A Friendly Tour of Post-Quantum Digital Signatures

A White Paper for Mathematics Students

Sara Malik and Naveed A.A.
PakCrypt NPO

May 2026

Abstract

In 2024 the U.S. National Institute of Standards and Technology (NIST) finalized its first batch of *quantum-safe* cryptographic standards, and on 14 May 2026 it advanced nine further digital-signature candidates to a third round of evaluation. This white paper explains, from the ground up, the mathematics behind thirteen of these schemes: the three signature standards already finalized (ML-DSA, SLH-DSA, and the forthcoming FN-DSA), together with the nine round-three candidates (FAEST, HAWK, MAYO, MQOM, QR-UOV, SDitH, SNOVA, SQIsign, and UOV). Our intended reader has just begun an undergraduate mathematics degree. We therefore assume only high-school algebra and a willingness to think carefully; every more advanced idea — modular arithmetic, vectors, polynomials, lattices, elliptic curves — is built up slowly, with pictures, analogies, and small numerical examples. The goal is not to make you a cryptographer in forty pages, but to let you see that each of these intimidating acronyms rests on one or two genuinely beautiful, genuinely elementary mathematical ideas.

Contents

1 Why This White Paper Exists

Imagine you want to send a sealed letter to a friend, and you want them — and everyone else — to be able to verify that the letter genuinely came from you and was not altered along the way. In the physical world you might use a wax seal stamped with a signet ring that only you possess. Anyone can *look* at the seal and recognize it; nobody else can *produce* it. This is exactly the job of a **digital signature**: a short string of data that you alone can create but that anyone can check.

Modern digital signatures (the ones quietly protecting your software updates, your bank transfers, and the very web page you might be reading this on) almost all rely on one of two mathematical facts:

- It is hard to factor a large number back into its prime pieces.
- It is hard to "undo" multiplication on an elliptic curve (the so-called discrete logarithm problem).

A Word of Caution

In 1994 the mathematician Peter Shor discovered an algorithm that, *if run on a sufficiently large quantum computer*, would solve *both* of the problems above quickly. No such computer exists today at the required scale. But encrypted data can be recorded now and decrypted later — the so-called "harvest now, decrypt later" threat — so the cryptographic world is migrating *today* to schemes that have no known quantum (or classical) attack. These are the **post-quantum** schemes.

The phrase "no known quantum attack" is doing a lot of work. We do not have mathematical *proofs* that these new problems are hard; we have decades of very clever people failing to break them. Cryptography is, in this honest sense, an empirical science resting on a mathematical scaffold. Part of the pleasure of this subject — and a recurring theme of this document — is watching how a simple, almost childish-looking puzzle ("solve these equations", "find the nearest grid point", "find a path in a graph") becomes, when the numbers are large enough, a fortress.

1.1 What NIST did, and the cast of characters

NIST ran a multi-year public competition. By 2024 it had finalized:

- **ML-KEM** (FIPS 203) — for *encryption / key exchange*. We mention it only for context; this paper is about signatures.
- **ML-DSA** (FIPS 204) — a lattice-based signature standard.
- **SLH-DSA** (FIPS 205) — a hash-based signature standard.
- **FN-DSA** (forthcoming FIPS 206) — a second lattice-based signature, based on the FALCON submission.

Because three signature schemes from one or two mathematical families is a fragile basket — if lattices fell, two of them would fall together — NIST opened a separate "Additional Digital Signatures" track to find schemes resting on *different* mathematics. On 14 May 2026, NIST's status report (NIST IR 8610) advanced **nine** candidates to a third round of evaluation:

Scheme	Mathematical family	One-line idea
FAEST	Symmetric / MPC-in-the-head	Prove you know an AES key
HAWK	Lattices (over special rings)	Closest-point on a nice lattice
MAYO	Multivariate quadratics	"Whipped" oil-and-vinegar
MQOM	Multivariate + ZK proofs	Prove you solved a quadratic system
QR-UOV	Multivariate (structured)	Oil-and-vinegar with a field trick
SDitH	Codes + ZK proofs	Prove you can decode a codeword
SNOVA	Multivariate (matrix ring)	Compact oil-and-vinegar
SQIsign	Isogenies (elliptic curves)	A secret walk between curves
UOV	Multivariate quadratics	The classic oil-and-vinegar

This document treats all thirteen named schemes: the four families they come from are *hash-based*, *lattice-based*, *multivariate*, *code-based*, *isogeny-based*, and *symmetric/proof-based*. We will build each family's mathematics from scratch.

1.2 How to read this document

You can read linearly, or you can read Section 2 (the toolkit) and then jump to whichever scheme interests you. Throughout, three kinds of boxes appear. A green "Big Idea" box states the single sentence you should remember if you remember nothing else. A blue "Building Intuition" box offers an analogy or a tiny worked example. An amber "A Word of Caution" box flags a place where the honest story is subtler than the slogan. Equations are never decoration here; if one appears, it is worth pausing on.

2 A Shared Mathematical Toolkit

Every scheme below is assembled from a small number of elementary building blocks. This section builds them once, carefully, so that later sections can move quickly. If you are comfortable with modular arithmetic, vectors over a finite field, and the idea of a one-way function, you may skim.

2.1 Clock arithmetic: working modulo n

Look at a 12-hour clock. If it is 10 o'clock and we add 5 hours, we do not get "15 o'clock" — we get 3 o'clock. We computed $10+5=15$, then subtracted 12. This is **modular arithmetic**, and it is the single most important idea in this paper.

Definition 2.1 (Arithmetic modulo n)**.** Fix a positive integer n, the *modulus*. We say two integers a and b are *congruent modulo* n, written $a \equiv b \pmod{n}$, if n divides $a-b$. Equivalently, a and b leave the same remainder when divided by n. We do all arithmetic by computing normally and then keeping only the remainder after division by n. The set of possible remainders is $\{0,1,2,\ldots,n-1\}$, which we call $\mathbb{Z}_n$.

Example 2.1. Modulo 7: $3+6=9\equiv 2$, since $9-2=7$. Also $4\times 5=20\equiv 6$, since $20=2\cdot 7+6$. Subtraction wraps too: $2-5=-3\equiv 4$, since $-3+7=4$.

Building Intuition

Modular arithmetic is a "finite world." There are only n numbers, and they form a circle, not a line. Addition is rotation around the circle. The crucial cryptographic consequence: many operations are easy to do forwards but scramble information so thoroughly that going backwards is hopeless without a secret. A clock hand at position 3 tells you almost nothing about how many hours have elapsed since the clock started.

When the modulus is a **prime number** p, something wonderful happens: every nonzero element has a multiplicative inverse (a "reciprocal" that lives inside the finite world). For example modulo 7, the inverse of 3 is 5, because $3 \times 5 = 15 \equiv 1$. A finite world in which you can add, subtract, multiply, *and* divide (by anything nonzero) is called a **finite field**, written $\mathbb{F}_p$. Finite fields are the playground for almost every scheme in this paper.

2.2 Vectors, matrices, and systems of equations over $\mathbb{F}_p$

You have met vectors as arrows. For us a vector is just an ordered list of numbers, each living in $\mathbb{F}_p$. We add vectors slot by slot and we can scale them. A **matrix** is a rectangular grid of numbers; multiplying a matrix A by a vector $\mathbf{x}$ produces a new vector $A\mathbf{x}$ whose entries are weighted sums of the entries of $\mathbf{x}$. A **linear system** $A\mathbf{x} = \mathbf{y}$ asks: given the grid A and the target $\mathbf{y}$, which $\mathbf{x}$ works?

Building Intuition

Linear systems — equations where the unknowns appear only to the first power, with no products of unknowns — are *easy*. The method you learned in school (Gaussian elimination: add multiples of one equation to another to cancel variables) always works and is fast, even over $\mathbb{F}_p$ and even with thousands of variables. **Remember this asymmetry**: linear is easy. Almost every scheme in this paper achieves its security by hiding an easy linear problem inside a hard *nonlinear* one.

A **quadratic** equation allows products of *two* unknowns, e.g. $3x_1x_2 + x_3^2 + 5x_1 = 4$. A system of many such equations in many unknowns over $\mathbb{F}_p$ has no general fast solution method known — this single fact powers the entire "multivariate" family (MAYO, UOV, QR-UOV, SNOVA, MQOM).

2.3 One-way functions and hash functions

Definition 2.2 (One-way function, informally)**.** A function f is *one-way* if computing $f(x)$ from x is easy, but finding *any* x with $f(x) = y$, given only y, is computationally infeasible.

Building Intuition

Mixing paint is one-way. Given blue and yellow, producing green is trivial. Given a can of green, recovering the exact original shades of blue and yellow is hopeless. Every signature scheme needs some operation with this flavor: the signer uses a secret to go "backwards" through a one-way door that the rest of the world can only push forwards.

A **cryptographic hash function** H (e.g. SHA-3) takes a message of any length and outputs a fixed-length "fingerprint," say 256 bits. It has three properties we will lean on repeatedly: it is one-way; it is *collision resistant* (you cannot find two messages with the same fingerprint); and it behaves like a "random oracle" — changing the input by one bit scrambles the output completely.

Hash functions are believed to be quantum-resistant essentially as-is (a quantum computer only square-roots the difficulty, which we compensate for by doubling the output length).

2.4 The shape of every signature scheme

Strip away the mathematics and all thirteen schemes share one skeleton.

The Big Idea

A digital signature scheme is three algorithms:

- **KeyGen:** produce a *secret key* sk (kept private) and a *public key* pk (published to the world).
- **Sign:** using sk and a message m, produce a signature σ.
- **Verify:** using pk, m, and σ, output *accept* or *reject*.

Correct signatures must always verify. The security requirement: someone who knows only pk (and has even seen many valid signatures) still cannot forge a signature on a new message.

The schemes differ entirely in *which hard problem* links sk to pk, and in *which clever protocol* lets the signer prove "I know the secret" without revealing it. We now meet the families.

3 Hash-Based Signatures: SLH-DSA (FIPS 205)

We begin here because hash-based signatures need *only* the toolkit above — no number theory, no lattices, nothing exotic. If you trust that SHA-3 is one-way and collision resistant, you can build a signature scheme whose security needs nothing more. This conceptual minimalism is why many cryptographers regard SLH-DSA as the most conservative, "no surprises" option.

3.1 From a hash function to a one-time signature

Suppose you must sign a single bit, either 0 or 1, exactly once. Here is a construction due to Lamport.

- **KeyGen:** Pick two random secret strings s_0 and s_1. Publish their hashes $p_0 = H(s_0)$ and $p_1 = H(s_1)$. The pair (p_0, p_1) is the public key.
- **Sign** the bit b: reveal s_b (and nothing else).
- **Verify:** check that $H(s_b) = p_b$.

Building Intuition

Think of s_0 and s_1 as two keys locked in two glass boxes p_0, p_1 that everyone can see but nobody can open. To "sign" the bit 1, you smash open box p_1 and show the world the key inside. Anyone can confirm it fits the box. But nobody could have produced that key in advance — the box was one-way sealed. The catch is glaring: once you have smashed a box, that box is gone. This is a *one-time* signature; reusing it leaks your secrets.

To sign a whole 256-bit message digest, use 256 such box-pairs (one per bit). To remove the "one-time" restriction we need a way to authenticate *many* one-time public keys using a single small public key. That mechanism is the Merkle tree.

3.2 Merkle trees: one root to certify a million keys

Definition 3.1 (Merkle tree)**.** Take 2^h data items (here: one-time public keys). Hash each one to get the leaves. Then repeatedly hash *pairs* of adjacent values together to form a parent, halving the count at each level, until a single value remains: the **root**.

Building Intuition

Picture a single-elimination tournament bracket with 2^h players. Each match's result depends on its two players; the final champion depends, transitively, on *everyone*. The champion (root) is a 256-bit summary of all 2^h leaves. To convince a skeptic that "player number 37 was really in this exact bracket," you do not replay every match — you just exhibit the results along the single path from player 37 up to the champion (about h values, its *authentication path*). The skeptic recomputes that one path and checks it lands on the known champion.

Now the public key is just the root: 32 bytes, no matter how many one-time keys hang below it. A signature consists of the relevant one-time signature plus its authentication path proving that one-time key is genuinely under the root.

3.3 Going stateless: SLH-DSA

A naive Merkle scheme must remember which leaves it has already used (reusing a one-time key is fatal). Remembering state across millions of signatures is a deployment nightmare. SLH-DSA — the standardized form of the SPHINCS+ submission — removes the bookkeeping with two ideas: stack many Merkle trees in a *hypertree* (a tree of trees, so the total number of usable leaves is astronomically large), and select which leaf to use by *hashing the message itself* rather than by a counter. With a vast enough address space, the chance of ever colliding on the same leaf is negligible, so no memory of the past is needed. At the bottom it uses a *few-time* signature (called FORS) that tolerates the rare accidental reuse.

The Big Idea

SLH-DSA's entire security reduces to one assumption: that the underlying hash function is a good one-way, collision-resistant function. No algebra to break, no number theory to factor. Its price is comparatively large signatures (kilobytes), which is the cost of building everything out of nothing but hashing.

4 Lattice-Based Signatures: ML-DSA (FIPS 204), FN-DSA (FIPS 206), HAWK

Three of our thirteen schemes — two standards and one round-three candidate — live in the world of **lattices**. We build the geometric picture first, because once you *see* a lattice the hard problems become obvious.

4.1 What is a lattice?

Definition 4.1 (Lattice)**.** Take a few vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ (the *basis*). A **lattice** is the set of *all* points you can reach by adding integer multiples of these vectors: all combinations $a_1\mathbf{b}_1 + \cdots + a_n\mathbf{b}_n$ with each a_i a (positive, negative, or zero) integer.

Building Intuition

In two dimensions, a lattice is just an infinite grid of dots — but not necessarily the square graph-paper grid. Tilt and stretch the grid and it is still a lattice. The subtle part: the *same* grid of dots can be described by many different bases. Some bases use short, nearly perpendicular arrows (a "nice" basis: it is easy to see where the dots are). Others use long, nearly parallel, skewed arrows (a "bad" basis: the same dots, but now hard to reason about). This gap between a nice and a nasty description of the identical object is the secret trapdoor of lattice cryptography.

4.2 The two hard problems

- **Shortest Vector Problem (SVP):** among all lattice points except the origin, find the one closest to the origin. With a bad basis this is notoriously hard in high dimensions.
- **Closest Vector Problem (CVP):** given an arbitrary point in space (not necessarily on the lattice), find the lattice point nearest to it. Also hard with a bad basis.

Building Intuition

Imagine being dropped at a random spot in a vast, tilted city laid out on a skewed grid you can only describe with terrible directions ("go 1,000 blocks northeast, then 998 blocks east-northeast ..."). "Which street corner is nearest?" is the Closest Vector Problem. With a good map (a nice basis) you glance and answer. With only the awful directions, you are lost. The owner of the good map has a genuine, checkable advantage that the map-less crowd cannot replicate — exactly the asymmetry a signature needs.

In practice these schemes use *structured* lattices built from polynomials modulo $x^n + 1$ with coefficients modulo q. This algebraic structure (working in a *ring*) shrinks keys dramatically and speeds everything up, at the cost of assuming the structured versions are still hard — a trade-off the community has scrutinized heavily.

4.3 ML-DSA: signing via "Fiat–Shamir with aborts"

ML-DSA (standardized from CRYSTALS-Dilithium) does *not* directly solve CVP to sign. Instead it uses the **Fiat–Shamir** paradigm, a recurring hero of this paper, so we explain it once in general.

The Big Idea

The identification-to-signature trick (Fiat–Shamir). Suppose you have an interactive way to prove "I know a secret" to a live challenger: you send a random-looking *commitment*, they send a random *challenge*, you send a *response* that only the secret-holder can compute, and they check it. To turn this into a non-interactive *signature*, replace the human challenger by a hash function: compute the challenge as $H(\text{commitment} \,\|\, \text{message})$. Now the message is welded into the proof, the challenger is removed, and the transcript *is* the signature.

In ML-DSA the secret is a matrix of small-coefficient polynomials $\mathbf{s}$; the public key is $\mathbf{t} = A\mathbf{s}$ for a public A (recovering $\mathbf{s}$ from $\mathbf{t}$ is the hard lattice problem). To sign: pick a small random $\mathbf{y}$, commit to $A\mathbf{y}$, derive the challenge c by hashing it with the message, and respond with $\mathbf{z} = \mathbf{y} + c\,\mathbf{s}$. The deep subtlety: $\mathbf{z}$ must not leak $\mathbf{s}$. If $\mathbf{z}$ would stray into a range that reveals information, the signer

simply *aborts* and retries with fresh randomness — the "rejection sampling" or "aborts" technique. After a few tries a safe signature emerges, provably independent of the secret.

4.4 FN-DSA: signing by genuinely solving CVP, carefully

FN-DSA (FIPS 206, from the FALCON submission) takes the other route: it really does use a trapdoor to solve a closest-vector-type problem. Its secret key is a *nice* basis of a structured lattice (built using number-theoretic objects called NTRU lattices); its public key is a *bad* basis of the same lattice. To sign a message, FN-DSA hashes it to a target point in space and uses the nice secret basis to find a lattice point very close to that target. The pair (target, nearby lattice point) is the signature; verifying just checks the point is on the public lattice and close enough.

A Word of Caution

The danger is that each signature is a lattice point "leaning toward" the secret nice basis; a careless sampler would, over many signatures, trace out the secret basis like footprints in snow. FN-DSA prevents this with *discrete Gaussian sampling*: it deliberately adds carefully shaped randomness so the output looks like a perfectly round blur centered on the target, leaking nothing about the *shape* of the secret basis. This requires high-precision floating-point arithmetic and is the reason FN-DSA is mathematically elegant but implementation-delicate. The reward: very compact signatures.

4.5 HAWK: the same geometry, but over a friendlier lattice

HAWK is a round-three candidate that keeps FN-DSA's "solve a closest-vector problem with a trapdoor" philosophy but changes the lattice. It works over a specific, highly symmetric lattice (a module over a cyclotomic ring, with the quadratic form built from that ring's natural geometry). The point of the change: the relevant short-vector / closest-vector operations become simpler integer arithmetic, sidestepping FALCON's notoriously tricky floating-point Gaussian sampler. HAWK's pitch to NIST is essentially "FALCON-like sizes and speed, but far easier to implement correctly and to protect against side-channel attacks," at the cost of resting on a somewhat more specialized lattice assumption that the community is still stress-testing.

The Big Idea

All three lattice schemes monetize the *same* asymmetry: a nice basis makes nearest-point questions trivial; a bad basis (the public key) makes them intractable. ML-DSA proves knowledge of the trapdoor indirectly (Fiat–Shamir with aborts); FN-DSA and HAWK use the trapdoor directly to find close points, differing only in which lattice — and hence which sampling machinery — they employ.

5 Multivariate Signatures: UOV, QR-UOV, MAYO, SNOVA, MQOM

Five of the nine round-three candidates rest on one elementary-sounding fact: **solving a system of many quadratic equations in many unknowns over a finite field is hard.** This is the "$\mathcal{MQ}$ problem." We meet the classic scheme first, then its four relatives.

5.1 The hard problem you can write on a napkin

Consider equations like

$$\begin{aligned} 3x_1^2 + x_1x_2 + 5x_2x_3 + 2x_1 &= 7, \\ x_1x_3 + 4x_2^2 + x_3 &= 1, \quad \dots \end{aligned} \pmod{p}$$

Given the equations, find $(x_1, x_2, \dots)$ satisfying all of them. If they were *linear* (no squared or product terms) you could solve instantly by elimination. The quadratic terms wreck elimination: substituting one equation into another only makes the degree explode. For random quadratic systems over a finite field, no efficient algorithm is known — classical or quantum.

Building Intuition

A linear system is a set of straight cuts through space; they meet in a single clean point you can locate by bookkeeping. A quadratic system is a tangle of curved surfaces. Asking where dozens of high-dimensional curved surfaces all intersect, in a finite field where you cannot even "follow the curve" continuously, is the combinatorial equivalent of finding a needle in a hay-galaxy. *Yet checking a proposed solution is instant* — plug in and see if every equation holds. Easy to check, hard to find: the signature-shaped asymmetry once more.

5.2 UOV: the classic Oil and Vinegar idea

How can the legitimate signer solve a quadratic system that is supposed to be hard? By building the system with a hidden structure that collapses the quadratics into *linear* equations — but only if you know the secret.

The Big Idea

The Oil-and-Vinegar trick. Split the variables into two groups: "vinegar" variables $v_1, \dots, v_m$ and "oil" variables $o_1, \dots, o_n$. Design the secret quadratic equations so that oil variables *never multiply other oil variables* — only vinegar×vinegar and oil×vinegar terms appear. Now here is the magic: if you randomly fix all the vinegar variables to concrete numbers, every remaining term is at most *linear* in the oil variables. The fearsome quadratic system collapses into an easy linear system, which you solve by elimination.

Building Intuition

Oil and vinegar do not mix — shake a vinaigrette and the oil droplets touch vinegar but never bond oil-to-oil. Same here: oil variables interact with vinegar, never with each other. "Pouring in" fixed vinegar values is like adding the vinegar to the bottle: the oil layer (the oil variables) is left obeying only *straight-line* rules, which you can solve in your head.

To turn this into a signature: the *secret key* is this nicely structured map together with a secret invertible linear "scrambler" T. The *public key* is the composition — the structured map smeared by T — which to an outsider looks like a generic, structureless hard quadratic system. To sign a message m: hash it to a target vector $\mathbf{y}$; using the secret structure, fix random vinegar values, solve the resulting linear system for the oil variables to find a preimage $\mathbf{x}$ with (structured map)$(\mathbf{x}) = \mathbf{y}$; unscramble with T^{-1}. The signature is $\mathbf{x}$. Verification just evaluates the public quadratic map at $\mathbf{x}$ and checks it equals $\mathbf{y}$ — fast, and needing no secret.

The Big Idea

UOV is, in NIST's words, the scheme that "has survived decades of intense public scrutiny." Its security rests on the plain $\mathcal{MQ}$ problem plus the difficulty of spotting the hidden oil-and-vinegar structure inside the scrambled public key. Its weakness is large public keys; the next three schemes are all attempts to keep UOV's battle-tested core while shrinking it.

5.3 QR-UOV: shrinking the key with a field trick

QR-UOV ("Quotient-Ring UOV") keeps the oil-and-vinegar engine untouched but represents the big public matrices using elements of a larger structured ring (a quotient ring of polynomials). Because one ring element compactly encodes a whole block of field elements that would otherwise be written out individually, the public key shrinks substantially. The bet: this extra algebraic structure does not hand attackers a new shortcut. It is UOV wearing a smaller, denser coat.

5.4 MAYO: "whipping" oil and vinegar

MAYO starts from a deliberately *broken* UOV: it uses an oil space that is too small to sign with directly (which, helpfully, makes the public key small). It then "whips" the map — combining several copies of the small public map with random linear mixing to temporarily enlarge the effective oil space *only during signing*. Like whipping mayonnaise, a small amount of oil is beaten up into a large, usable volume. The result keeps a tiny public key while restoring the ability to sign, with fast verification and small signatures.

5.5 SNOVA: oil and vinegar over a matrix ring

SNOVA replaces the individual field elements of UOV with small *matrices* (it works over a non-commutative matrix ring). Each variable now carries more information per symbol, so equivalent security is reached with far fewer variables and a much smaller public key. The price is the extra structure of the matrix ring, whose security implications the community is actively probing — but the headline is "UOV-grade security at exceptionally small sizes."

5.6 MQOM: proving you solved a quadratic system, instead of using a trapdoor

MQOM ("$\mathcal{MQ}$ on my mind") takes a strikingly different stance. The other four hide a trapdoor inside the equations. MQOM uses a *plain, random* quadratic system with *no* hidden structure. The secret key is simply a solution $\mathbf{x}$ to a public random system; finding $\mathbf{x}$ from the system is the full, unweakened $\mathcal{MQ}$ problem. To sign, the signer produces a *zero-knowledge proof* that "I know a solution to this system" — compressed into a signature via Fiat–Shamir.

The Big Idea

The philosophical split inside the multivariate family: UOV / QR-UOV / MAYO / SNOVA weaken the problem just enough to plant a usable trapdoor, then hide the weakening. MQOM keeps the problem at full hardness and instead invents a clever *argument* that convinces a verifier you solved it, without revealing how. To understand that "argument," we need the MPC-in-the-head idea — which is exactly the engine behind our next family.

6 Proving Without Revealing: Zero-Knowledge and MPC-in-the-Head

Three remaining schemes — SDitH, FAEST, and (as just noted) MQOM — share a single beautiful engine. Before meeting them we build that engine, because it is one of the most delightful ideas in modern cryptography and needs no prerequisites beyond "a hash function exists."

6.1 Zero-knowledge proofs: the cave

Building Intuition

The Ali Baba cave. A ring-shaped cave has a magic door at the back that opens only with a secret word. Peggy claims she knows the word. She walks in and takes the left or right passage — out of sight. Victor then shouts "come out the left side!" If Peggy truly knows the word she can always comply (passing through the door if needed). If she is bluffing she only had a 50% chance of having guessed Victor's call. Repeat twenty times: a bluffer survives with probability 2^{-20}, about one in a million. Victor ends *certain* Peggy knows the word — yet has learned nothing *about the word itself.* That is a **zero-knowledge proof**: convince someone a statement is true while revealing nothing beyond its truth.

For signatures the "secret word" is the secret key, the "statement" is "this public key was generated honestly," and Fiat–Shamir (Section 4.3) turns the interactive cave game into a non-interactive signature by hashing the message to generate Victor's challenges.

6.2 MPC-in-the-head: a one-person committee

How do you build the cave game for an arbitrary statement like "I know $\mathbf{x}$ with $F(\mathbf{x}) = \mathbf{y}$"? The ingenious answer is **MPC-in-the-head**.

The Big Idea

Secret sharing. Split a secret number s into, say, three "shares" s_1, s_2, s_3 that are random subject only to $s_1 + s_2 + s_3 = s$. Any one or two shares look completely random and reveal nothing; only all three together reconstruct s. **MPC-in-the-head:** the prover imagines a committee of virtual parties, gives each one share of the secret, and has them jointly run a computation (a "multiparty computation," MPC) that verifies $F(\mathbf{x}) = \mathbf{y}$ *without any party seeing the whole secret.* The prover plays *all* parties in their own head, commits (via hashes) to every party's transcript, and the verifier then demands to inspect a random subset of the parties.

Building Intuition

Why is this convincing yet zero-knowledge? Inspecting a strict subset of parties shows the prover ran honest, consistent participants — yet, exactly like seeing only two of three shares, it never reveals the secret. If the prover had cheated, the cheating must live in some party, and the random inspection catches it with good probability. Hash everything and apply Fiat–Shamir, and the whole imagined committee meeting collapses into one static signature string.

This engine is profound because it manufactures a signature from *any* hard one-way function F

whatsoever — you only need to be able to *check* $F(\mathbf{x}) = \mathbf{y}$, never to invert it with a trapdoor. The remaining schemes simply choose different F.

7 Code-Based Signatures: SDitH

SDitH ("Syndrome Decoding in the Head") plugs the hardest problem of *coding theory* into the MPC-in-the-head engine.

7.1 Error-correcting codes and the syndrome

When data is sent over a noisy channel, we add redundancy so a few flipped bits can be detected and corrected. A *linear code* does this with a public matrix H: a string $\mathbf{c}$ is a valid codeword exactly when $H\mathbf{c} = \mathbf{0}$. If noise flips some bits, turning $\mathbf{c}$ into $\mathbf{c} + \mathbf{e}$ for an unknown sparse *error vector* $\mathbf{e}$, then $H(\mathbf{c} + \mathbf{e}) = H\mathbf{e}$. This quantity $\mathbf{s} = H\mathbf{e}$ is the **syndrome**: it depends only on the error, not the message.

The Big Idea

The Syndrome Decoding Problem. Given the public matrix H and a syndrome $\mathbf{s}$, find a *low-weight* error vector $\mathbf{e}$ (few nonzero entries) with $H\mathbf{e} = \mathbf{s}$. Computing $\mathbf{s}$ from $\mathbf{e}$ is one matrix multiply — trivial. Recovering the sparse $\mathbf{e}$ from $\mathbf{s}$ is a classic problem proven NP-hard in general and unbroken after fifty years of attack, classical and quantum.

Building Intuition

The asymmetry, once more in plain words: scrambling a clean codeword by sprinkling a few errors and reporting the syndrome is effortless. Being handed only the syndrome and asked "which tiny set of bit-flips produced this?" — with astronomically many candidate error patterns and no structure to exploit — is the needle in the hay-galaxy.

7.2 Assembling SDitH

The recipe is now mechanical given our toolkit. **KeyGen:** pick a secret low-weight error $\mathbf{e}$ (the secret key); publish H and $\mathbf{s} = H\mathbf{e}$ (the public key). **Sign:** produce an MPC-in-the-head zero-knowledge proof of "I know a low-weight $\mathbf{e}$ with $H\mathbf{e} = \mathbf{s}$," bound to the message via Fiat–Shamir. **Verify:** replay the hashed challenges and check the revealed parties are consistent. SDitH's appeal is that its hardness assumption — plain syndrome decoding of random codes — is among the oldest and most trusted in all of cryptography.

8 Symmetric-Key Signatures: FAEST

FAEST pushes the MPC-in-the-head philosophy to its minimalist conclusion: it uses no number theory, no lattices, no codes — only the **Advanced Encryption Standard (AES)**, the same block cipher already trusted to protect essentially all encrypted data on Earth.

8.1 A block cipher as a one-way function

AES is a function $\mathrm{AES}_k(b)$ that scrambles a block b using a key k. Fix a public input block b and a public output $c = \mathrm{AES}_k(b)$. Define $F(k) = \mathrm{AES}_k(b)$. Given k, computing c is a fast standard

operation. Given only (b, c), finding the key k is exactly "break AES by brute force" — believed infeasible, and the bedrock assumption of modern symmetric cryptography. So F is a one-way function we already trust enormously.

8.2 Assembling FAEST

The Big Idea

KeyGen: pick a random AES key k (secret key); publish a block b and $c = \mathrm{AES}_k(b)$ (public key). **Sign:** produce an MPC-in-the-head zero-knowledge proof of "I know the key k such that $\mathrm{AES}_k(b) = c$," welded to the message by Fiat–Shamir. **Verify:** re-derive the hashed challenges and check the opened virtual parties. (FAEST uses an optimized variant of the engine called VOLE-in-the-head, but the conceptual skeleton is identical.)

The Big Idea

FAEST's security rests *only* on the secrecy of AES — arguably the single most studied, most trusted primitive in all of cryptography. There is no new algebraic structure for an attacker to exploit, because there is no new structure at all. That conservatism is exactly its selling point; the cost is relatively large signatures, the recurring tax of building signatures from nothing but symmetric primitives.

9 Isogeny-Based Signatures: SQIsign

We save the most geometrically exotic — and most compact — for last. SQIsign produces by far the smallest public keys and signatures of any post-quantum signature, at the cost of the deepest mathematics in this paper. We will not reach the bottom of it, but we can honestly see its shape.

9.1 Elliptic curves, gently

An **elliptic curve** over $\mathbb{F}_p$ is the set of solutions (x, y) to an equation like $y^2 = x^3 + ax + b$, together with a special "point at infinity." The remarkable fact, discovered in the 19th century, is that you can *add* two points on the curve to get a third point on the curve, via an explicit geometric rule, making the points of the curve into an algebraic system you can compute in. These curves already underlie much of today's (pre-quantum) cryptography.

9.2 Isogenies: structure-preserving maps between curves

Definition 9.1 (Isogeny, informally)**.** An **isogeny** is a map from one elliptic curve to another that respects the addition rule — it sends the sum of two points to the sum of their images. Each elliptic curve thus becomes a *node*, and each isogeny a *directed edge*, in a vast graph of curves. These "isogeny graphs" are extraordinarily well-mixed (technically, expander graphs): from any curve, a short random walk along isogenies lands you at an essentially unpredictable curve.

Building Intuition

Picture an immense, intricate subway map where each station is an elliptic curve and each line is an isogeny. **KeyGen:** start at a fixed public station, take a long secret sequence of train

transfers, and publish only your *destination* station. The hard problem: given the start and end stations, reconstruct *a* valid route between them. The graph is so huge and so well-mixed that an outsider faces an astronomical maze, while you simply remember the route you took. This "find a secret path" problem (the isogeny path problem) has, so far, resisted even quantum attackers.

9.3 From a secret path to a signature

SQIsign uses the secret isogeny path as the trapdoor in a Fiat–Shamir identification scheme. Roughly: the signer commits to a random auxiliary curve; the message-derived hash challenge selects a constraint; the signer answers by producing an isogeny that simultaneously connects the right curves *and* implicitly demonstrates knowledge of the secret path — without revealing it. The machinery that makes "produce the connecting isogeny on demand" possible relies on a deep correspondence (the Deuring correspondence) between isogenies of curves and arithmetic in associated algebraic objects called quaternion orders. This is genuine third-year-and-beyond mathematics; the honest summary for a first-year reader is the subway picture plus the slogan below.

The Big Idea

SQIsign trades mathematical accessibility for spectacular compactness: its keys and signatures are only a few hundred bytes — the smallest in the entire post-quantum signature landscape, comparable to today's pre-quantum signatures. The cost is twofold: its security rests on the youngest and least battle-tested assumptions here, and signing is computationally heavy. It is the high-risk, high-reward entry.

10 Standing Back: The Whole Landscape on One Page

You have now met thirteen schemes. The deeper lesson is that they are not thirteen unrelated tricks; they are a handful of *ideas* recombined. Three patterns recur:

1. **Trapdoor inversion.** A one-way map with a secret shortcut for going backwards. (FN-DSA, HAWK, UOV, QR-UOV, MAYO, SNOVA, SQIsign.)
2. **Prove-knowledge-without-revealing.** Keep a hard problem at full strength; sign by proving you know a solution. (SDitH, FAEST, MQOM, and ML-DSA's identification core.)
3. **Build from hashing alone.** Trust nothing but a hash function. (SLH-DSA.)

And a single conversion — **Fiat–Shamir**, "replace the live challenger by a hash of the message" — turns almost every one of them from an interactive game into a static signature. If you internalize Fiat–Shamir and the easy/hard asymmetry, you understand the skeleton of the entire field.

Scheme	Hard problem	How signing works	Trade-off
SLH-DSA	Hash is one-way / collision-resistant	Reveal hash preimages, certified by a Merkle hypertree	Most conservative; large signatures
ML-DSA	Structured lattice (Module-LWE)	Fiat–Shamir identification with aborts	Balanced standard; moderate sizes
FN-DSA	Structured lattice (NTRU)	Trapdoor closest-vector + Gaussian sampling	Compact; delicate floating-point
HAWK	Module lattice, special form	Trapdoor closest-vector, integer arithmetic	Compact & implementable; newer assumption
UOV	$\mathcal{MQ}$ + hidden oil/vinegar	Fix vinegar, solve linear system for oil	Battle-tested; large public key
QR-UOV	$\mathcal{MQ}$ over a quotient ring	UOV engine, compact ring representation	Smaller key; extra structure
MAYO	$\mathcal{MQ}$ (small oil space)	"Whip up" the oil space during signing	Tiny key, small signatures
SNOVA	$\mathcal{MQ}$ over a matrix ring	UOV engine over matrices	Exceptionally small; new structure
MQOM	Plain $\mathcal{MQ}$ (no trapdoor)	ZK proof of knowing a solution	Conservative assumption; bigger signatures
SDitH	Syndrome decoding of random codes	MPC-in-the-head ZK proof	Very old trusted problem; bigger signatures
FAEST	AES is one-way	MPC-in-the-head proof of knowing the key	No new assumptions; bigger signatures
SQIsign	Isogeny path problem	Fiat–Shamir using a secret isogeny path	Smallest sizes; youngest math, slow

A Word of Caution

No row is strictly best. Cryptographic standardization is portfolio management against an uncertain future: NIST keeps schemes from *different* mathematical families so that a breakthrough against one (say, all lattices) cannot collapse the whole edifice. "Diversity of hard problems" is itself the security goal — which is the entire reason the nine additional candidates exist alongside the finalized standards.

11 What You Should Take Away

If you are a first-year student wondering which parts of your degree this connected to: *all of them, eventually.* Modular arithmetic and finite fields are a first course in abstract algebra. Lattices and closest-vector problems are linear algebra plus a pinch of geometry of numbers. The oil-and-vinegar trick is nothing more than the fact, which you will prove rigorously in linear algebra, that linear systems are solvable by elimination while quadratic ones are not. Elliptic curves and isogenies are a thread running from a second algebra course up into research mathematics. Zero-knowledge proofs are where mathematics meets the theory of computation.

The single most important habit to carry away is to always ask, of any cryptographic construction: *where is the easy/hard asymmetry, and what secret converts the hard direction back into the easy one?* Every scheme in this paper is a different, ingenious answer to that one question. The mathematics

that makes them work is, at its foundation, mathematics you are already beginning to learn. The only thing separating "a fun puzzle" from "a fortress guarding the world's information" is how large you are willing to make the numbers — and how honestly the community has tried, and failed, to break in.

The numbers do not get harder. We just get braver about how big to make them.

Sources and further reading. The status of the schemes described here follows NIST's official Post-Quantum Cryptography project pages and the finalized standards FIPS 203 (ML-KEM), FIPS 204 (ML-DSA), FIPS 205 (SLH-DSA), the forthcoming FIPS 206 (FN-DSA), and NIST Internal Report 8610, *Status Report on the Second Round of the Additional Digital Signature Schemes* (finalized May 2026), which advanced FAEST, HAWK, MAYO, MQOM, QR-UOV, SDitH, SNOVA, SQIsign, and UOV to the third round. Mathematical descriptions are deliberately simplified for a first-year audience; each scheme's official specification is the authoritative technical reference.