education, Culture, and Science through Gravitation project "Challenges in Cyber Security – 024.006.037", by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972, and by the Swiss SNSF grant 10.008.425 PretZeL. Large parts of this work were done while the first author was employed at MPI-SP.

References

[AAB+25] José Bacelar Almeida, Gustavo Xavier Delerue Marinho Alves, Manuel Barbosa, Gilles Barthe, Luís Esquível, Vincent Hwang, Tiago Oliveira, Hugo Pacheco, Peter Schwabe, and Pierre-Yves Strub. Faster verification of faster implementations: Combining deductive and circuit-based reasoning in EasyCrypt. In Marina Blanton, William Enck, and Cristina Nita-Rotaru, editors, *2025 IEEE Symposium on Security and Privacy*, pages 3820–3838. IEEE Computer Society Press, May 2025. 19

[ABB+17] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, Arthur Blot, Benjamin Grégoire, Vincent Laporte, Tiago Oliveira, Hugo Pacheco, Benedikt Schmidt, and Pierre-Yves Strub. Jasmin: High-assurance and high-speed cryptography. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1807–1823. ACM Press, October / November 2017. 7

[ABB+23] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, Benjamin Grégoire, Vincent Laporte, Jean-Christophe Léchenet, Tiago Oliveira, Hugo Pacheco, Miguel Quaresma, Peter Schwabe, Antoine Séré, and Pierre-Yves Strub. Formally verifying Kyber episode IV: Implementation correctness. *IACR TCHES*, 2023(3):164–193, 2023. 7, 14, 19

[ABC+25] Gorjan Alagic, Maxime Bros, Pierre Ciadoux, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, Hamilton Silberg, Daniel Smith-Tone, and Noah Waller. Status report on the fourth round of the nist post-quantum cryptography standardization process. Technical Report NIST IR 8545, National Institute of Standards and Technology, March 2025. https://csrc.nist.gov/pubs/ir/8545/final. 1

[ABD+17] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber — algorithm specifications and supporting documentation. Submission to the NIST Post-Quantum Cryptography Standardization Project, November 2017. Round 1 submission, 2017-11-30, https://pq-crystals.org/kyber/data/kyber-specification.pdf. 5, 6

[ABD+19] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber (version 2.0) — algorithm specifications and supporting documentation. Submission to round 2 of the NIST Post-Quantum Cryptography Standardization Project, March 2019. Round 2 submission, 2019-03-30, https://pq-crystals.org/kyber/data/kyber-specification-round2.pdf. 6

[ADPS16] 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 2016*, pages 327–343. USENIX Association, August 2016. 5

[AOB+24] José Bacelar Almeida, Santiago Arranz Olmos, Manuel Barbosa, Gilles Barthe, François Dupressoir, Benjamin Grégoire, Vincent Laporte, Jean-Christophe Léchenet, Cameron Low, Tiago Oliveira, Hugo Pacheco, Miguel Quaresma, Peter Schwabe, and Pierre-Yves Strub. Formally verifying Kyber - episode V: Machine-checked IND-CCA security and correctness of ML-KEM in EasyCrypt. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part II*, volume 14921 of *LNCS*, pages 384–421. Springer, Cham, August 2024. 19

[AOP+25] Amin Abdulrahman, Felix Oberhansl, Hoang Nguyen Hien Pham, Jade Philipoom, Peter Schwabe, Tobias Stelzer, and Andreas Zankl. Towards ML-KEM & ML-DSA on OpenTitan. In Marina Blanton, William Enck, and Cristina Nita-Rotaru, editors, *2025 IEEE Symposium on Security and Privacy*, pages 1–19. IEEE Computer Society Press, May 2025. 7, 11

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *J. Math. Cryptol.*, 9(3):169–203, 2015. http://www.degruyter.com/view/j/jmc.2015.9.issue-3/jmc-2015-0016/jmc-2015-0016.xml. 6

[BBF+19] Nina Bindel, Jacqueline Brendel, Marc Fischlin, Brian Goncalves, and Douglas Stebila. Hybrid key encapsulation mechanisms and authenticated key exchange. In Jintai Ding and Rainer Steinwandt, editors, *PQCrypto 2019*, pages 206–226. Springer, Cham, 2019. 2

[BCD+16] Joppe W. 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 *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1006–1018. ACM, 2016. 1, 2

[BCLvV17] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU Prime: Reducing attack surface at low cost. In Carlisle Adams and Jan Camenisch, editors, *Selected Areas in Cryptography - SAC 2017 - 24th International Conference, Ottawa, ON, Canada, August 16-18, 2017, Revised Selected Papers*, volume 10719 of *Lecture Notes in Computer Science*, pages 235–260. Springer, 2017. 1

[BDGL16] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In Robert Krauthgamer, editor, *27th SODA*, pages 10–24. ACM-SIAM, January 2016. 5

[BDK+18] Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé.

CRYSTALS-Kyber: A CCA-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*, pages 353–367. IEEE, 2018. 1, 2

[BGR+21] Joppe W. Bos, Marc Gourjon, Joost Renes, Tobias Schneider, and Christine van Vredendaal. Masking Kyber: First- and higher-order implementations. *IACR TCHES*, 2021(4):173–214, 2021. 10

[BKL+25] Manuel Barbosa, Matthias J. Kannwischer, Thing-Han Lim, Peter Schwabe, and Pierre-Yves Strub. Formally verified correctness bounds for lattice-based cryptography. In Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng Shieh, David Lie, and Véronique Cortier, editors, *ACM CCS 2025*, pages 156–169. ACM Press, October 2025. 3, 4, 6, 10, 11

[BL24] Daniel J. Bernstein and Tanja Lange, editors. *eBACS: ECRYPT Benchmarking of Cryptographic Systems*, 2024. https://bench.cr.yp.to. Accessed: 2026-10-15. 18

[CMST25] Kévin Carrier, Charles Meyer-Hilfiger, Yixin Shen, and Jean-Pierre Tillich. Assessing the impact of a variant of MATZOV's dual attack on kyber. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part I*, volume 16000 of *LNCS*, pages 444–476. Springer, Cham, August 2025. 2

[CRY] CRYSTALS-Kyber team. Security estimate and decryption-failure scripts for CRYSTALS-Kyber. https://github.com/pq-crystals/security-estimates. GitHub repository. Accessed: 2026-10-15. 5, 6

[DEEK24] Léo Ducas, Andre Esser, Simona Etinski, and Elena Kirshanova. Asymptotics and improvements of sieving for codes. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VII*, volume 14657 of *LNCS*, pages 151–180. Springer, Cham, May 2024. 2

[DKRV18] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. In Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *AFRICACRYPT 18*, volume 10831 of *LNCS*, pages 282–305. Springer, Cham, May 2018. 1

[DRV20] Jan-Pieter D'Anvers, Mélissa Rossi, and Fernando Virdia. (One) failure is not an option: Bootstrapping the search for failures in lattice-based encryption schemes. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 3–33. Springer, Cham, May 2020. 5

[Fro] FrodoKEM Team and Microsoft Research. PQCrypto-LWEKE: FrodoKEM. https://github.com/microsoft/PQCrypto-LWEKE. GitHub repository. Accessed: 2026-09-07. 15, 16, 17

[GHH26] Yiming Gao, Xuyuan Han, and Honggang Hu. Provable subexponential algorithms for NIST third-round lattice families. Cryptology ePrint Archive, Paper 2026/2386, 2026. 2

[GHP18] Federico Giacon, Felix Heuer, and Bertram Poettering. KEM combiners. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 190–218. Springer, Cham, March 2018. 2

[GJ21] Qian Guo and Thomas Johansson. Faster dual lattice attacks for solving LWE with applications to CRYSTALS. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 33–62. Springer, Cham, December 2021. 2

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 341–371. Springer, Cham, November 2017. 6

[HQC] HQC Team. HQC: Hamming Quasi-Cyclic. https://gitlab.com/pqc-hqc/hqc. GitLab repository. Accessed: 2026-09-07. 15, 16, 17

[HQC25] HQC Resources Team. HQC specifications. Documentation PDF, http://pqc-hqc.org, August 2025. Accessed: 2026-09-16. 1

[ISO26] ISO/IEC. Information technology — Security techniques — Encryption algorithms — Part 2: Asymmetric ciphers — Amendment 2. Standard ISO/IEC 18033-2:2006/Amd 2:2026, International Organization for Standardization, Geneva, Switzerland, June 2026. https://www.iso.org/standard/86890.html. 2

[KKWS26] Kris Kwiatkowski, Panos Kampanakis, Bas Westerbaan, and Douglas Stebila. Post-quantum hybrid ECDHE-MLKEM key agreement for TLSv1.3. RFC 10024, Internet Engineering Task Force, August 2026. https://www.rfc-editor.org/rfc/rfc10024. 2

[Laa16] Thijs Laarhoven. *Search problems in cryptography: From fingerprinting to lattice sieving*. PhD thesis, Eindhoven University of Technology, 2016. 5

[LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *DCC*, 75(3):565–599, 2015. 5

[MAT22] MATZOV. Report on the security of LWE: Improved dual lattice attack. Report, Zenodo, April 2022. 2, 6

[MB25] Victor Duarte Melo and Willian J. Buchanan. KyFrog: A high-security LWE-based KEM inspired by ML-KEM. arXiv preprint arXiv:2512.06411, 2025. 2, 15

[Nat16] National Institute of Standards and Technology. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process, 2016. https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf. 11

[Nat20] National Institute of Standards and Technology. Post-quantum cryptography: Round 3 submissions. https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions, 2020. Accessed: 2026-07-27. 1

[Nat24a] National Institute of Standards and Technology. Module-lattice-based digital signature standard. Technical Report Federal Information Processing Standards Publication (FIPS) 204, U.S. Department of Commerce, Washington, D.C., August 2024. https://doi.org/10.6028/NIST.FIPS.204. 1

[Nat24b] National Institute of Standards and Technology. Module-lattice-based key-encapsulation mechanism standard. Technical Report Federal Information Processing Standards Publication (FIPS) 203, U.S. Department of Commerce, Washington, D.C., August 2024. https://doi.org/10.6028/NIST.FIPS.203. 1, 2, 4, 6

[Nat24c] National Institute of Standards and Technology. Stateless hash-based digital signature standard. Technical Report Federal Information Processing Standards Publication (FIPS) 205, U.S. Department of Commerce, Washington, D.C., August 2024. https://doi.org/10.6028/NIST.FIPS.205. 1

[NIS25] NIST PQC Forum. How is HQC substantially different from RLWE? NIST PQC Forum discussion, October 2025. https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/PBC-KPOCSF0. Accessed: 2026-08-29. 2

[NP26] Ruben Niederhagen and Hoang Nguyen Hien Pham. Improving ML-KEM and ML-DSA on OpenTitan – efficient multiplication vector instructions for OTBN. *IACR TCHES*, 2026(2):495–519, 2026. 11

[Per25] Ray Perlner. FIPS 206: FN-DSA (Falcon). Presentation at the 6th NIST PQC Standardization Conference, September 2025. https://csrc.nist.gov/presentations/2025/fips-206-fn-dsa-falcon. 1

[PQ 26] PQ Code Package. mlkem-native: A unified, fast and verified ML-KEM library. https://github.com/pq-code-package/mlkem-native, 2026. GitHub repository. Accessed: 2026-09-17. 7

[San26] Sanketh. Re: [pqc-forum] Re: Quasipolynomial attack on Classic McEliece. Post to the pqc-forum mailing list, 2026. https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/2kfyUnLObWQ/m/Zmntq4UcAQAJ. Accessed: 2026-10-15. 18

[SBG+23] Basavesh Ammanaghatta Shivakumar, Gilles Barthe, Benjamin Grégoire, Vincent Laporte, Tiago Oliveira, Swarn Priya, Peter Schwabe, and Lucas Tabary-Maujean. Typing high-speed cryptography against spectre v1. In *2023 IEEE Symposium on Security and Privacy*, pages 1094–1111. IEEE Computer Society Press, May 2023. 14

[The26] The FrodoKEM Team. FrodoKEM: Practical quantum-secure key encapsulation from generic lattices. https://frodokem.org/#news, 2026. News section. Accessed: 2026-07-27. 2