Performance Comparison of NIST Post-Quantum Cryptography Round 4 KEM Submissions

CS 4850 Spring Semester 2024

MR-1 (Green) Cryptography

Team: Charles Foshee, Chris Freel, Amyr Murray

Sharon Perry 2-14-24

Website

Github

Lines of Code: 450

Number of Components: 4
(C, LibOQS, MATLAB, OverLeaf)

Performance Comparison of NIST Post-Quantum Cryptography Round 4 KEM Submissions

Charles Foshee cfoshee2@students.kennesaw.edu
Chris Freel cfreel@students.kennesaw.edu
Advisor: Manohar Raavi mraavi3@kennesaw.edu

April 27, 2024

Abstract

This paper provides a performance analysis of the three algorithms from round 4 of the National Institute of Standards and Technology (NIST) Post-Quantum Cryptography (PQC) Standardization. Runtime and storage requirements are studied for the three algorithms: Hamming Quasi-Cyclic (HQC), Bit Flipping Key Encapsulation (BIKE), and Classic McEliece. Using the LibOQS library for evaluation, metrics such as key size and time for key generation, encryption, and decryption were analyzed across several NIST security levels. Results highlight HQC as a promising candidate for standardization, with strong advantages in most metrics. The algorithms are further compared to CRYSTALS-Kyber, one of the algorithms standardized after round 3 of NIST PQC.

1 Introduction

With the potential rise of quantum computing, the current methods of securing digital data over the internet may become obsolete. Once fully developed, a quantum computer could brute-force the typical prime-factorization-based encryption used for online communication in [number] minutes/seconds (cite). In response to this potential threat, the National Institute of Standards and Technology (NIST) initiated the Post-Quantum Cryptography (PQC) Standardization project to find algorithms that can continue to keep data secure even in a post-quantum world.

This paper focuses on the three submissions to Round 4 of NIST PQC: HQC, BIKE, and Classic McEliece. The runtime performance and storage efficiency of the algorithms are analyzed and compared to each other and to the existing standardized algorithm CRYSTALS-Kyber.

2 Background

Quantum computers utilize the properties of quantum physics to run using quantum bits, or qbits. Rather than the binary 1 or 0 of a traditional bit, a qbit can effectively store all quantum states between 0 and 1, then converge onto a desired state, allowing them to do calculations in parallel. It is this property, along with a method called Shor's algorithm, that could allow a quantum computer to easily decrypt data sent using the current cryptographic schemes [1].

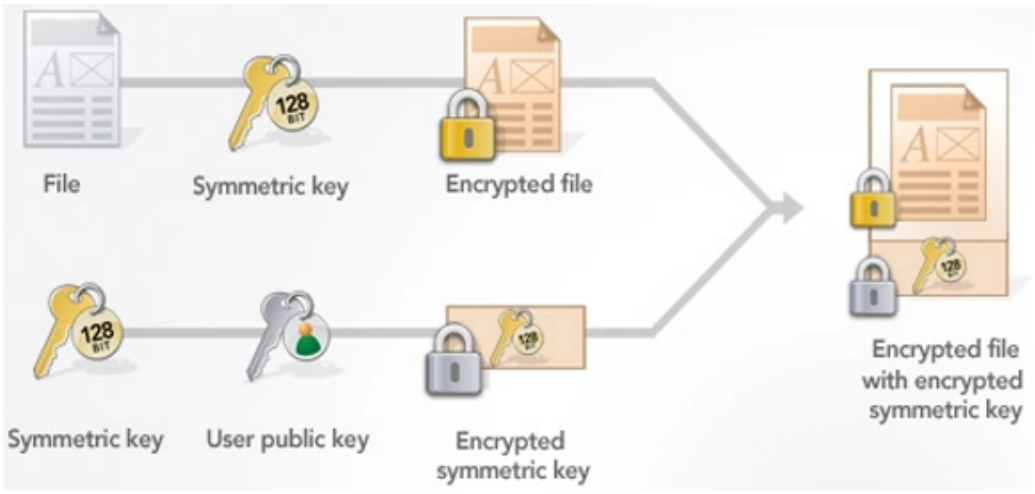

Figure 1: Key Encapsulation Mechanism

While quantum computers powerful enough to break encryption on a relevant timescale are unlikely to appear before the 2030s [1], it would be best to be fully prepared for the eventuality. Additionally, an attack known as "store now decrypt later," in which a hostile actor stores encrypted data now and decrypts it once quantum computers become viable, is still a current concern. Any data that will still be relevant within the next 10-20 years is potentially vulnerable to it [1].

With these concerns in mind, in 2016, the National Institute of Standards and Technology (NIST) put out a call for new public-key cryptographic systems that could be standardized for widespread use. The dozens of submissions have gone through several rounds of detailed analysis, and at the end of round 3, NIST selected several cryptographic schemes for standardization: Kyber, Dilithium, Falcon, and SPHINCS+. However, the competition is still ongoing, and three algorithms are still being evaluated in round 4: Classic McEliece, HQC, and BIKE. These are the algorithms that will be evaluated in this paper.

Classic McEliece, HQC, and BIKE are Key-Encapsulation Mechanism (KEM) algorithms. This means that instead of encoding an entire plaintext, they use public-key cryptography to encode a symmetric key (a key that both the sender and receiever have access to), which is then used to encode a larger plaintext. This is useful because symmetric keys are both more efficient and much easier to secure against quantum attacks, so one only needs a secure way to send that symmetric key over the internet. Thus, a KEM is used, as seen in Figure 1. A file is encrypted with a symmetric key, then the symmetric key is encrypted with the user's public key using a KEM, then both the encrypted file and encrypted symmetric key are sent to the receiver. The receiver can then decrypt the symmetric key, then use it to decrypt the file.

The three round 4 algorithms are all code-based. This means they are based on error-correction codes similar to the Hamming codes used in data communication to correct errors in bits sent from transmitter to receiver. The codes are defined as a subset of values in a space (for example, 111 and 000 out of all possible 3-bit numbers), and then if a number is received that is outside of the space, it is corrected to the closest valid code. The algorithm is able to use this property to add random noise to a plaintext in such a way that the resulting ciphertext is only able to be corrected by the

intended receiver. While all three are code-based, they use different codes. HQC uses Quasi-Cyclic Moderate Density Parity-Check codes, Classic McEliece uses Goppa codes, and BIKE is based on linear codes over a quaternary alphabet.

It is still important to continue to develop algorithms beyond the ones already standardized because all of the standardized algorithms are based on the same underlying theory. They are lattice-based, which means unique combinations of vectors are used in their encoding, the same way prime factorization is used in current traditional encryption and error correction codes are used in the round 4 algorithms. Thus, it is imperative to have standardized algorithms with different underlying principles in case a vulnerability is discovered in lattice-based algorithms in general. Additionally, it is best to have a robust set of encryption algorithms so that users are able to select one that is best suited for their specific application.

3 Methodology

LibOQS will serve as a primary tool for researching and evaluating the PQC algorithms. This library supports the three algorithms (HQC, BIKE, and classic Mceliece). This makes development and testing much simpler and quicker.

The development environment where LibOQS was configured was on a Linux virtual machine. This virtual machine ran Ubuntu 22.04. This environment was chosen for ease of getting any tools needed set up and working. The testing script that uses LibOQS was set up to perform key generation, encryption, decryption, and show key stats as it performs these operations. Because LibOQS supports these algorithms directly, this script can do this quite easily. The script will start by specifying which algorithm will be performing the operations (This is manually specified at the beginning of the script before it is run). After that specification, the script will perform the operations of key generation, encrypting, and decrypting. These operations because of LibOQS are done through API calls. Each call is times through the "clock" method in C to show how much time was taken (expressed in milliseconds). When the script is done all pertinent information is printed to the console. This includes the times for each of the operations performed, and the sizes of the public key, private key, and ciphertext (expressed in bytes).

4 Experimentation

The primary objective was to compare the results of two significant factors between the algorithms. One factor being the average runtime of each operation and the sizes of the public key, private key, and cipher-text size.

Each algorithm contains a different set of parameters pertaining to its NIST security level. In general with these security levels, the higher the security the longer the runtime. So it would depend on the practical application for the appropriate balance of performance and security. However for an in-depth comparison of all parameters for each algorithm. Once all these results were retrieved, the data was examined using MATLAB. To help visually compare the PQC algorithms MATLAB scripts were produced. The first script produces three line plots for comparing the runtime deviation of all three algorithms for key generation, encryption, and decryption. Where the x-axis is the run number and the y-axis is the runtime. On the key generation and decryption graphs the y-axis is on a logarithmic scale so the lines do not become too compressed. This compression is due to the Classic McEliece algorithm usually having a far slower runtime when compared to the same security

level parameters of HQC and BIKE for key generation and decryption. However, on the encryption graph, a linear scale for the y-axis can be used since the runtimes are similar. The second script produces a bar graph where the average time of each operation is calculated and then graphed. This again is on a logarithmic scale due to the runtime of Classic McEliece. In the third script, the averages of each operation are placed on a bar graph. The fourth script makes a direct comparison between the most promising algorithm (HQC) and Krystals-Kyber the current NIST standard with a step plot. The y-axis on this step plot represents the time and the x-axis represents the public key size. Each point on the step plot represents, respectively, the lowest point on the line, the midpoint on the line, and the highest point on the line for each algorithm at level 1, level 3, and level 5.

5 Results and Analysis

The tables below show the average times for all operations across each security level for the algorithms. Note that Classic McEliece has multiple parameter sets for each NIST security level requirement. For runtime comparison to HQC and BIKE the fastest parameter set for levels 1, 3, and 5 were chosen.

NIST Security Level 1						
Algorithm	**Keygen Time (ms)**	**Encryption Time (ms)**	**Decryption Time (ms)**	**Public Key Size (bytes)**	**Private Key Size (bytes)**	**Ciphertext Size (bytes)**
Classic-McEliece-348864f	44.568	0.11	20.846	261120	6492	96
BIKE-L1	0.303	0.093	1.766	1541	5223	1573
HQC-128	0.066	0.145	0.229	2249	2289	4481
Kyber512	0.037	0.042	0.047	800	1632	768

NIST Security Level 3

Algorithm	Keygen Time (ms)	Encryption Time (ms)	Decryption Time (ms)	Public Key Size (bytes)	Private Key Size (bytes)	Ciphertext Size (bytes)
Classic-McEliece-460896f	139.702	0.21	47.281	524160	13608	156
BIKE-L3	0.895	0.242	5.452	3083	10105	3115
HQC-192	0.201	0.429	0.632	4522	4562	9026
Kyber768	0.058	0.065	0.073	1184	2400	1088

NIST Security Level 5						
Algorithm	**Keygen Time (ms)**	**Encryption Time (ms)**	**Decryption Time (ms)**	**Public Key Size (bytes)**	**Private Key Size (bytes)**	**Ciphertext Size (bytes)**
Classic-McEliece-6688128f	236.299	0.305	89.291	1044992	13932	208
BIKE-L5	2.032	0.598	13.421	5122	16494	5154
HQC-256	0.323	0.713	1.073	7245	7285	14469
Kyber1024	0.098	0.105	0.117	1568	3168	1568

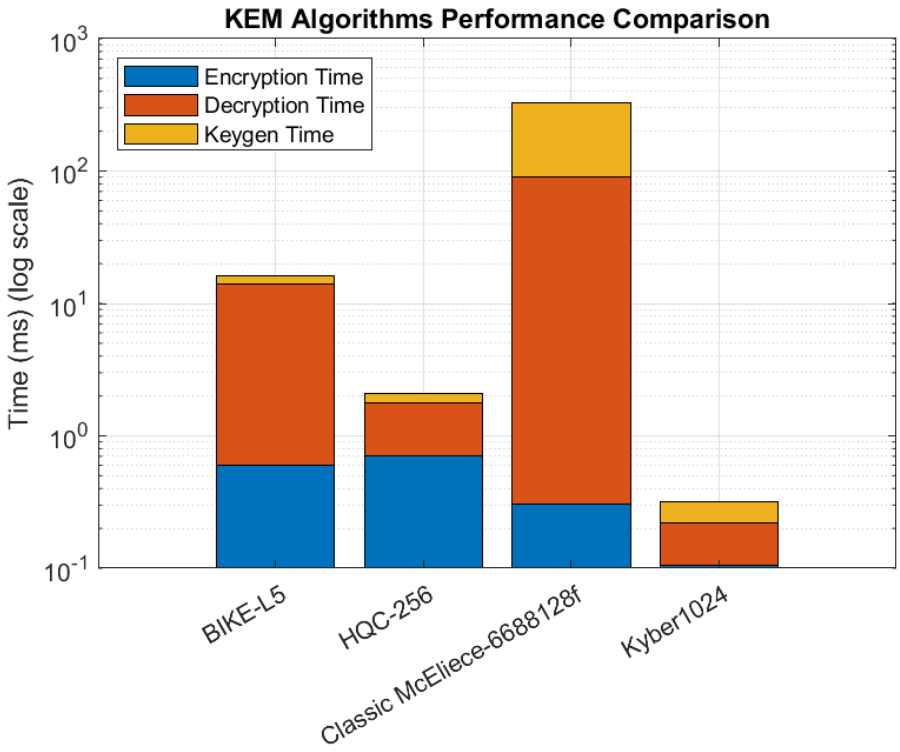

Figure 2: Average Performances

5.1 Runtime analysis

On average these times show for key generation and decryption, HQC has the fastest runtime for these operations. For the decryption operation Classic McEliece does it the quickest on average. However, examining the data we can see that HQC for both operations of key generation and decryption is nearly 100% faster across all security levels. So while Classic McEliece does perform encryption approximately 50% faster than HQC, its times to perform key generation and encryption are far slower than HQC's times. Another factor to consider is that key generation is not a frequent operation like encryption and decryption (encryption occurring every time a message is sent and decryption every time a message is received). Therefore, a greater priority can be placed on better encryption and decryption times. However, because Classic McEliece is so much slower at decryption and only slightly faster at encryption, therefore it may not be a viable option in this regard.

5.2 Runtime deviation analysis

Another key factor to analyze is runtime deviation. To see how much of a time difference there is on each run with all algorithms, the 1000 runs used to calculate the averages were all graphed and can be seen in Figures 2, 3, and 4. Figure 2 shows the deviation in key generation and keep in mind that this graph is on a logarithmic scale. This figure shows that for key generation HQC shows the

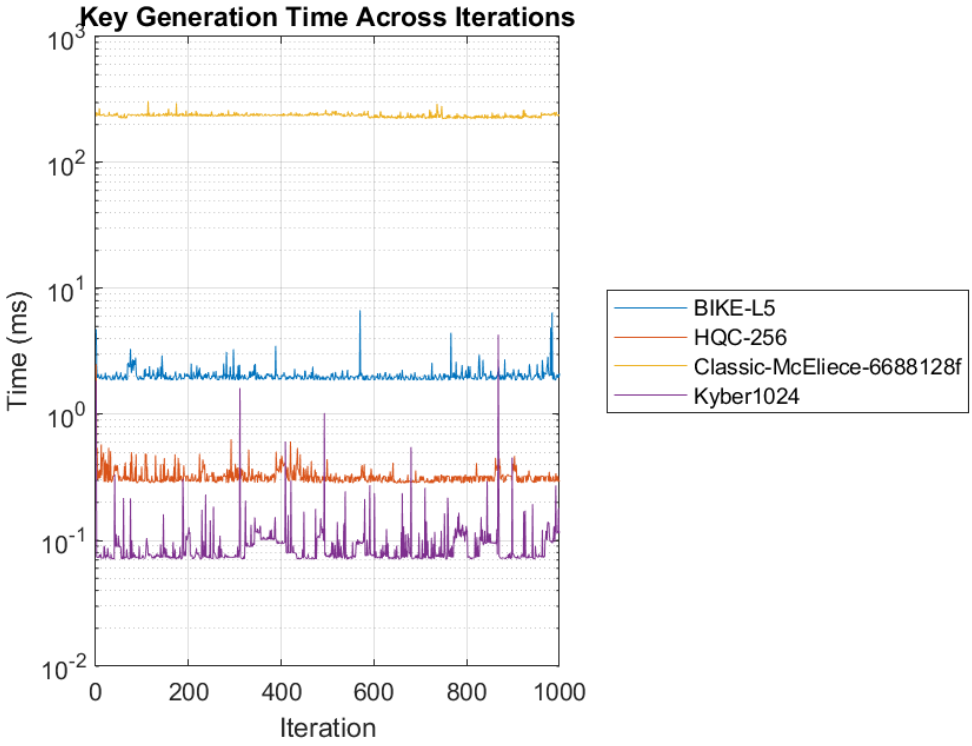

Figure 3: Keygen Time Across Iterations

fastest time and lowest deviation (usually within 0.5 milliseconds between runs). The slowest key generation and highest deviation is Classic McEliece (usually deviates 50 – 100 milliseconds between runs). Figure 3 shows the deviation in encryption and is on a linear scale. Here all algorithms have a similar performance. Figure 4 shows the deviation in decryption time and is on a logarithmic scale. For decryption, HQC shows the fastest time and lowest deviation (usually within 6 milliseconds between runs). The slowest decryption and highest deviation is Classic McEliece (usually deviates 50 – 100 milliseconds between runs).

5.3 Storage requirements analysis

The tables also show the storage requirements in bytes of the public key, private key, and cipher text. Across all three security levels, BIKE has the smallest public key size, HQC has the smallest private key size and Classic has the smallest cipher text size. However, Classic McEliece has a very large public key size compared to BIKE and HQC. This larger storage requirement may not be suitable for most standard applications. HQC and BIKE have very similar storage requirements overall, but BIKE is slightly smaller.

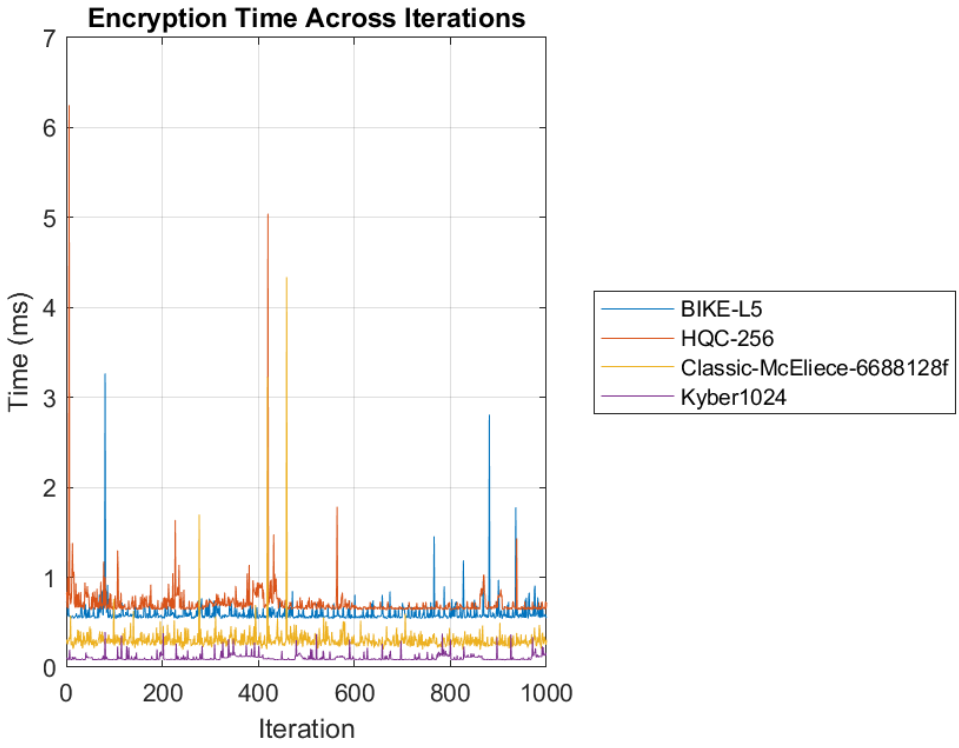

Figure 4: Encryption Time Across Iterations

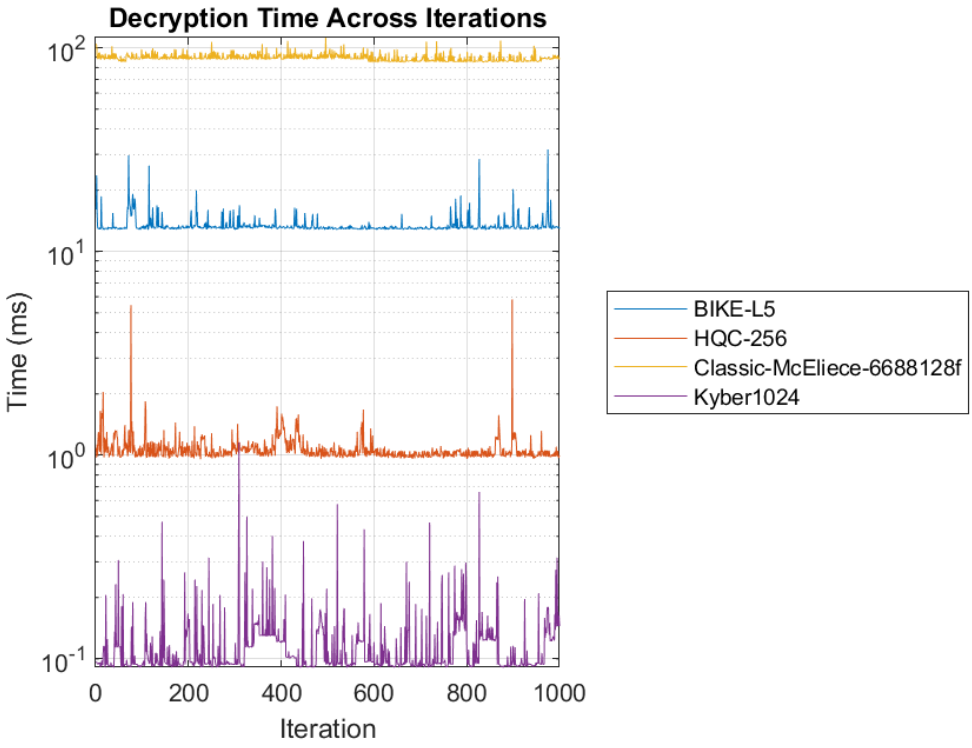

Figure 5: Decryption Time Across Iterations

5.4 Analysis summary

To summarize the analysis a few key points are derived. The Classic McEliece algorithm may not be suitable for most applications, due to its large public key size and long operation runtimes. BIKE and HQC have better runtimes and sizes making either a good choice for standardization. However, HQC appears to be the better choice over BIKE and should be the proposed algorithm for standardization. While BIKE may have slightly better sizes, HQC provides better runtimes for key generation and encryption, especially at higher security levels. At level 5 HQC is 84.24% faster at key generation and 92.03% faster at decryption and only 18.33% slower at decryption. This faster performance that HQC provides is most likely more important for most applications than saving 1 or 2 kilobytes for storage that BIKE offers. Therefore, HQC should be the algorithm proposed for standardization, but how does HQC compare with the current standardized PQC KEM algorithm? Crystals-Kyber is the current KEM algorithm standardized by NIST. It is currently in use for Apple's iMessage encryption. Referring to the tables and figures above, it can be seen that all operations for HQC and Kyber on average are completed within or near 1 millisecond even at a security level 5. However, Kyber does offer a smaller storage requirement. Utilizing Kyber, an application may save several kilobytes at higher security levels compared to HQC, but the performance will stay approximately the same in terms of runtime. Overall, HQC is a code-based algorithm that gives another KEM algorithm with similar statistics to the current NIST KEM PQC standard. If standardized this algorithm will give another option for a quantum-safe algorithm giving more variety in what applications can be used to be quantum-secure. This is important so that should a vulnerability or attack work on one algorithm it is unlikely that it will work on the other since they are based on concepts.

6 Conclusion

This paper presents a performance comparison of the NIST Post-Quantum Cryptography Round 4 submissions HQC, BIKE, and Classic McEliece. Insights into runtime performance of key generation, encryption, and decryption were gleaned, as well as into storage requirements.

HQC has standout performance among the three algorithms, with comparable storage requirements to BIKE and requiring much less than Classic McEliece. HQC has significant runtime advantages in key generation and decryption, with similar times to the other two algorithms in encryption. Classic McEliece generally lags behind, requiring a significantly larger public key size and having much slower decryption times. It is, however, the fastest at encryption by a small margin. CRYSTALS-Kyber is ahead of the three round 4 algorithms in most metrics, which is to be expected from one that was already standardized, but it is important that research and development continue on other algorithms in the case of vulnerabilities being discovered in lattice-based algorithms.

Thus, HQC is likely to be the best for general applications, with BIKE as a close second. We cannot recommend Classic McEliece for practical use except in cases where encryption time is paramount, to the exclusion of all else.

7 References

[1] “Algorithms,” Open Quantum Safe. https://openquantumsafe.org/liboqs/algorithms (accessed Feb. 27, 2024)
[2] NIST, “Post-Quantum Cryptography — CSRC,” CSRC — NIST, Jan. 03, 2017. https://csrc.nist.gov/projects/post-quantum-cryptography (accessed Mar. 30, 2024).
[3] E. Parker, “When a Quantum Computer Is Able to Break Our Encryption, It Won’t Be a Secret,” Lawfare, Sep. 13, 2023. https://www.lawfaremedia.org/article/when-a-quantum-computer-is-able-to-break-our-encryption-it-won-t-be-a-secret (accessed Apr. 08, 2024).