Post-Quantum Cryptography for Big Data: A Comparative Performance Evaluation of NIST-Selected Post-Quantum Cryptographic Schemes

Arnab Ghosh

March 29, 2024

Abstract

Post-quantum cryptography has developed significantly in response to the acceleration of the possibility of a quantum computer capable of running Shor's algorithm, which theoretically may break the current state-of-the-art cryptographic standards based on RSA (factorization) and ECC (elliptic curve) cryptography. Currently, the projected cryptographic standard by the National Institute for Standards in Technology (NIST) has identified four lattice-based learning-with-errors cryptographic algorithms as the likely successors for a quantum-resistant cryptographic standard. However, LWE and Lattice-based cryptographic algorithms are known to be more memory inefficient than their state-of-the-art counterparts. This paper measures the comparative performance of the 4 NIST-selected post-quantum cryptographic algorithms: Dilithium, Kyber, Falcon, and SPHINCS to the current state-of-the-art in traditional cryptography using large encryption payloads and strict performance constraints.

Introduction Cryptography Cryptography is the discipline concerning the protection of information. Primarily, cryptography is used to secure communication between two parties or to secure stored data. Cryptographers work to either *encrypt* information (obfuscate the message delivered from all non-intended parties), *decrypt* information (de-obfuscate the information for the intended party), or communicate the obfuscated message from one party to another in a secure manner. In general, we define a cryptographic algorithm as an algorithm that achieves one or more of the specified functions. We define a cryptographic scheme as a set of cryptographic algorithms or a general approach for satisfying all of these requirements [@schneierAppliedCryptographyProtocols1996].

When a cryptographic algorithm encrypts information, we call the message we wish to encrypt the *plaintext*. The algorithm successfully encrypts this plaintext into a *ciphertext*, which is obfuscated from all non-intended parties. The intended party, then, decrypts the ciphertext back to its plaintext form using a decryption algorithm [@lamCryptographyComputationalNumber2001].

The vast majority of cryptographic schemes utilize a special set of mathematical problems we define as *intractable*. These intractable problems are mathematical problems that are infeasible to solve without some information, which we may call a key. The key is generated as some part of the cryptographic scheme, then used by both parties to encrypt or decrypt the desired message [@lamCryptographyComputationalNumber2001]. KEM For the purposes of this paper, we shall consider a cryptographic scheme (cryptosystem) as any KEM, or Key Exchange Mechanism. A KEM is a corresponding set of three algorithms. There are many more complexities to a KEM than specified in this paper; however, those details are not necessary for the purposes of understanding this paper. The first algorithm in a KEM is a key generation algorithm, which generates two keys: a public key and a private key (also called a secret key).

The second KEM algorithm is called an encryption or encapsulation algorithm. As the name suggests, the encryption algorithm obfuscates the plaintext and returns a ciphertext. Crucially, this function takes the aforementioned public key as an input. The third algorithm, then, is called a decryption or decapsulation algorithm. This algorithm performs the inverse of the encryption algorithm, taking both the generated ciphertext and the private key. RSA The most prominent cipher in modern cryptography, [[RSA]] (named after its architects, Rivest-Shamir-Adleman), utilizes the *factoring into primes problem*, which considers the prime factorization of very large numbers [@schneierAppliedCryptographyProtocols1996]. The scheme is named for the three scientists who devised it at the Massachusetts Institute of Technology in 1978. We define the factorization problem as follows: given a positive integer $x \in \mathbb{Z}^+$, find the unique set of prime numbers $p_1, p_2, p_3, \cdots p_n$ and integer exponents $k_1, k_2, k_3, \cdots, k_n$ such that $x = p_1^{k_1} p_2^{k_2} p_3^{k_3} \cdots p_n^{k_n}$. Interestingly, it has been mathematically proven that there is *no efficient way* to factor large numbers; even the fastest methods are little more effective than trial and error.

However, the factorization problem becomes trivial with a crucial bit of information: just one of these factors and an associated exponent. Given such a pair, we may simply divide the factor by x: $p^k \cdot x^{-1}$ in order to calculate the rest of the factorization trivially. Logically, then, [[RSA]] creates a cryptosystem using the prime factorization problem as defined below. Mathematical Definition Key Generation First, we pick two very large numbers $p, q \in \mathbb{Z}$:

$$p, q \in \mathbb{Z} \text{ (and very large)}$$

$$n := pq$$

Then, we generate a number e using this huge number n. Note that λ is any totient function, usually chosen as Carmichael's totient but sometimes Euler's. A totient function is any function that counts up the amount of prime numbers to an integer $n \in \mathbb{Z}^+$.

$$2 \leq e \leq \lambda(n) \text{ and } \gcd(e, \lambda(n)) = 1$$

e can be chosen from random with the parameters above. Randomness is an extremely important tool within cryptography. Note that it is mathematically dubious to pick a number at random but the threat this poses to literally any mathematical operation that is not purely theoretical with no consequences is trivial. We set the public key to be distributed freely as (n, e). Finally, we can generate the private key d as follows:

$$d := e^{-1} \mod \lambda(n)$$

All of this satisfies the first requirement of [[KEM]]: key generation. Encapsulation/Encryption We are now ready to encrypt a message. First, we have to take the given message and hash the message; in other words, turn it into numbers.

Hashing the message has the secondary benefit of obfuscating the method enough such that you can't derive the original message. While this may seem counterintuitive, consider the case that we want to simply *verify* a message rather than send it. By hashing the message, even if a third party were to get access to the hash somehow, they would not be able to use it in any way. Password verification, for example, utilizes hashes to this effect. However, if we do not wish to obfuscate the message, we can use a simple hash such as Base64 that can be inverted.

For mathematical purposes, define the hashing function $h : A \to \mathbb{Z}$ where A is the set containing the message.

$$m := h(\text{message})$$

$$c := m^e \mod n \text{ where } 0 \leq m < n$$

We call c the ciphertext, which we distribute in this case. Note that anyone can generate the ciphertext, as long as they have the message they wish to encrypt and the public key they wish to use. Decapsulation/Decryption After encrypting the message, we are now going to decrypt the message. Remember that as the decrypting party, we have access to three variables: 1. The ciphertext, c 2. The public key, (n, e). 3. The private key, d. Also remember that the goal of the algorithm is to derive m, the original hashed message. RSA allows us to compute this cleverly:

$$c^d \equiv m^{ed} \equiv m \mod n$$

Notice that we did not generate m; rather, we found $m \mod n$. However, because $0 \leq m < n$, $m \mod n = m$. We have finished decrypting our message, upon which we can either de-hash or verify.

You may have noticed a flaw with the above appraisal: we assume that both parties already have the public and private keys. One may consider that previously, we presumed most cryptosystems to be using a singular key in order to crack the ciphertext. There is a problem with this single-key approach, however: we must convey the key somehow to the intended party without the prying eyes of unintended parties. In the past, this was achieved through physical means beforehand; however, in the modern era, most cryptosystems implement the Diffie-Hellman key exchange algorithm (or something very similar) to achieve this.

The details of the Diffie-Hellman algorithm are not necessary to understand this paper; simply recall that Diffie-Hellman results in the distribution of *two* keys: a public key and a private key. The public key is freely distributed, and the private key is solely in the hands of the two verified parties. The first algorithm of a KEM cryptosystem generates these two keys. Elliptic Curve Cryptography Recently, an alternative to RSA cryptosystems has arisen: Elliptic-Curve Cryptography, or ECC for short. ECC relies on a different mathematical construction: namely, the discrete logarithm problem [@schneierAppliedCryptographyProtocols1996]. Define the discrete logarithm problem as follows: given a group G and generator $g \in G$ and element $h \in G$, we can say that $\log_g(h) = x$, if x exists such that x is an integer and $g^x = h$. To take advantage of the discrete logarithm problem, elliptic curve cryptography defines elliptic curves over finite fields. In doing so, they are able to use much smaller key sizes than other algorithms while maintaining similar or even improved levels of security, especially in security-critical IoT infrastructure often afforded significant performance hindrances [@dhillonEllipticCurveCryptography2016] [@liuIoTNUMSEvaluatingNUMS2019][@tiwariNovelMethodDNABased2018].

Elliptic Curve Cryptography is largely considered preferential to RSA and Diffie-Hellman based approaches to security. NIST has standardized the set of elliptic curves in Federal Information Processing Standard(s) (FIPS) 186, currently in its 5th iteration (note that the fourth generation of elliptic curve standards set by FIPS 186-4 is set to be retired in the early weeks of February 2024) [@DigitalSignatureStandard2023].

With a basic understanding of the cryptographic primitives necessary for this paper, we shall now explore the requisite quantum computing terminology relevant to this paper. Quantum Computing The vast majority of computers follow a classical computing model. In fact, classical computing has become (rightfully so) synonymous with computation as a whole. Classical computers are precise and exact: they store data in a concrete and binary manner, in units of bits. Computation occurs through logic gates and electrical circuits; these calculations have little to no noise associated with them. As a result, ironically, classical computers are slow: classical computers perform jobs thoroughly, exactly, yet slowly.

The preciseness and reliability of classical computers, however, are extremely useful. At scale, classical computers are still fast enough to solve most problems demanded of them; additionally, algebraic and computational theory has evolved to a sufficient maturity to which the drawbacks associated with classical computers are not true drawbacks in the majority of cases. However, due to their fundamentally thorough nature, classical computers still remain limited.

Quantum computing, however, acts as a complement to classical computing models. Fittingly taking advantage of the notoriously unpredictable principles of quantum mechanics, such as superposition, quantum computers are notoriously difficult to work with. Rather than classical bits, which can either be 0 or 1 (denoting on or off on a circuit board, and 0 or 1 mathematically), quantum computers store data in qubits, which oscillate freely between 0 and 1 and may even simultaneously occupy multiple values. Where classical computers emphasize accuracy and preciseness, quantum computers are noisy and unpredictable. In fact, the very act of attempting to gauge the value of a qubit can effect not only its value, but the value of other qubits.

However, the seemingly paradoxical and imprecise nature of quantum computing also belies its greatest strength. Because quantum computers can store and process massive amounts of data simultaneously, they are a great fit for applications such as scientific simulations, which often do not require the accuracy of classical computers but desire the power of quantum computers. For this, and many other tasks which often require brute-force methods on classical computers, quantum computers show promise as the compute ideology of the future. One such application, named Shor's Algorithm, threatens nearly all modern security.

However, while quantum computers have matured at an extraordinary pace, quantum computers are, compared to their classical counterparts, infants. Even the largest quantum computers struggle to reach qubit counts above 100, rendering current quantum computers useless for the vast majority of applications. For this reason, most optimism regarding quantum computing is limited to future possibilities. However, quantum computers have grown exponentially in the past decade; with increasing interest from government and private parties alike, all of which are eager to unearth its power, quantum computing may become viable much sooner than current estimates project. Time Complexity As a small aside, we note that we may refer to terms such as linear or exponential run time. This simply refers to the mathematical function class which best approximates the time taken by an algorithm when given larger and larger inputs. For example, an algorithm which runs in linear time has a time signature approximated by a function of the class $T : x \mapsto ax + b$; an exponential algorithm has a time signature of the class $T : x \mapsto ba^x$. We can imagine the many different types of time complexities which exist for many different algorithms. Only elementary familiarity of time complexity is required for comprehension of our experiment.

Literature Review Shor's Algorithm Recall from the previous section that quantum computers pose an existential threat to current cryptographic standards, compromising both the RSA and ECC cryptosystems. This threat is due to an algorithm devised by Mathematician Peter Shor in 1994; ironically, it remains the only concrete feasible implementation of a quantum algorithm for a useful problem.

Named Shor's Algorithm after its creator, the program solves the aforementioned prime factorization and discrete logarithm problems without the need for a key, thus bypassing all security guaranteed by the two protocols. Shor's Algorithm solves both problems by solving a related problem, the order finding problem, defined as follows for a given positive integer $a \in \mathbb{Z}^+$, and larger integer N:

$$\text{ord}_a = \text{ the smallest } r \text{ where } a^r \equiv 1 \mod N$$

Utilizing some clever mathematics, one can convert both the discrete logarithm and prime factorization problem to the order finding problem. Shor's Algorithm, then, by finding a solution to the order finding solution, by proxy, has found a solution to the other two problems. In fact, Shor's Algorithm disqualifies a whole host of cryptographic schemes; RSA and KEM are simply the most important.

With two of the state-of-the-art in cryptographic systems rendered vulnerable to quantum-based attacks, it becomes vital that secure post-quantum cryptography (PQC) is developed, immune to Shor's Algorithm. Quantum Cryptography Note that this paper concerns *post-quantum cryptography*, not quantum cryptography. Quantum Cryptography is a separate field of cryptography concerning how to use the properties and expanded capabilities of quantum computers to *produce* security- we are simply concerned with preventing quantum computers from breaking existing security utilizing classical computers [@gisinQuantumCryptography2002].

Learning with Errors In response to the threat presented by Shor's Algorithm, The National Institute for Standards in Technology (NIST) held a public competition to standardize a post-quantum cryptographic scheme resistant to Shor's Algorithm. Of the four algorithms eventually selected to proceed by NIST, three of the four algorithms utilized the *Learning with Errors* (LWE) problem in order to guarantee security. LWE effectively combines two mathematical problems, the Closest Vector Problem (CVP) and the Shortest Vector Problem (SVP) in order to guarantee security [@blanco-chaconRingLearningErrors2019]. SVP is defined as follows. Note that the magnitude of a vector v is given with the following formula [@schneierAppliedCryptographyProtocols1996] :

$$||v|| = \sqrt{\sum_{i=1}^{n} v_i^2}$$

Now, given a lattice L, imagine finding a vector $x \in L$ such that for every $y \in L$, the magnitude $||x|| \leq ||y||$ - essentially, the smallest vector problem. CVP is very similar: given a point $y \in \mathbb{R}^n$, find a vector x such that for all other vectors $z \in L$, $||y - x|| \leq ||y - z||$ - in essence, you are finding the closest vector to a point.

LWE effectively combines both the CVP and SVP problems for a lattice of many dimensions in order to guarantee security - while these problems are feasible to solve in a few dimensions, adding dimensions to this problem greatly increases its difficulty to a level considered unbreakable by quantum computers with current knowledge.

However, LWE is also much less efficient than the state-of-the-art ECC or RSA. Whereas ECC is commonly accepted as fast, viable, and secure even in low performance devices [@dhillonEllipticCurveCryptography2016], LWE is only approaching [@khalidLatticebased-CryptographyIoT2019] [@guillenPostquantumSecurityIoT2017], [@seyhanLatticebasedCryptosystemsSecurity2022] that designation. However, early results are very promising in regards to CPU performance [@akleylekNewLatticebasedAuthentication2022]. NIST We note that in August 2023, NIST authored three reports which outline the post-quantum cryptographic standards of all United States federal communications: FIPS 203, FIPS 204. and FIPS 205 [@ThreeDraftFIPS2023]. These reports outline the use of Kyber in particular as an encryption standard and outline the procedure for creating a KEM using Kyber. In order to gauge the effectiveness of post-quantum cryptography, we opt to use the standards outlined by NIST in order to ensure the applicability of our experiment. The Kyber specification itself outlined by PQ-Crystals may be adapted using a variety of different implementations with different efficiencies; we opt to use the most official specification via NIST.

Methods We shall begin by stating the choices in the design of our experiment, then following up this overview with a justification of these choices.

In order to gauge the feasibility of the NIST-selected post-quantum algorithms, we opt to test two algorithms: Kyber and ECC-KEM (the Elliptic Curve Integrated Encryption Scheme, or ECIES). Our approach is purely quantitative in nature, collecting the following performance metrics: CPU usage, as a percentage; memory usage, in megabytes; and time, in seconds. We use a Haswell-based 12-core Intel Xeon processor; however, we opt to test each algorithm on a single thread. Additionally, in order to guarantee a level of separation from the base operating system, we run each benchmark inside a Docker virtual machine. Why Quantitative In line with most cryptographic benchmarks, we choose a quantitative study primarily to optimize ease of use. Quantitative figures are easily available when measuring CPU and memory usage; therefore, in order to take advantage of our mathematical toolkit to analyze figures, we opt to use quantitative data. Crucially, we plan to collect data at a scale in which manual data analysis is not feasible - quantitative data allows us to computationally analyze these results simultaneously to find the patterns we seek efficiently. Past Methods We primarily attempt to adhere to PQ-Crystals' own methodology for benchmarking Kyber in the design of this experiment [@westerbaanX25519Kyber768Draft00HybridPostquantum2023]. However, we must reconcile this goal with the reality that Kyber and ECIES are very different algorithms; therefore, because we must ensure that there are minimal differences in the benchmarking setup between the two algorithms, we opt to build our own benchmarking script for both ECIES and Kyber [@liuIoTNUMSEvaluatingNUMS2019]. Let us describe the individual benchmarking script for each algorithm a *test saddle*, and relegate the benchmark label to the script producing the performance data. Research Instruments Our experiment, then, in line with other experiments, designs a test saddle and runs this test saddle with a benchmark for each algorithm. We then compare the results for each of the discrete combinations we wish to test. One may notice a large discrepancy in our experiment compared to other experiments: whereas the majority of other experiments measure CPU performance primarily through cycles, we opt to measure CPU performance as a percentage of total CPU usage. We recognize that there are several advantages to using cycles over a percentage of total usage. Cycles are far more standardized and accurate when compared percentages. However, recall that the focus of this experiment is not to compute a direct performance comparison between the two algorithms; rather, we wish to gauge how *viable* post-quantum algorithms are. Also recall that it is well established that compute limitations on post-quantum algorithms are not limited even by small CPUs; rather, we are far more concerned with memory usage. As a consequence of testing two very different algorithms on Linux systems, it therefore makes sense to prioritize the convenience of gathering memory usage. In essence, we are sacrificing good CPU usage data in order to gain more reliable and useful information in the form of memory data.

Procedures It is true that NIST has selected other post-quantum algorithms: namely, Dilithium; however, Dilithium and Kyber share the same mathematical foundation (the Learning With Errors problem). If the purpose of this study were to be a direct performance comparison between different cryptographic algorithms, we would be obligated to test both Dilithium and Kyber. However, because we solely wish to test the feasibility of these algorithms rather than their direct performance, we conclude that it is adequate to simply test Kyber. Additionally, PQ-Crystals, the organization who developed both Kyber and Dilithium, provides direct CPU performance comparisons under small payloads; therefore, a performance comparison of the two is not necessary. Finally, note that we choose the official implementation of Kyber [@hekkalaImplementingPostquantumCryptography2023].

Also note that this study does not attempt to use multi-threading to optimize either cipher. We choose not to take advantage of multithreading primarily to objectify the study as much as possible. This study does not make any assertions on the effect of effects of multithreading on the effectiveness of either ECIES or KEM, nor does it wish to. Because multithreaded approaches to neither are standardized, we wish to relegate the potential effects of multithreading to future studies and instead solely focus on the feasibility of the official specifications of Kyber and ECIES. Attempting to multithread each would introduce significant differences between the two, compromising any attempt at a comparison [@ebrahimiPostQuantumCryptoprocessorsOptimized2019]. We elaborate on this choice and its consequences in the limitations section of this paper.

Earlier, we note that we use Docker as a virtualization platform. We will briefly justify the use of a virtualization platform. A platform such as Docker allows us to minimize performance overhead while retaining a key advantage of virtualization: independence from the host system. This allows each test to remain as independent and equal as possible, with little to no difference in the starting environments.[@singhAdvancedLightweightEncryption2017] This can remove sources of random bias, such as the operating system mysteriously choosing to allocate more resources to one process over the other. The primary disadvantage of virtualization is the significant performance overhead induced by a virtual platform; however, Docker is minimal as to minimize this effect to where the loss in performance is negligible. Therefore, we opt to use Docker to increase the consistency of our study. Data Analysis

Our benchmarking process generates CSV files which report: a record number, the instantaneous time, the instantaneous CPU usage, the instantaneous memory usage. We aggregate this data to generate the following performance indicators for each combination of payload and algorithm: time elapsed, minimum CPU usage, maximum CPU usage, average CPU usage, minimum memory usage, maximum memory usage, average memory usage. For special cases, Excel was used to generate visuals to manually inspect trends. Truncated revisions of the raw data collected will be provided in the Appendix; however, the size of this data dictates that including all of the data simply is not feasible.

Results In the process of executing our experiment, we unexpectedly come to the conclusion that presently, the application of post-quantum cryptographic algorithms are, in fact, infeasible with large payloads. Furthermore, we conclude that the official specifications of Kyber are fundamentally incompatible with any task that wishes to operate on a larger than normal file, and certainly infeasible even with large computational resources.

Our de-facto control group, ECIES, produced as-expected performance results. As show in Figure 1, with a maximum payload size of a sixteen gigabyte random-byte scratch-file, memory follows a modest, linear increase from around 192MB to 243MB. Note that we achieve such low memory footprint using a segmented approach (splitting our large file into many small files), which is considered standard in the encryption of large files. We do so at the sacrifice of speed. We may nominally consider that there are methods to expend more resources to optimize this process; however, because we wish solely to obtain a benchmark for which to compare Kyber, we choose not to make these optimizations. Make note of this for future reference.

As a result, the elapsed time benchmarks for ECIES increased linearly: with a half-gigabyte sized payload, the benchmark took 14 seconds; with a one-gigabyte sized payload, the benchmark took 30 seconds; with a two-gigabyte sized payload, the benchmark took 58 seconds; with a sixteen-gigabyte sized payload, the benchmark too 487 seconds. In this manner, segmentation is quite palatable because we are able to maintain a linear time complexity. Note that a linear time complexity is not necessarily always indicative of good performance; however, it is generally a desirable label to attain because linear time complexities scale excellently.

— Benchmark — Payload Size (GB) — Time Elapsed (s) — CPU Min (— ——— — ————— — —————- — ———— — ———— — ———— — ————— — ————— — ———— — — ECIES — 0.5 — 14 — 0.13 — 3.9 — 2.4 — 248 — 293 — 291 — — ECIES — 1 — 30 — 0.12 — 3.9 — 2 — 220 — 293 — 270 — — ECIES — 2 — 58 — 0.14 — 4.9 — 1.5 — 208 — 293 — 250 — — ECIES — 4 — 116 — 0.13 — 5.2 — 4 — 302 — 310 — 305 — — ECIES — 8 — 232 — 0.13 — 5.4 — 3 — 309 — 403 — 340 — — ECIES — 16 — 487 — 0.11 — 4 — 3.5 — 192 — 503 — 451 — — Kyber — 0.5 — 42339 — 0.14 — 4.3 — 3.8 — 263 — 504 — 283 —

In line with memory usage, CPU usage using ECIES for the largest payload remained inline and relatively low, osc illating between less than one percent and three percent usage as shown in Figure 1. Once again, we may attribute this relatively low performance overhead to segmentation. Perhaps most surprisingly, the performance impact of differently sized payloads had little variation. This is, once again, most likely a product of segmentation.

Our results begin to break down when inspecting Kyber's performance data. For even a half-gigabyte payload, Kyber takes significantly longer - 42339 seconds, or 11.76 hours - over

3024 times the amount of time expended by ECIES. Kyber's significant time requirement dictated that even for a single gigabyte, we could not feasibly run a full benchmark. Therefore, we must conclude that the Kyber specification outlined for use in FIPS 203-205 are not feasible for use with even medium-sized payloads (the 16GB payload we planned to test), let alone the ones that will be required to formulate a post-quantum transition (TB's of data). Analysis We reiterate that the current state of post-quantum cryptography, at least that outlined by NIST, is infeasible with large payloads. However, we note that this does not appear to be a purely computational limit: while time usage was significantly higher for Kyber, actual performance usage characteristics were not abnormal; in fact, they were much lower than expected. Therefore, we speculate that Kyber's infeasibility is a result of segmentation, as noted earlier. Segmentation allowed the ECIES benchmarks to maintain low levels of resource usage, in addition to slower than expected benchmarks; however, we note that segmentation kept ECIES running in linear time. Even so, we speculate that in real-world scenarios, the size of each segment would expand in order to better utilize the available resources.

Upon further inspection of the FIPS protocols of which our official Kyber implementation is based on, we find that Kyber internally uses 20-bit blocks in their encryption process, ensuring segmentation. 20 bytes is quite small; the mathematical implementation of Kyber requires that bytes are this size in order to permit the usage of certain operations that guarantee Kyber's security. We speculate that Kyber's inefficiency originates from this 20-bit requirement: while certainly usable for smaller payloads, as found in earlier experiments on smaller computers, such segmentation significantly hinders performance on large systems. Note that we do not craft our own implementation of Kyber to specification; we use a pre-implemented official version of Kyber to the FIPS specifications.

Furthermore, we may calculate that the 20 bit requirement required that the half-gigabyte test which ran for 11.76 hours processed a single 20-bit block 25 million times, or processing close to 580 blocks per second. We can conclude, therefore, that even for a much more capable computer, allowing for a larger block size will be required to attain feasibility on larger payloads. Until then, we may rest assured that Kyber works wonderfully for smaller payloads, consuming a small performance footprint. Additionally, the importance of FIPS 203-205 remains of the utmost importance to the post quantum transition, because as prior research notes, it is quite feasible in low-performance environments in scenarios which segmentation is not problematic.

Conclusions Segmentation We come to the unfortunate conclusion that we, unfortunately, cannot conclude that the current NIST standards are feasible for the encryption of larger payloads. In fact, the feasibility for even medium-sized payloads are not currently feasible. While post-quantum cryptographic algorithms may be adequate for most use cases with smaller payloads , significant progress and perhaps a complete retool is required to facilitate usability with big data. We speculate in the analysis section that segmentation makes up the majority of this problem; the current block sizes outlined by NIST for Kyber are too small. Perhaps a larger block, at the cost of using more resources, could be used to facilitate the usage of Kyber with larger payloads. However, it is important to note that the underlying implementation of many cryptographic algorithms rely on blocks of small sizes in order to perform a desired mathematical operation. The security of these algorithms are contingent on these mathematical operations; thus, increasing the block size may be a tall order and may require the development of an entirely new algorithm, or even entirely disqualify certain approaches. Therefore, we fail to make a conclusion on the merits of segmentation.

Multithreading Allow us to consider the case of a medium-sized data-transition: a 10TB database. Let us label the efficiency of our benchmark γ_0. Note that we tested on a 0.5GB sized payload; therefore, assuming a linearly segmented encryption process, we can speculate that our task will take $\frac{10TB}{0.5GB} = 20,000$ times longer than our benchmark assuming a constant efficiency of γ_0, we arrive at 235,200 hours: or, 9,800 days which is nearly 27 years. If we were to assume that our task could run on multiple threads; for the sake of argument, 64, we still require 154 days of continuous data processing. Additionally, this assumes a strict, best-case scenario where the performance benefits afforded by multithreaded parallel-processing occurs as efficiently as possible.

Earlier, we stated that we do not attempt to multithread in this study. We do not multithread precisely because the performance benefits afforded by parallelism and multi-threaded processing are not completely efficient. In other words, allowing our process 2 cores compared to one core would not necessarily double our performance; in fact, in some cases, multithreading can actually decrease performance. While we may hypothesize that our experiment would have been much faster had we decided to multithread, the applicability of our results would be compromised as our specific implementation of multithreading would not have been official. In fact, we recommend further research exploring the efficacy of multithreading in post-quantum encryption schemes; a general idea of the performance benefits yielded by multithreading could pave the way for future development of a more efficiently multithreaded post-quantum encryption algorithm in order to guarantee effectiveness. Limitations There are many limitations associated with our interpretation of this question. For example, we use a quite old hardware setup that may have hardware constraints. Additionally, we do not use multithreading; this is elaborated on in the previous section. In respect to algorithms, we opt only to test Kyber; however, different post-quantum algorithms may have different results (though that is unlikely given that they all utilize the same mathematical framework). Additionally, we tested only one payload with Kyber due to feasibility issues; a more powerful setup may have been able to compute a result for a larger Kyber benchmark that could be used to produce additional conclusions regarding Kyber's performance at scale. We will examine each limitation in full and provide an explanation for each.

We are unfortunately limited by the computational resources at our disposal. We choose a notably older Intel Haswell architecture with an older Xeon processor. Therefore, it may not be completely accurate to extrapolate conclusions on a Haswell CPU to the modern CPUs that the majority runs on today. However, while this is true, it is unlikely that purely architectural differences play a major role in the validity of our conclusions. Rather, our resource limitations prevent us from speculating how performance may scale up. Perhaps a newer processor might have demonstrated a greater ability to scale with larger payloads than our Haswell processors, via features such as a larger cache of improved multithreading or otherwise.

One may notice that we only experimentally conclude infeasibility for a single post-quantum cryptographic algorithm: Kyber. We choose to do so because it is the algorithm specifically denoted by NIST in FIPS 203 - 205 [@ThreeDraftFIPS2023]. However, doing so does create further questions on perhaps the increased scalability or feasibility of other algorithms, perhaps an algorithm which relies less on segmentation. Therefore, it may be advisable for future research to study the performance of other cryptographic algorithms with larger payloads, in addition to exploring the effects of multithreading. Combined with a future study on segmentation in particular, one my find differing conclusions than ours regarding the feasibility of a post-quantum algorithm.