Elliptic Curve and Post-Quantum Cryptography

Daniel Zappala

CS 465 · Introduction to Security and Privacy · Fall 2026

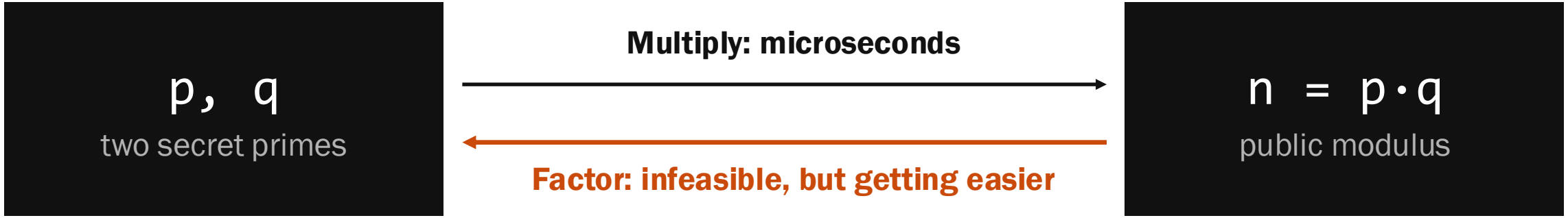
BYU

01

Elliptic curves

The same security as RSA with much smaller keys

RSA keys must keep growing because factoring attacks keep improving



A trapdoor function

Easy to compute, infeasible to reverse unless you hold the secret: here, the primes p and q

Attacks keep improving

The number field sieve beats brute force, so keys need to grow fast: 128-bit security (about 2^{128} steps to break) needs a 3,072-bit key; 256-bit security needs a 15,360-bit key!

Big keys are slow

Going from 2048 to 3072 bits makes private-key operations about 3× slower and every key 50% larger

01

Elliptic curves

ECC matches RSA security with keys 6 to 30 times smaller

Security level	RSA key	ECC key	RSA ÷ ECC
80 bits	1,024 bits	160 bits	**6×**
112 bits	2,048 bits	224 bits	**9×**
128 bits	3,072 bits	256 bits	**12×**
192 bits	7,680 bits	384 bits	**20×**
256 bits	15,360 bits	521 bits	**30×**

Source: NIST SP 800-57 Part 1, comparable strengths

12×

smaller keys at the 128-bit level most systems use today: RSA-3072 vs P-256

Why the gap widens

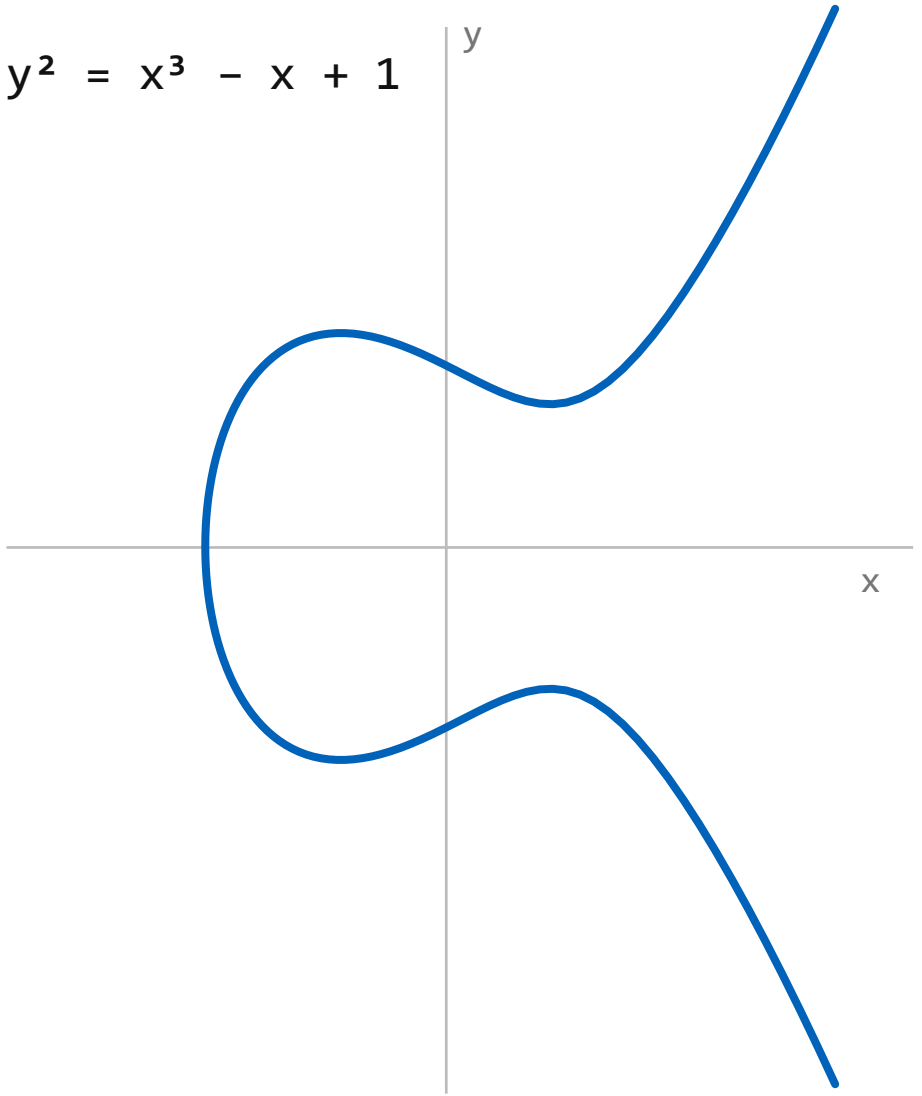
The best attack on a good curve is generic: about $\sqrt{n}$ steps. An ECC key only needs twice the security level in bits, while RSA keys grow much faster

01

Elliptic curves

An elliptic curve is every point satisfying $y^2 = x^3 + ax + b$

Example curve with a = −1, b = 1, plotted over the real numbers



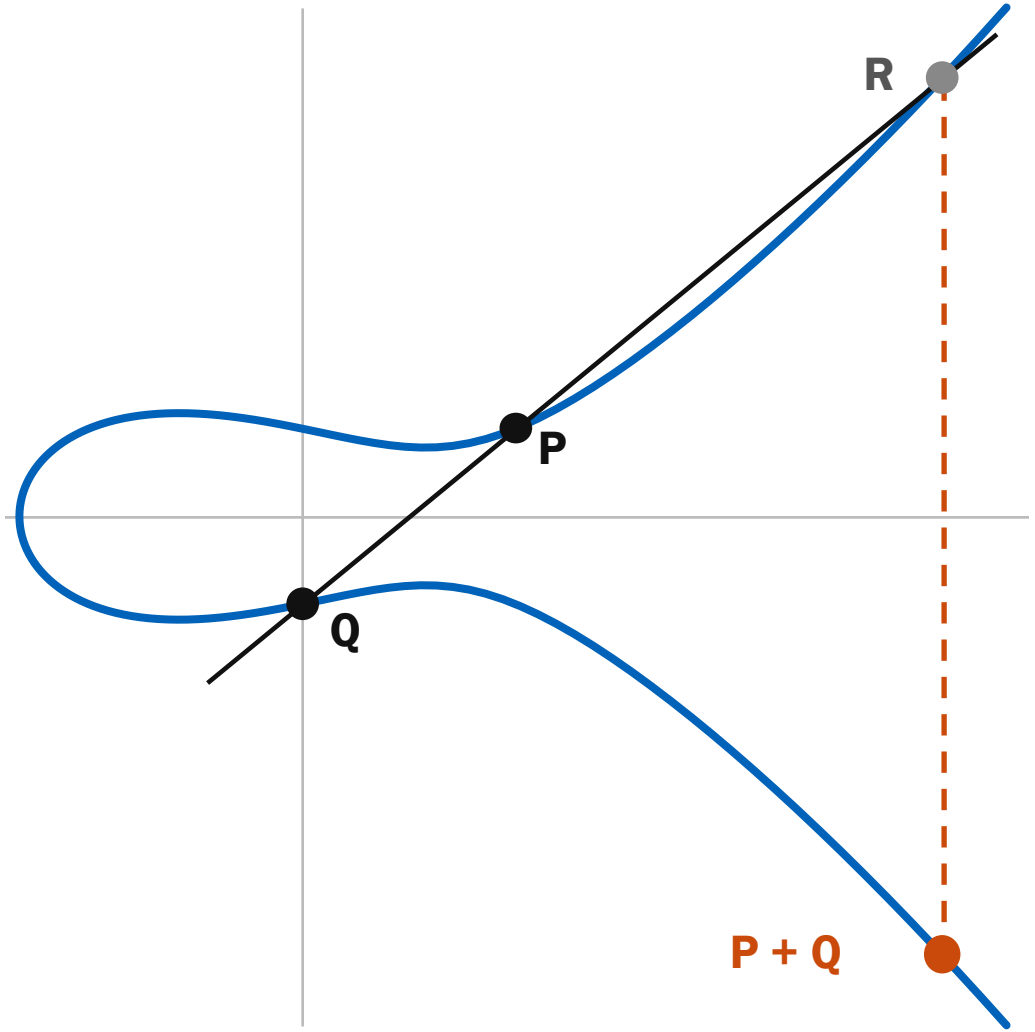
Mirror symmetry

If (x, y) is on the curve, so is (x, −y)

Curves are fixed

Real systems pick a and b once and share them: P-256 and Curve25519 are the common choices

Point addition



1 Draw a line through P and Q

$m = (y_Q - y_P) \div (x_Q - x_P) = 2$, so the line is $y = 2x - 1$

2 Find the third hit, R

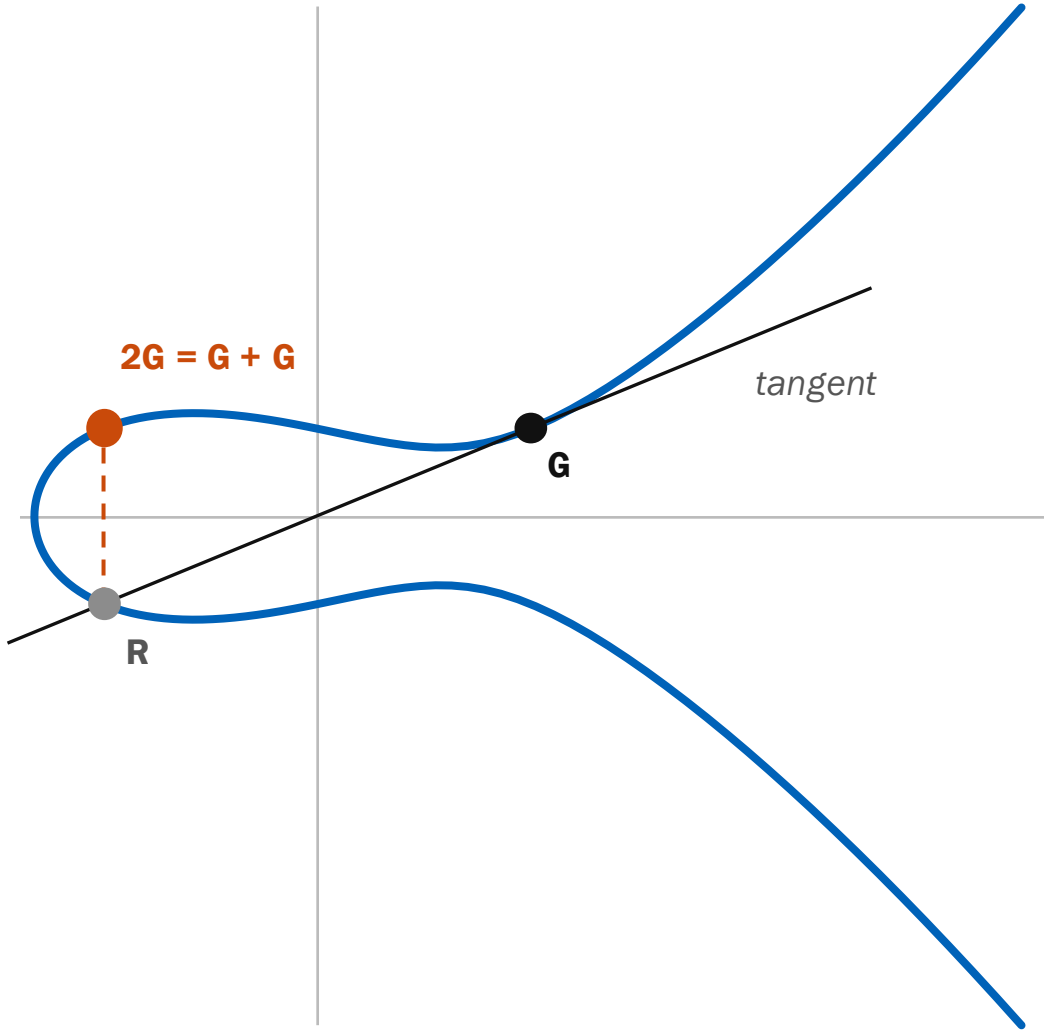
The line meets the curve exactly once more, at R = (3, 5)

3 Reflect R across the x-axis

The mirror image (3, −5) is defined to be P + Q

Only basic arithmetic: subtract, multiply, divide. All operations are done in a finite field mod p, so dividing by d means multiplying by its modular inverse d^{-1} ($d \cdot d^{-1} \equiv 1$)

Point doubling: adding G to itself



1 Draw the tangent at G

With both points at G = (1, 1), the line through them is the tangent: y = x

2 Find the third hit, R

The tangent touches the curve at G and meets it exactly once more, at (−1, −1)

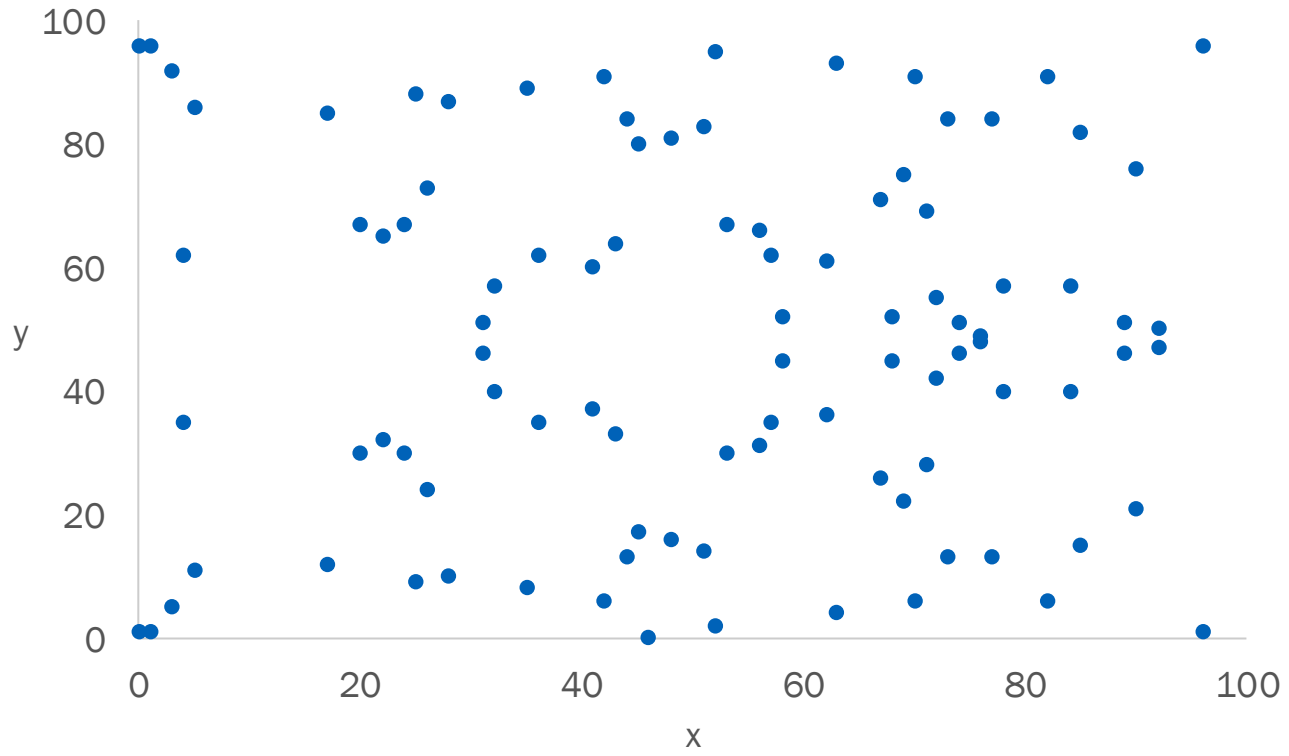
3 Reflect R across the x-axis

The mirror image (−1, 1) is 2G = G + G

The line through 2G and G gives 3G, and the same steps repeat for 4G, 5G …

Adding G again and again, mod a prime, scatters points with no visible pattern

G, 2G, 3G … 97G on $y^2 = x^3 - x + 1 \bmod 97$



Using a prime field

ECC can use a finite field whose elements are the remainders (0,…,p-1). After point addition, reduce the result modulo the prime (p).

Example: x = 5

$5^3 - 5 + 1 = 121$, and $121 \bmod 97 = 24$. Since $11^2 = 121 \equiv 24$, both (5, 11) and (5, 86) are on the curve

Every dot is a multiple of G

Adding G = (0, 1) to itself again and again visits all 97 points, hopping in this scattered order

P-256: a 256-bit prime gives about 2^{256} points

01

Elliptic curves

Adding G to itself k times hides k, which gives ECC its trapdoor

1 One addition hides nothing

Given P and P + Q, anyone can add −P and get Q back, so a single addition can’t protect a secret

2 So repeat it from a public point G

Add G again and again: G, 2G, 3G … Over a prime field each hop lands somewhere with no visible link to the last

3 The number of hops becomes the secret

Seeing only the final point k·G, nobody knows a fast way to work back to k

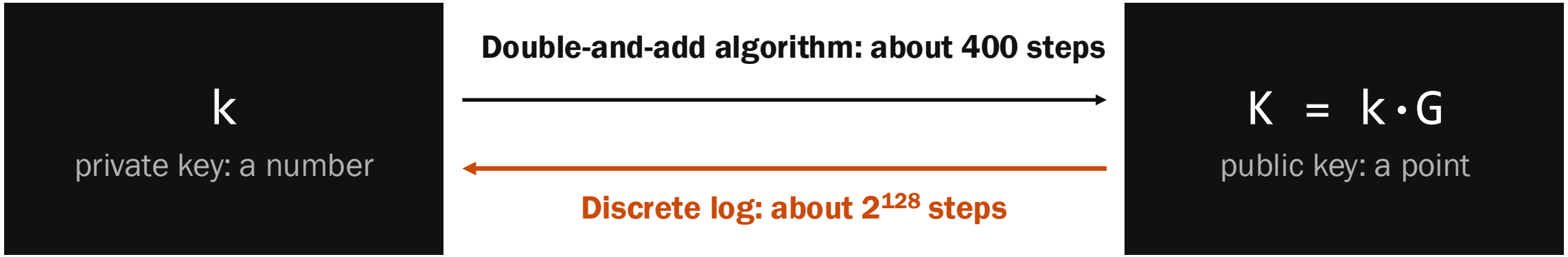
$y^2 = x^3 - x + 1 \bmod 97$, $G = (0, 1)$

k	k·G
1	(0, 1)
2	(73, 84)
3	(56, 31)
4	(57, 62)
5	(67, 26)
6	(28, 87)
…	…
99	(0, 1) again

01

Elliptic curves

A private key is a number k; the public key is the point k·G



G is public

Everyone uses the same starting point G, fixed by the curve standard along with a and b

Fast forward

Double G repeatedly (2G, 4G, 8G …) and add the doublings that match the bits of k

No shortcut back

Recovering k from K is the elliptic curve discrete log problem; the best known attack on P-256 needs about 2^{128} additions

01

Elliptic curves

ECC does RSA's jobs: key exchange and signatures

Key exchange: ECDH

$a \cdot (b \cdot G) = b \cdot (a \cdot G)$

Alice sends a·G, Bob sends b·G, and both compute the same secret point. Thursday covers this as Diffie-Hellman

Used in: TLS 1.3 (X25519), Signal, SSH

Signatures: ECDSA, EdDSA

$\text{sign}(k, m) \rightarrow \sigma$

Only the holder of k can make a signature that verifies against the public point K = k·G

Used in: TLS certificates, code signing, today's exercise

Standard curves

P-256, Curve25519

P-256 (NIST, 2000) dominates certificates; Curve25519 (2006) was designed to be fast and hard to implement wrong

Used in: Ed25519 signatures, X25519 key exchange

Smaller: an Ed25519 signature is 64 bytes; an RSA-3072 signature is 384

01

Elliptic curves

Who picks the constants matters: Dual_EC showed a standard can hide a backdoor

2006

NIST standardizes Dual_EC

A random number generator built on two curve points, P and Q, with no explanation of where Q came from

2007

A possible backdoor

Shumow and Ferguson show that whoever knows how P and Q relate can predict the output

2013

Snowden documents

Reporting ties the design to the NSA; NIST tells users to stop and removes Dual_EC from the standard in 2015

The lesson: a constant with no stated origin can hide a trapdoor for whoever chose it

01

Elliptic curves

Curve25519 explains every constant; P-256 asks you to trust a seed

Constant	Curve25519	P-256
Prime p	**$2^{255} - 19$** Prime just below a power of 2, so mod p is fast	**$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$** Also chosen for fast mod p
Equation	**$y^2 = x^3 + 486662x^2 + x$** Smallest coefficient that passes the published security checks	**$y^2 = x^3 - 3x + b$** b is the SHA-1 hash of a seed, c49d3608..., never explained
Base point	**$x = 9$** Smallest x that gives a valid generator	**Fixed G** Unexplained, but harmless: any point works equally well

Based on rules

Anyone can rerun Curve25519's rules and get the same numbers, so a secretly weak curve couldn't be chosen

Uses an unexplained seed

A designer could hash seed after seed until one gave a curve with a flaw only they knew. None has been found, and a **$12k bounty** awaits whoever explains the seed. Bounty claimed October 2026.

The quantum threat and post-quantum cryptography

02

ECC is no safer than RSA against a quantum computer

Why we moved to elliptic curves

A cost problem

- **Cause:** factoring attacks keep improving, so RSA keys keep growing
- **Fix:** the same security from keys 12× smaller
- **Payoff:** faster handshakes and smaller certificates

Why we are moving to post-quantum

A threat problem

- **Cause:** Shor's algorithm breaks RSA and ECC alike
- **Fix:** new hard problems with no known quantum attack
- **Price:** keys and signatures grow 35–40×; each handshake carries about 2 KB more

Qubits: why ECC is no safer, and maybe less

A qubit is a quantum bit that holds a mix of 0 and 1 at once. Shor's algorithm needs enough qubits to hold the numbers it attacks

Both need about **1,200–1,450 error-corrected qubits**, but P-256 needs about **100× fewer steps**, so it breaks in minutes where RSA-2048 takes days[1]

[1] Babbush et al., Google Quantum AI, arXiv 2603.28846 (2026): ECDLP-256 in 1,200–1,450 logical qubits, 70–90M Toffoli gates; Gidney, arXiv 2505.15917 (2025): RSA-2048 in ~1,400 logical qubits, 6.5B Toffoli gates

Shor's algorithm, part 1: factoring reduces to finding when a sequence repeats

One period: r = 4

x	0	1	2	3	4	5	6	7
$7^x \bmod 15$	**1**	**7**	**4**	**13**	**1**	**7**	**4**	**13**

1 Find the period

$7^4 \equiv 1 \pmod{15}$

so $r = 4$

2 Halve it

$7^2 = 49 \equiv 4 \pmod{15}$

3 Take two gcds

$\gcd(4 - 1, 15) = 3$

$\gcd(4 + 1, 15) = 5$

Why it works: 15 divides $7^4 - 1 = (7^2 - 1)(7^2 + 1) = 48 \cdot 50$, but neither factor alone, so each holds part of 15

The catch: for a 2048-bit N the period can be nearly as large as N, far too long to find by listing values

Shor's algorithm, part 2: a quantum computer finds the period through interference

1 Superpose every x

Put 4 qubits into an equal mix of all x from 0 to 15; if you measure at this point, you get a random x, each with chance 1/16

2 Compute 7^x mod 15 once

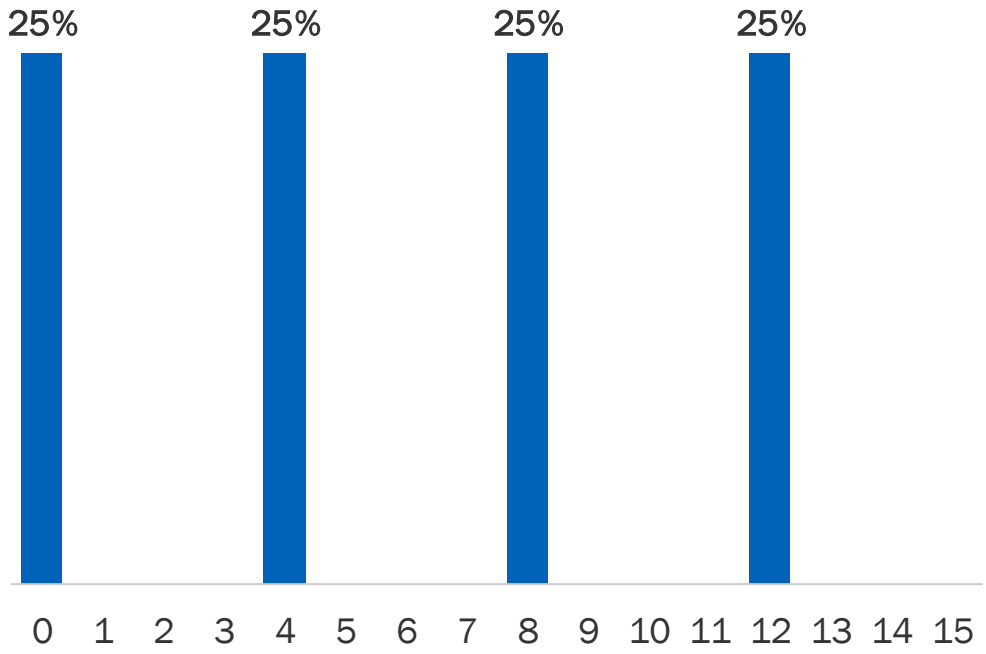
A single evaluation acts on the whole mix, entangling each x with its value

3 Apply the quantum Fourier transform

Like noise-canceling headphones: wrong answers cancel to zero, while outcomes that fit the period reinforce. This is the interference step.

Not "every answer at once": you get one measurement, and interference makes it a useful one

Chance of each outcome in a single run, N = 15, a = 7



Each run returns 0, 4, 8 or 12; 4 or 12 gives r = 4, the others mean run again, and a few runs is usually enough

02

The quantum threat

The same trick breaks the discrete log behind ECC

Shor's algorithm needs one thing: a function with a hidden period

PROBLEM	HIDDEN PERIODIC FUNCTION	THE PERIOD REVEALS
Factoring behind RSA	$f(x) = a^x \bmod N$ repeats every r	p and q
Discrete log behind ECC and DH	$f(a, b) = a \cdot G + b \cdot K$ repeats along k	the private key k

Shor breaks both

Doubling key sizes buys almost nothing: Shor runs in polynomial time, so the fix has to be a different hard problem

Shor's algorithm breaks public-key crypto; Grover's only weakens symmetric crypto

Primitive	Used for	Classical attack	Quantum attack	Fix
RSA-3072	Key exchange, signatures	2^{128}	**Broken (Shor)**	**Replace**
ECC P-256	ECDH, ECDSA	2^{128}	**Broken (Shor)**	**Replace**
AES-128	Symmetric encryption	2^{128}	2^{64} (Grover)	Move to AES-256
AES-256	Symmetric encryption	2^{256}	2^{128} (Grover)	Already safe
SHA-256	Hashing	2^{256}	2^{128} (Grover)	Already safe

Why AES and SHA-256 have no shortcut

Public keys need algebra strangers can use, like RSA's $(m^e)^d \equiv m \pmod n$. That algebra makes values repeat: the period Shor finds. AES and SHA-256 are scrambles with no pattern, so only Grover's search applies

Why 2^{128} is the safe line

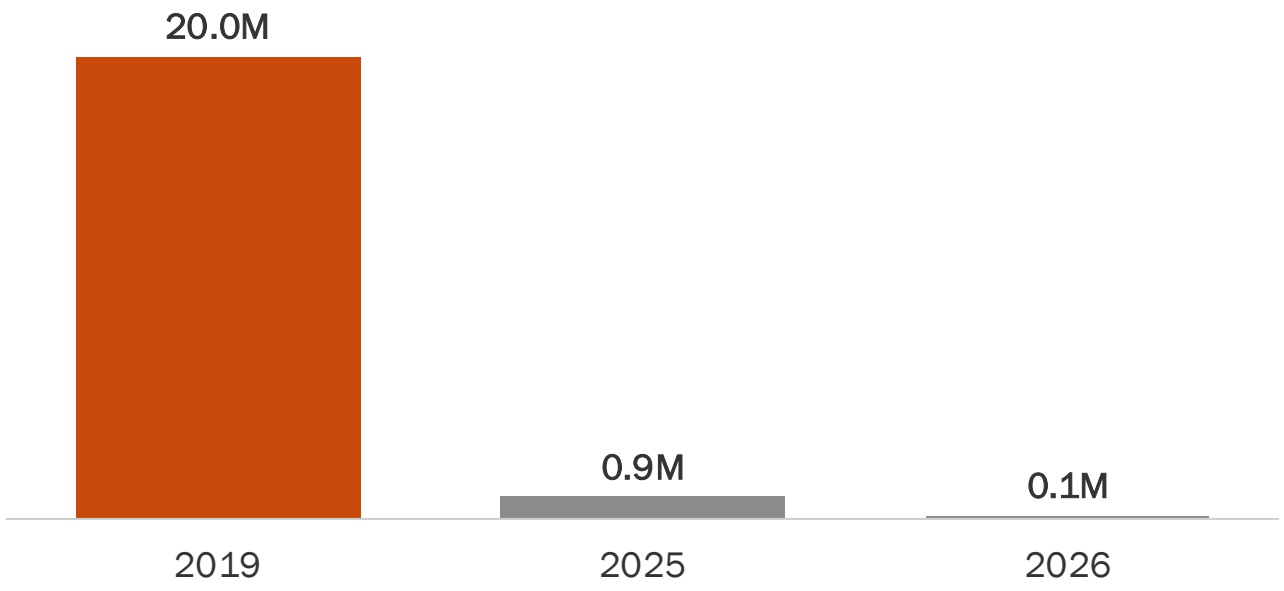
$2^{128} \approx 3 \times 10^{38}$ tries: a billion machines at a billion tries a second would need 800× the age of the universe. Grover's steps run one after another, so extra machines barely help

02

The quantum threat

The machine needed to break RSA-2048 shrank 20× in six years

Physical qubits needed to factor RSA-2048, millions



2019: Gidney and Ekerå · 2025: Gidney · 2026: Iceberg Quantum

The 2026 bar is a design, not a machine: it needs wiring and error correction no one has built, and a month to run

Better algorithms did it

Researchers found cheaper ways to run Shor's algorithm, with the same hardware assumptions as in 2019

ECC is cheaper to break

Google (2026): under 500,000 qubits and about 9 minutes for a 256-bit curve key, versus about 5 days for RSA-2048

Today's hardware

Google's largest chip, Willow, has 105 qubits; these attacks need roughly 1,000 to 10,000 times more

Sources: Gidney, arXiv 2505.15917 (2025); Webster et al., Iceberg Quantum, arXiv 2602.11457 (2026); Babbush et al., Google Quantum AI, arXiv 2603.28846 (2026)

02

The quantum threat

Experts put a code-breaking quantum computer at 28–49% likely within 10 years

28–49%

chance of a code-breaking quantum computer within 10 years

51–70%

within 15 years

Where the numbers come from

A yearly survey of quantum computing researchers, run since 2019; the 2025 round asked 26 of them

The odds keep rising

The 10-year estimate is the highest the survey has recorded, but it is still a guess, not a date

The question to ask

Would you protect 10-year secrets with a lock that has a coin-flip chance of failing?

Source: Mosca and Piani, Quantum Threat Timeline Report 2025, Global Risk Institute and evolutionQ (Mar 2026)

02

The quantum threat

Attackers can record traffic now and decrypt it once quantum computers arrive

Record today
encrypted with RSA or ECC

→

Store for years
storage is cheap

→

Decrypt after Q-day
a quantum computer finds the keys

2022
Cloudflare turns on post-quantum key exchange for all sites

2024
NIST publishes the first post-quantum standards

2029
Cloudflare and Google target full post-quantum security

2030
US deprecates RSA and ECC; Germany by 2030–31

2035
US guidance: disallow RSA and ECC

Mosca's rule: if years data must stay secret + years to migrate > years until Q-day, you are already late

Why ship a draft in 2022: traffic recorded now can never be protected later, so Cloudflare used draft Kyber, paired with X25519 in case the draft was flawed

Sources: Cloudflare blog, "Cloudflare targets 2029 for full post-quantum security" (Apr 2026) and post-quantum executive order post (2026); Global Risk Institute briefing on BSI guidance (Mar 2026); NIST FIPS 203–205 (Aug 2024)

Upgrade encryption first: recorded traffic is at risk today

URGENT NOW

Encryption: key exchange

- **Attack already running:** recorded traffic can be decrypted after Q-day
- **Can’t fix later:** upgrading then won’t protect what was already recorded
- **Easy to swap:** negotiated fresh on every connection

URGENT BY Q-DAY

Signatures and certificates

- **Checked once:** a signature only has to hold up during the handshake
- **No harm in hindsight:** a future quantum computer can’t forge a past connection
- **Slow to swap:** every certificate must be reissued

Both need a new hard problem

Bigger RSA or ECC keys won’t help, so we need a problem with no hidden period for Shor to find: **lattices**

03

Lattice cryptography

Hide the secret under noise

03

Lattice cryptography

Lattice problems have no hidden period for Shor to find

What Shor's algorithm needs, problem by problem

BROKEN BY SHOR

Factoring and discrete log

A hidden period exists, so the quantum Fourier transform finds it in polynomial time

BELIEVED QUANTUM-SAFE

Lattice problems

No period to find: the best quantum attacks are only slightly faster than the best classical ones

Believed, not proven

No one has proven lattice problems are hard, but nearly 30 years of attacks have found no shortcut, classical or quantum

03

Lattice cryptography

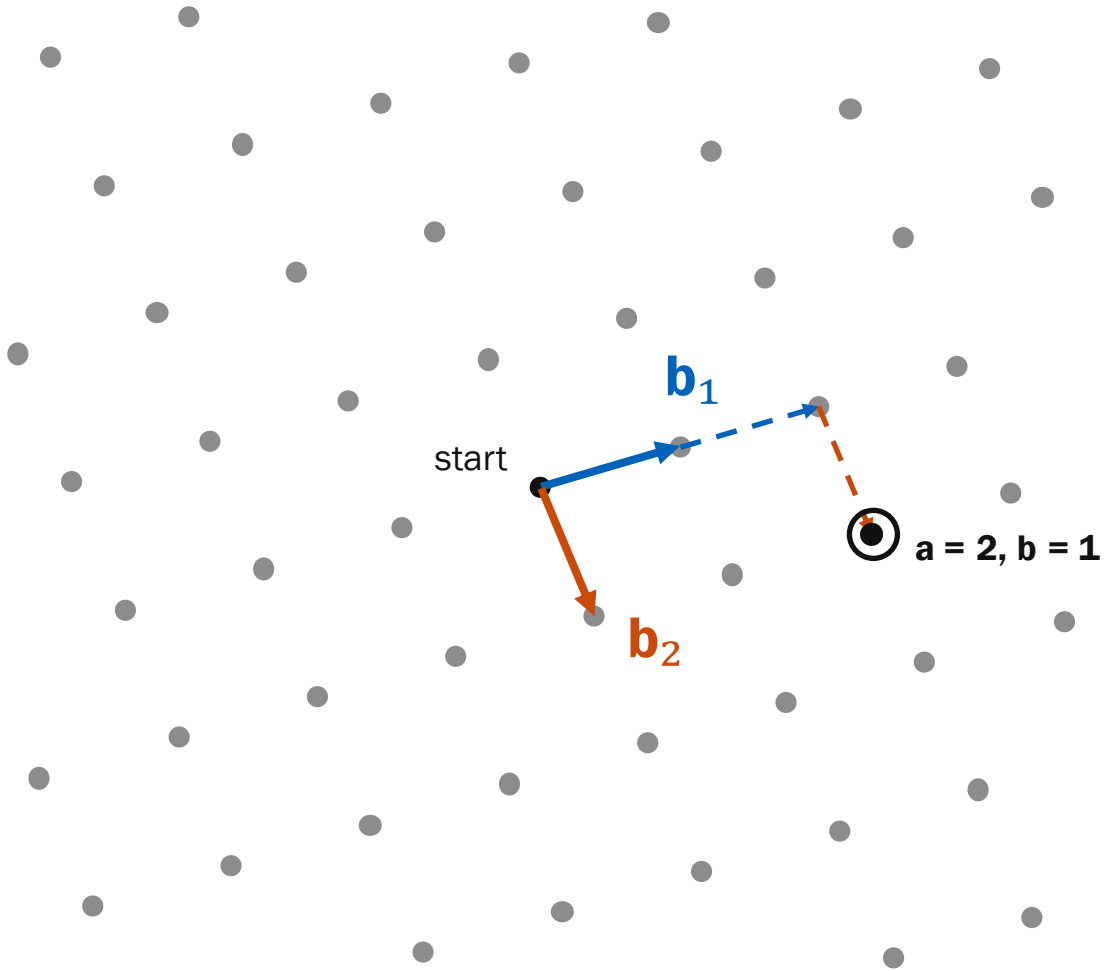
Every trapdoor has the same three parts; lattices swap in a new hard problem

Scheme	Easy direction	Hard direction	Secret that undoes it
RSA	Multiply $p \cdot q$	Factor n	The primes p and q
ECC	Compute $k \cdot G$	Find k from $k \cdot G$	The number k
Lattices	Add small noise to a lattice point	Remove the noise	The secret vector s

The plan for this section

See what a lattice is, why finding a point hidden under noise is hard, how the secret s makes it easy, then use it to agree on a key

A lattice is a grid of points built from a few basis vectors



Built from a basis

Every point is $a \cdot b_1 + b \cdot b_2$ for whole numbers a and b: start at the black point, step a times along b_1 and b times along b_2. The circled point is $\mathbf{2 \cdot b_1 + 1 \cdot b_2}$, so a = 2, b = 1

Regular and endless

The pattern repeats forever in every direction, like atoms in a crystal

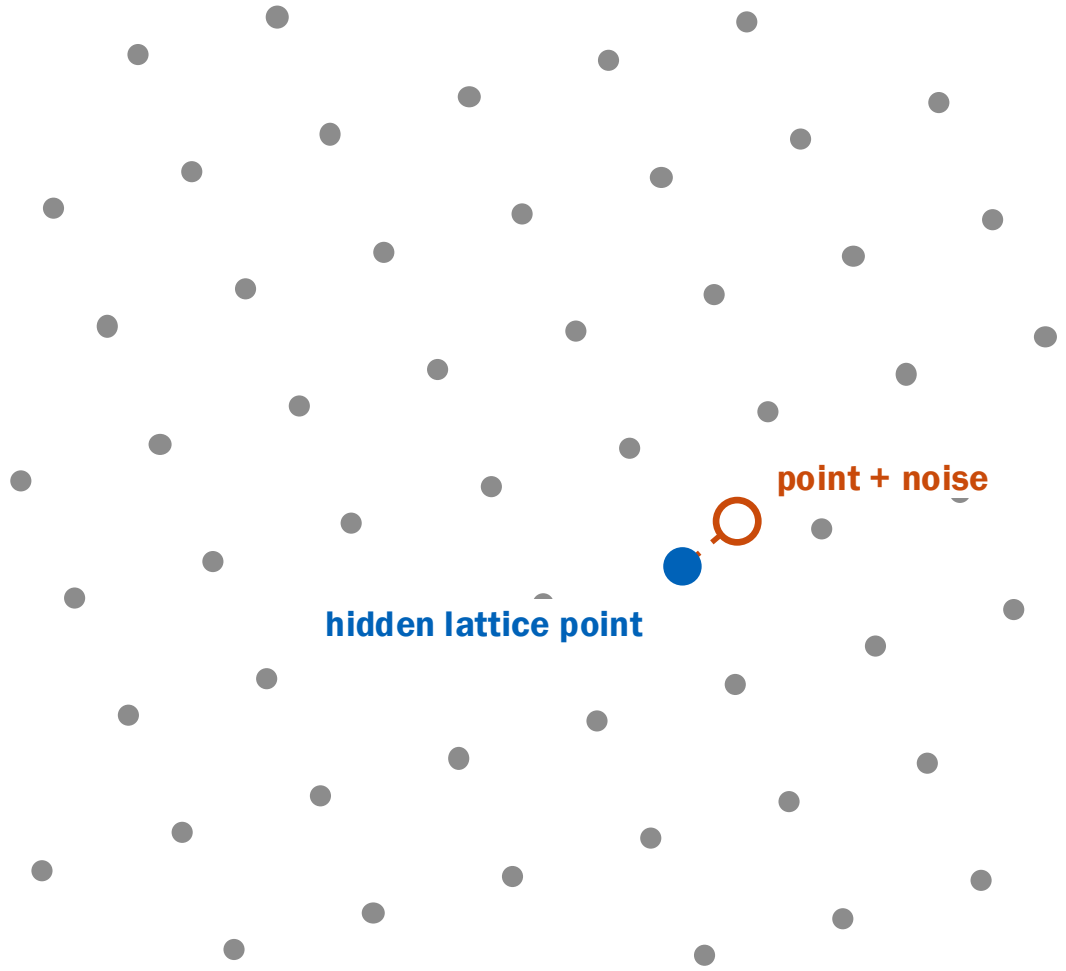
Hundreds of dimensions

Our picture has 2 dimensions; ML-KEM-768 works in 768, where intuition about grids stops helping

03

Lattice cryptography

The hard problem: find the lattice point hidden under a little noise



1 Pick a hidden point

Choose a lattice point and keep it secret

2 Nudge it off the grid

Add a small random shift: this is the noise

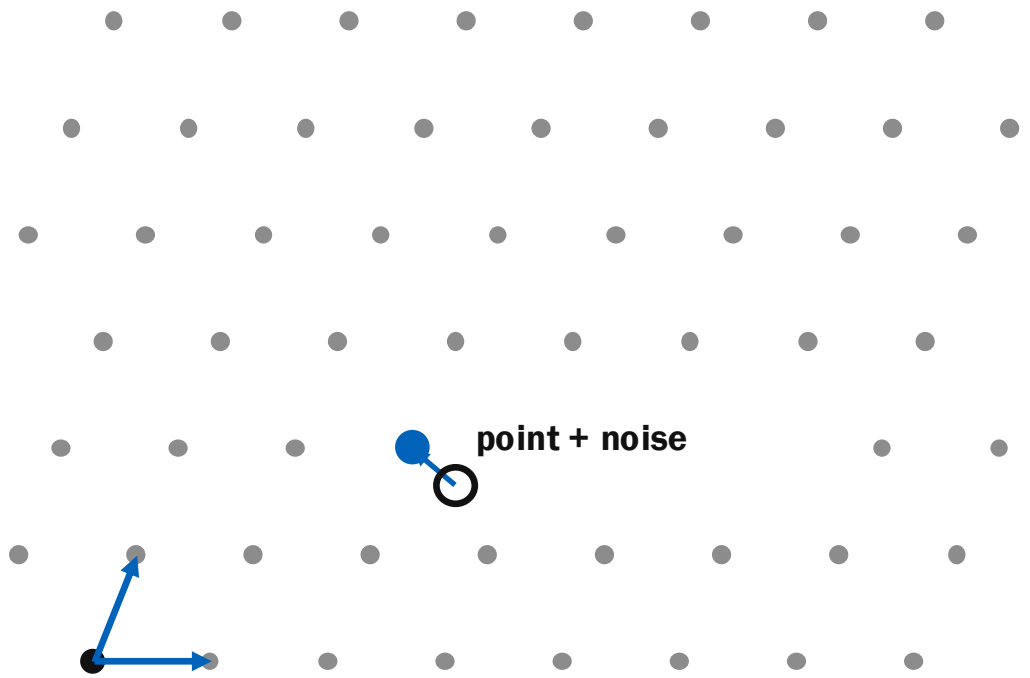
3 Ask where it came from

In 2D you can eyeball it. In 768 dimensions nobody knows a fast way, even with a quantum computer

Called the closest vector problem

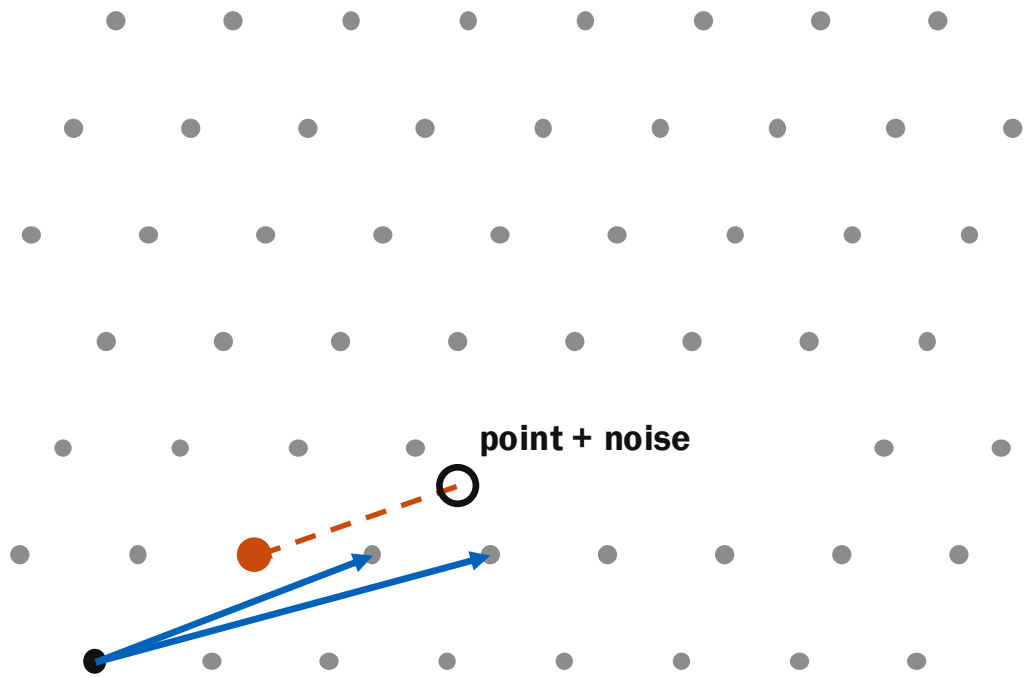
Intuition: The secret vector s works like a good basis: with it, rounding finds the hidden point

Picture intuition: in ML-KEM the secret is a vector s; knowing it strips the noise the way a good basis rounds correctly



Good basis: short and square

Private key: point + noise = $2.3*b_1 + 0.8*b_2$ – you can round to 2 and 1 and land on the hidden point



Bad basis: long and skewed

Public key: this basis produces the same lattice, but rounding lands on the wrong point

The same hidden-point problem, written as equations

IN THE PICTURE

Lattice point

IN EQUATIONS

$A \cdot s$

A is a public random matrix; s is a secret vector of small numbers. Every product of A and s lands on a lattice point

IN THE PICTURE

Noise

IN EQUATIONS

e

A short vector of small errors, each −1, 0 or +1

IN THE PICTURE

Point + noise

IN EQUATIONS

$b = A \cdot s + e$

What everyone sees: a point just off the lattice

Public key: (A, b) **Private key:** s

Recovering s from (A, b) is **learning with errors (LWE)**: the closest vector problem written as equations

Linear equations are easy to solve until you add small noise

Secret $s = (s_1, s_2, s_3)$, all arithmetic mod 17

Exact equations

$$4s_1 + 7s_2 + 1s_3 \equiv 15$$
$$2s_1 + 9s_2 + 5s_3 \equiv 10$$
$$8s_1 + 3s_2 + 6s_3 \equiv 0$$
$$1s_1 + 6s_2 + 11s_3 \equiv 4$$

Gaussian elimination recovers s = (3, 5, 2) in a few steps

Each answer off by −1, 0 or +1

$$4s_1 + 7s_2 + 1s_3 \equiv 16$$
$$2s_1 + 9s_2 + 5s_3 \equiv 9$$
$$8s_1 + 3s_2 + 6s_3 \equiv 0$$
$$1s_1 + 6s_2 + 11s_3 \equiv 5$$

Elimination multiplies the errors until the answer is noise; at 768 unknowns no known method recovers s

Learning with errors (LWE): given many pairs (a, a·s + e) with small e, find s. ML-KEM is built on this problem

Key exchange in three steps: only the holder of s can strip the noise

Bob wants to share a random key with Alice. We follow one bit of it; ML-KEM sends all 256 bits at once



Eavesdropper: sees A, b, u and v, but recovering the bit without s is an LWE problem

Encryption and decryption in equations: only s cancels the big terms

ALICE: KEYS

Pick small s and e

$$b = A \cdot s + e$$

Publish (A, b); keep s

BOB: ENCRYPT ONE BIT

Pick small r, e_1, e_2

$$u = r \cdot A + e_1$$

$$v = r \cdot b + e_2 + bit \times 1665$$

ALICE: DECRYPT

Compute with her secret s

$$v - u \cdot s$$

then round to 0 or 1 (next slide)

Why decryption works: the big terms cancel

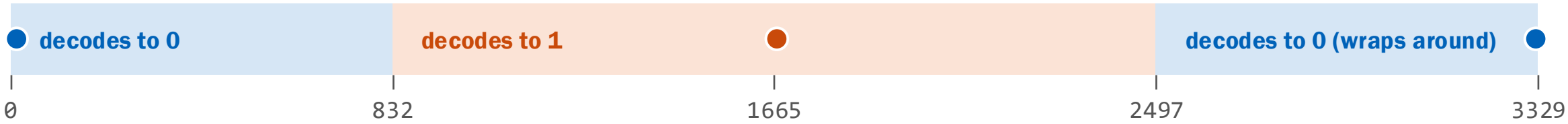
$$v - u \cdot s = bit \times 1665 + (r \cdot e + e_2 - e_1 \cdot s)$$

Both sides contain $r \cdot A \cdot s$, which subtracts away. The leftover noise is small because r, s and every e are small

Eavesdropper: without s, $r \cdot A \cdot s$ never cancels; finding s from (A, b) is LWE

Alice’s last step: round away the noise to read Bob’s bit

What decryption leaves behind, mod q = 3329: the dots at 12, 1671 and 3325 decode to 0, 1 and 0



1 Bob adds his bit

Add bit × 1665 to a noisy LWE value built from the public key

2 Alice cancels with s

The secret key cancels the big terms, leaving bit × 1665 plus small noise

3 Alice rounds

Near 0 reads as 0, near 1665 as 1; it works while the noise stays under 832

In ML-KEM: v is a polynomial with 256 coefficients, so Bob sends a whole 256-bit key in one step, and both sides hash it into a session key. This means each coefficient is multiplied by 1665 with its own noise.

03

Lattice cryptography

NIST standardized three post-quantum schemes in August 2024

FIPS 203 · **LATTICE**

ML-KEM

formerly Kyber

Key encapsulation: two parties agree on a shared secret, built on learning with errors

Used for: TLS key exchange today

FIPS 204 · **LATTICE**

ML-DSA

formerly Dilithium

Digital signatures from the same lattice family; the expected default replacement for ECDSA

Used for: certificates, code signing

FIPS 205 · **HASH-BASED**

SLH-DSA

formerly SPHINCS+

Signatures built only from hash functions, a backup if lattices fall; larger and slower

Used for: firmware, long-lived root keys

A familiar foundation: SLH-DSA rests on the same SHA-2 and SHA-3 hashes from last week’s lecture

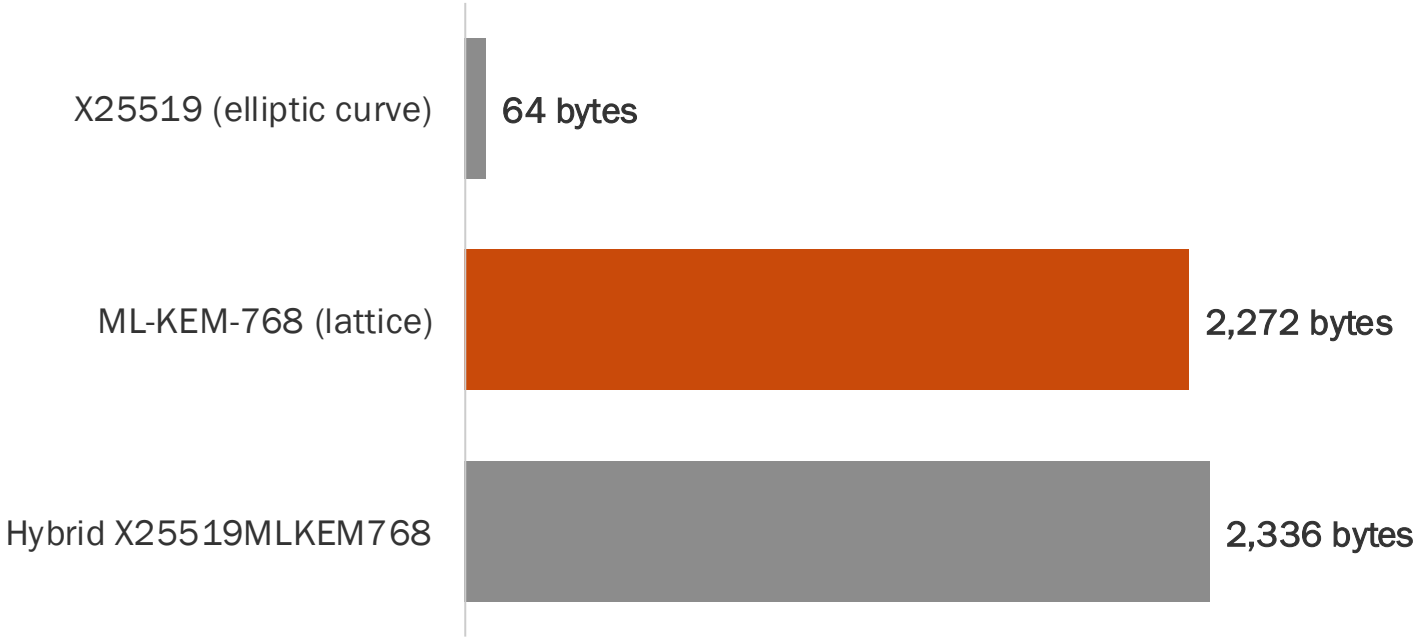
Backup KEM coming: HQC, built on codes instead of lattices so a lattice break won’t take down key encapsulation too; standard expected 2027

03

Lattice cryptography

Lattice key exchange sends 35 times more bytes than elliptic curves

Bytes exchanged to agree on one key, client plus server



35×

more data for ML-KEM-768 than for X25519

Still fast enough

2.3 KB adds little delay to a handshake; the hard engineering was packet sizes and old middleboxes that choked on big messages

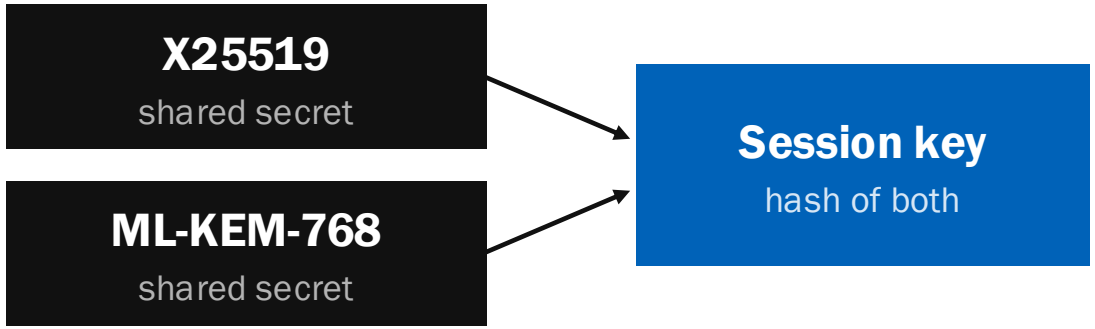
Source: Cloudflare blog, “Prepping for post-quantum: a beginner’s guide to lattice cryptography” (Mar 2025); hybrid total is the sum of both shares

03

Lattice cryptography

Encryption has largely moved to post-quantum; signatures are the harder half

Encryption: hybrid key exchange



Safe if either one holds: a lattice flaw still leaves X25519

2/3+ **of human TLS traffic to Cloudflare used this hybrid by April 2026**

Signatures: the harder half

Scheme	Public key	Signature
Ed25519	32 B	64 B
ML-DSA-44	**1,312 B**	**2,420 B**

Sizes multiply: a handshake carries a certificate chain with several keys and signatures, so 64 bytes becomes kilobytes

Less urgent: a forgery has to happen live, so signatures only need to switch before Q-day

Sources: Cloudflare blog, post-quantum IPsec post (Apr 2026); FIPS 204 parameter sizes

Summary: three trapdoors

Scheme	Hard problem	Why we adopted it	Public key[1]	Quantum-safe
RSA	Factoring $n = p \cdot q$	First practical public-key scheme (1977)	384 B	**No**
ECC	Elliptic curve discrete log	Same security from 12× smaller keys	32 B	**No**
Lattices	Finding a lattice point under noise (LWE)	No known quantum attack	1,184 B	**Believed yes**

Two different motivations: ECC solved a cost problem; lattices solve a threat problem, at a higher cost

1. RSA-3072 and X25519 at 128-bit security; ML-KEM-768 targets a level comparable to AES-192

Class exercise: ECDSA signatures with Rust Crypto

- **Crates:** Rust Crypto, p256 with ECDSA
- **Code examples:** ECDSA signatures in the Rust Cryptography repo
- **Extension:** make an ML-KEM-768 key pair with the ml-kem crate and compare its size with a p256 key