

Learning With Errors, by Hand

The mathematics MedLattice is named after — noisy equations worked by hand, the lattice underneath them, one bit locked and unlocked, the road from a toy to ML-KEM-1024, and the assumption put on trial against everything 2026 threw at it.

Dr Neal Aggarwal

Medicine · Surgery · Engineering · Information Technology · Artificial Intelligence

19 OCTOBER 2026

My patient record is called MedLattice, and across four essays I have spent something like twenty-five thousand words on its cryptography. The mathematics it is actually named after — the lattice problem that ML-KEM rests on — got one paragraph. I took Shor's algorithm apart by hand and then asked you to take its replacement on trust. This post repays that debt. It builds Learning With Errors from nothing: clock arithmetic, two simultaneous equations you can solve on a napkin, the same two equations with a whisper of noise added, and the moment the napkin method collapses. It shows why the problem is secretly a question about finding the nearest point on a grid in hundreds of dimensions, locks and unlocks a single bit by hand with numbers you can check, rebuilds the whole thing in a spreadsheet and in forty lines of Python, and then walks from that toy to the real ML-KEM-1024 that protects every MedLattice record — rings, modules, the number-theoretic transform, the decryption-failure budget and the transform that makes a fragile scheme safe to use. It then puts the assumption on trial against everything 2026 has thrown at it: a claimed quantum break of Module-LWE posted in May and dismantled within weeks, a provable new analysis that shaves up to 68 bits off one classical attack estimate, quantum resource estimates for RSA falling towards a hundred thousand qubits, and a side-channel paper presented eight days ago. The mathematics held. The honest conclusion is about where it did not need to.

Contents

Download this essay as a PDF	5
Part 1 — Clock arithmetic in five minutes	7
Part 2 — Two equations you can solve	8
Part 3 — The same equations, slightly wrong	9
The brute-force rescue — and why it only works on toys	10
Part 4 — The grid underneath	11
A good description and a bad description of the same grid	11
What the attackers actually do: lattice reduction	13
Part 5 — Locking one bit by hand	14
The key	14
Encrypting	14
Decrypting	15
Part 6 — Check it in a spreadsheet	16
Part 7 — Check it in Python	18
Part 8 — From the toy to ML-KEM	20
Idea one: let one equation do the work of hundreds	20
Idea two: a little flexibility — modules	20
Idea three: why 3,329 — the number-theoretic transform	21
Idea four: compression, and the transform that makes it safe	22
Part 9 — Why Shor’s algorithm gets no grip	23
Part 10 — The assumption on trial: 2026	25
The precedent: April 2024	25
The claim: May 2026	25
The quiet result: September 2026	26
The other clock: quantum hardware for RSA	27
The chip that talks: side channels	27
Part 11 — What this means for MedLattice	29
What this does not do	31
Where this leaves the argument	31

Further reading and references	33
A short glossary	34

Learning With Errors, by Hand

The mathematics MedLattice is named after

Equations anyone can solve, made unsolvable by a whisper of noise — and why a quantum computer still finds nothing to grip.

THE PROBLEM · NOISY EQUATIONS

Easy — until you add a whisper

$3x + 7y = 7, 10x + 7y = 8 \pmod{17}$ gives $(5, 11)$.
Nudge each answer by ± 1 : the same method gives $y = 16$.
Dividing by 7 means multiplying by 5. Errors explode.

THE GEOMETRY · A LATTICE

Find the nearest grid point

Short arrows round to the right point.
Long arrows, same grid, round 6.5 units away.
Reducing a basis costs exponentially in dimension.

THE LOCK - ONE BIT

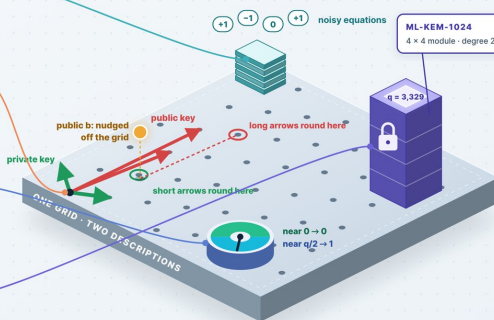
Shift it half-way round the clock

Sum a secret handful of public equations.
Add $q/2$ to send a 1; add nothing to send a 0.
The key-holder strips all but noise: which half?

THE REAL THING • ML-KEM-1024

Polynomials, modules, a fast transform

Degree-256 polynomials stand in for whole matrices.
1,568-byte keys; the NTT is why $q = 3,329$.
A re-encryption check rejects tampered ciphertexts.



THE ASSUMPTION ON TRIAL, 2026

- 1 **A claimed quantum break**
Luo, May: examined, and found broken itself
- 2 **A classical estimate moved**
Up to 68 bits off one dual attack — still far out of reach
- 3 **RSA's clock ticks faster**
~100,000 qubits (preprint): threatens X448, not lattices
- 4 **The chip, not the maths**
KyberSlash, SEAL3G: implementations leak keys

MEDLATTICE'S ANSWER

what the specification already commits to

HYBRID

- ✓ ML-KEM-1024 + X448: secure if either holds

AGILITY

- ✓ Retire a suite on-chain; re-wrap keys in hours

HISTORY

- ✓ SLH-DSA notarises it — no lattices at all

OPEN ITEM

- Lattice monoculture: HQC or FrodoKEM next



THE NOISE IS THE LOCK.

The key-holder can ignore it; nobody else can remove it. Shor finds periods — noise leaves none.

drnealaggarwal.info · 2026

*I write here in a personal capacity. This post is the third companion on MedLattice's cryptography, after *Not Shown Is Not Locked*, which describes the record, and *The Spinning Coin and the Stolen Key*, which explains what Shor's algorithm breaks. That essay explained the lock that fails. This one explains the lock that holds. Every number in the worked examples was computed by hand and checked in code; if your arithmetic disagrees with mine, one of us has made a mistake and it is worth finding out which.*

Download this essay as a PDF

The whole argument as a printable essay — Learning With Errors worked by hand on numbers you can check with a pencil, the lattice geometry underneath it, one bit locked and unlocked, the spreadsheet and the Python, the walk from that toy to ML-KEM-1024, and what 2026 threw at the assumption.

Aggarwal N. *Learning With Errors, by Hand: The Mathematics Med-Lattice Is Named After*. 2026.

Download PDF (580 KB, 35 pages, 3 figures)

Listen to this post (71 minutes)

Listen online at drnealaggarwal.info

Imagine four blood results on a single patient, each from an analyser that is off by a small amount nobody has written down. One reads a fraction high, one a fraction low, one dead on, one a fraction high again. Each result is a known combination of two things you cannot measure directly, and you want those two things back.

Any clinician knows the instinct. You solve for the two unknowns from the first two results, then check the answer against the other two. If the instruments were perfect, the check would pass exactly. Because they are not, you expect it to be slightly off, and slightly off is fine.

Now suppose that the arithmetic involved has a peculiar property: **the smallest error in any one result, pushed through the algebra of solving, comes out the other**

side not slightly off but completely wrong — as wrong as a random guess. The check against the remaining results fails by miles, not by a fraction. And the only way to recover the true values is to try every possibility in turn, which at realistic sizes would take longer than the universe has existed.

That is Learning With Errors. It is the problem underneath ML-KEM, the key-establishment algorithm NIST standardised in August 2024 as FIPS 203, and therefore the problem underneath every patient record I have designed MedLattice to protect. The record is named after the geometry this problem secretly is — a lattice — and in four essays I have explained it in exactly one paragraph.

In [The Spinning Coin and the Stolen Key](#) I took Shor’s algorithm apart by hand, on numbers small enough to check with a pencil, because I did not think anybody should have to take on trust a claim that the cryptography protecting their medical record is about to fail. Then, at the end, I asked you to take on trust the claim that its replacement will not. One paragraph: a grid in hundreds of dimensions, a point nudged off it, nothing periodic for a quantum computer to grip.

That paragraph was accurate, but it was an assertion, and this post replaces it with an argument. By the end you will have solved a Learning With Errors problem by hand, watched the obvious method fail, locked and unlocked a bit of information with a public key you built yourself, checked all of it in a spreadsheet and in Python, and followed the toy all the way to the real parameters inside MedLattice. Then I want to put the assumption on trial, because 2026 has been a genuinely interesting year for it — including one paper that claimed to break it outright.

What this post explains, in one page

The problem. Solving simultaneous equations is easy — schoolchildren do it. Solving simultaneous equations in which every right-hand side has been nudged by a small random amount is, in clock arithmetic and at large enough sizes, believed to be hard for every computer, classical or quantum. That is Learning With Errors (LWE).

Why the noise matters. Solving equations means dividing, and in clock arithmetic dividing scrambles small numbers into large ones. A nudge of 1 becomes an error of 5, then of 12, then indistinguishable from noise. The method that works perfectly on clean equations gives garbage on noisy ones.

The geometry. Every LWE problem is secretly a question of finding the nearest point on a regular grid — a lattice — to a point that has been nudged slightly off it. With a good description of the grid, that is easy. With a bad one, which is what the public key is, it is hard, and the best algorithms known get exponentially slower as the number of dimensions grows.

The encryption. To send one bit, add up a secret random handful of the public equations and shift the total halfway round the clock for a 1. The holder of the secret can strip out everything except the accumulated noise and see which half of the clock the answer landed on. Nobody else can.

The real thing. ML-KEM replaces numbers with polynomials so that one equation does the work of hundreds, uses a fast transform to multiply them, compresses the result, and wraps the whole scheme in a re-encryption check that turns a fragile primitive into one that can be safely exposed to an attacker.

Why a quantum computer does not help much. Shor’s algorithm finds hidden periods. Lattice problems have no hidden period of the kind it needs. The best quantum attacks known give a modest speed-up of the same order as Grover’s against AES, not the collapse Shor causes for RSA.

The 2026 trial. One claimed quantum break, examined and found to be broken itself. One classical estimate lowered by up to 68 bits, leaving it astronomically out of reach. Quantum hardware estimates falling for RSA, which matters for MedLattice’s classical half but not its lattice half. And repeated evidence that the real danger is the chip, not the mathematics.

Part 1 — Clock arithmetic in five minutes

Everything in this post happens on a clock.

An ordinary clock has twelve positions. Four hours after ten o’clock is two o’clock, not fourteen, because after twelve you start again. Mathematicians write that as $10 + 4 = 2 \pmod{12}$ and call it arithmetic *modulo* 12. The number you wrap around at is the *modulus*.

I am going to use a clock with **17** positions, numbered 0 to 16, because 17 is prime and that makes division behave. On this clock:

- $9 + 12 = 21$, and $21 - 17 = 4$, so **$9 + 12 = 4$** .
- $3 - 5 = -2$, and $-2 + 17 = 15$, so **$3 - 5 = 15$** . Going backwards past zero wraps round to the top.
- $6 \times 5 = 30$, and $30 - 17 = 13$, so **$6 \times 5 = 13$** .

Multiplication just means “do the ordinary multiplication, then take the remainder after dividing by 17”. Nothing more exotic than that.

Division is the only interesting part. On an ordinary number line, dividing by 7 means multiplying by one-seventh. There are no fractions on a clock, so instead we ask: *which number, multiplied by 7, gives 1?* Try them: $7 \times 5 = 35$, and $35 - 34 = 1$. So on the 17-clock, **5 plays the role of one-seventh**. Dividing by 7 is multiplying by 5. Mathematicians call 5 the *inverse* of 7.

Every non-zero number on a prime clock has exactly one inverse. Here are the ones we will need:

Number	2	3	5	7	10
Its inverse on the 17-clock	9	6	7	5	12

Check any of them: $3 \times 6 = 18 = 1 \pmod{17}$. $10 \times 12 = 120 = 7 \times 17 + 1 = 1 \pmod{17}$. Good.

Hold on to one fact about this table, because the whole post turns on it: **the inverse of a small number is usually a large-ish, unremarkable number**. Dividing by 7 means multiplying by 5. Dividing by 3 means multiplying by 6. Dividing on a clock does not shrink things the way dividing on a number line does. It scrambles them.

Part 2 — Two equations you can solve

Here is a secret: two numbers on the 17-clock. Call them x and y . I will tell you that

$$\begin{aligned} 3x + 7y &= 7 \pmod{17} \\ 10x + 7y &= 8 \pmod{17} \end{aligned}$$

and ask you to find them. This is exactly the simultaneous-equations exercise from school, carried out on a clock.

Step one: eliminate y . Both equations contain $7y$, so subtract the first from the second:

$$(10 - 3)x = 8 - 7, \text{ which is } 7x = 1.$$

Step two: divide by 7. On the 17-clock that means multiplying by 5:

$$x = 1 \times 5 = 5.$$

Step three: substitute back. Put $x = 5$ into the first equation: $3 \times 5 + 7y = 7$, so $15 + 7y = 7$, so $7y = 7 - 15 = -8$, which wraps round to **9**.

Step four: divide by 7 again. $y = 9 \times 5 = 45$, and $45 - 34 = 11$.

So the secret is $(x, y) = (5, 11)$. Check it in the second equation: $10 \times 5 + 7 \times 11 = 50 + 77 = 127$, and $127 - 7 \times 17 = 127 - 119 = 8$. Correct.

That is Gaussian elimination, the same method a computer uses to solve a system of a thousand equations in a thousand unknowns in a fraction of a second. It is fast, it is mechanical, and on clean equations it never fails. If MedLattice's security rested on equations like these, a laptop would read every record before you finished your coffee.

Part 3 — The same equations, slightly wrong

Now I do one small thing. Before I publish the right-hand sides, I nudge each one by a tiny random amount: +1, 0 or −1, chosen secretly, different for each equation. I also give you two more equations, nudged the same way, so you have something to check your answer against.

Here is what you receive. Same secret, (5, 11), underneath.

Equation	Coefficients	Published right-hand side	(True value)	(Hidden nudge)
1	$3x + 7y$	8	7	+1
2	$10x + 7y$	9	8	+1
3	$16x + 12y$	7	8	−1
4	$2x + 2y$	16	15	+1

You only see the first three columns. The last two are there so you can follow what happens.

Run exactly the same method as before on the first two equations.

Eliminate y: subtract equation 1 from equation 2. $7x = 9 - 8 = 1$. So $x = 5$. Correct — but only by luck: both nudges were +1, so they cancelled when I subtracted. Had they been +1 and −1, x would already be wrong.

Substitute back: $3 \times 5 + 7y = 8$, so $7y = 8 - 15 = -7$, which wraps to 10. Divide by 7, which means multiply by 5: $y = 50$, and $50 - 34 = 16$.

The true y is 11. You got 16. **A nudge of 1 went into the division and came out as an error of 5**, because dividing by 7 means multiplying by 5, and multiplying a nudge of 1 by 5 turns a whisper into a shout. That is the property from Part 1, doing its damage.

Now check (5, 16) against the two remaining equations, the way the clinician checks against the other two blood results:

- Equation 3: $16 \times 5 + 12 \times 16 = 80 + 192 = 272 = 16 \times 17 + 0$, so it predicts **0**. Published value: 7. Off by **7**.
- Equation 4: $2 \times 5 + 2 \times 16 = 42 = 2 \times 17 + 8$, so it predicts **8**. Published value: 16. Off by **8**.

The nudges were never bigger than 1. The check is off by 7 and 8 on a clock that only goes up to 16 — which is to say, about as wrong as it is possible to be. A random guess would do as well.

The correct answer, (5, 11), by contrast, fits all four equations with every discrepancy inside ± 1 . That is how you would recognise it if you found it. The problem is finding it.

Why the obvious repair does not work

An experienced data analyst's instinct is to use all four equations at once and fit by least squares — the method that handles noisy measurements everywhere from regression to radiotherapy planning. On a number line, that works beautifully, because errors average out. On a clock it does not, because there is no notion of “average” that survives the wrap-around: is 16 close to 0 or far from it? It is one step away going forwards and sixteen going backwards, and the algebra of least squares has no way to know which. The clock destroys the geometry that every averaging method depends on.

The brute-force rescue — and why it only works on toys

On a 17-clock with two unknowns, there are only $17 \times 17 = \mathbf{289}$ possible secrets. You could try every one, compute what each predicts for all four equations, and keep the candidates whose predictions are all within ± 1 of the published values.

I did exactly that. Out of 289 candidates, **exactly one** passes: (5, 11), with discrepancies of +1, +1, -1, +1 — the hidden nudges, recovered.

So at this size the problem is trivial. Its difficulty is entirely a matter of scale, and scale is cheap to buy:

- With **two** unknowns on a 17-clock, there are 289 candidates.
- With **ten** unknowns, there are $17^{10} \approx 2 \times 10^{12}$ — a few hours on a laptop.
- In ML-KEM-1024, the secret is **1,024** numbers. The scheme deliberately draws them small — each one is -2, -1, 0, 1 or 2 — so that the noise stays manageable. Even so, that is $5^{1024} \approx \mathbf{2^{2378}}$ candidates. For comparison, the number of atoms in the observable universe is around 2^{266} .

Brute force is not the attack anybody worries about. The serious attacks are cleverer, and to see what they are doing you have to look at the problem from a completely different direction.

Part 4 — The grid underneath

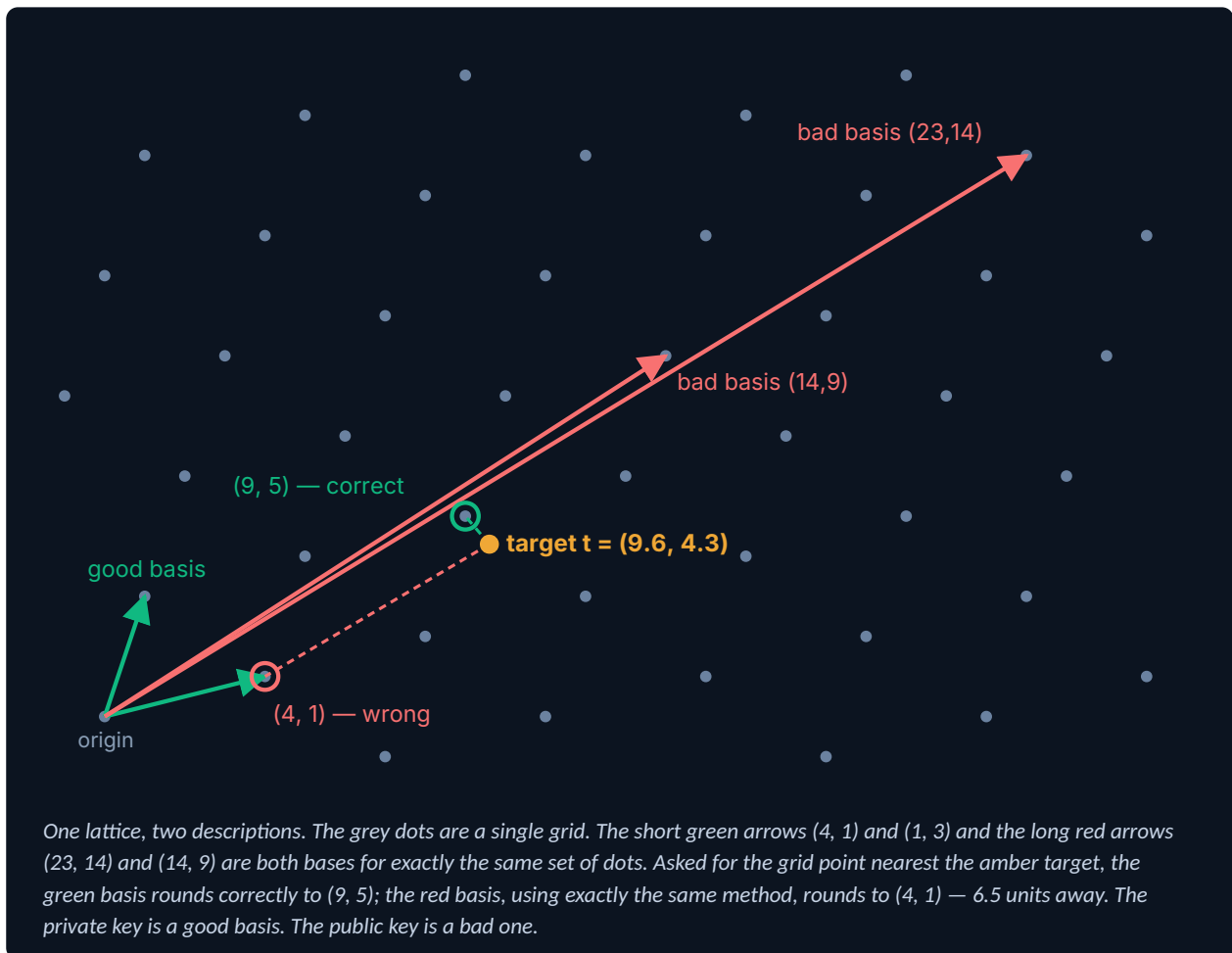
Take every possible secret (x, y) , push it through the left-hand sides of the equations without any noise, and write down the list of answers it produces. Each secret gives you one list — one point, in a space with as many dimensions as there are equations. Do this for every secret, allow the wrap-around of the clock, and the points you get are not scattered at random. **They form a perfectly regular grid**, extending in every direction, the same pattern repeating forever. That grid is a *lattice*.

The published, noisy right-hand sides are one more point in the same space — a point that sits *near* one of the grid points but not on it, because every coordinate was nudged by a small amount. Finding the secret is exactly the same problem as finding **the grid point nearest to that nudged target**. The nearest grid point tells you which secret produced it.

Stated like that, it sounds easy. In two dimensions, it is easy — you can see the nearest point. The difficulty lives in two places: the number of dimensions, and the way the grid is described.

A good description and a bad description of the same grid

A lattice is described by a *basis*: a small set of arrows such that every grid point can be reached by walking a whole number of steps along each arrow. The surprising thing is that the same grid has infinitely many bases, and they are not equally useful.



In the figure, the grey dots are one lattice. The two short green arrows, $(4, 1)$ and $(1, 3)$, are a basis for it: every dot is some whole number of the first plus some whole number of the second. The two long red arrows, $(23, 14)$ and $(14, 9)$, are **also** a basis for exactly the same set of dots — every dot can be reached by whole-number combinations of them too, and the two bases enclose exactly the same area per grid cell, 11 square units. They describe precisely the same grid.

Now take the amber target, $(9.6, 4.3)$, and ask each basis for the nearest grid point using the most natural method there is — write the target as a combination of the two arrows, and round each coefficient to the nearest whole number. (This is called *Babai rounding*, after László Babai, who analysed it in 1986.)

- **With the good basis:** the target is 2.23 lots of $(4, 1)$ plus 0.69 lots of $(1, 3)$. Round to 2 and 1, and you land on $(9, 5)$, at distance 0.92. **Correct** — it is the true nearest point.
- **With the bad basis:** the target is 2.38 lots of $(23, 14)$ plus -3.23 lots of $(14, 9)$. Round to 2 and -3 , and you land on $(4, 1)$, at distance 6.5. **Wrong** — and not slightly wrong. Rounding errors of less than half a step, multiplied by long, nearly parallel arrows, become an enormous error in position.

That is the same phenomenon as Part 3, seen geometrically. Small errors pushed through a bad description of the problem come out large. **The private key is, in effect,**

a good description of the grid. The public key is a bad one. Anyone can see the grid; only the holder of the short arrows can navigate it.

What the attackers actually do: lattice reduction

So the real attack is not brute force. It is *lattice reduction*: given a bad basis, search for a good one — shorter arrows, closer to perpendicular. The founding algorithm is LLL, published by Arjen Lenstra, Hendrik Lenstra and László Lovász in 1982, and it is one of the most useful algorithms in computational mathematics. It runs quickly and always improves a basis. It just does not improve it *enough* in high dimensions: the arrows it finds are short only relative to a factor that grows exponentially with the dimension.

The modern workhorse is BKZ, which improves on LLL by repeatedly solving the shortest-vector problem exactly in small blocks of the basis — say 50 or 100 dimensions at a time — and stitching the results together. The bigger the block, the better the basis, and the cost of each block grows exponentially in its size. The best known classical method for those blocks, *lattice sieving*, costs roughly $2^{0.292 \beta}$ operations for a block of size β . The best known quantum version costs roughly $2^{0.265 \beta}$.

That second number is worth pausing on, because it is the whole quantum story in miniature. A quantum computer does help lattice sieving, by using Grover-style search inside it. **But it only shaves the constant in the exponent, from 0.292 to 0.265.** It does not turn an exponential problem into a polynomial one, which is what Shor’s algorithm does to factoring. It is the same kind of modest, survivable speed-up that Grover’s algorithm gives against AES — the reason I explained in the previous essay that AES-256 needed no replacement.

For ML-KEM-1024, breaking the scheme by the best known lattice attack needs a block size of roughly 870. That works out to about 2^{256} operations classically and roughly 2^{230} quantumly, in the standard accounting cryptographers use (called “core-SVP”, which deliberately counts only the dominant cost and is therefore generous to the attacker). Real attacks cost more than that, not less. NIST places ML-KEM-1024 in its highest security category, Category 5, defined as at least as hard to break as recovering an AES-256 key by exhaustive search.

An honest note on what “hard” means here

Nobody has proved that lattice reduction cannot be done fundamentally faster. These estimates are the cost of the best attacks known after about four decades of work on lattice algorithms, and roughly fifteen years of intense, well-funded effort aimed specifically at the parameter sets that became ML-KEM. That is the same kind of evidence on which RSA’s security rested for forty years. I come back to what that does and does not justify in the 2026 section below.

Part 5 — Locking one bit by hand

So far we have a problem that is easy to set and hard to solve. To turn that into encryption you need one more idea, and it is due to Oded Regev, who introduced Learning With Errors in 2005. (He won the Gödel Prize for it in 2018.) The idea is so simple that we can do the whole thing by hand.

The 17-clock is too cramped for encryption — you will see why in a moment — so I am going to move to a **97**-clock. Same rules: wrap around at 97, everything else as before.

The key

My secret is two numbers: **$s = (29, 63)$** .

My public key is six noisy equations about that secret. For each one I pick two random coefficients, compute the true answer, and add a secret nudge of -1 , 0 or $+1$:

Row	Coefficients (a_1, a_2)	True value: $29a_1 + 63a_2 \pmod{97}$	Nudge	Published b
1	(23, 61)	48	+1	49
2	(70, 8)	12	-1	11
3	(45, 90)	88	0	88
4	(12, 37)	60	+1	61
5	(88, 54)	37	+1	38
6	(31, 76)	61	-1	60

Check row 1 yourself: $29 \times 23 + 63 \times 61 = 667 + 3,843 = 4,510$. And $97 \times 46 = 4,462$, so $4,510 = 48 \pmod{97}$. Add the nudge, publish 49.

The public key is the coefficients and the published b column. The secret $(29, 63)$ and the nudges stay with me. Anyone who wants to recover my secret from the public key faces exactly the problem in Part 3.

Encrypting

You want to send me a single bit. You do this:

1. **Pick a secret random handful of the rows.** Say rows 1, 3, 4 and 6. You tell nobody which.
2. **Add up their coefficients**, column by column, to get a new pair u :
 - first column: $23 + 45 + 12 + 31 = 111 = \mathbf{14} \pmod{97}$
 - second column: $61 + 90 + 37 + 76 = 264 = \mathbf{70} \pmod{97}$
3. **Add up their published b values:** $49 + 88 + 61 + 60 = 258$.
4. **If your bit is 1, add 48** — very nearly half of 97, i.e. half-way round the clock. If your bit is 0, add nothing.

To send a **1**: $258 + 48 = 306 = 306 - 291 = \mathbf{15} \pmod{97}$.

The ciphertext is the pair $u = (14, 70)$ and the single number $v = 15$. That is all you send.

What you have actually done is build a *new* noisy equation — the sum of four old ones — whose noise is the sum of their four nudges. Then you have shifted its right-hand side halfway round the clock to encode the bit. An eavesdropper sees a fresh, legitimate-looking noisy equation and cannot tell whether its right-hand side has been shifted, because they cannot tell what its true value is. That is the LWE problem again.

Decrypting

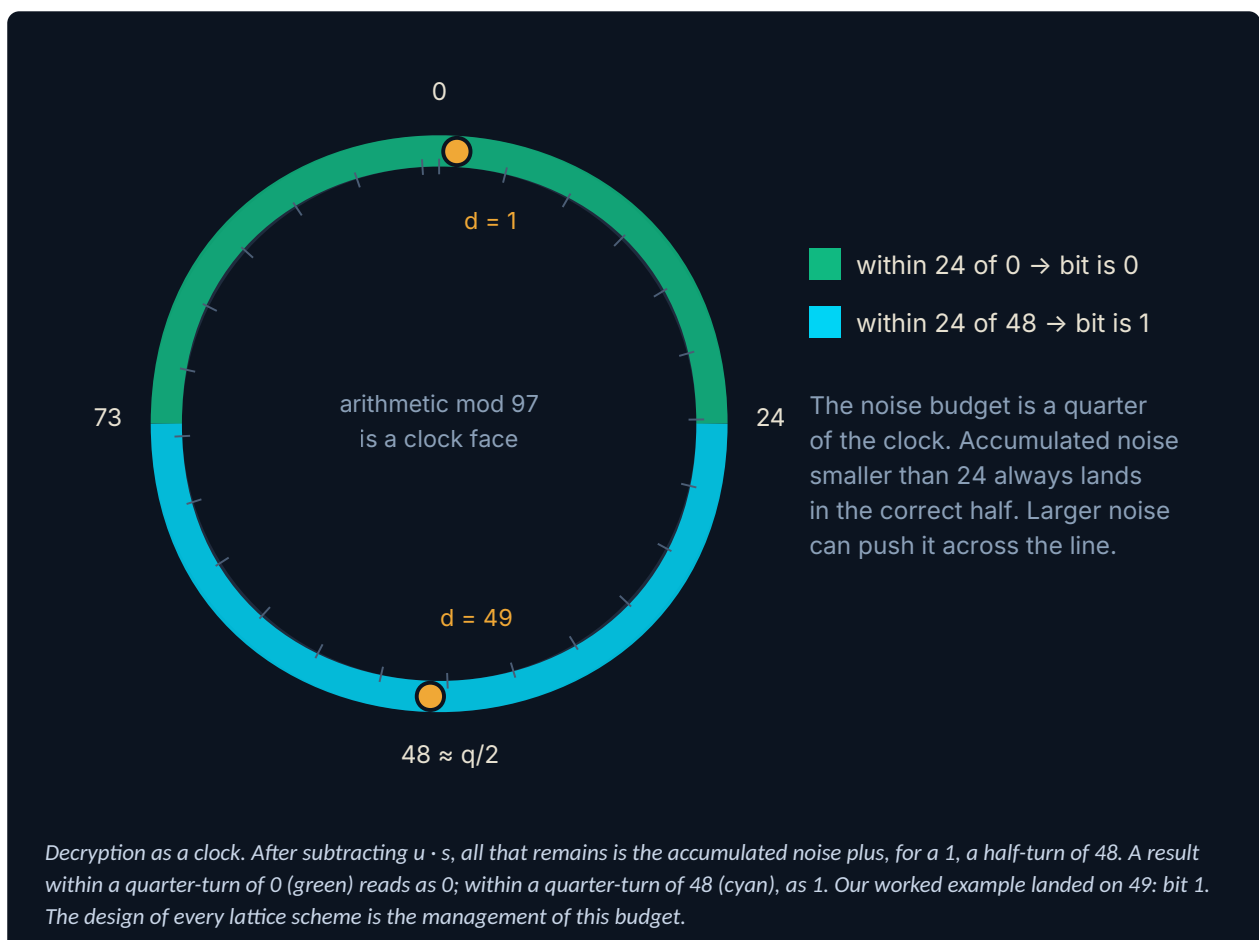
I hold the secret, so I can compute what the new equation's right-hand side *would* have been without noise or message:

$$u \cdot s = 14 \times 29 + 70 \times 63 = 406 + 4,410 = 4,816 = 97 \times 49 + 63 = \mathbf{63} \pmod{97}.$$

Now subtract it from what you sent: $v - 63 = 15 - 63 = -48$, which wraps round to **49**.

What is left is exactly *the accumulated noise plus the message*: the four nudges $(+1, 0, +1, -1)$ sum to $+1$, and the message added 48, so I should see 49. I do.

I do not know the nudges, so I cannot subtract them. I do not need to. I only need to ask which half of the clock 49 is in.



49 is right next to 48, half-way round, so **the bit is 1**. Had you sent a 0, v would have been 64, I would have computed $64 - 63 = 1$, right next to 0, and read the bit as 0.

That picture is the whole design problem of lattice encryption. The noise is what makes the scheme secure, and the noise is also what threatens to make it wrong. **Every operation adds noise, and decryption only works while the accumulated noise is less than a quarter of the way round the clock.** Here the worst case is six nudges of 1 all in the same direction — a total of 6, comfortably inside the budget of 24. On the 17-clock the budget would have been only 4, and six rows could have blown straight through it. That is why I moved to 97.

Every real lattice scheme is an exercise in managing that budget: enough noise to defeat lattice reduction, little enough that decryption essentially never fails. Hold on to the word *essentially*; we will need it.

Part 6 — Check it in a spreadsheet

In [Backpropagation by Hand](#) I rebuilt everything in a spreadsheet, because watching a cell change when you change its inputs makes arithmetic real in a way reading it never does. The same works here. Set it up like this:

Cell(s)	What goes in it
A2:B7	The six coefficient pairs from the table above
C2:C7	The nudges: 1, -1, 0, 1, 1, -1
F1, G1	The secret: 29 and 63
D2	<code>=MOD(A2*\$F\$1+B2*\$G\$1+C2, 97)</code> — fill down to D7. These are the published b values; check they read 49, 11, 88, 61, 38, 60
E2:E7	Your secret handful: 1 for rows you include, 0 for rows you skip (1, 0, 1, 1, 0, 1 for the worked example)
H1	The bit you want to send: 0 or 1
H2	<code>u_1: =MOD(SUMPRODUCT(A2:A7,E2:E7), 97)</code>
H3	<code>u_2: =MOD(SUMPRODUCT(B2:B7,E2:E7), 97)</code>
H4	<code>v: =MOD(SUMPRODUCT(D2:D7,E2:E7)+H1*48, 97)</code>
H5	What decryption sees: <code>=MOD(H4 - (H2*\$F\$1+H3*\$G\$1), 97)</code>
H6	The decrypted bit: <code>=IF(MIN(H5, 97-H5)<24, 0, 1)</code>

Now play with it. Flip the bit in H1 and watch H6 follow. Change the handful in column E. Then make the nudges in column C larger — try values around ± 10 — and watch decryption start to fail on some handfuls and not others. That is the noise budget, failing in front of you.

Finally, try to recover F1 and G1 *without looking*, from columns A, B and D alone. You will find it easy at this size, because there are only $97 \times 97 = 9,409$ candidates and a spreadsheet can try them all. The point of the exercise is not that it is hard here. It is to see that **the only way you found it was by trying them all**.

Part 7 — Check it in Python

The same scheme, scaled up so that you can run thousands of encryptions and watch the noise budget behave statistically. It is deliberately the simplest possible code. **It is for learning only and must never be used to protect anything** — real implementations are a different discipline altogether, as the 2026 section will make painfully clear.

```

"""Toy Regev encryption – for learning only. Never use this to protect anything."""
import random

def keygen(n, m, q, eta, rng):
    s = [rng.randrange(q) for _ in range(n)] # secret
    A = [[rng.randrange(q) for _ in range(n)] for _ in range(m)] # public matrix
    e = [rng.randint(-eta, eta) for _ in range(m)] # small noise
    b = [(sum(a * x for a, x in zip(row, s)) + err) % q for row, err in zip(A, e)]
    return s, (A, b)

def encrypt(pk, bit, q, rng):
    A, b = pk
    rows = [i for i in range(len(A)) if rng.random() < 0.5] # secret random handful
    n = len(A[0])
    u = [sum(A[i][j] for i in rows) % q for j in range(n)]
    v = (sum(b[i] for i in rows) + bit * (q // 2)) % q
    return u, v

def decrypt(s, ct, q):
    u, v = ct
    d = (v - sum(x * y for x, y in zip(u, s))) % q # = accumulated noise + bit *
    q/2 # nearer 0, or nearer q/2?
    return 0 if min(d, q - d) < q // 4 else 1

def failure_rate(n, m, q, eta, trials=20000, seed=1):
    rng = random.Random(seed)
    s, pk = keygen(n, m, q, eta, rng)
    bad = 0
    for _ in range(trials):
        bit = rng.randrange(2)
        bad += decrypt(s, encrypt(pk, bit, q, rng), q) != bit
    return bad / trials

if __name__ == "__main__":
    for n, m, q, eta in [(2, 6, 97, 1), (16, 64, 97, 1), (16, 64, 97, 3),
                        (16, 64, 97, 6), (16, 64, 3329, 6)]:
        print(f"n={n:3d} m={m:3d} q={q:5d} noise=±{eta}: "
              f"failure rate {failure_rate(n, m, q, eta):.4f}")

```

Run it and you should see something very close to this:

Unknowns n	Public rows m	Clock q	Nudges	Decryption failures
2	6	97	±1	0.00%
16	64	97	±1	0.00%
16	64	97	±3	1.13%
16	64	97	±6	16.64%
16	64	3,329	±6	0.00%

Read down the table. Increase the noise on the same clock and decryption starts failing — about one bit in ninety at ± 3 , one in six at ± 6 . Keep the large noise but make the clock much bigger and the failures vanish again, because the quarter-clock budget has grown from 24 to 832.

That last clock size is not arbitrary. **3,329 is the modulus inside ML-KEM.** The reason for that exact number is coming.

A clinical way to think about the noise budget

Think of it as the therapeutic window of a drug. Too little noise and the scheme is ineffective: lattice reduction strips it straight off and recovers the secret. Too much and it is toxic: decryption fails and the scheme stops working. Parameter selection in lattice cryptography is the same exercise as dose selection — finding the range between the minimum effective dose and the maximum tolerated one, with a margin either side, for every patient, every time.

Part 8 — From the toy to ML-KEM

Regev’s scheme as we have built it works, and it is believed secure at large enough sizes. It is also wildly inefficient. To send one bit, you sent two numbers plus one more; to make it secure, n has to be in the high hundreds, which means sending hundreds of numbers per bit, and the public key is a matrix of hundreds of thousands of entries. Four engineering ideas turn it into ML-KEM.

Idea one: let one equation do the work of hundreds

The first idea is to stop using plain numbers and use *polynomials* — expressions like $3 + 5x + 2x^2 + 7x^3$ — with a special rule for multiplying them. This variant, Ring-LWE, was introduced by Vadim Lyubashevsky, Chris Peikert and Oded Regev in 2010.

The rule is: whenever a power of x reaches 4 (in the real scheme, 256), replace x^4 with -1 . So x^4 becomes -1 , x^5 becomes $-x$, x^6 becomes $-x^2$, and so on. Powers wrap around, like the clock, but **with a sign flip** each time they do.

Work one example. Multiply $a = 3 + 5x + 2x^2 + 7x^3$ by a small secret $s = 1 - x^2 + x^3$:

- Ordinary multiplication gives $3 + 5x - x^2 + 5x^3 + 3x^4 - 5x^5 + 7x^6$.
- Fold the high powers back with the sign flip: $3x^4$ becomes -3 ; $-5x^5$ becomes $+5x$; $7x^6$ becomes $-7x^2$.
- Collect: $(3 - 3) + (5 + 5)x + (-1 - 7)x^2 + 5x^3 = 0 + 10x - 8x^2 + 5x^3$.
- On the 17-clock, -8 wraps to 9 : the product is **$10x + 9x^2 + 5x^3$** .

Here is why that matters. That one polynomial multiplication is *exactly* the same as multiplying the secret by a particular 4×4 matrix built from a ’s four coefficients, each row a shifted copy of the one above with the wrapped-around entries negated:

3	-7	-2	-5
5	3	-7	-2
2	5	3	-7
7	2	5	3

Multiply that matrix by the secret $(1, 0, -1, 1)$ and you get $(0, 10, -8, 5)$ — the same four numbers. **One polynomial of four coefficients stands in for a whole matrix of sixteen.** At ML-KEM’s size, one polynomial of 256 coefficients stands in for a 256×256 matrix — and one ring equation carries 256 noisy equations’ worth of information. Instead of sending hundreds of numbers per bit, you send a few bits per bit.

Idea two: a little flexibility — modules

Pure Ring-LWE puts all its eggs in one ring of a fixed size. ML-KEM uses a compromise called *Module-LWE*: instead of one big polynomial, a small $k \times k$ matrix of polynomials,

each of degree 256. The ring structure gives the efficiency; the small matrix gives a dial for security. That is the entire difference between the three ML-KEM parameter sets:

	ML-KEM-512	ML-KEM-768	ML-KEM-1024
Matrix size k	2	3	4
Polynomial degree	256	256	256
Clock q	3,329	3,329	3,329
Secret / noise range (η_1, η_2)	$\pm 3, \pm 2$	$\pm 2, \pm 2$	$\pm 2, \pm 2$
Public key	800 bytes	1,184 bytes	1,568 bytes
Ciphertext	768 bytes	1,088 bytes	1,568 bytes
Decryption failure probability	about 2^{-139}	about 2^{-165}	about 2^{-175}
NIST security category	1 ($\approx$ AES-128)	3 ($\approx$ AES-192)	5 ($\approx$ AES-256)

MedLattice uses ML-KEM-1024 for everything at rest, and the hybrid X25519 + ML-KEM-768 that browsers already speak for connections in transit. The specification explains why: a session key protects data for seconds, a compartment key for the life of a patient, and “the size penalty is paid once per wrap rather than once per packet”.

For a sense of scale, the unstructured, plain-LWE alternative to ML-KEM — a scheme called FrodoKEM, which deliberately avoids the ring structure — has public keys and ciphertexts of around 21 kilobytes at the equivalent security level. ML-KEM-1024’s are 1.6 kilobytes. That is what the polynomials buy. Whether the structure that buys it is also a weakness is a fair question, and one of the 2026 papers below is an attempt to exploit exactly that structure.

Idea three: why 3,329 — the number-theoretic transform

Multiplying two polynomials of degree 256 the schoolbook way takes $256 \times 256 = 65,536$ multiplications. ML-KEM does it in a few thousand, using the *number-theoretic transform* — the clock-arithmetic cousin of the fast Fourier transform that every MRI scanner uses to turn raw signal into an image.

The FFT needs special numbers called roots of unity: a number which, multiplied by itself enough times, comes back to 1. On a clock, those exist only if the clock size is chosen carefully. 3,329 is chosen so that $3,329 - 1 = 3,328 = 256 \times 13$. That guarantees a number which, multiplied by itself 256 times, returns to 1 on the 3,329-clock (ML-KEM uses 17). (The original Kyber submission used 7,681, which allows a complete transform; the designers moved to 3,329 because it is small enough for each coefficient to fit in 12 bits while still leaving a comfortable noise budget, at the cost of a transform that stops one step short.) The transform is why ML-KEM runs faster than the elliptic-curve key exchange it is replacing.

Idea four: compression, and the transform that makes it safe

Two last steps. First, the ciphertext is *compressed*: ML-KEM-1024 rounds each coefficient of u down to 11 bits and of v down to 5, throwing away the low-order bits. That rounding adds a little more noise of its own — deterministic, but counted against the same quarter-clock budget — in exchange for a much smaller ciphertext.

Second, and much more important: the encryption we built in Part 5 is fragile. If an attacker can send me deliberately malformed ciphertexts and watch whether I decrypt them successfully — even just whether I complain — they can learn about my secret one probe at a time. This is not hypothetical; it is how many real lattice schemes have been attacked in the laboratory.

ML-KEM closes it with a construction called the Fujisaki-Okamoto transform, after its 1999 inventors. The idea is a re-encryption check. The sender's random choices — the handful, in our toy — are not truly random but derived, through a hash function, from the message itself. So when I decrypt, I can take the message I recovered, re-derive the sender's choices, re-encrypt, and check that I get *exactly* the ciphertext I received, bit for bit. If anything has been tampered with, the check fails, and instead of an error message I return a key computed from a separate secret value — a key that is useless to the attacker and indistinguishable, to them, from a real one. This is called *implicit rejection*. The attacker probing for reactions gets no reaction to probe.

This is also why the decryption-failure probabilities in the table above are so absurdly small. With the transform in place, a decryption failure is itself a form of leak, so the budget must make failure not just rare but negligible. About 2^{-175} for ML-KEM-1024 means that if every person on earth performed a billion decapsulations a second for the age of the universe, the expected number of failures would still be indistinguishable from zero.

That is the real ML-KEM, and every step of it is the toy from Part 5 with engineering bolted on. Secret small, public equations noisy, a random handful summed, the message shifted halfway round the clock, the noise stripped by the key-holder down to which half of the clock the answer landed in.

Part 9 — Why Shor’s algorithm gets no grip

In the previous essay I showed how Shor’s algorithm works: turn the problem into finding a hidden *period* — the length of a repeating cycle — and let the quantum Fourier transform pick that period out of a superposition by interference. Factoring has a period hiding in it. So does the elliptic-curve discrete logarithm. That is the entire reason both fall.

For the mathematically curious, the precise version is this. Shor’s algorithm solves what is called the *hidden subgroup problem* for *abelian* groups — structures where the order of operations does not matter. Factoring and discrete logarithms are both instances of it. In 2002, Regev showed that lattice problems can also be phrased as a hidden subgroup problem — but over the *dihedral* group, the group of symmetries of a polygon, which is not abelian. Shor’s method does not work there. The best quantum algorithm known for the dihedral problem, due to Greg Kuperberg in 2003, runs in subexponential time rather than polynomial time, and in more than two decades nobody has managed to turn it into a practical attack on LWE.

The plain-language version is the one from my original paragraph, now with the reason behind it. **The noise destroys the periodicity.** Without the nudges, the equations from Part 2 are a perfectly periodic structure, and a quantum computer would make short work of them — but so would a laptop, using Gaussian elimination. The nudges are precisely what break the clean repeating pattern that interference needs to sculpt towards. There is no hidden cycle for the quantum Fourier transform to resonate with, because the noise smears it out.

There is one more piece of theory that makes cryptographers trust LWE more than they trusted RSA at the same age, and it is the reason Regev’s 2005 paper won the Gödel Prize. He proved that solving *random* LWE instances — the kind a cryptosystem generates — is at least as hard as solving the *hardest* instances of certain lattice problems, in the worst case, for a quantum computer. That kind of guarantee is rare. RSA has nothing like it: nobody has proved that the random numbers used in RSA keys are as hard to factor as the hardest numbers. For LWE, an attack that works on typical keys would automatically be an attack on every instance of a problem mathematicians have studied for decades.

The two caveats to that guarantee

First, the reduction is asymptotic. ML-KEM’s actual parameters are not chosen by plugging numbers into Regev’s proof — at those sizes the proof’s guarantee is too loose to be useful — but by estimating the cost of the best known attacks, as in Part 4. The proof says the problem is the right kind of hard; the cryptanalysis says how hard.

Second, the proof is for plain LWE. ML-KEM uses Module-LWE, which has extra algebraic structure, and structure is exactly what an attacker looks for. There are matching reductions for the structured versions, but they reduce to lattice problems with the same structure, which have had less independent study. In 2016, Ronald Cramer, Léo Ducas,

Chris Peikert and Oded Regev showed that a related problem in the same family of rings — finding a short generator of a principal ideal, a problem on *ideal* lattices — can be solved efficiently with a quantum computer. It does not touch Module-LWE as ML-KEM uses it. But it is the reason that every serious attempt to break ML-KEM with a quantum computer starts from that 2016 paper, and it is where the most interesting paper of 2026 started too.

Part 10 — The assumption on trial: 2026

Explanations of post-quantum cryptography usually stop at the theory. But the theory is only as good as the people who have tried to break it, and the most useful thing I can do here is show you, concretely, what that looks like when it happens. This year it happened several times.

The precedent: April 2024

On 10 April 2024, Yilei Chen of Tsinghua University posted a paper titled *Quantum Algorithms for Lattice Problems*, claiming a polynomial-time quantum algorithm for LWE. Chen is a serious, respected cryptographer, and the claim, if true, would have undermined the entire post-quantum migration a few months before NIST published the standards. The cryptographic community dropped what it was doing and read it.

Eight days later, Chen added a note to the paper: “Step 9 of the algorithm contains a bug, which I don’t know how to fix.” The bug had been found independently by Hongxun Wu and by Thomas Vidick. Even had it worked, the claimed algorithm’s parameters would not have reached ML-KEM directly. But the episode established what an attack on LWE would actually have to survive — and how fast it would be checked.

The claim: May 2026

Between 17 April and 17 May 2026, Ming-Xing Luo of Southwest Jiaotong University posted a four-part series on arXiv titled *Module Lattice Security*. Part IV claimed a quantum attack, extending the 2016 Cramer-Ducas-Peikert-Regev approach, that would break ML-KEM, along with the signature schemes Falcon and Hawk and variants of NTRU, using roughly 1,400 logical qubits and on the order of 10^8 logical gates. For ML-KEM-1024, the paper claimed a comfortable margin of about sixteen times.

If true, that would have been the most important result in cryptography in thirty years. It would also have taken down MedLattice’s key encapsulation, and I would be writing a very different post.

It did not survive examination. By 25 May, an analysis published by PostQuantum.com had listed a series of mathematical errors, among them a base case that treats $\sqrt{2}$ as a unit when its norm is 4, a recursion using the wrong norm, a precision formula that implies around 2.9 million qubits at the top level rather than 1,400, a step that depends on a bound the paper itself lists as unproven, a citation to a Part V that did not exist, and no implementation or test on toy parameters. Thomas Espitau, a co-author of the Hawk paper Luo cites, reportedly said that the reduction was dimensionally wrong.

The decisive objection, as reported, came from Daniel Apon, formerly a NIST cryptographer. It is worth understanding, because it is a beautiful piece of reasoning that anybody who has followed this post can check. Apon’s point was that the object the attack targets would be **the same for any two keys that share the same public matrix**. But two such keys can have completely different secrets. An algorithm whose answer

does not depend on the secret cannot possibly recover the secret. You do not need to find the bug in the algebra; the structure of the claim rules it out.

Translated into our toy: imagine a method that claims to recover my secret (29, 63) but only ever looks at the coefficient columns of my public key, never the b values. Give it a second public key with exactly the same coefficients but a different secret, and it must return the same answer — so it is wrong at least half the time. Whatever it is computing, it is not the secret.

What the episode actually demonstrated

Two claimed quantum breaks of LWE in two years, both found to be wrong within weeks. That is not evidence that lattice cryptography is proven secure — nothing is. It is evidence that the claim “no known quantum algorithm breaks this” is being actively, adversarially tested by people with every incentive to break it, and that the tests are fast. That is precisely the kind of scrutiny that turns a young assumption into a mature one. The analysts who examined Luo’s paper drew the sensible conclusion: migration plans should not change because of it, hybrid deployment should remain the default, and systems should be built so that an algorithm can be replaced. MedLattice was already built that way; I come back to how in Part 11.

The quiet result: September 2026

The paper that matters more, in my judgement, made no headlines at all. On 26 September, IACR ePrint published *A Provable Correctness Analysis of the MATZOV-Style FFT Dual Attack on ML-KEM*, by Jiale Li, Li-Ping Wang and Huaxiong Wang.

Some background. In 2022 a group at the Israeli cryptographic research centre MATZOV published an improved “dual” lattice attack on what became ML-KEM, which under some cost models brought the smallest parameter set slightly below its target security level. That triggered a long argument, because the attack’s analysis rested on heuristic assumptions — reasonable-looking guesses about independence — that other researchers then questioned. The honest state of affairs for several years was that nobody was entirely sure how well dual attacks actually work.

The September paper replaces the heuristics with proofs. Under the rigorous framework it adopts, it estimates attack costs of 2^{210} , 2^{300} and 2^{410} for ML-KEM-512, -768 and -1024 — between 28 and 68 bits lower than the previous provable baseline from 2024.

Two things need saying about those numbers, and the second matters more than the first. First, **68 bits is a large reduction**: it means the attack is about 2^{68} — roughly three hundred billion billion — times cheaper than the previous provable estimate. Second, **2^{410} is not comparable to the 2^{256} figure from Part 4**, because the two are computed under different accounting conventions; the provable framework counts costs the heuristic one ignores. What the paper shows is that the attack estimates are moving, and moving downwards, which is what attack estimates do over time. What it does not

show is anything close to a practical attack on any ML-KEM parameter set, and certainly not on ML-KEM-1024.

This is what normal cryptanalysis looks like: not a dramatic collapse, but a slow erosion of margin, tracked in public, years ahead of it mattering. It is the reason to choose the largest parameter set for data that must stay secret for decades. A system that chose its security margin precisely, to the bit, would have no room for erosion of this kind. MedLattice chose Category 5 for data at rest precisely so that it would.

The other clock: quantum hardware for RSA

While the lattice side held, the threat it exists to answer moved closer. In May 2025, Craig Gidney of Google Quantum AI published *How to factor 2048 bit RSA integers with less than a million noisy qubits*, estimating that a machine of fewer than a million physical qubits could do it in under a week — a twentyfold reduction from his own 2019 estimate. In February 2026, a preprint describing the “Pinnacle” architecture, using a newer family of error-correcting codes called QLDPC codes, put the figure at roughly 100,000 physical qubits running for about a month. That one has not yet been peer-reviewed, and it assumes classical error decoding inside a ten-microsecond window, which nobody has yet demonstrated at that scale.

I said in the previous essay that the year of the first cryptographically relevant quantum computer is genuinely disputed, and it still is. But the direction of travel over eighteen months has been unambiguous: the machine needed keeps getting smaller.

For MedLattice, this matters in a specific and slightly counter-intuitive way. **It does not touch the lattice half of the hybrid at all.** Shor’s algorithm breaks RSA and elliptic curves, including X448, MedLattice’s classical partner to ML-KEM. What falling qubit estimates shorten is the life expectancy of the classical half of the hybrid. That was always expected — the specification says in so many words that X448 “is quantum-broken”. The classical half is there in case the lattice half turns out to have an unknown weakness. The falling estimates simply make it clearer which of the two locks is carrying the long-term weight.

The chip that talks: side channels

The final category of 2026 news is the one I take most seriously, because it concerns the thing that actually breaks deployed cryptography.

The mathematics of ML-KEM can be perfect and an implementation can still leak the secret through its physical behaviour — how long an operation takes, how much power the chip draws, what electromagnetic noise it gives off. In 2024, a team including Daniel J. Bernstein published *KyberSlash*: several implementations of what became ML-KEM, **including the official reference code**, performed a division whose running time depended on secret data. On a Raspberry Pi 2 and an Arm Cortex-M4 microcontroller, the researchers recovered secret keys within minutes for one variant of the flaw and within hours for the other. It was disclosed responsibly and patched quickly. The lattice problem was never at risk. The keys were.

On 11 October 2026 — eight days ago — researchers from the secure-chip company SEALSQ presented a peer-reviewed paper at LightSec 2026, a workshop affiliated with CHES, the main conference on cryptographic hardware. Titled *Cross-information lattice attack on ML-KEM*, it targets the encoding step of key generation and combines profiling of the physical device with lattice reduction, recovering secret information more effectively than either approach alone. The paper reportedly includes countermeasures; the published coverage does not say which hardware was tested or how many measurements the attack needs, so I would not draw conclusions about any particular product from it.

The lesson is the same as KyberSlash, and it is the most practically important sentence in this post: **a post-quantum algorithm is only as strong as the specific implementation running it.** The maths of Part 4 protects against an attacker with the public key and unlimited computing power. It does nothing against an attacker who can time your server or put a probe on your hardware.

Part 11 — What this means for MedLattice

Everything above maps onto a specific decision in the [MedLattice specification](#). I want to set them out together, because the design choices were made before most of 2026’s news, and it is worth checking honestly whether they still look right.

The hybrid. Every key MedLattice wraps is encapsulated twice — once with ML-KEM-1024 and once with X448 — and the two shared secrets are combined using the X-Wing construction of Connolly, Schwabe and Westerbaan, generalised from its standard ML-KEM-768 + X25519 form to the Category 5 pairing (§5.2). The combined key is secure if either component is. Against the Luo claim, the hybrid was the insurance: had the paper been right, X448 would still have held against every attacker without a large quantum computer. Against the falling RSA estimates, the insurance runs the other way: ML-KEM holds when X448 falls. This is exactly the posture the analysts recommended after the Luo episode, and it was the right call before it.

The kill-switch and the re-wrap. Every identity, every wrapped key and every anchor on the ledger carries a four-byte identifier for the cryptographic suite it uses, and a suite can be retired on-chain, deactivating every identity still bound to it within one block (§5.10). More importantly, migrating to a new suite **re-wraps the 32-byte keys; it does not re-encrypt the content**. Moving a petabyte imaging archive to a new algorithm means re-encapsulating keys — hours of work — rather than re-encrypting the archive, which would take months. That is the single property that makes “we can replace ML-KEM if we have to” a real capability rather than a reassuring sentence. If a corrected Luo-style attack ever did appear, the response is a migration measured in hours.

The implementation. The specification already says plainly that its reference implementations — the Python libraries `kyber-py` and `dilithium-py` — are faithful to the standard but not hardened against side channels, and that production deployment requires a FIPS 140-3 validated module, with compartment root keys held in a FIPS 140-3 Level 3 hardware security module or equivalent key-management service (§3.5, §5.12). KyberSlash and the SEALSQ paper are the reason that clause exists. I would go further than the specification currently does, and add two procurement requirements: evidence of constant-time verification of the ML-KEM implementation (the tooling the KyberSlash authors built makes this checkable rather than a matter of vendor assurance), and a statement from the hardware security module vendor of which published side-channel attacks on ML-KEM key generation their product has been tested against.

The history. The record’s history is notarised with SLH-DSA, a hash-based signature scheme that rests on nothing but the properties of a hash function — no lattices at all. Even a complete, correct break of Module-LWE tomorrow would not let anybody rewrite what MedLattice recorded. Of all the decisions in the design, this one looks better after 2026 than before it.

The open item: lattice monoculture

The specification is candid that key establishment is currently “lattice-monocultural” (§5.12). ML-KEM and ML-DSA rest on closely related structured-lattice assumptions, and the X448 half of the hybrid has a known expiry date. If structured lattices fell and a large quantum computer existed, past ciphertext would be lost, and the specification says so in its threat table.

There are two ways to diversify, and 2026 has sharpened the case for both. The first is HQC, a code-based scheme with no lattices in it, which NIST selected in March 2025 as a backup to ML-KEM. NIST said at the time that it expected a draft standard about a year later and a final standard in 2027. The specification already names it as the candidate for a future suite, QMED-KEM-2 (decision D-09). The second is FrodoKEM, the plain-LWE scheme from Part 8, which keeps the lattice assumption but drops the ring structure that the Luo paper — and every serious quantum attempt since 2016 — has tried to exploit. Germany’s federal security office, the BSI, already recommends it. Its cost is size: about 21 kilobytes per wrap instead of 1.6. By the specification’s own argument for Category 5 — that the size penalty at rest is “paid once per wrap rather than once per packet” — that cost may well be affordable for the most sensitive compartments. I have not decided between them, and I am not sure the answer is either-or. But it is the right question, and it is the one I will be putting to the next design review.

What this does not do

This post explains why the mathematics behind ML-KEM is believed to be hard, and why a quantum computer is not believed to change that. It does not, and cannot, prove that it is hard. Nobody can prove that about any practical cryptosystem; the honest claim is “secure against every attack known after sustained, adversarial scrutiny”, and that claim is only as good as the scrutiny.

The worked examples are toys. A 2-dimensional secret on a 97-clock teaches the mechanism, but the security of the real scheme depends on details — the exact noise distribution, compression, the Fujisaki-Okamoto transform, the precise parameters — that the toy leaves out or simplifies. In particular, the toy Regev scheme in Part 7 is insecure against an attacker who can submit chosen ciphertexts, which is exactly the attack the transform in Part 8 exists to stop.

I have reported the Luo paper’s errors and the expert objections as they were published by others; I have not independently verified every step of the critique, and the author has not, as far as I can find, published a response. Likewise, I have reported the September dual-attack estimates under the conventions the authors use, and warned against comparing them with figures computed under other conventions. Anybody who puts the two numbers side by side and concludes that ML-KEM-1024 has gained 154 bits of security, or lost them, has misread both.

Finally, none of this says anything about the parts of MedLattice that are not lattice cryptography. The EVM transaction signatures on the consortium chain are still classical secp256k1, which Shor’s algorithm breaks; the specification states that gap plainly and sets out its mitigations (§5.11). Traffic analysis and metadata leakage are separate problems. A perfect key encapsulation mechanism protects keys, not everything else.

Where this leaves the argument

In the previous essay I said that the uncertainty in the quantum story is not whether RSA falls, only when. This essay is the other half of that sentence. For the lattice mathematics MedLattice is named after, the uncertainty is of a completely different kind. It is not “when”, because there is no known algorithm whose arrival we are waiting for. It is “whether anybody ever finds one”, and 2026 was a year in which several people tried, publicly, and the assumption held.

What I find most reassuring is not any single result. It is the shape of the year. A dramatic claim of a break was examined and dismantled in weeks. A careful, unglamorous paper moved a classical estimate by 68 bits, and the design absorbed it because it had been built with margin to spare. The quantum threat to the classical half of the hybrid got closer, as expected, and the lattice half was there to carry the weight. And the one real, repeatedly demonstrated danger — a chip that leaks its secret through its timing

or its power draw — is one that the specification already treats as a procurement requirement rather than a design afterthought.

The lock is not unpickable. Nothing is. But you have now seen the lock taken apart, worked it by hand, and watched people with every incentive spend a year trying to pick it. That is a better basis for trust than a paragraph — and, I think, the only one a patient whose record will outlive both of us should be asked to accept.

Further reading and references

The mathematics

- Regev O. *On lattices, learning with errors, random linear codes, and cryptography*. STOC 2005; J ACM 2009. The paper that introduced LWE and proved the worst-case reduction.
- Lyubashevsky V, Peikert C, Regev O. *On ideal lattices and learning with errors over rings*. EUROCRYPT 2010.
- Langlois A, Stehlé D. *Worst-case to average-case reductions for module lattices*. Designs, Codes and Cryptography 2015.
- Lenstra AK, Lenstra HW, Lovász L. *Factoring polynomials with rational coefficients*. Math Ann 1982. The LLL algorithm.
- Regev O. *Quantum computation and lattice problems*. FOCS 2002. The dihedral hidden subgroup connection.
- Cramer R, Ducas L, Peikert C, Regev O. *Recovering short generators of principal ideals in cyclotomic rings*. EUROCRYPT 2016.
- Peikert C. *A Decade of Lattice Cryptography*. Foundations and Trends in Theoretical Computer Science 2016. The best single survey.

The standard

- NIST. FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard. August 2024.
- NIST selects HQC as a fifth post-quantum algorithm: SecurityWeek, March 2025.

The 2026 trial

- Chen Y. Quantum Algorithms for Lattice Problems. IACR ePrint 2024/555, with the author's 18 April 2024 note on the bug.
- Luo M-X. Module Lattice Security (Part IV). arXiv 2605.17412, May 2026.
- Ivezic M, Sandler J. Analysis of the Luo ML-KEM quantum attack claim. PostQuantum.com, 25 May 2026.
- Li J, Wang L-P, Wang H. A Provable Correctness Analysis of the MATZOV-Style FFT Dual Attack on ML-KEM. IACR ePrint 2026/2166.
- Gidney C. How to factor 2048 bit RSA integers with less than a million noisy qubits. arXiv 2505.15917, May 2025.
- Shor, QLDPC codes and the compression of RSA-2048 resource estimates. Quantum Computing Report, March 2026.
- Bernstein DJ et al. KyberSlash: Exploiting secret-dependent division timings in Kyber implementations. IACR ePrint 2024/1049; TCHES 2025.

- SEALSQ presents research on side-channel attacks against ML-KEM. The Quantum Insider, 14 October 2026.

Earlier essays in this series

- Not Shown Is Not Locked: The Patient Record I Am Building
- Twenty Generals, One Ledger: Why MedLattice Runs on Hyperledger Besu and QBFT
- The Spinning Coin and the Stolen Key: What Shor’s Algorithm Actually Threatens
- Backpropagation by Hand — the same method, applied to neural networks
- MedLattice Technical Specification v1.1

A short glossary

Basis. A set of arrows from which every point of a lattice can be reached by whole-number steps. The same lattice has infinitely many bases; short, nearly perpendicular ones are good, long, nearly parallel ones are bad.

BKZ. The standard lattice-reduction algorithm: it improves a basis by solving the shortest-vector problem exactly in small blocks. Its cost grows exponentially in the block size.

Core-SVP. A conservative way of estimating the cost of a lattice attack by counting only its dominant step. Generous to the attacker by design.

Decryption failure. The rare case in which accumulated noise crosses the quarter-clock line and a lattice ciphertext decrypts to the wrong value. In ML-KEM-1024, about 2^{-175} per decapsulation.

Fujisaki-Okamoto transform. The re-encryption check that makes ML-KEM safe against attackers who submit tampered ciphertexts and watch the response.

Hybrid. Combining a post-quantum algorithm with a classical one so that the result is secure if either is. MedLattice pairs ML-KEM-1024 with X448.

Lattice. A regular, infinitely repeating grid of points in many dimensions.

Learning With Errors (LWE). Recovering a secret from many equations in clock arithmetic, each nudged by a small random error. Easy without the errors; believed hard with them.

Modulus. The size of the clock: the number at which arithmetic wraps around. 17 and 97 in this post’s examples; 3,329 in ML-KEM.

Module-LWE. The version of LWE used by ML-KEM: a small matrix of polynomials rather than a large matrix of numbers, trading a little structure for a great deal of efficiency.

Number-theoretic transform (NTT). The clock-arithmetic version of the fast Fourier transform, used to multiply polynomials quickly. The reason ML-KEM's modulus is 3,329.

Side channel. Information leaked by the physical behaviour of a device — timing, power, electromagnetic emission — rather than by the mathematics. The most practical threat to deployed post-quantum cryptography.