-rootdigit®

ROOT DIGIT · SECURITY SERIES

Post-Quantum Security

Cryptographic Agility for the Modern Enterprise — a practical guide to the NIST PQC standards, hybrid migration, and building systems that survive the quantum transition.

AN ENTERPRISE TECHNICAL GUIDE

By Root Digit Security Lab

rootdigit.com

Post-Quantum Security

Cryptographic Agility for the Modern Enterprise

First edition · 2026 · Root Digit Security Series

Published by Root Digit LLC. Root Digit operates from the Dubai International Financial Centre (DIFC) with a registered presence in Wyoming, USA. The firm designs and delivers production AI, robotics, and connected-systems platforms for enterprises in regulated and safety-critical industries.

The guidance in this book reflects engineering practice and published standards as of its publication date. Cryptographic choices should be validated against your own regulatory, security, and reliability requirements, and against the latest guidance from NIST and your sector regulators. Product, algorithm, and standard names are the property of their respective owners.

Contact: info@rootdigit.com · **Web:** rootdigit.com

EXECUTIVE SUMMARY

The Clock Is Already Running

A cryptographically relevant quantum computer does not yet exist. The threat to your data, however, is not in the future — it is happening now. Adversaries are recording encrypted traffic today on the bet that they will be able to decrypt it within the lifetime of its value. The strategy has a name: *harvest now, decrypt later*.

Public-key cryptography — RSA, Diffie-Hellman, and elliptic-curve schemes — secures essentially every TLS session, code-signing chain, VPN tunnel, and digital identity in production today. Its security rests on mathematical problems that a sufficiently large quantum computer running Shor's algorithm would solve efficiently. When that machine arrives, the affected algorithms do not weaken gracefully; they break completely. Any secret protected by them, and recorded before migration, becomes readable.

This book sets out a practical response. In August 2024 the U.S. National Institute of Standards and Technology finalised its first post-quantum cryptography (PQC) standards. The migration is no longer a research topic — it is an engineering programme with published algorithms, government deadlines, and a clear architectural prize: **crypto-agility**, the ability to change cryptographic algorithms without rewriting the systems that depend on them.

3 FINALISED NIST STANDARDS (203/204/205)	**8** CHAPTERS, THREAT TO GOVERNANCE
2035 TARGET FOR U.S. FEDERAL MIGRATION	**1** CORE GOAL: CRYPTOGRAPHIC AGILITY

What you will take away

- A clear account of what quantum computing actually breaks — and, importantly, what it does not.
- A working knowledge of the finalised NIST PQC standards: ML-KEM, ML-DSA, SLH-DSA, and the forthcoming FN-DSA.
- The intuition behind lattice-based cryptography and why it is believed to resist quantum attack.
- Concrete patterns for hybrid key exchange, crypto-agility architecture, and a phased migration roadmap.

- The implementation pitfalls — larger keys, side channels, library maturity — and the governance and compliance landscape, from CNSA 2.0 to ISO.

HOW TO READ THIS BOOK

Chapters 1–3 establish the threat and the new mathematics: what breaks, the standards that replace it, and the lattices underneath. Chapters 4–6 are operational — hybrid key exchange, crypto-agility architecture, and a migration roadmap. Chapters 7–8 cover the practical engineering pitfalls and the governance, mandates, and compliance you will answer to. Start now: the discovery work in Chapter 6 takes longer than the cryptography.

CONTENTS

What's Inside

CHAPTER ONE

The Quantum Threat to Cryptography

The cryptography that protects the modern internet was not designed to resist a quantum computer. Most of it rests on two mathematical problems — integer factorisation and the discrete logarithm — that are hard for classical computers and easy for a large enough quantum one. Understanding precisely which problems break, and which do not, is the foundation of a sound migration.

In 1994 Peter Shor described a quantum algorithm that factors large integers and computes discrete logarithms in polynomial time. The security of RSA depends on factoring being infeasible; the security of Diffie-Hellman and elliptic-curve cryptography (ECC, including ECDSA and ECDH) depends on the discrete logarithm being infeasible. **Shor's algorithm dissolves both.** A cryptographically relevant quantum computer — one with enough stable, error-corrected logical qubits — would recover an RSA private key from its public key, or an ECC private key from a public point, in a matter of hours. The keys do not get weaker; they cease to be secret.

What actually breaks — and what does not

The damage is concentrated in **public-key (asymmetric) cryptography**. Symmetric cryptography is a different story. The best known quantum attack on a symmetric cipher is Grover's algorithm, which provides only a quadratic speed-up to brute-force search. In practical terms, Grover effectively halves the security level of a symmetric key: AES-128 drops to roughly 64 bits of quantum security, while **AES-256 retains around 128 bits — still comfortably out of reach.** The same logic applies to hash functions: doubling the output size (preferring SHA-384/SHA-512 and SHA3 over SHA-256 where margins matter) restores the margin. The remedy for symmetric primitives is to grow key and digest sizes; the remedy for asymmetric primitives is to *replace the algorithm entirely*.

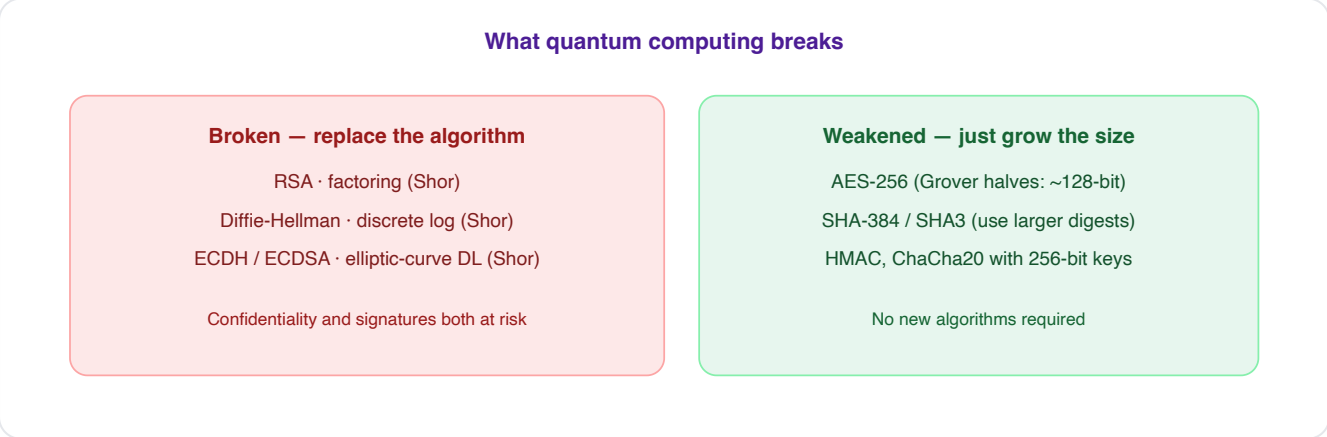

Figure 1.1 — Asymmetric algorithms must be replaced; symmetric primitives only need larger keys and digests.

Harvest now, decrypt later

The most important idea in this book is that the threat is already live. An adversary does not need a quantum computer today to harm you; it only needs to **record your encrypted traffic today** and wait. When a capable machine arrives, the key exchange that protected each captured session can be broken, and the session decrypted retroactively. This is the *harvest-now, decrypt-later* (HNDL) attack, sometimes called "store now, decrypt later".

The practical consequence is a simple time calculation. Let X be how many years your data must stay confidential, Y the years it will take you to migrate, and Z the years until a cryptographically relevant quantum computer exists. **If X + Y > Z, you are already exposed.** This formulation, due to cryptographer Michele Mosca, explains why organisations with long-lived secrets — health records, state secrets, financial master keys, signing roots whose certificates span decades — cannot wait for the quantum computer to be announced. By then the harvested data is already lost.

Two distinct problems to solve

The migration splits cleanly along the two jobs public-key cryptography does. The first is **confidentiality**: protecting key exchange so that recorded traffic cannot be decrypted later. This is urgent precisely because of HNDL, and it is the focus of hybrid key exchange in Chapter 4. The second is **authentication**: digital signatures for certificates, code signing, and identity. A signature only needs to resist forgery *at the moment it is verified* — a quantum computer cannot retroactively forge a signature you accepted last year — so signature migration is important but less time-critical than confidentiality. Keeping these two problems distinct will sharpen every prioritisation decision you make later.

> **IN PRACTICE**
>
> When a Root Digit client first quantified its exposure, the alarming number was not the quantum timeline — it was the data-confidentiality requirement. Several master encryption keys had a mandated lifetime of 25 years. Under any credible quantum estimate, X + Y already exceeded Z. Migration moved from a 2030s line item to a current-year programme overnight.

CHAPTER TWO

The NIST PQC Standards

After an eight-year, open, multi-round competition that began in 2016, NIST published its first finalised post-quantum standards in August 2024. They are real, named Federal Information Processing Standards (FIPS), and they are what you should be designing toward today.

The standardisation effort separated the two jobs from Chapter 1. One algorithm family solves key establishment (confidentiality); the others solve digital signatures (authentication). Knowing which standard does which job is the first step to using them correctly.

FIPS 203 — ML-KEM (Kyber): key establishment

ML-KEM (Module-Lattice-based Key-Encapsulation Mechanism), standardised in FIPS 203 and derived from the scheme known as CRYSTALS-Kyber, is NIST's primary tool for protecting key exchange. A key-encapsulation mechanism (KEM) is the modern replacement for Diffie-Hellman: one party generates an ephemeral public key, the other "encapsulates" a freshly generated shared secret to it, and both sides derive the same symmetric key. ML-KEM ships in three parameter sets — ML-KEM-512, -768, and -1024 — corresponding to NIST security categories 1, 3, and 5. **ML-KEM-768 is the common default** for general use; it is what major browsers and TLS stacks have adopted in their hybrid deployments.

FIPS 204 — ML-DSA (Dilithium): the default signature

ML-DSA (Module-Lattice-based Digital Signature Algorithm), standardised in FIPS 204 and derived from CRYSTALS-Dilithium, is NIST's primary signature scheme. Like ML-KEM it is lattice-based, which gives it good all-round performance: reasonably fast signing and verification with moderate key and signature sizes. For most signing needs — TLS certificates, software updates, document signing — **ML-DSA is the recommended default.**

FIPS 205 — SLH-DSA (SPHINCS+): the conservative hedge

SLH-DSA (Stateless Hash-Based Digital Signature Algorithm), standardised in FIPS 205 and derived from SPHINCS+, is a signature scheme built entirely from hash functions. Its security rests only on the well-understood properties of cryptographic hashes — not on the lattice assumptions shared by ML-KEM and ML-DSA. That makes it the **diversity hedge:** if a future advance ever undermined lattices, hash-based signatures would stand unaffected. The trade-off is steep — SLH-DSA signatures are large (tens of kilobytes) and signing is slow — so it is reserved for low-frequency, high-assurance uses such as firmware and root-of-trust signing.

FN-DSA (FALCON): compact signatures, coming soon

A fourth scheme, **FN-DSA** (FFT over NTRU lattices, derived from FALCON), is being standardised as FIPS 206 and was not yet finalised at the time of writing. Its appeal is small signatures and public keys, valuable where bandwidth or storage is tight — for instance in certificate chains. The catch is that FN-DSA signing uses floating-point Gaussian sampling, which is genuinely difficult to implement in constant time; this implementation sensitivity is part of why it has taken longer to finalise.

The standards at a glance

Standard	Algorithm (origin)	Job	Use it for
FIPS 203	ML-KEM (Kyber)	Key encapsulation	TLS / VPN key exchange — the HNDL priority
FIPS 204	ML-DSA (Dilithium)	Digital signature	Default signing: certificates, code, documents
FIPS 205	SLH-DSA (SPHINCS+)	Digital signature	Conservative hedge: firmware, roots of trust
FIPS 206*	FN-DSA (FALCON)	Digital signature	Bandwidth-constrained signing (*draft)

One KEM, three signatures — why?

The asymmetry is deliberate. For confidentiality, a single strong, well-performing KEM (ML-KEM) is enough, and there is no value in carrying two interoperable KEMs. For signatures, NIST chose to standardise more than one scheme to provide *cryptographic diversity*: a lattice default (ML-DSA), a hash-based alternative resting on different assumptions (SLH-DSA), and a compact option (FN-DSA). Should one mathematical family ever be weakened, the others provide a fallback — which is itself an argument for the crypto-agility we build in Chapter 5.

A practical consequence flows from this design. For nearly every enterprise the working defaults are simple: **ML-KEM-768 for key exchange and ML-DSA for signing**, with SLH-DSA reserved for the few signing operations — firmware images, the root of a code-signing hierarchy — where you are willing to trade size and speed for the strongest possible assurance and the comfort of a non-lattice foundation. Resist the temptation to deploy every algorithm everywhere; carry the minimum your risk profile justifies, and let the abstraction layer make the rare exceptions a matter of policy rather than rewriting. The standards give you a small, deliberate toolkit, not a menu to be exhausted.

It is also worth being clear about what these standards are *not*. They do not replace your symmetric ciphers or hashes — AES-256 and SHA3 carry forward unchanged, as Chapter 1 explained. They do not, on their own, make a system quantum-safe; an ML-KEM handshake that still presents a classical-only certificate, or that silently downgrades, gains you little. And finalised though they are, they will evolve: pa-

rameter recommendations are refined, and FN-DSA's arrival will add an option rather than retire an existing one. The standards are a stable foundation to build on, not a box to be ticked once.

NAMING NOTE

You will see the research names (Kyber, Dilithium, SPHINCS+, FALCON) and the standard names (ML-KEM, ML-DSA, SLH-DSA, FN-DSA) used interchangeably. They are not perfectly identical — the FIPS versions include tweaks made during standardisation — so in production always specify the **FIPS algorithm and parameter set** (e.g. "ML-KEM-768"), never just "Kyber".

CHAPTER THREE

Lattice-Based Cryptography

Three of the four NIST schemes are built on lattices. You do not need to be a number theorist to migrate, but a working intuition for why lattices are believed to resist quantum attack will help you reason about sizes, parameters, and the residual risks worth hedging.

The hard problem: learning with errors

The security of ML-KEM and ML-DSA reduces to the difficulty of the **Learning With Errors** (LWE) problem. The intuition is approachable. Imagine a system of linear equations over a finite field: given a matrix A and the products $A \cdot s$, recovering the secret vector s is trivial schoolbook algebra. Now perturb each equation by adding a small, random *error* term. The equations no longer hold exactly — each is "almost right" — and suddenly recovering s becomes extraordinarily hard. The noise turns clean algebra into a search through an enormous space of near-solutions. That is LWE: solving noisy linear equations.

Geometrically, the public data defines a **lattice** — a regular grid of points stretching through high-dimensional space — and the secret corresponds to finding the lattice point closest to a given target (the Closest Vector Problem) or the shortest non-zero vector in the lattice (the Shortest Vector Problem). In hundreds of dimensions, with the right structure, these problems are believed to be hard even for a quantum computer.

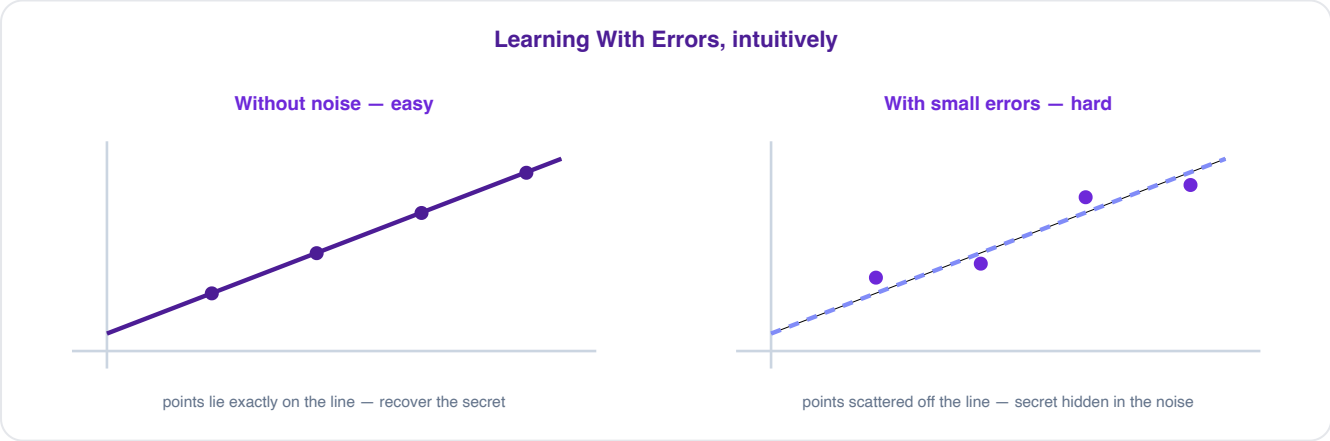

Figure 3.1 — Adding a small error term to each equation turns trivial linear algebra into a hard search problem.

Module-LWE: structure for speed

Plain LWE with a fully random matrix would yield enormous keys. The NIST lattice schemes instead use **Module-LWE**, a structured variant that arranges the secret and matrix as small modules over a polynomial ring. This structure dramatically shrinks key sizes and speeds up arithmetic (it admits the fast Number-Theoretic Transform for polynomial multiplication) while — on current understanding — pre-

serving hardness. The "Module" in ML-KEM and ML-DSA refers exactly to this. The trade is a modest, carefully studied assumption that the added algebraic structure does not introduce an exploitable weakness; this is the part of the design most watched by cryptanalysts, and the reason a non-lattice hedge (SLH-DSA) exists.

Why we believe it is quantum-hard

Two reasons give confidence. First, despite decades of study, no efficient quantum algorithm is known for the general lattice problems underlying LWE — Shor's algorithm simply does not apply, because there is no hidden periodic structure to exploit. Second, lattice problems enjoy unusually strong **worst-case to average-case** reductions: breaking a random instance would imply solving the hardest cases of well-studied lattice problems. That said, "believed hard" is not "proven hard". The honest engineering posture is to deploy the standards, prefer higher parameter sets where margins matter, and retain agility so a parameter or algorithm can be changed if cryptanalysis advances.

The cost: everything gets bigger

Quantum resistance is not free, and the bill is paid in size. Where an X25519 public key is 32 bytes and an Ed25519 signature 64 bytes, the post-quantum equivalents are an order of magnitude larger. An ML-KEM-768 ciphertext is about 1,088 bytes and its public key about 1,184 bytes; an ML-DSA-65 signature is roughly 3,300 bytes; an SLH-DSA signature can exceed 17,000 bytes. These numbers drive the protocol, bandwidth, and storage concerns of Chapters 4 and 7.

Approximate sizes — classical vs post-quantum

Scheme	Public key	Ciphertext / signature	Note
X25519 (classical KEM)	32 B	32 B	Baseline for comparison
ML-KEM-768	~1,184 B	~1,088 B (ciphertext)	Common KEM default
Ed25519 (classical sig)	32 B	64 B (signature)	Baseline for comparison
ML-DSA-65	~1,952 B	~3,309 B (signature)	Default PQ signature
SLH-DSA-128s	32 B	~7,856 B (signature)	Hash-based; sizes vary widely

Sizes are approximate and parameter-dependent; treat them as orders of magnitude. The lesson is consistent: post-quantum keys and signatures are large enough to matter, and any system that hard-codes assumptions about cryptographic object sizes will need attention.

CHAPTER FOUR

Hybrid Key Exchange & Migration

The first thing most enterprises deploy is not pure post-quantum cryptography but a *hybrid*: a classical algorithm and a post-quantum one run side by side, combined so that the connection is secure if *either* survives. Hybrid is the bridge across the transition, and for key exchange it is the single highest-value move you can make this year.

Why hybrid, and why first

Hybrid key exchange runs an established classical KEM — typically X25519 — alongside a post-quantum KEM such as ML-KEM-768, and derives the session key from **both** shared secrets concatenated together. The reasoning is risk management. The classical algorithm is decades-proven against classical attack but doomed against quantum. The PQ algorithm resists quantum attack but its implementations are young and its lattice assumptions, while well-studied, are newer. Combining them means an attacker must break *both* to recover the key: you inherit the maturity of the classical scheme and the quantum resistance of the new one. If the PQC has a flaw, you are no worse off than today; if the classical is quantum-broken, the PQC still holds.

It goes first because of harvest-now, decrypt-later. Every TLS or VPN session you negotiate today with classical-only key exchange is a session an adversary can record and decrypt in the future. Switching key exchange to hybrid **stops the bleeding** — from the moment hybrid is enabled, newly recorded traffic is protected against the future quantum computer. Signatures, by contrast, do not have this retroactive exposure and can follow.

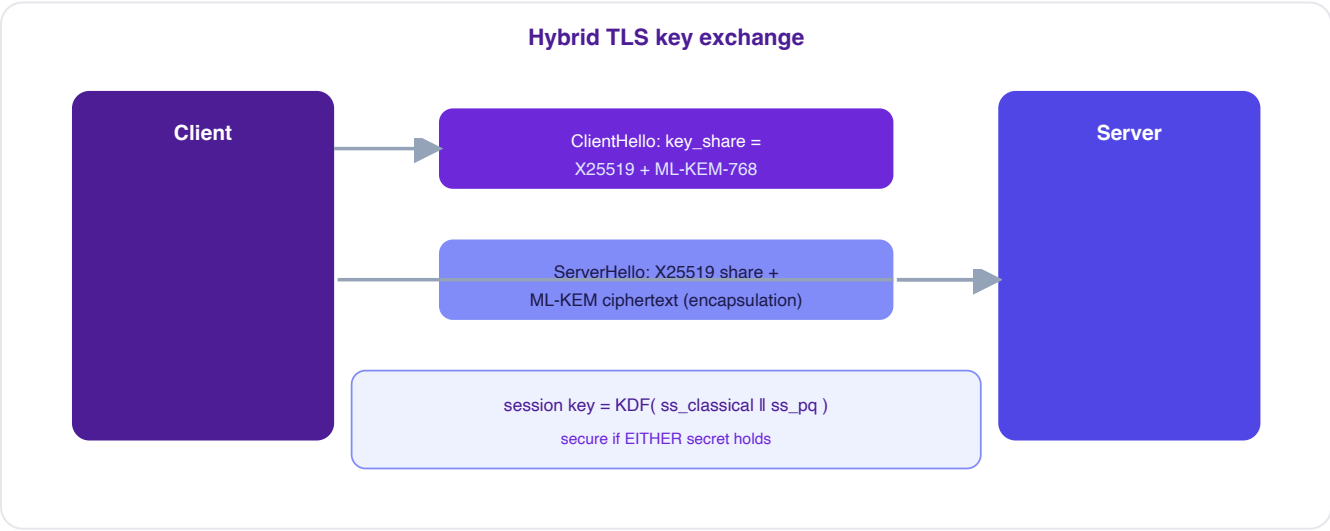

Figure 4.1 — In hybrid TLS 1.3 both a classical and a PQ secret are established; the session key derives from both.

Where it plugs in

Hybrid key exchange lives in the protocols you already operate. In **TLS 1.3** it is expressed as a named group — the IANA-registered `X25519MLKEM768` — negotiated in the same `key_share` extension as classical curves, requiring no protocol redesign. Major browsers and servers shipped this group in 2024–2025, and for many enterprises the fastest win is simply enabling it on load balancers, CDNs, and reverse proxies. **SSH** has carried a hybrid method (`sntrup761x25519-sha512`, with ML-KEM variants following) for some time. **IPsec/IKEv2** supports multiple additional key exchanges so a PQ KEM can be layered onto existing tunnels. In each case the pattern is the same: negotiate two secrets, combine them, lose nothing if one fails.

```
# Hybrid KEM — combine a classical and a PQ shared secret
def hybrid_key_exchange(peer):
    cl_pub, cl_priv = X25519.keygen()              # classical, decades-proven
    pq_pub, pq_priv = ML_KEM_768.keygen()          # post-quantum (FIPS 203)
    send(peer, cl_pub + pq_pub)                    # both key shares in one ClientHello

    cl_peer, pq_ct = recv(peer)
    ss_cl = X25519.derive(cl_priv, cl_peer)        # classical shared secret
    ss_pq = ML_KEM_768.decapsulate(pq_priv, pq_ct) # PQ shared secret

    # concatenate, then run through a KDF — break BOTH to win
    return HKDF(ss_cl + ss_pq, info=b"tls13 hybrid")
```

Listing 4.1 — Hybrid combiner pseudocode: the session key depends on both the classical and the post-quantum secret.

Migration sequencing

A sound order follows the threat. **First**, enable hybrid key exchange everywhere confidentiality is recorded or long-lived — external TLS, VPNs, internal service mesh — to neutralise HNDL. **Second**, migrate signatures and certificates to PQ or hybrid certificates, starting with the longest-lived roots and the slowest-to-rotate firmware. **Third**, retire classical-only paths once the ecosystem and your own inventory allow. Throughout, hybrid lets you advance without betting everything on a young algorithm — exactly the agility the next chapter makes structural.

One caveat keeps hybrid honest: the combiner must be done correctly. Concatenating the two shared secrets and passing them through a key-derivation function is the standardised approach precisely because it guarantees the result is at least as strong as the stronger input — neither secret alone reveals the session key. Naive shortcuts, such as XORing the secrets or simply picking one when the other is unavailable, can quietly undermine the property you deployed hybrid to obtain. Use the construction your protocol stack already implements for the registered hybrid groups rather than inventing your own; this is one more reason hybrid belongs in vetted libraries and at well-understood integration points, not in bespoke application code.

IN PRACTICE

The lowest-effort, highest-impact step we recommend to clients is enabling `X25519MLKEM768` at the edge — the CDN, load balancer, and TLS-terminating proxy. Modern stacks support it as a configuration flag. One change protects the bulk of inbound traffic against harvest-now, decrypt-later, often before a single application is touched.

CHAPTER FIVE

Crypto-Agility Architecture

The deepest lesson of the PQC transition is not which algorithm to choose — it is that you will have to choose again. Parameters will shift, FN-DSA will finalise, cryptanalysis will advance. Crypto-agility is the architectural property that lets you change cryptographic algorithms without rewriting the systems that depend on them. Build it once and every future transition becomes a configuration change.

You cannot migrate what you cannot find

Agility begins with knowledge. Most enterprises have no accurate picture of where cryptography lives: it is buried in application code, embedded in libraries, baked into certificates, hidden in hardware modules and third-party services. The first artefact of an agile programme is a **cryptographic bill of materials** (CBOM) — a machine-readable inventory of every algorithm, key, certificate, and protocol in use, where it runs, and what data it protects. Automated discovery (scanning binaries, certificate stores, TLS endpoints, and source) builds it; governance keeps it current. The CBOM is the map without which Chapter 6's roadmap is guesswork.

```yaml
# Cryptographic inventory entry (CBOM-style)
asset: payments-api
endpoint: tls://payments.internal:443
protocol:    TLS1.3
key_exchange:
  current:  X25519                # classical only — HNDL exposed
  target:   X25519MLKEM768        # hybrid (FIPS 203)
signature:
  current:  ECDSA-P256
  target:   ML-DSA-65             # FIPS 204
data_sensitivity: cardholder      # long-lived → high priority
owner: team-payments@rootdigit.com
status: migration-scheduled
```

Listing 5.1 — A cryptographic inventory entry makes each system's current and target algorithms explicit and auditable.

Abstraction: never hard-code an algorithm

The structural enemy of agility is the hard-coded primitive — `RSA.encrypt(...)`, a pinned curve name, an algorithm OID compiled into a binary. Each such call site is a place you must find and edit when the algorithm changes. The remedy is a **cryptographic abstraction layer**: application code requests a *capability* ("encapsulate a key to this peer", "sign this artefact at high assurance") and a central provider decides which concrete algorithm satisfies it, driven by policy and the CBOM. Algorithms be-

come a configuration value, not a code dependency. Crucially, the abstraction must carry an **algorithm identifier** alongside every ciphertext, signature, and key, so the system always knows how to verify or decrypt material produced under an older policy.

```
// Crypto-agility interface — capabilities, not algorithms
interface CryptoProvider {
  // caller asks for a capability + assurance tier; provider picks the algorithm
  encapsulate(peerKey, tier: "hybrid" | "pq"): { ciphertext, sharedSecret, algId }
  decapsulate(privKey, ciphertext, algId): sharedSecret    // algId selects the routine

  sign(payload, tier: "default" | "high-assurance"): { signature, algId }
  verify(payload, signature, algId): boolean           // verify old material too
}
// policy maps tiers -> algorithms; changing it migrates without touching callers
policy = { hybrid: "X25519MLKEM768", pq: "ML-KEM-768",
           default: "ML-DSA-65", "high-assurance": "SLH-DSA-128s" }
```

Listing 5.2 — An abstraction that exposes capabilities and assurance tiers, with an algorithm identifier on every artefact.

Key management for an agile world

Keys must be addressable, rotatable, and versioned independently of the code that uses them. Centralise issuance and storage in a key-management system or hardware security module, reference keys by identifier rather than embedding material, and design for **multiple algorithms coexisting** — an asset may hold a classical key, a PQ key, and a hybrid pair simultaneously during transition. Verify that your HSMs and KMS support the PQ algorithms before committing; some hardware roots of trust will need firmware updates or replacement, and that lead time belongs in the roadmap.

Rotation deserves particular attention. A system that cannot rotate a key cannot, in practice, change its algorithm either — the two operations share the same machinery. If your application can already issue a new key version, retire an old one, and re-encrypt or re-sign material under the new key without downtime, then swapping a classical algorithm for a hybrid or post-quantum one is the same operation with a different parameter. Conversely, every place where a key is pinned, cached indefinitely, or assumed never to change is a place a future migration will stall. Treat the ability to rotate as the litmus test of agility: exercise it regularly in normal operations, not for the first time under a quantum deadline.

Agility is a capability, not a project

It is tempting to frame crypto-agility as a one-time refactor that ends when the post-quantum algorithms are in place. That framing misses the point. The transition to PQC is the first of many cryptographic changes you will make: parameter sets will be revised, FN-DSA will arrive, deprecated curves will be withdrawn, and — should cryptanalysis ever advance against lattices — you may need to pivot to the hash-based hedge in a hurry. An organisation that has built the inventory, the abstraction layer, and the rotation discipline has not merely migrated to post-quantum cryptography; it has acquired a standing

capability to absorb whatever the next two decades of cryptographic change demand, at the cost of a configuration update rather than a programme.

DESIGN PRINCIPLE

Treat the algorithm as data, never as code. Every encrypted blob, signature, certificate, and key should self-describe the algorithm and parameters that produced it. Systems that can read their own metadata can be migrated; systems that assume a single hard-coded algorithm must be rewritten — and rewriting under deadline is how migrations fail.

CHAPTER SIX

The Migration Roadmap

With the threat understood, the standards chosen, and an agile architecture in mind, migration becomes a programme rather than a panic. It has four phases — discover, prioritise, transition, and validate — and the most time-consuming is the first.

Phase 1 — Discovery

You cannot protect what you cannot see. Discovery produces the cryptographic inventory of Chapter 5: every algorithm, key, certificate, library, and protocol, mapped to the systems that use it and the data it protects. Cast the net wide — application code, TLS endpoints, certificate authorities, code-signing pipelines, VPN concentrators, databases at rest, message queues, HSMs, and the cryptography embedded in vendor products and SaaS you consume. **Budget more time here than feels reasonable.** In practice, discovery routinely consumes the largest share of a migration timeline, because cryptography hides in places no one documented.

Phase 2 — Risk prioritisation

You will not migrate everything at once, so order the work by exposure. The dominant factor is the Mosca calculation from Chapter 1: **data with the longest confidentiality lifetime, protected by key exchange, comes first**, because it is most exposed to harvest-now, decrypt-later. A health record that must stay private for fifty years is more urgent than a session cookie that expires in an hour. Rank assets by data sensitivity and required confidentiality lifespan, then by exposure (internet-facing before internal), then by difficulty to change (slow-to-rotate firmware and long-lived certificate roots need early starts even if individually lower risk).

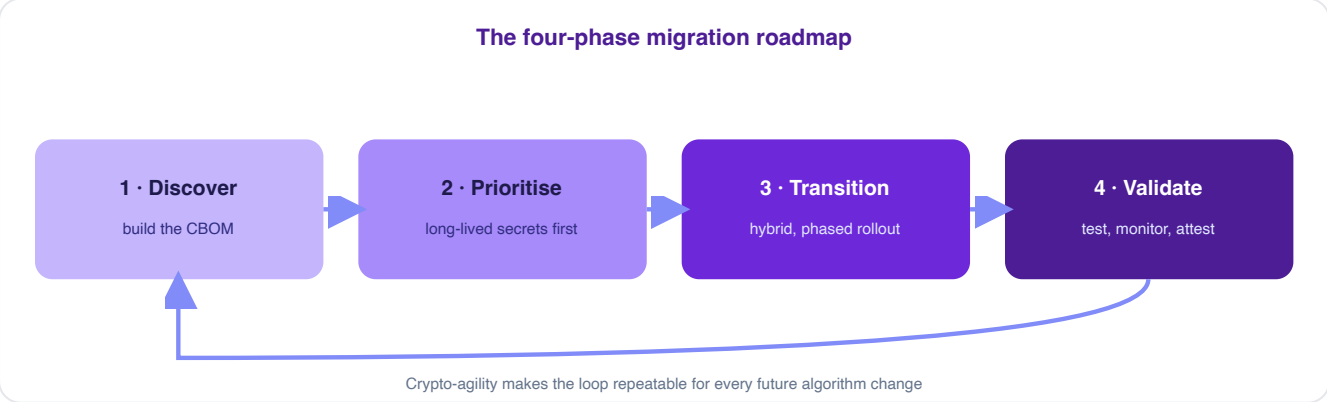

Figure 6.1 — A repeatable migration loop; agility (Chapter 5) turns a one-off project into a standing capability.

Phase 3 — Phased transition

Roll out in waves, almost always starting with **hybrid** rather than pure PQ. Enable hybrid key exchange at the network edge first (the high-leverage move from Chapter 4), then work inward to service-to-service communication, then to signatures and certificates. Use feature flags and the abstraction layer of Chapter 5 to switch algorithms per service, so a problem in one wave does not block the others. Run dual-stack where needed — accepting both classical and hybrid — to maintain interoperability with peers and vendors who have not yet migrated. Resist a flag-day cutover; gradual rollout with the ability to roll back is what keeps availability intact.

Phase 4 — Validation

Each wave needs testing before and monitoring after. Confirm correct negotiation (that hybrid groups are actually selected, not silently downgraded), measure the latency and handshake-size impact discussed in Chapter 7, and watch for interoperability failures with older peers. Critically, test the **rollback path**: the abstraction layer must be able to revert a service to its previous algorithm without data loss, since material encrypted under the new algorithm must remain decryptable. Then attest the change in the inventory and reporting that governance (Chapter 8) will rely on.

A pragmatic first year

- **Months 1–4:** stand up automated discovery; build the first cryptographic inventory; assign owners.
- **Months 3–6:** rank assets by confidentiality lifetime and exposure; identify the long-lived-secret hot list.
- **Months 5–9:** enable hybrid key exchange (X25519MLKEM768) at the edge and on the highest-risk endpoints.
- **Months 8–12:** introduce the crypto-agility abstraction; begin migrating signatures on the longest-lived roots.

Two cross-cutting practices hold the phases together. The first is **vendor engagement**: much of your cryptography is not yours to change directly — it lives in SaaS products, managed databases, payment gateways, and appliances. Begin asking suppliers for their PQC roadmaps during discovery, not at transition; their timelines constrain yours, and the request itself nudges the market. The second is **communication**: a migration touches interoperability with every external partner, so coordinate cutover windows, dual-stack periods, and certificate changes with the peers you connect to, and publish your own readiness so others can plan against it.

The goal of year one is not "all post-quantum". It is to neutralise the harvest-now exposure on your most sensitive data and to install the agility that makes every subsequent change routine. Done in this order — discover widely, prioritise by confidentiality lifetime, transition through hybrid, and validate including

rollback — the programme stays a controlled engineering effort rather than the emergency it becomes for those who start once a quantum computer is already on the horizon.

CHAPTER SEVEN

Performance & Implementation Pitfalls

Post-quantum algorithms are practical — the lattice schemes are fast — but they are not drop-in replacements. Larger keys and signatures stress protocols and storage, and the implementations are young enough that side-channel safety and library maturity deserve real scrutiny.

Bigger objects, real consequences

As Chapter 3 showed, PQ keys and signatures are an order of magnitude larger than their classical counterparts. The most visible effect is on **handshake size**. An ML-KEM-768 key share plus ciphertext adds roughly two kilobytes to a TLS handshake. That sounds trivial, but it can push the ClientHello past a single network packet (the typical ~1,500-byte MTU), causing fragmentation, and it interacts badly with TCP's initial congestion window — measurably increasing connection-setup latency on lossy or high-latency links. PQ signatures compound the problem in **certificate chains**: an ML-DSA-signed chain is several kilobytes larger per certificate, and SLH-DSA's tens-of-kilobytes signatures can dominate a handshake entirely. Where signature size is the binding constraint, this is the argument for the compact FN-DSA, and for keeping chains shallow.

Computation: mostly fine, with sharp edges

For raw speed the lattice schemes are competitive — ML-KEM and ML-DSA operations are often comparable to or faster than RSA, though typically slower than the extremely lean elliptic-curve operations. The sharp edges are specific. **SLH-DSA signing is slow** and produces huge signatures, so reserve it for infrequent, high-assurance signing. **FN-DSA signing** is fast but relies on floating-point Gaussian sampling that is notoriously hard to make constant-time and portable. Measure on your actual hardware — including constrained IoT and embedded devices, where the larger memory footprint of PQ key generation can itself be a limiting factor.

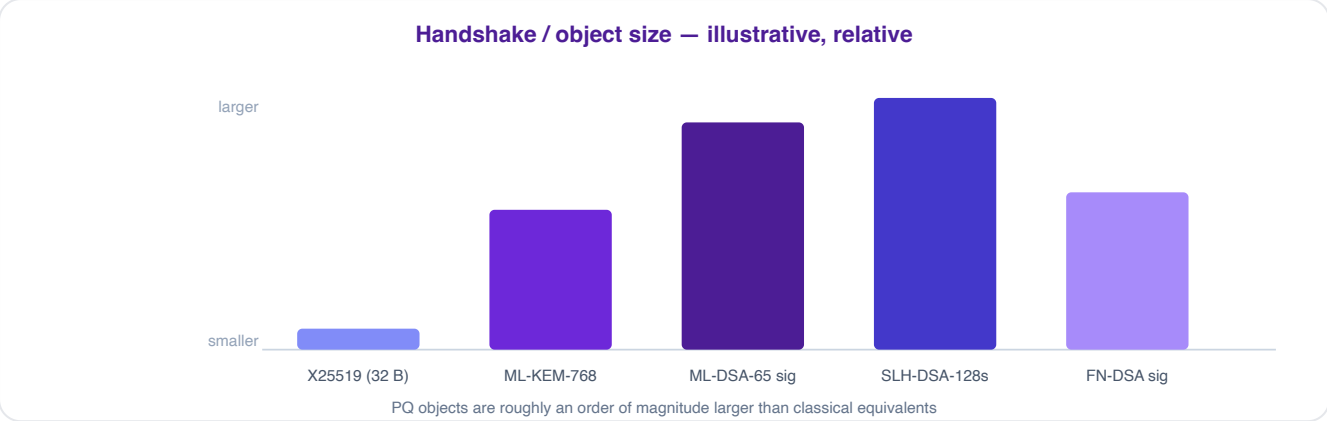

Figure 7.1 — Relative size of PQ keys and signatures versus a classical baseline. Sizes are illustrative, not to scale.

Side channels and constant-time code

The most dangerous pitfall is not performance but **side-channel leakage**. A cryptographic implementation whose timing, power draw, or memory-access pattern depends on secret data can leak that secret regardless of the algorithm's mathematical strength. Lattice schemes have their own hazards — secret-dependent rejection sampling in signing, the modular arithmetic of the NTT, and the FN-DSA Gaussian sampler are all places where a naive implementation can leak. The rule is absolute: **cryptographic code that touches secrets must be constant-time**, with no secret-dependent branches or memory accesses. This is exactly why you should not write your own — use vetted libraries and keep them patched.

Choosing libraries

Implementation maturity is improving quickly but remains uneven. Favour widely reviewed, actively maintained implementations: the algorithms are increasingly available in mainstream TLS stacks and in established cryptographic libraries, with NIST's reference and test vectors available to validate against. Where the platform offers it, prefer **FIPS-validated** modules for regulated workloads, and confirm the specific PQ algorithms and parameter sets are covered by the validation. Test against the official Known-Answer Tests, pin versions, and track advisories — a young algorithm in a young library is precisely where you want the discipline of a managed dependency.

A related decision is *where* the cryptography runs. Offloading PQ operations to hardware — a network card, an HSM, or a dedicated accelerator — can absorb the extra computation and keep secret material off the general-purpose CPU, but only if that hardware actually supports the chosen algorithms; much of the installed base does not yet, and confirming support is part of discovery. For software paths, memory is an underappreciated constraint: ML-KEM and ML-DSA key generation needs noticeably more working memory than elliptic-curve operations, which matters on constrained devices and in environments that create many short-lived contexts. Profile real workloads under realistic concurrency rather than trusting a single-operation benchmark.

Plan for the long migration tail

Finally, accept that the ecosystem will be mixed for years. You will run hybrid and classical-only peers side by side, support old certificates while issuing new ones, and decrypt data sealed under several generations of algorithm. This is not a failure of the migration; it is its normal steady state. Design protocols and storage to **carry the algorithm identifier with the data** (the agility principle of Chapter 5), so a system encountering an artefact always knows how it was produced. The implementations that age well are not the fastest ones — they are the ones that never assumed there would only ever be one algorithm.

> **COMMON PITFALL**
>
> Silent downgrade. A misconfigured server or middlebox can negotiate a classical-only group while everyone believes hybrid is active, leaving traffic exposed with no error raised. Always verify the *actually negotiated* group in monitoring — do not assume the configuration took effect. Re-run the check after every infrastructure change.

CHAPTER EIGHT

Governance & Compliance

The quantum transition is not only an engineering programme; it is increasingly a compliance obligation with published timelines. Governance turns ad-hoc remediation into an accountable, audited capability — and answers the regulator who will, sooner than you expect, ask what your plan is.

Mandates and timelines

In the United States, the direction is set. National Security Memorandum 10 (NSM-10, 2022) directed federal agencies to prepare for the migration, and the NSA's **Commercial National Security Algorithm Suite 2.0** (CNSA 2.0) specifies the post-quantum algorithms — ML-KEM and ML-DSA among them — for national-security systems, with a phased schedule that begins immediately and targets broad adoption through the early-to-mid 2030s. Federal civilian agencies, under OMB guidance and CISA/NIST coordination, are required to inventory their cryptographic systems and plan migration, with the U.S. government broadly aiming to complete its transition by **2035**. Other jurisdictions are moving in parallel — Europe's ENISA and national agencies have issued PQC roadmaps, and several recommend hybrid deployment during the transition. Even outside government, these schedules cascade through supply chains: vendors and contractors inherit the deadlines of the institutions they serve.

Standards to align with

Beyond the FIPS algorithm standards of Chapter 2, two bodies of standards shape governance. ISO/IEC is incorporating the post-quantum schemes into its cryptographic standards, giving internationally recognised reference points for procurement and audit. And the concept of the **cryptographic bill of materials** is being formalised (for example as an extension of the CycloneDX SBOM standard), turning the inventory of Chapter 5 from an internal spreadsheet into an interoperable, auditable artefact. Aligning your inventory format to an emerging standard now saves rework when auditors and customers begin to ask for it in a defined form.

Governance reference points

Source	What it sets	Implication for you
FIPS 203 / 204 / 205	Approved PQ algorithms	Design toward these; specify parameter sets
NSM-10 / CNSA 2.0 (US)	National-security migration mandate & schedule	Binding for NSS; cascades to suppliers
OMB / CISA / NIST guidance	Federal inventory & ~2035 target	Inventory now; report progress
ENISA & national roadmaps	EU / international guidance, often hybrid-first	Multi-jurisdiction programmes must track several
ISO/IEC & CBOM standards	International algorithm & inventory standards	Procurement, audit, interoperable reporting

Reporting and accountability

Governance needs the same artefacts that make the engineering work: a maintained cryptographic inventory, a prioritised migration plan tied to data-confidentiality lifetimes, named owners per asset, and a regular progress report against milestones. Assign clear executive accountability — a quantum-readiness owner who reports to risk leadership — and integrate the programme into existing risk management rather than running it as an island. Vendor and supply-chain governance matters too: require that critical suppliers disclose their own PQC roadmaps and the algorithms in the products you depend on, because their timelines become yours.

Start now — the recurring theme

Every strand of this book converges on a single instruction: begin. The standards are finalised, the mandates are published, the harvest-now threat is live today, and the most time-consuming work — discovery and crypto-agility — must precede the cryptography itself. Organisations that wait for a quantum computer to be announced will be migrating under emergency conditions, with their most sensitive data already harvested. Those that build the inventory, enable hybrid key exchange, and install crypto-agility now will treat the eventual arrival of the quantum computer as a non-event — exactly as good engineering intends.

IN PRACTICE

Root Digit bakes cryptographic inventory and algorithm metadata into its platform templates, so every new service is born crypto-agile and self-describing. Quantum readiness then becomes a property the system has by construction, and the governance report writes itself from the inventory — rather than being a separate forensic exercise undertaken under deadline.

IN CLOSING

Key Takeaways & Further Reading

Ten things to remember

1. The threat is live today: harvest-now, decrypt-later means recorded traffic is already exposed. If X + Y > Z, act now.
2. Quantum computing breaks public-key crypto (RSA, DH, ECC via Shor); symmetric crypto only needs larger keys (AES-256, SHA-384).
3. Separate the two problems: confidentiality (key exchange) is urgent because of HNDL; signatures can follow.
4. The standards are real and final — FIPS 203 ML-KEM for key exchange, FIPS 204 ML-DSA as the default signature.
5. FIPS 205 SLH-DSA is the conservative, hash-based hedge; FN-DSA (FALCON) is the compact, still-finalising option.
6. Lattice security rests on Learning With Errors — believed quantum-hard, but newer; keep a non-lattice hedge available.
7. Deploy hybrid first: classical + PQ combined, secure if either holds. Enable X25519MLKEM768 at the edge for the biggest win.
8. Build crypto-agility: inventory everything (a CBOM), abstract away algorithms, tag every artefact with its algorithm.
9. Migrate in phases — discover, prioritise by confidentiality lifetime, transition with hybrid, validate and test rollback.
10. Watch the pitfalls — handshake bloat, side channels, library maturity — and align with CNSA 2.0, the 2035 targets, and ISO.

Further reading

- FIPS 203 — Module-Lattice-Based Key-Encapsulation Mechanism Standard (ML-KEM), NIST, 2024.
- FIPS 204 — Module-Lattice-Based Digital Signature Standard (ML-DSA), NIST, 2024.
- FIPS 205 — Stateless Hash-Based Digital Signature Standard (SLH-DSA), NIST, 2024.
- NSA Commercial National Security Algorithm Suite 2.0 (CNSA 2.0) and U.S. National Security Memorandum 10.
- NIST SP 1800-38 — Migration to Post-Quantum Cryptography (NCCoE practice guide); ENISA PQC guidance.
- Root Digit Security Series — companion volumes on Zero-Trust Architecture and on Cryptographic Key Management.

ABOUT ROOT DIGIT

Root Digit is an applied AI, robotics, and connected-systems firm headquartered in the Dubai International Financial Centre, with a registered presence in Wyoming, USA. We design and operate production intelligence and security platforms for enterprises in financial services, healthcare, manufacturing, energy, and defence. Learn more at **rootdigit.com**.

Be ready before the quantum clock runs out.

Root Digit helps enterprises navigate the post-quantum transition described in this book — building the cryptographic inventory, enabling hybrid key exchange, and installing the crypto-agility that turns the arrival of a quantum computer into a non-event. We bring the standards and the architecture; you keep your secrets secret.

Talk to our security engineers →

rootdigit.com

General: info@rootdigit.com · **Consultation:** rootdigit.com/contact/consultation
Dubai International Financial Centre (DIFC) · Wyoming, USA