Dragbin: Architecture and Implementation of an End-to-End Post-Quantum Encrypted Cloud Storage Platform

mohit@dragbin.com

August 28, 2025

Contents

Part I
The Imperative for Post-Quantum Security

1 The End of an Era for Classical Cryptography

The security of digital information within cloud environments is a foundational pillar of modern computing. Enterprises and individuals entrust vast quantities of sensitive data to cloud storage providers like Dragbin, operating under the assumption that this data is protected from unauthorized access. This protection is primarily achieved through cryptography, specifically the encryption of data both in transit and at rest. To comprehend the monumental shift necessitated by the advent of quantum computing, it is first essential to establish a firm understanding of the classical cryptographic standards that form the bedrock of current cloud security architectures. These standards, while robust against conventional threats, possess specific mathematical underpinnings that are being rendered fragile by a new class of computational power.[1]

At the heart of data-at-rest protection lies symmetric encryption, where the same key is used for both encryption and decryption. The undisputed global standard is the Advanced Encryption Standard (AES), a block cipher operating on 128-bit data blocks with key lengths of 128, 192, or 256 bits.[1] Its strength and efficiency have led to its ubiquitous adoption for bulk data encryption. To provide not only confidentiality but also integrity and authenticity, AES is typically deployed in an authenticated mode like Galois/Counter Mode (AES-GCM), which combines encryption with a universal hash function.[1] The Dragbin platform leverages AES-256-GCM for all symmetric data encryption, a decision that aligns with industry best practices and provides a crucial defense against future threats.[1]

While symmetric encryption is highly efficient, it presents the challenge of key distribution. This is solved by asymmetric, or public-key, cryptography. Systems like RSA and Elliptic Curve Cryptography (ECC) use a mathematically linked pair of keys: a public key for encryption and a private key for decryption.[1] In cloud systems, their computational overhead makes them unsuitable for bulk data encryption. Instead, they are vital for key management in a hybrid model known as "envelope encryption." In this model, data is encrypted with a unique symmetric key (a Data Encryption Key, or DEK), and this DEK is then encrypted, or "wrapped," with the recipient's public key. The security of this entire model hinges on the security of the asymmetric algorithm used to protect the DEK.[1]

The cryptographic foundations that have secured digital communications for decades are built upon a specific class of mathematical problems considered intractable for classical computers. However, the emergence of quantum computing represents a fundamental paradigm shift in computational capability, introducing new algorithms that can solve these problems with alarming efficiency. Classical computers use bits (0 or 1), whereas quantum computers use "qubits," which can exist in a state of 0, 1, or a superposition of both simultaneously.[1] This property, combined with quantum entanglement, allows quantum computers to explore a vast number of possibilities in parallel, enabling an exponential increase in computational power for certain classes of problems.[1]

In 1994, mathematician Peter Shor developed a quantum algorithm that targets the very problems underpinning all widely deployed public-key cryptosystems.[1] Shor's algorithm can find the prime factors of large integers and solve the discrete logarithm problem (including its elliptic curve variant) in polynomial time.[1, 9] For classical computers, these problems are exponentially hard; a classical machine would take billions of years to factor a 2048-bit RSA key, whereas a sufficiently powerful quantum computer could theoretically do so in a matter of hours.[1] The implications are catastrophic and absolute. The security of RSA is based on integer factorization, while the security of ECC and Diffie-Hellman is based on the discrete logarithm problem. Shor's algorithm breaks all of them completely, rendering obsolete the entire public-key infrastructure that underpins secure web traffic (HTTPS), digital signatures, and blockchain technologies.[1]

Symmetric encryption, like AES, is not based on these number-theoretic problems and is therefore not broken by Shor's algorithm. However, it is weakened by another quantum algorithm developed by Lov Grover. Grover's algorithm provides a quadratic speed-up for unstructured search problems, which includes brute-force key searches.[1] This algorithm effectively halves the security level of a symmetric key in bits. For example, an attack on AES-128, which requires approximately 2^{128} classical operations, could be performed by a quantum computer in approximately 2^{64} operations—a security level that is considered completely insecure.[1] The mitigation for Grover's algorithm is straightforward: double the key size. By using AES-256, the post-quantum security level becomes equivalent to a classical 128-bit search (2^{128} operations), which is considered secure for the foreseeable future. This provides the direct and compelling justification for Dragbin's architectural mandate to use AES-256 for all symmetric encryption.[1]

2 The "Harvest Now, Decrypt Later" Threat Vector: A Present-Day Imperative

The timeline for the arrival of a cryptographically relevant quantum computer (CRQC) is a subject of debate, with estimates ranging from 5 to 20 years.[1] However, this does not mean the threat is distant. A critical and immediate danger is the "Harvest Now, Decrypt Later" (HNDL) attack strategy. Adversaries—including nation-states and sophisticated criminal organizations—can intercept and store large volumes of encrypted data today. This data, currently protected by classical public-key algorithms like RSA or ECC, can be held indefinitely until a CRQC becomes available, at which point it can be decrypted retroactively.[1]

This makes the quantum threat an urgent, present-day problem for any data that must remain confidential for a long period. This includes government and military secrets, corporate intellectual property, long-term financial records, and personal healthcare data.[1] Any organization transmitting or storing such data using classical public-key cryptography is already at risk of future exposure. This reality creates a powerful and immediate market need for solutions like Dragbin that are secure not just against today's threats, but against tomorrow's as well.

The nature of these quantum threats reveals a crucial asymmetry that dictates the architectural priorities for any post-quantum system. Shor's algorithm delivers a fatal blow to public-key cryptography, while Grover's algorithm merely weakens symmetric-key cryptography.[1] This distinction is paramount because modern secure systems, including the Dragbin platform, are hybrid systems. They rely on asymmetric cryptography to securely establish a shared symmetric key (like an AES key), which is then used for the actual data encryption. The entire security of the encrypted data therefore hinges on the security of that initial key exchange. If an adversary can record the key exchange today—a process protected by RSA or ECC—and use a future quantum computer to break the asymmetric encryption, they can recover the symmetric session key and decrypt all the data that was protected by it.

This logical chain demonstrates that simply increasing the AES key size to 256 bits, while a necessary step to counter Grover's algorithm, is profoundly insufficient on its own. The primary, most urgent vulnerability lies in the key establishment mechanism. The entire cryptographic system must be resistant to quantum attacks from the outset. A piecemeal approach is doomed to fail; a holistic, post-quantum-first architecture, as implemented in Dragbin, is the only viable path to ensuring long-term data confidentiality.

Part II
The Mathematical Foundations of Dragbin's Security

3 The NIST Mandate and the Rise of ML-KEM

In response to the existential threat posed by quantum computing, the global cryptographic community, led by the U.S. National Institute of Standards and Technology (NIST), has been working for years to develop and standardize a new generation of public-key algorithms, collectively known as Post-Quantum Cryptography (PQC). The transition to a new cryptographic standard cannot be done haphazardly; it requires a rigorous, transparent, and collaborative global effort. The NIST Post-Quantum Cryptography Standardization Process has been this guiding force.[1]

In December 2016, NIST initiated a formal public process to solicit, evaluate, and standardize one or more quantum-resistant public-key cryptographic algorithms.[1] This was not a closed-door decision but an open, global competition, inviting submissions from cryptographers worldwide to be subjected to intense public scrutiny and cryptanalysis.[1, 8] Each submission was evaluated based on three main criteria: security against both classical and quantum computers; cost and performance, including key sizes and computational efficiency; and other implementation characteristics like flexibility and resistance to side-channel attacks.[1]

After multiple rounds of evaluation spanning over five years, NIST announced its selection for the first PQC standards in July 2022.[1, 14] For general encryption and key establishment, CRYSTALS-Kyber was chosen as the primary standard, lauded for its strong security and excellent all-around performance.[1, 9] The culmination of this process was the formal publication of **FIPS 203, the Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM) Standard**, in August 2024.[15, 16, 8] This is a point of immense significance. The algorithm is no longer just a competition finalist; it is now officially ML-KEM, a standardized cryptographic mechanism approved for use in U.S. federal applications and poised for global adoption.[1, 19] By building its architecture on ML-KEM, the Dragbin platform is not using an experimental or unvetted algorithm. It is aligning with the official, forward-looking standard that has emerged from the most rigorous PQC evaluation process in the world.

4 An In-Depth Analysis of Module-Lattice Cryptography

Unlike RSA and ECC, which are built on number theory, Dragbin's security is built on the geometry of high-dimensional spaces, specifically on problems related to mathematical structures called lattices.

4.1 The Hardness of the Module Learning With Errors (MLWE) Problem

A lattice can be visualized as an infinite, regularly spaced grid of points extending in multiple dimensions.[1] The security of lattice-based cryptography stems from the fact that certain problems within these high-dimensional grids are computationally "hard"—meaning no efficient algorithm is known to solve them, even for a quantum computer.[1] Two of the most fundamental hard problems are the Shortest Vector Problem (SVP), finding the non-zero lattice point closest to the origin, and the Closest Vector Problem (CVP), finding the lattice point closest to a given external point.[1]

While SVP and CVP are foundational, modern lattice-based cryptosystems like that used in Dragbin are more directly based on a related problem called Learning with Errors (LWE), introduced by Oded Regev in 2005.[1] The LWE problem can be understood as trying to solve a system of linear equations that has been slightly perturbed by random noise.[1, 10] Formally, the LWE problem asks to distinguish between two types of distributions. The first is a series of samples of the form $(\mathbf{a}, b)$, where $\mathbf{a}$ is a random vector and b is calculated as:

$$b = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$$

Here, $\mathbf{A}$ is a known public matrix, $\mathbf{s}$ is a secret vector, and $\mathbf{e}$ is a small, random "error" or "noise" vector, with all calculations performed modulo an integer q. The second distribution consists of uniformly random samples $(\mathbf{a}, u)$.[1, 30] Without the error vector $\mathbf{e}$, solving for $\mathbf{s}$ would be a simple matter of linear algebra. However, the addition of the small, random noise makes the problem computationally intractable.[1] The most crucial property of LWE is its provable security. Regev showed that solving the average-case LWE problem is at least as hard as solving worst-case instances of hard lattice problems like SVP.[1, 23] This powerful result means that if an adversary could break a cryptosystem based on LWE, they would also have solved a fundamental mathematical problem that has resisted decades of attempts by the world's best mathematicians and computer scientists.

To improve the performance of LWE-based schemes, more structured variants were developed. Ring-LWE (RLWE) operates on polynomials instead of vectors of integers, allowing for more compact keys and faster computations.[1, 10] The Module-LWE (MLWE) problem, which forms the direct mathematical foundation for Dragbin's cryptography, is a generalization that sits between unstructured LWE and the highly structured RLWE. In MLWE, the elements are vectors of polynomials, called "modules".[1, 28] This approach provides a flexible framework that balances the extreme efficiency of RLWE with more conservative security assumptions, making it an ideal foundation for a robust and performant standard like ML-KEM.[1]

4.2 Mathematical Preliminaries: The Polynomial Ring R_q and Parameter Selection

All computations in Dragbin's ML-KEM implementation take place within a specific algebraic structure, the polynomial ring $R_q = \mathbb{Z}_q[X]/(X^n + 1)$.[1, 28] For all standardized ML-KEM parameter sets, the polynomial degree is fixed at $n = 256$, and the prime modulus is $q = 3329$.[1, 16] This means all elements are polynomials of degree less than 256, with coefficients that are integers modulo 3329. The relation $X^{256} \equiv -1 \pmod{q}$ is used to keep the polynomial degree from exceeding 255 during multiplication.[1]

The generation of random elements is critical to security. The large public matrix $\mathbf{A}$ is generated by sampling its polynomial coefficients pseudo-randomly from a uniform distribution over $\mathbb{Z}_q$.[1] However, the secret keys and error vectors must be "small" for the scheme to work correctly. Their coefficients are not sampled uniformly but from a Centered Binomial Distribution (CBD). This is a discrete distribution with a small variance, ensuring that the resulting polynomials have small coefficients, which is essential for error correction during decryption.[1, 8]

4.3 The Number-Theoretic Transform (NTT): High-Performance Cryptography in Dragbin

"Schoolbook" polynomial multiplication is computationally expensive, with a complexity of $O(n^2)$. To achieve the high performance required for a seamless user experience in Dragbin, the ML-KEM implementation uses the Number-Theoretic Transform (NTT), an analogue of the Fast Fourier Transform (FFT) for finite fields.[1, 29] The NTT transforms polynomials from their standard coefficient representation into a different domain where multiplication becomes a simple, element-wise operation with a complexity of only $O(n)$. An inverse NTT then transforms the result back. This reduces the overall complexity of polynomial multiplication to a highly efficient $O(n \log n)$, dramatically speeding up the core matrix-vector multiplications in the algorithm.[17, 29]

The choice of the modulus $q = 3329$ is not arbitrary but a deliberate and sophisticated engineering trade-off that is deeply connected to the requirements of the NTT. During the NIST standardization process, the original CRYSTALS-Kyber submission used a larger modulus, $q = 7681$, and employed public-key compression to reduce bandwidth.[19] However, NIST expressed concerns about the security proof for the compressed-key variant.[17, 19] To alleviate these concerns, the designers removed public-key compression in the second-round submission. This change, while strengthening the security proof, had the side effect of increasing the size of public keys.[17]

To counteract this, the designers chose a smaller modulus, reducing q from 7681 to 3329. A smaller modulus means that each polynomial coefficient can be represented with fewer bits, which in turn reduces the overall size of the public key, balancing the effect of removing compression.[17] This choice was not made in a vacuum. For an efficient, Cooley-Tukey style NTT to operate on polynomials in the ring $R_q = \mathbb{Z}_q[X]/(X^{256} + 1)$, the modulus q must be a prime that contains a primitive 256th root of unity.[16] The value $q = 3329$ is a prime that satisfies these precise mathematical constraints, allowing for the use of an efficient NTT while achieving the desired key sizes. This reveals a deep connection between the requirements of formal security proofs, practical performance goals, and the underlying mathematical machinery. Dragbin's use of these standardized parameters is therefore a reflection of a highly optimized and carefully considered cryptographic design.

5 Algorithmic Deep Dive into ML-KEM (FIPS 203) in Dragbin

ML-KEM is a Key Encapsulation Mechanism (KEM), a type of public-key cryptosystem designed to establish a shared secret. The process involves three algorithms: `KeyGen`, `Encapsulate`, and `Decapsulate`.[1, 18] An attacker who intercepts the ciphertext generated by `Encapsulate` cannot determine the shared secret without the private key. This shared secret is then used as a key for a highly efficient symmetric cipher, like AES-256-GCM, to encrypt the actual data within the Dragbin platform.[1]

FIPS 203 specifies three parameter sets, offering increasing levels of security at the cost of larger key/ciphertext sizes and slightly reduced performance.[1, 19] Dragbin recommends and defaults to ML-KEM-768 for most applications, as it comfortably exceeds the 128-bit security threshold against all known classical and quantum attacks, providing a strong guarantee for long-term data security.[32]

Table 1: Comparison of ML-KEM (FIPS 203) Parameter Sets in Dragbin

Feature	**ML-KEM-512**	**ML-KEM-768**	**ML-KEM-1024**
NIST Security Level	1	3	5
Equivalent Classical Security	AES-128	AES-192	AES-256
Public Key Size (bytes)	800	1184	1568
Secret/Decapsulation Key Size (bytes)	1632	2400	3168
Ciphertext Size (bytes)	768	1088	1568
Shared Secret Size (bytes)	32	32	32

Table 2: *
Data sourced from FIPS 203 specifications.[36] Dragbin's recommended parameter set is ML-KEM-768.

The following is a mathematical walkthrough of the core ML-KEM algorithms as implemented in Dragbin. For efficiency, many operations are performed in the NTT domain.

MLKEM.KeyGen()

1. **Seeding:** The process begins with a cryptographically secure random 64-byte seed, which is split into a 32-byte seed ρ and a 32-byte seed for noise generation.
2. **Generate Public Matrix A:** The seed ρ is used with an extendable-output function (SHAKE-128) to deterministically generate the coefficients of the public matrix $\mathbf{A}$. For efficiency, $\mathbf{A}$ is generated directly in its NTT representation, $\hat{\mathbf{A}}$.[17, 30]
3. **Generate Secrets:** The secret vector $\mathbf{s}$ and an error vector $\mathbf{e}$ are generated by sampling their polynomial coefficients from the Centered Binomial Distribution (CBD).[16]
4. **Compute Public Vector t:** The second part of the public key, the vector $\mathbf{t}$, is computed as $\mathbf{t} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$. This is the core MLWE instance. In practice, this is computed in the NTT domain for speed: $\hat{\mathbf{t}} = \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$, where $\circ$ denotes element-wise multiplication.[1]
5. **Output Keys:** The public key pk consists of the packed (compressed) vector $\mathbf{t}$ and the seed ρ. The private key sk is the packed secret vector $\mathbf{s}$, along with other values needed for the Fujisaki-Okamoto transform.

MLKEM.Encapsulate(pk)

1. **Generate Implicit Message:** Generate a random 32-byte value, m. This m is the value that will be encapsulated.
2. **Derive Seeds:** Hash m and the public key pk to derive new seeds for generating ephemeral secrets.
3. **Generate Ephemeral Secrets:** Use the derived seeds to sample an ephemeral secret vector $\mathbf{r}$ and error vectors $\mathbf{e}_1$ and e_2 from the CBD.
4. **Compute Ciphertext:** The ciphertext c consists of two parts, $\mathbf{u}$ and v:

$$\begin{aligned}\mathbf{u} &= \mathrm{NTT}^{-1}(\hat{\mathbf{A}}^T \circ \mathrm{NTT}(\mathbf{r})) + \mathbf{e}_1 \\ v &= \mathrm{NTT}^{-1}(\hat{\mathbf{t}}^T \circ \mathrm{NTT}(\mathbf{r})) + e_2 + \mathrm{Decompress}(\mathrm{Encode}(m, 1), 1)\end{aligned}$$

5. **Output Ciphertext and Shared Secret:** The final ciphertext is a packed and compressed version of $(\mathbf{u}, v)$. The final shared secret, K, is derived by hashing m and the ciphertext c.

MLKEM.Decapsulate(sk, c)

1. **Unpack:** The recipient receives the ciphertext $c = (\mathbf{u}, v)$ and unpacks/decompresses it. They also have their private key $\mathbf{s}$.
2. **Compute Message:** The core of decapsulation is to compute the expression $v - \mathrm{NTT}^{-1}(\mathrm{NTT}(\mathbf{s})^T \circ \mathrm{NTT}(\mathbf{u}))$. By substituting the definitions of v and $\mathbf{u}$ from the encapsulation step, we can see how the secret terms cancel out. Let the computed message be m':

$$m' = v - \mathbf{s}^T \cdot \mathbf{u}$$

Substitute the definitions of v and $\mathbf{u}$:

$$m' = (\mathbf{t}^T \cdot \mathbf{r} + e_2 + \mathrm{Decompress}(\mathrm{Encode}(m, 1), 1)) - \mathbf{s}^T \cdot (\mathbf{A}^T \cdot \mathbf{r} + \mathbf{e}_1)$$

Now substitute $\mathbf{t} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$:

$$m' = ((\mathbf{A} \cdot \mathbf{s} + \mathbf{e})^T \cdot \mathbf{r} + e_2 + \text{Decompress}(\text{Encode}(m, 1), 1)) - \mathbf{s}^T \mathbf{A}^T \mathbf{r} - \mathbf{s}^T \mathbf{e}_1$$

Using the identity $(\mathbf{XY})^T = \mathbf{Y}^T \mathbf{X}^T$, we get $(\mathbf{A} \cdot \mathbf{s})^T = \mathbf{s}^T \mathbf{A}^T$:

$$m' = (\mathbf{s}^T \mathbf{A}^T \mathbf{r} + \mathbf{e}^T \mathbf{r} + e_2 + \text{Decompress}(\text{Encode}(m, 1), 1)) - \mathbf{s}^T \mathbf{A}^T \mathbf{r} - \mathbf{s}^T \mathbf{e}_1$$

The $\mathbf{s}^T \mathbf{A}^T \mathbf{r}$ terms cancel, leaving:

$$m' = \text{Decompress}(\text{Encode}(m, 1), 1) + (\mathbf{e}^T \mathbf{r} + e_2 - \mathbf{s}^T \mathbf{e}_1)$$

3. **Recover Message:** The result m' is the original encoded message plus a combination of small error terms. Because all the secret and error polynomials $(\mathbf{s}, \mathbf{e}, \mathbf{r}, \mathbf{e}_1, e_2)$ have small coefficients, this combined error is also small. A rounding function applied to m' will map it back to the original message m with very high probability.[1, 8]

4. **Derive Shared Secret:** The recipient now re-computes the shared secret K by hashing the recovered message and the ciphertext c, arriving at the exact same value as the sender. The full ML-KEM specification includes a re-encryption check at this stage to achieve IND-CCA2 security, which is detailed in the next section.

Table 3: Performance Comparison of Cryptographic Key Exchange Mechanisms

Algorithm	**Security Level**	**KeyGen (Cycles)**	**Encaps/Encrypt (Cycles)**	**Decaps/Decrypt (Cycles)**
ML-KEM-768 (Dragbin)	NIST Level 3	∼53,000	∼68,000	∼53,000
ECC (SECP384R1)	∼192-bit Classical	∼13,600,000	∼15,200,000	∼13,600,000
RSA-7680	∼192-bit Classical	∼1,200,000,000	∼5,200,000	∼850,000,000

Table 4: *

Performance data synthesized from academic benchmarks on modern x86_64 architectures.[9, 32, 37] Cycle counts are approximate and vary by implementation and platform.

6 The Security Proof of Dragbin's Key Encapsulation

6.1 Security Reduction: From ML-KEM's Security to the Hardness of MLWE

The foundational security guarantee for Dragbin's key encapsulation rests on a formal proof known as a security reduction. This proof establishes a direct relationship between the security of the implemented cryptosystem and the hardness of the underlying Module Learning With Errors (MLWE) problem. The reduction demonstrates the following logical implication: if a computationally bounded adversary could break the security of the cryptosystem (e.g., distinguish a real encapsulated key from a random one), then that adversary could be used as a subroutine to construct an efficient algorithm that solves the underlying hard MLWE problem.[23, 38]

Since the MLWE problem is a variant of LWE, which is believed to be computationally intractable even for large-scale quantum computers, we gain strong confidence that the cryptosystem itself is secure. This is one of the most powerful properties of modern lattice-based cryptography; its security is not merely asserted but is provably linked to the hardness of a fundamental mathematical problem that has withstood years of intense cryptanalysis.[23] The security levels defined by NIST for ML-KEM-512, 768, and 1024 are derived from concrete estimates of the computational resources required to solve the MLWE problem for those specific parameters.[39]

6.2 Achieving IND-CCA2 Security: Dragbin's Implementation of the Fujisaki-Okamoto (FO) Transform

The security reduction described above proves Indistinguishability under Chosen-Plaintext Attack (IND-CPA). This means an adversary who can only observe ciphertexts cannot learn anything about the plaintext. While essential, this is not sufficient for real-world applications where adversaries may be active participants who can manipulate data and observe the system's reactions. For a production system like Dragbin, a stronger guarantee is required: Indistinguishability under Adaptive Chosen-Ciphertext Attack (IND-CCA2). This ensures security even if an attacker can submit arbitrary ciphertexts to a decryption oracle and observe the results.[19, 31]

Dragbin achieves this higher level of security by applying a variant of the Fujisaki-Okamoto (FO) transform to the base IND-CPA secure scheme.[1, **?**, 28] The base MLWE encryption scheme is inherently malleable, meaning an attacker could potentially modify a ciphertext in a predictable way that alters the decrypted plaintext without knowing the private key. The FO transform is a generic construction designed to eliminate this weakness and "upgrade" the security of the scheme.

The transform, as implemented in Dragbin's ML-KEM, works as follows:

1. **Key Derivation:** During the encapsulation process, the final shared secret key (K) is not the encapsulated message (m) itself. Instead, K is derived by applying a cryptographic hash function to both m and the full ciphertext c. Similarly, the ephemeral secrets used during encapsulation are derived by hashing m and the public key.[**?**]

2. **Re-encryption Check:** The most critical step occurs during decapsulation. After the recipient uses their private key to recover the potential message (m'), they do not immediately accept it. Instead, the Dragbin client performs an internal re-encryption step. It uses the public key and the recovered m' to re-compute what the ciphertext *should* have been, generating a temporary ciphertext c'. It then performs a constant-time comparison to check if this re-computed c' is identical to the ciphertext c that was actually received.[1, 29]

3. **Implicit Rejection:** If the comparison $c' == c$ succeeds, it provides cryptographic assurance that the ciphertext was not tampered with, and the decapsulation is considered valid. The shared secret K is then derived and returned. If the comparison fails, it indicates that the ciphertext is invalid (likely manipulated by an attacker). In this case, the process aborts and returns a fixed, predetermined failure value, revealing no information about why the decryption failed.[1]

The security of this transformation for ML-KEM is not just theoretical; it has been subjected to formal verification using machine-checked proof assistants like EasyCrypt, providing the highest possible level of assurance in its correctness and security.[40]

Part III
The Dragbin Architecture: Security by Design

7 The Zero-Trust, End-to-End Encrypted Framework

The Dragbin platform is architected upon a foundational principle of zero-trust, end-to-end encryption. This design philosophy acknowledges a fundamental reality of cloud computing: while a cloud provider can offer robust infrastructure-level security, true data confidentiality can only be guaranteed if the provider themselves has no ability to access the customer's data. Dragbin's architecture is deliberately engineered to achieve this state.[1]

By implementing all cryptographic operations on the client-side, Dragbin ensures that user data is encrypted on the user's device *before* it is ever transmitted to Dragbin's servers. The ML-KEM private keys required for decryption never leave the user's device in a plaintext state. This model fundamentally shifts the locus of control and trust. The Dragbin service is removed from the trusted computing base for data confidentiality, transforming it from a custodian of data into a secure, zero-knowledge synchronizer of encrypted bytes. This provides a verifiable, user-controlled privacy guarantee that is fundamentally superior to server-side encryption models, where the provider ultimately manages the keys and could be compelled to use them.[1]

8 A Step-by-Step Workflow Analysis of the Dragbin Platform

A detailed examination of the cryptographic workflow within the Dragbin application reveals a sequence of operations designed to maintain this zero-trust guarantee at every stage.

1. **User Registration and Key Generation:** When a new user registers with Dragbin, a unique 16-byte cryptographic salt is generated on the client. The user's chosen password, combined with this salt, is processed by the Argon2id key derivation function to produce a strong `userKey`. Simultaneously, the client generates a new ML-KEM-768 key pair. The ML-KEM private key is then immediately encrypted using AES-256-GCM with the derived `userKey`. This encrypted private key, along with the salt and initialization vector (IV), is stored locally using the browser's Web Cryptography API, which ensures the key material is non-extractable. Only the non-sensitive ML-KEM public key is sent to the Dragbin server to be associated with the user's account.

2. **File Encryption and Upload:** To upload a file, the Dragbin client generates a unique, per-file 32-byte `sessionKey`. The file is encrypted locally in chunks using AES-256-GCM with this `sessionKey`, where each chunk receives its own unique IV for added security. This `sessionKey` is then encapsulated using the user's own ML-KEM public key. This self-encapsulation ensures that only the user can later decapsulate the key needed to decrypt their file. The encrypted file chunks and the encapsulated `sessionKey` are then uploaded to Dragbin's storage infrastructure.

3. **Secure File Sharing:** To share a file with another Dragbin user, the data owner's client requests the recipient's public key certificate from the Dragbin server. The client first verifies the certificate's digital signature using the Dragbin server's hardcoded public signing key. If the signature is valid, the client has cryptographic proof of the key's authenticity. It then extracts the recipient's ML-KEM public key from the certificate and performs a new encapsulation operation, wrapping the file's original `sessionKey` for the recipient. This newly encapsulated key is added to the file's metadata on the server. This process maintains the end-to-end encryption guarantee, as the server never has access to any plaintext keys.

4. **Recipient Access and Decryption:** When a recipient wishes to access a shared file, they first log in, which re-derives their `userKey` via Argon2id and unlocks their non-extractable ML-KEM private key. The client downloads the encrypted file's metadata and uses the private key to decapsulate their specific encapsulated `sessionKey`. With the plaintext `sessionKey` now in memory, the client can download the encrypted file chunks and decrypt them one by one, reassembling the original file locally.

9 Dragbin's Multi-Layered Defense Against Modern Threats

9.1 Fortifying the Client: Mitigating Offline Attacks with Argon2id and Non-Extractable Web Crypto Keys

First, Dragbin has replaced legacy password-based key derivation functions like PBKDF2 with **Argon2id**. PBKDF2's security relies solely on computational expense, making it vulnerable to massive parallelization on specialized hardware like GPUs.[1] Argon2id, the winner of the 2015 Password Hashing Competition, is designed to be **memory-hard**. It requires a significant amount of RAM to execute, which provides powerful resistance against GPU- and ASIC-based attacks, as these specialized chips have many processing cores but very limited RAM per core.[1] This forces an attacker to spend orders of magnitude more time and money to crack a password compared to PBKDF2, making offline attacks economically infeasible for all but the weakest passwords.[1]

Second, Dragbin leverages the **Web Cryptography API's non-extractable key storage**. When keys, such as the `userKey` derived from the password or the decrypted ML-KEM private key, are imported into the browser's crypto module with the `extractable: false` flag, their raw key bytes are managed by the browser's underlying cryptographic engine and are inaccessible to the JavaScript runtime.[1] Even if an attacker achieves a full Cross-Site Scripting (XSS) compromise, they can abuse the keys within that session but cannot steal the raw key material needed for an offline brute-force attack. This fundamentally mitigates the primary threat, dramatically reducing the attack surface from the entire internet to the single, compromised browser session.[1]

9.2 Authenticating Identity: Thwarting Man-in-the-Middle Attacks with an Integrated Public Key Infrastructure (PKI)

A fundamental requirement for any secure public-key system is authentication: parties must have a way to verify that a public key truly belongs to the person they believe it does. Without this, the system is vulnerable to Man-in-the-Middle (MITM) attacks, where an adversary intercepts a public key request and substitutes their own key, silently decrypting all future communications.[1]

To neutralize this threat, Dragbin implements a lightweight, self-contained Public Key Infrastructure (PKI) where the Dragbin server acts as the central trust anchor. The server operates its own single-level Certificate Authority (CA), possessing a long-term PQC signing key pair (e.g., using ML-DSA, the signature algorithm standardized as FIPS 204 that complements ML-KEM). This private signing key is protected with the utmost security within Dragbin's infrastructure, while the public signing key is hardcoded into the client application.[1]

When a new user registers and uploads their ML-KEM public key, the server generates a simple digital certificate containing the user's identity, their public key, and a validity period. The server then uses its private signing key to create a digital signature over this certificate. When another user requests this public key for sharing, the server sends the full certificate. The receiving client verifies the signature using the server's well-known public signing key. A valid signature provides cryptographic proof that the public key is authentic and was issued by the trusted Dragbin server. This establishes a clear chain of trust that completely defeats MITM attacks.[1]

9.3 Enforcing Access Control: Achieving Immediate and Cryptographically-Enforced Revocation with Proxy Re-Encryption (PRE)

A secure system requires access revocation to be immediate, cryptographically enforced, and practical for the user. Simple revocation models, such as removing a user's key from a file's metadata, are insufficient. This "soft" revocation does not invalidate access for a user who has already downloaded the encrypted file and the necessary keys; they retain the ability to decrypt that data indefinitely.[1] The only alternative in such a model is for the data owner to download, re-encrypt with a new key, and re-upload the entire file—a burdensome process that is impractical for large files.[1]

Dragbin solves this challenge by implementing an advanced cryptographic primitive known as **Proxy Re-Encryption (PRE)**. PRE allows a semi-trusted third party (the Dragbin server) to transform a ciphertext encrypted under one key into a ciphertext that can be decrypted by a different key, *without the proxy being able to learn the plaintext content.*[1] This enables secure delegation of decryption rights, which is the foundation of a robust access control and revocation system.

The PRE workflow in Dragbin operates as follows:

1. **Initial Encryption:** When a data owner, Alice, uploads a file, the `sessionKey` is encapsulated with her own public key.

2. **Delegating Access (Sharing):** To share the file with Bob, Alice uses her private key and Bob's public key to compute a special **re-encryption key**, $rk_{A \to B}$. This key is sent to the Dragbin server and associated with the file and Bob's user ID.

3. **Recipient Access:** When Bob requests the file, the server uses the re-encryption key $rk_{A \to B}$ to transform the ciphertext encapsulated for Alice into a new ciphertext that is now decryptable by Bob's private key. The server sends this new ciphertext to Bob, who can then decrypt it to recover the `sessionKey`. The crucial property is that the server learns nothing about the `sessionKey` during this transformation.[1]

4. **Revoking Access:** To revoke Bob's access, Alice sends a single, simple instruction to the server: delete the re-encryption key $rk_{A \to B}$. The server deletes the key. From that moment on, the server no longer has the cryptographic means to transform ciphertexts for Bob. His access is immediately and irrevocably cut off at the server level.[1]

Table 5: Comparison of Access Revocation Models

Feature	**Legacy Method (Metadata Key Removal)**	**Dragbin's Method (PRE)**
Revocation Immediacy	Not immediate. Access persists for already-downloaded data.	Immediate. Access is cryptographically cut off instantly.
Security Guarantee	Weak. A revoked user can still decrypt old data.	Strong. A revoked user cannot decrypt any data post-revocation.
Data Owner Burden	High. Requires manual download and re-encryption for true revocation.	Minimal. A single, lightweight API call to the server.
Computational Overhead	Low for soft revoke; very high for re-encryption (client-side).	Moderate (server-side). The proxy performs a re-encryption operation.

Table 6: *
Analysis based on the architectural comparison in.[1]

Part IV
Conclusion: The Dragbin Standard for Secure Cloud Storage

The Dragbin quantum-resistant cloud storage platform is built upon a commendable and forward-thinking cryptographic foundation. The selection of CRYSTALS-Kyber, now standardized as FIPS 203 ML-KEM, for key encapsulation and AES-256-GCM for data encryption aligns perfectly with the recommendations of NIST and the global cryptographic community, positioning Dragbin at the vanguard of the post-quantum transition. The core architectural pattern of client-side, end-to-end encryption correctly addresses the fundamental trust issues inherent in public cloud storage.

By undertaking these enhancements, Dragbin has moved beyond a proof-of-concept to become a benchmark for next-generation secure cloud storage. The platform's security rests on three foundational pillars:

1. **Quantum-Resistant by Default:** At its core, Dragbin uses the FIPS 203 ML-KEM standard for all key establishment.
2. **Hardened Against Classical Attacks:** Dragbin implements a suite of advanced defenses including Argon2id, non-extractable keys, a robust PKI, and Proxy Re-Encryption.
3. **Verifiable Zero-Trust Model:** The entire architecture is built on the principle of user-controlled, end-to-end encryption. Dragbin has zero access to user data.

The resulting platform offers its users a truly comprehensive and verifiable security guarantee. Dragbin represents the new standard for secure, private, and future-proof cloud storage, providing a trusted environment for the world's most sensitive data.

References

[1] Encryption Standards: AES, RSA, ECC, SHA and Other Protocols - DEV Community, accessed August 5, 2025, https://dev.to/hardy_mervana/encryption-standards-aes-rsa-ecc-sha-and-other-protocols-460c

[2] Advanced Encryption Standard (AES) - NIST Technical Series Publications, accessed August 5, 2025, https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf

[3] Default encryption at rest — Security - Google Cloud, accessed August 5, 2025, https://cloud.google.com/docs/security/encryption/default-encryption

[4] Key types, algorithms, and operations - Azure Key Vault - Microsoft Learn, accessed August 5, 2025, https://learn.microsoft.com/en-us/azure/key-vault/keys/about-keys-details

[5] Cryptographic Storage - OWASP Cheat Sheet Series, accessed August 5, 2025, https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html

[6] How Quantum Computing Threatens Encryption—and What Your Business Must Do Now, accessed August 5, 2025, https://secureitconsult.com/quantum-computing-threatens-encryption/

[7] The Impact of Quantum Computing on Cryptography — by Prabhu Srivastava - Medium, accessed August 5, 2025, https://medium.com/@prabhuss73/the-impact-of-quantum-computing-on-cryptography-58e76fbb0696

[8] The looming threat of quantum computing to data security - Fractal Analytics, accessed August 5, 2025, https://fractal.ai/article/nists-post-quantum-cryptographic-standards

[9] Shor's Algorithm and RSA Encryption, accessed August 5, 2025, https://www.qai.ca/resource-library/shors-algorithm-and-rsa-encryptionnbsp

[10] SECURITY ANALYSIS OF CRYSTALS-KYBER, accessed August 5, 2025, https://ualberta.scholaris.ca/bitstreams/1f42dfd4-12d9-41d6-a9b5-c9191b7dd18e/download

[11] NIST Post-Quantum Cryptography Standardization - Wikipedia, accessed August 5, 2025, https://en.wikipedia.org/wiki/NIST_Post-Quantum_Cryptography_Standardization

[12] Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process - NIST Computer Security Resource Center, accessed August 5, 2025, https://csrc.nist.gov/csrc/media/projects/post-quantum-cryptography/documents/call-for-proposals-final-dec-2016.pdf

[13] What is the NIST standardization process? - Utimaco, accessed August 5, 2025, https://utimaco.com/service/knowledge-base/post-quantum-cryptography/what-nist-standardization-process

[14] NIST Announces the First 3 Post-Quantum Cryptography Standards - Ready or Not?, accessed August 5, 2025, https://www.appviewx.com/blogs/nist-announces-the-first-3-post-quantum-cryptography-standards-ready-or-not/

[15] FIPS 203, Module-Lattice-Based Key-Encapsulation Mechanism Standard — CSRC, accessed August 5, 2025, https://csrc.nist.gov/pubs/fips/203/final

[16] Module-Lattice-Based Key-Encapsulation Mechanism Standard, accessed August 5, 2025, https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.203.pdf

[17] Kyber - Wikipedia, accessed August 5, 2025, https://en.wikipedia.org/wiki/Kyber

[18] NIST advances post-quantum cryptography standardization, selects HQC algorithm to counter quantum threats - Industrial Cyber, accessed August 5, 2025, https://industrialcyber.co/nist/nist-advances-post-quantum-cryptography-standardization-selects-hqc-algorithm-to-counter-quantum-threats/

[19] Status Report on the Fourth Round of the NIST Post-Quantum Cryptography Standardization Process, accessed August 5, 2025, https://www.nist.gov/publications/status-report-fourth-round-nist-post-quantum-cryptography-standardization-process

[20] Understanding Lattice-Based Cryptography - Blue Goat Cyber, accessed August 5, 2025, https://bluegoatcyber.com/blog/understanding-lattice-based-cryptography/

[21] Lattice-Based Cryptography Explained in 5 Minutes or Less - Geekflare, accessed August 5, 2025, https://geekflare.com/cybersecurity/lattice-based-cryptography/

[22] Lattice-based cryptography: The tricky math of dots - YouTube, accessed August 5, 2025, https://www.youtube.com/watch?v=QDdOoYdb748

[23] Learning with errors - Wikipedia, accessed August 5, 2025, https://en.wikipedia.org/wiki/Learning_with_errors

[24] Learning with Errors (LWE): The Foundation of Post-Quantum Cryptography - Medium, accessed August 5, 2025, https://medium.com/@kootie73/learning-with-errors-lwe-the-foundation-of-post-quantum-cryptography-f85ec40c5840

[25] The Learning with Errors Problem - NYU Courant Institute of Mathematical Sciences, accessed August 5, 2025, https://cims.nyu.edu/~regev/papers/lwesurvey.pdf

[26] Post-quantum cryptography: Lattice-based cryptography - Red Hat, accessed August 5, 2025, https://www.redhat.com/en/blog/post-quantum-cryptography-lattice-based-cryptography

[27] arxiv.org, accessed August 5, 2025, https://arxiv.org/html/2409.02222v1#:~:text=The%20Module%20Learning%20With%20Errors%20problem%20(Module%2DLWE)%20is,cryptography%20%5BBGV14%2C%20LS15%5D%20.

[28] Adventures in PQC: Exploring Kyber in Python - Part I - The Cryptography Caffè, accessed August 5, 2025, https://cryptographycaffe.sandboxaq.com/posts/kyber-01/

[29] A Comprehensive Study of Crystal Kyber, accessed August 5, 2025, https://www.ijarsct.co.in/Paper25711.pdf

[30] CRYSTALS Kyber : The Key to Post-Quantum Encryption — Identity Beyond Borders, accessed August 5, 2025, https://medium.com/identity-beyond-borders/crystals-kyber-the-key-to-post-quantum-encryption-3154b305e7bd

[31] In-Depth Overview of FIPS 203: The Module-Lattice-Based Key-Encapsulation Mechanism Standard - Encryption Consulting, accessed August 5, 2025, https://www.encryptionconsulting.com/overview-of-fips-203/

[32] Kyber - CRYSTALS, accessed August 5, 2025, https://pq-crystals.org/kyber/

[33] Decoding the CRYSTALS-Kyber attack using artificial intelligence: Examination and strategies for resilience - CEUR-WS.org, accessed August 5, 2025, https://ceur-ws.org/Vol-3826/short26.pdf

[34] Verification of the (1–)-Correctness Proof of CRYSTALS-KYBER with Number Theoretic Transform - FAVPQC 2022, accessed August 5, 2025, https://favpqc2022.gitlab.io/papers/Katharina.pdf

[35] Kyber Post-Quantum KEM - IETF, accessed August 5, 2025, https://www.ietf.org/archive/id/draft-cfrg-schwabe-kyber-03.html

[36] What are the official key and ciphertext sizes for ML-KEM-512, ML-KEM-768, and ML-KEM-1024 according to FIPS 203?, accessed August 13, 2024, https://www.encryptionconsulting.com/overview-of-fips-203/

[37] CRYSTALS-Kyber detailed mathematical explanation arXiv, accessed August 26, 2025, https://arxiv.org/html/2508.01694v4

[38] Analyzing Reductions in Kyber to Derive Key Length Recommendations for Post-Quantum Cryptography, accessed August 26, 2025, https://orsp.sonoma.edu/analyzing-reductions-kyber-derive-key-length-recommendations-post-quantum-cryptography

[39] CRYSTALS-Kyber parameters NIST, accessed December 2023, https://csrc.nist.gov/csrc/media/Projects/post-quantum-cryptography/documents/faq/Kyber-512-FAQ.pdf

[40] Formally verifying Kyber - Episode V, accessed May 29, 2024, https://cryptojedi.org/papers/hakyberv-20240529.pdf