ISE Cryptography – Lecture 09

Post-Quantum Cryptography

The Threat

Why are we having this lecture? RSA still works.

Shor's Algorithm

- Shor (1997) showed a quantum computer can solve integer factorisation and discrete logarithm in *polynomial time*
  - Classical best: sub-exponential (number field sieve for factoring, index calculus for DL)
  - Quantum: polynomial – Shor reduces factoring to period-finding, which a quantum Fourier transform solves efficiently, a categorical leap no classical speedup can match
- If a sufficiently large fault-tolerant quantum computer (CRQC) is built, the following are broken:
  - RSA (factoring), finite-field DH (discrete log), ECDH and ECDSA (elliptic-curve DL)
- Not a side-channel attack, not a protocol weakness: a direct attack on the underlying hard problem
- CRQCs do not exist today: the question is when, not if

What Survives a Quantum Adversary?

- Grover's algorithm (1996) gives a *quadratic* speedup for unstructured search
  - Against symmetric encryption:
    - AES-128 drops to 2^{64} (security margin halved; Grover is serial so this is not equivalent to classical 2^{64}, but the margin is thin – not recommended for new systems)
    - AES-256 drops to 2^{128} (comfortable margin)
  - Against hash functions: preimage resistance loses half its bit-strength
- SHA-2 and SHA-3 under Grover:
  - SHA-256: 2^{128} quantum preimage security (**usable**), 2^{128} is the accepted minimum
  - SHA-384 / SHA-512: 2^{192} / 2^{256} (comfortable margins)
  - SHA3-256: same 2^{128} quantum security level as SHA-256, though a different construction
- Collision resistance is less exposed: without quantum RAM, the best known quantum collision attacks match the classical birthday bound
  - SHA-256 collision resistance stays at 2^{128} under realistic quantum threat models
  - With quantum RAM (speculative hardware), the BHT algorithm achieves $2^{n/3}$ collisions: 2^{85} for SHA-256; a distant concern, but relevant in high-assurance threat models

What Breaks and What Holds

Classical Asymmetric

- RSA
- Finite-field DH
- ECDH
- ECDSA / EdDSA

Broken by Shor's algorithm

Symmetric and Hash

- AES-256
- ChaCha20
- SHA-256 / SHA-384
- SHA3-256
- AES-128 (thin margin)

Weakened by Grover (use AES-256 for new systems)

Harvest Now, Decrypt Later (HNDL)

- An adversary can record encrypted traffic today and store it
  - They cannot decrypt it now, but they do not need to
  - When a CRQC arrives, they decrypt retrospectively
- The relevant question is not “when will CRQCs arrive?” but:
  - **“What is the confidentiality lifetime of my data?”**
  - Medical records, state secrets, intellectual property, key-exchange transcripts
- If your data needs to remain confidential for 10–20 years, and CRQCs may arrive in 10–20 years, the migration deadline is **today**
- Intelligence agencies have been collecting TLS traffic for years
- NIST IR 8547 (2024) identifies the migration deadline as the mid-2030s for critical systems

The Threshold Concept: Hardness Is Not a Fact

- We did not discover that factoring is hard
  - We observed that no efficient classical algorithm has been found in 50 years of looking
  - We built RSA on that empirical gap
- Shor's algorithm does not prove RSA was wrong: it proves the adversary model changed
  - The mathematical fact (factoring is in NP) did not change
  - The assumption (no efficient algorithm exists in the relevant adversary model) did change
- **Cryptographic hardness is a defended position in an ongoing argument with cryptanalysts, not a mathematical fact**

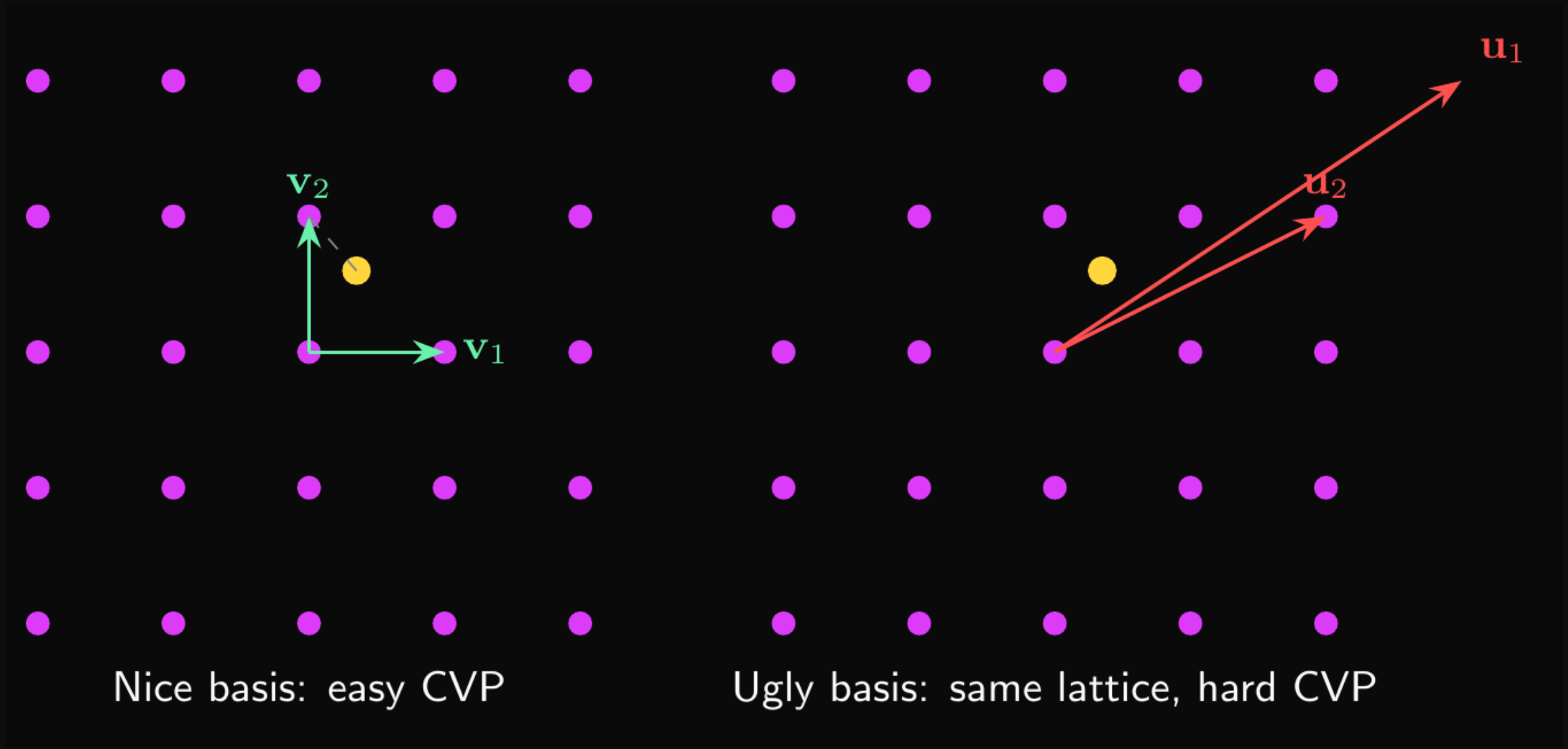
Lattices and LWE, Intuitively

Where does asymmetric hardness come from, if not from groups?

What Is a Lattice?

- Start with basis vectors $\mathbf{b}_1, \mathbf{b}_2 \in \mathbb{R}^2$: two linearly independent vectors pointing in different directions
  - In n dimensions: n linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^m$
- Scale each by any integer: $a_1\mathbf{b}_1$ for $a_1 \in \mathbb{Z}$, $a_2\mathbf{b}_2$ for $a_2 \in \mathbb{Z}$
  - Real multiples fill a continuous plane, while integer multiples give a discrete grid of points
- The lattice Λ is every point reachable by adding scaled basis vectors
  - $\Lambda(\mathbf{b}_1, \mathbf{b}_2) = \{a_1\mathbf{b}_1 + a_2\mathbf{b}_2 \mid a_1, a_2 \in \mathbb{Z}\}$
  - Geometrically: tile the plane with copies of the parallelogram spanned by $\mathbf{b}_1$ and $\mathbf{b}_2$
- The same set of lattice points can be described by infinitely many different bases
  - $(\mathbf{b}_1, \mathbf{b}_2)$ and $(2\mathbf{b}_1 + \mathbf{b}_2,\ \mathbf{b}_1 + \mathbf{b}_2)$ generate the same lattice
  - A *nice* basis (short, nearly orthogonal) makes the structure obvious, while an *ugly* basis (long, skewed) hides it

The Same Lattice, Two Different Bases



SVP and CVP

- Two canonical hard problems on lattices:
  - **Shortest Vector Problem (SVP):** given a basis, find the shortest non-zero vector in the lattice
  - **Closest Vector Problem (CVP):** given a basis and a target point, find the nearest lattice point
- With a nice (short, orthogonal) basis: both are easy, just round to the nearest combination
- With an ugly (long, skewed) basis: both are conjecturally hard, even for quantum adversaries
  - No known quantum algorithm gives a polynomial speedup for worst-case lattice problems
  - Best quantum algorithms for lattice problems: still exponential in the lattice dimension
- **The curse of dimensionality**: the number of candidate vectors grows exponentially with dimension: there is no known shortcut
  - In 2 dimensions SVP is trivial by inspection
  - In hundreds of dimensions even quantum search cannot navigate the space efficiently
  - Unlike RSA or DH (empirical hardness: no efficient algorithm found in decades of looking), lattice hardness has a geometric explanation
  - We choose the dimension as the security parameter: higher dimension means harder problems, at the cost of larger keys
    - This is the trade-off seen in the ML-KEM parameter comparison

Learning With Errors (LWE), Intuitively

- Start with a system of linear equations: find $\mathbf{s}$ given $\mathbf{A}$ and $\mathbf{b} = \mathbf{A}\mathbf{s}$
  - Straightforward: Gaussian elimination solves this in polynomial time
- Add small random noise to each equation: $b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i$
  - $\langle \mathbf{a}_i, \mathbf{s} \rangle$ is the dot product: $a_{i,1} s_1 + a_{i,2} s_2 + \cdots + a_{i,n} s_n$
  - Each e_i is a small integer sampled from a narrow distribution
  - The noise destroys the linear structure that Gaussian elimination exploits

LWE: Hardness and Trapdoor

- **The LWE problem:** recover $\mathbf{s}$ given many samples $(\mathbf{a}_i,\ \mathbf{b}_i)$ where $\mathbf{b}_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i$
- Regev's reduction (2005): LWE is as hard as solving worst-case SVP on any n-dimensional lattice
  - "Solving LWE would solve the hardest lattice problems" – not just an average-case assumption
  - The original reduction is itself quantum; Peikert et al. later gave a classical reduction with different parameters
- No quantum algorithm is known to give a polynomial speedup for SVP – unlike Shor for factoring, lattice algorithms remain exponential
- Publish $(\mathbf{A},\ \mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e})$ as your public key; keep $\mathbf{s}$ and $\mathbf{e}$ secret
  - An observer sees a noisy linear system and faces the LWE problem to find $\mathbf{s}$
- An encryptor uses $(\mathbf{A}, \mathbf{b})$ to wrap a message; you decrypt by using $\mathbf{s}$ to subtract the noise
- **The trapdoor is $\mathbf{s}$**: LWE hardness makes the public key look random, but the private key makes it structured

From Fields to Rings

- You know two algebraic structures from earlier in the module:
  - **Group**: one operation, e.g. $(\mathbb{Z}_p^*, \times)$ in Diffie-Hellman
  - **Field**: add and multiply (every nonzero element has a multiplicative inverse), e.g. $\mathbb{Z}_p$, $\mathrm{GF}(2^{128})$ in GCM
- In GCM you saw polynomial arithmetic mod an irreducible polynomial: bit coefficients, degree kept bounded by reduction
  - The irreducible modulus ensures every nonzero element is invertible, making it a field
- A **ring** has the same two operations but no guarantee of multiplicative inverses
  - The integers $\mathbb{Z}$: you can add and multiply freely, but $3^{-1} \notin \mathbb{Z}$
  - $\mathbb{Z}_q[x]/(x^n+1)$: polynomials with coefficients in $\mathbb{Z}_q$, reduced mod x^n+1: over $\mathbb{Z}_q$, the polynomial x^n+1 factors (for ML-KEM's $q = 3329$, $x^{256}+1$ splits into 128 quadratic factors), creating zero divisors – so this is a ring, not a field
- Module-LWE works in this ring because:
  - Arithmetic stays over integers: exact, no floating-point rounding
    - Reals would corrupt the small noise values that carry the LWE hardness
  - Key storage drops from $O(n^2)$ (plain integer matrix) to $O(n)$ (ring elements)
  - Multiplication uses NTT (number-theoretic transform): the factorisation of x^n+1 over $\mathbb{Z}_q$ is exactly what makes the NTT

Module-LWE: The Same Idea, More Efficient

- Module-LWE replaces plain integer vectors with vectors of elements from $\mathbb{Z}_q[x]/(x^n + 1)$
  - Same LWE problem structure, same hardness argument, now over ring elements
- The payoff: same security level with keys $O(n)$ in size rather than $O(n^2)$
- The cost: a slightly stronger assumption (hardness of Module-LWE rather than plain LWE)
  - Module-LWE has been extensively studied since 2012 (no structural weaknesses found)
- Both ML-KEM and ML-DSA rest on Module-LWE and Module-SIS hardness

ML-KEM

Can it actually replace ECDH in something I use?

The KEM Interface (Recall from HPKE, Lecture 07)

- A KEM $\mathcal{K} = (G, \text{Encaps}, \text{Decaps})$:
  - $G()$: probabilistic key generation, outputs (pk, sk)
  - $\text{Encaps}(pk)$: outputs (k, c), a shared secret and its encapsulation
  - $\text{Decaps}(sk, c)$: outputs k or `reject`
- Correctness: for all $(pk, sk) \leftarrow G()$, if $(k, c) \leftarrow \text{Encaps}(pk)$, then $\text{Decaps}(sk, c) = k$
- **This interface is unchanged for ML-KEM**, only the math behind each function changes

ML-KEM: Structure

- ML-KEM (FIPS 203, formerly Kyber) is built on Module-LWE
- Without the algebra:
  - **KeyGen:** sample a small secret $\mathbf{s}$ and error $\mathbf{e}$, compute a noisy linear combination as the public key, and keep $\mathbf{s}$ as the private key
  - **Encaps:** given pk, produce a ciphertext that is a noisy linear function of the public key, and the shared secret is derived from the output
  - **Decaps:** use the secret $\mathbf{s}$ to subtract the noise, recover the encoded shared secret, and output k
- The noise is what makes the scheme hard to invert: without $\mathbf{s}$, the ciphertext looks uniformly random
- The underlying PKE scheme is CPA-secure by reduction to Module-LWE hardness; the FO transform (next slide) lifts it to IND-CCA2

The Fujisaki-Okamoto Transform

- The PKE at the core of ML-KEM is IND-CPA secure but not IND-CCA2: given a decryption oracle, an attacker can exploit ciphertext malleability to recover key material
- The Fujisaki-Okamoto (FO) transform lifts the CPA-secure PKE to an IND-CCA2 KEM
  - On decapsulation, re-encrypt the ciphertext using the recovered message and check consistency
  - Any tampered ciphertext is rejected, making the decryption oracle useless
  - FIPS 203 uses implicit rejection: a bad ciphertext returns a pseudorandom value instead of $\perp$, preventing timing-based oracle leakage
- For the derivation: Boneh and Shoup §12
- **FIPS 203 ML-KEM is IND-CCA2 secure under Module-LWE hardness**

ML-KEM Parameter Sets

Parameter set	NIST category	Classical security	Key (pk)	Ciphertext
ML-KEM-512	Category 1	≈ AES-128	800 B	768 B
ML-KEM-768	Category 3	≈ AES-192	1184 B	1088 B
ML-KEM-1024	Category 5	≈ AES-256	1568 B	1568 B

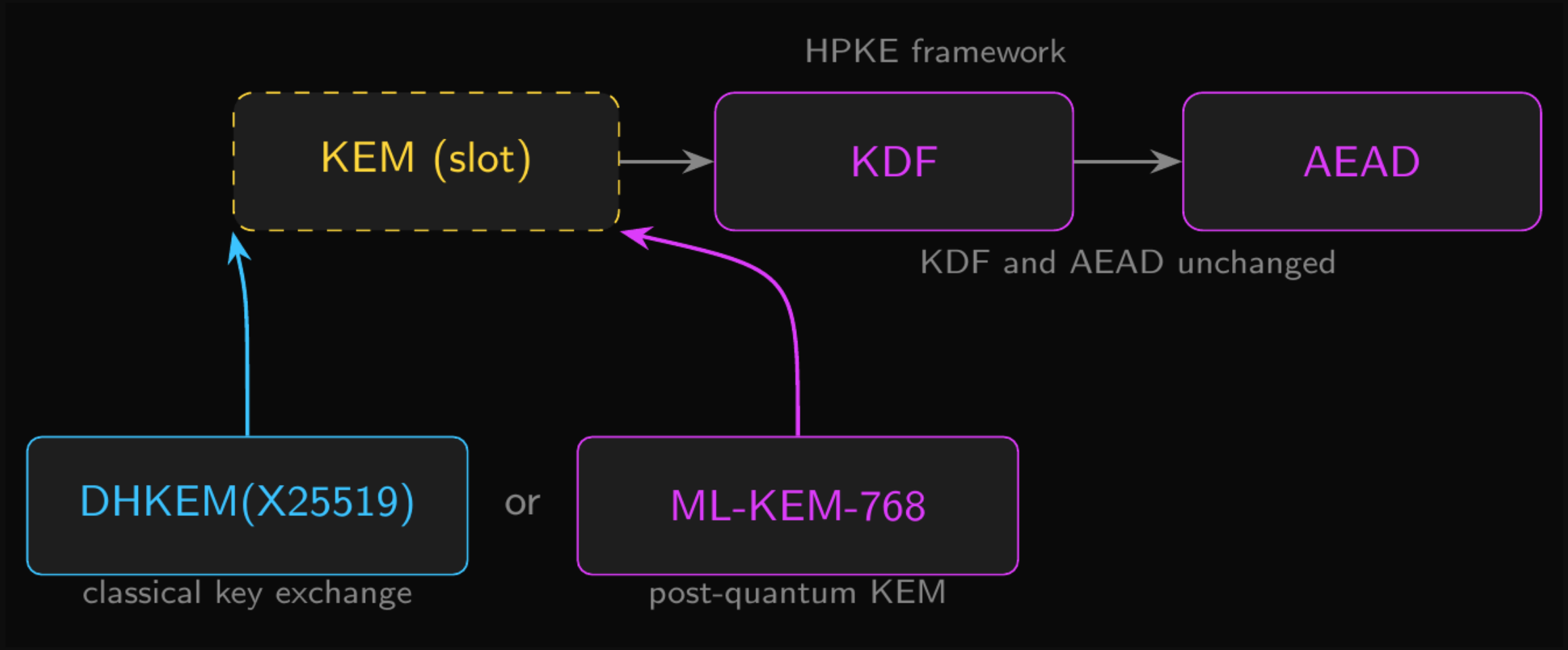
- **ML-KEM-768 is the deployment default** (TLS hybrids, HPKE, etc.)
- NIST security categories: Category 1 ≈ AES-128 classical security, Category 3 ≈ AES-192, Category 5 ≈ AES-256; Category 2 ≈ SHA-256 collision resistance (appears in ML-DSA-44)

ECDH vs ML-KEM: Same Interface, Different Sizes

	X25519 (ECDH)	ML-KEM-768
Public key	32 B	1184 B ($\mathbf{37}\times$)
Ciphertext / ephem.	32 B	1088 B ($\mathbf{34}\times$)
Shared secret	32 B	32 B (same)
KeyGen	$\approx 18\,\mu s$	$\approx 20\,\mu s$ (comparable)
Encaps / agree	$\approx 55\,\mu s$	$\approx 25\,\mu s$ (faster)

Bandwidth is the cost; CPU is not.
liboqs-python, Intel Core i7-1360P

ML-KEM in HPKE: Drop-In Replacement



ML-DSA and SLH-DSA

What about signatures, and why are there two PQ signature standards?

ML-DSA: Structure

- ML-DSA (FIPS 204, formerly Dilithium) is built on Module-LWE and Module-SIS
- **Module-SIS (Short Integer Solution):** given a random matrix $\mathbf{A}$, find a short non-zero vector $\mathbf{z}$ such that $\mathbf{Az} = 0 \pmod{q}$
  - Conjectured hard for both classical and quantum adversaries
- ML-DSA uses a *Fiat-Shamir with Aborts* construction:
  - Fiat-Shamir converts an interactive identification protocol to a signature by replacing the verifier's random challenge with a hash of the commitment
  - An interactive lattice-based protocol (Schnorr-style) is made non-interactive this way
  - The "with aborts" twist: some signing attempts produce responses that would leak the secret key, so the signer rejects and retries
  - Average number of retries per signature: roughly 4–6, a small and bounded constant

ML-DSA Parameter Sets

Parameter set	NIST category	Public key	Signature
ML-DSA-44	Category 2	1312 B	2420 B
ML-DSA-65	Category 3	1952 B	3293 B
ML-DSA-87	Category 5	2592 B	4595 B

- Compare Ed25519: pk = 32 B, sig = 64 B
- ML-DSA-65 public key is $\mathbf{61}\times$ larger, signature is $\mathbf{51}\times$ larger
- Signing and verification are fast (sub-millisecond on modern hardware)

SLH-DSA: A Completely Different Foundation

- SLH-DSA (FIPS 205, formerly SPHINCS+) is built on **hash functions alone**
  - No lattice assumption, no algebraic structure
  - Security reduces to collision and preimage resistance of the underlying hash (SHA-2 or SHA-3 variants)
- Structural sketch:
  - A Merkle tree (a hash tree where each parent node is the hash of its two children) of one-time signature (OTS) keys provides a large pool of signing keys
  - Each signing operation consumes a one-time key from the tree
    - The tree is large enough that this never repeats, and no state is needed to track which keys have been used
- The signing process traverses paths through the tree and produces authentication proofs, while verification re-hashes the path
- SLH-DSA is *stateless*: unlike the earlier hash-based schemes XMSS and LMS (NIST SP 800-208), no counter is needed to track spent one-time keys; XMSS and LMS are deployed in firmware signing today where stateful operation is acceptable

SLH-DSA Parameter Sets

Parameter set	Public key	Signature	Focus
SLH-DSA-SHA2-128f	32 B	17 088 B	fast signing
SLH-DSA-SHA2-128s	32 B	7 856 B	small signature
SLH-DSA-SHA2-256f	64 B	49 856 B	fast signing
SLH-DSA-SHA2-256s	64 B	29 792 B	small signature

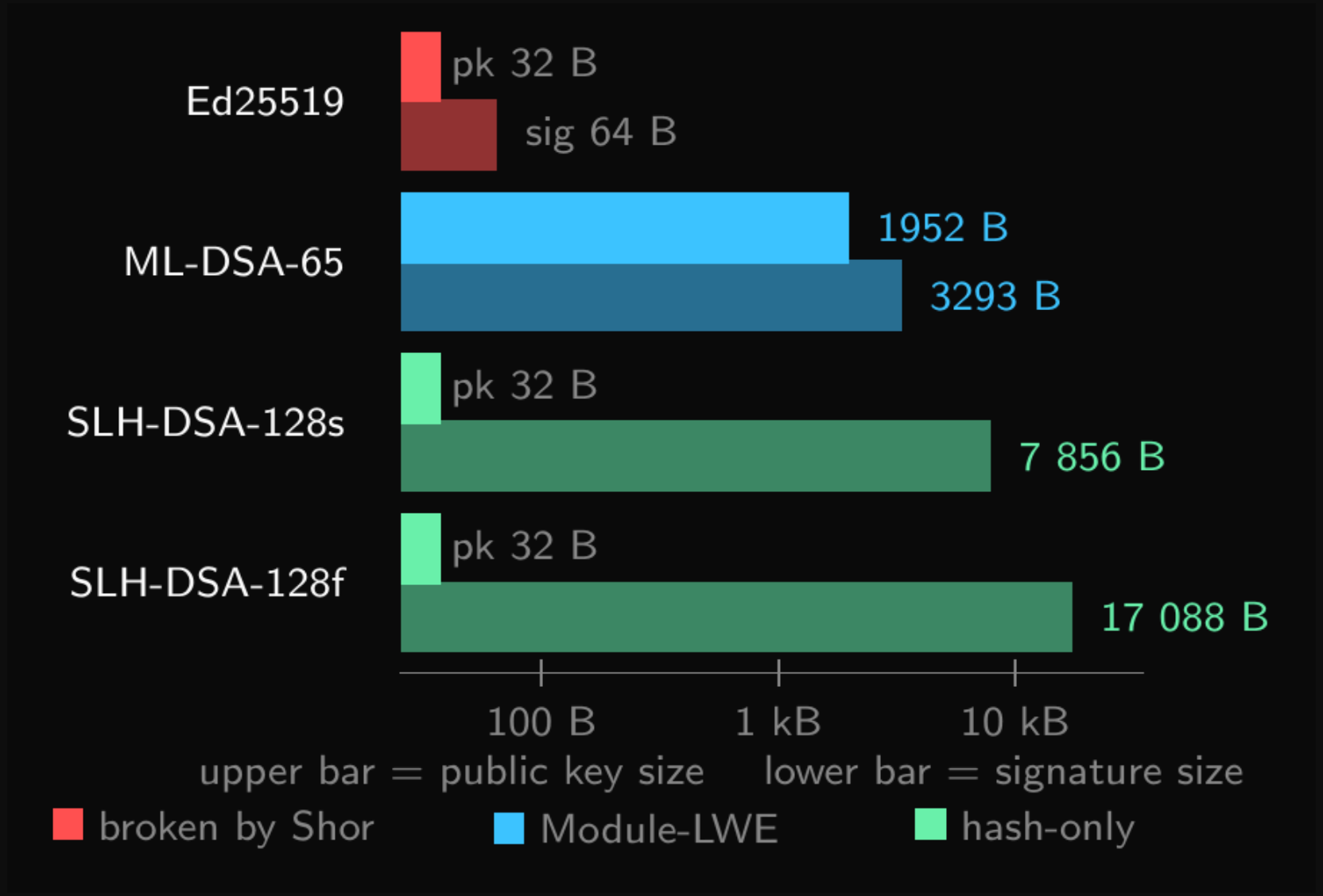
- “f” variants: more tree layers, faster signing, larger signatures
- “s” variants: fewer layers, slower signing, smaller signatures
- Public key is tiny (32–64 B), important for root CA and firmware use cases

Why Three Signature Standards?

The threshold concept from Segment 1 in action: cryptographic hardness is a defended position, not a fact.

- ML-DSA gives reasonable sizes and fast operations
  - The security assumption is Module-LWE/SIS: well-studied *for roughly a decade*
- SLH-DSA gives a fallback whose security rests on hash-function assumptions we have studied since the 1970s
  - If every lattice assumption collapses tomorrow, SLH-DSA still stands
- **The threshold concept revisited:** we are not selecting the "best" signature scheme
  - We are maintaining a defended position under the worst-case scenario
  - Three standards because no single scheme dominates on all axes: speed, size, and assumption diversity trade off against each other
- A third standard, FN-DSA (FIPS 206, Falcon), completes the picture:
  - Smaller signatures than ML-DSA ($\approx$ 1280 B at Category 1, vs ML-DSA-44's 2420 B) and fast verification
  - The cost: a floating-point lattice Gaussian sampler that is hard to implement without timing side-channels
- When to pick which:
  - ML-DSA: default for most applications (TLS certificates, code signing, authenticators)
  - SLH-DSA: long-lived signatures where assumption diversity matters more than signature size (root CAs, firmware, archives)

Signature Size Comparison



Hybrids: Combiners, X-Wing, X25519MLKEM768

Should we just deploy ML-KEM and hope?

The Trust Argument

- ML-KEM has been under public cryptanalytic scrutiny since Kyber's NIST submission in 2017
  - Roughly 8 years of analysis: no structural breaks, some parameter tightening
- X25519 (Curve25519 DH) has been under scrutiny since 2006 and in widespread deployment since ~2014
  - 20 years of analysis, billions of connections per day
- A hybrid scheme combines both: **if either component holds, the combined scheme holds**
  - Classical adversary: X25519 alone is sufficient
    - ML-KEM adds nothing against current threats
  - Quantum adversary: ML-KEM provides the post-quantum security
    - X25519 still holds until Shor is realised
- Hybrid is strictly stronger than either alone under any adversary model we can construct today

Combiner Design: Why Naïve Concatenation Fails

- A naïve combiner: $k_{\text{combined}} = k_{\text{X25519}} \| k_{\text{ML-KEM}}$
  - Entropy is preserved as long as either component stays secret
  - But the concatenation has no formal IND-CCA2 security reduction from the components' individual guarantees
  - Without binding ciphertexts and public keys into the derivation, an attacker can substitute one component's ciphertext independently of the other (context-confusion attacks)
  - The naïve combiner does not pass the secrets through a one-way function: the two keys are never mixed
- A robust combiner: $k_{\text{combined}} = \text{KDF}(k_{\text{X25519}} \| k_{\text{ML-KEM}} \| ct_{\text{X25519}} \| ct_{\text{ML-KEM}} \| pk_{\text{X25519}} \| pk_{\text{ML-KEM}})$
  - The KDF (e.g. HKDF) is one-way and mixing: both secrets contribute to every output bit
  - Binding the ciphertexts and public keys prevents context-confusion attacks

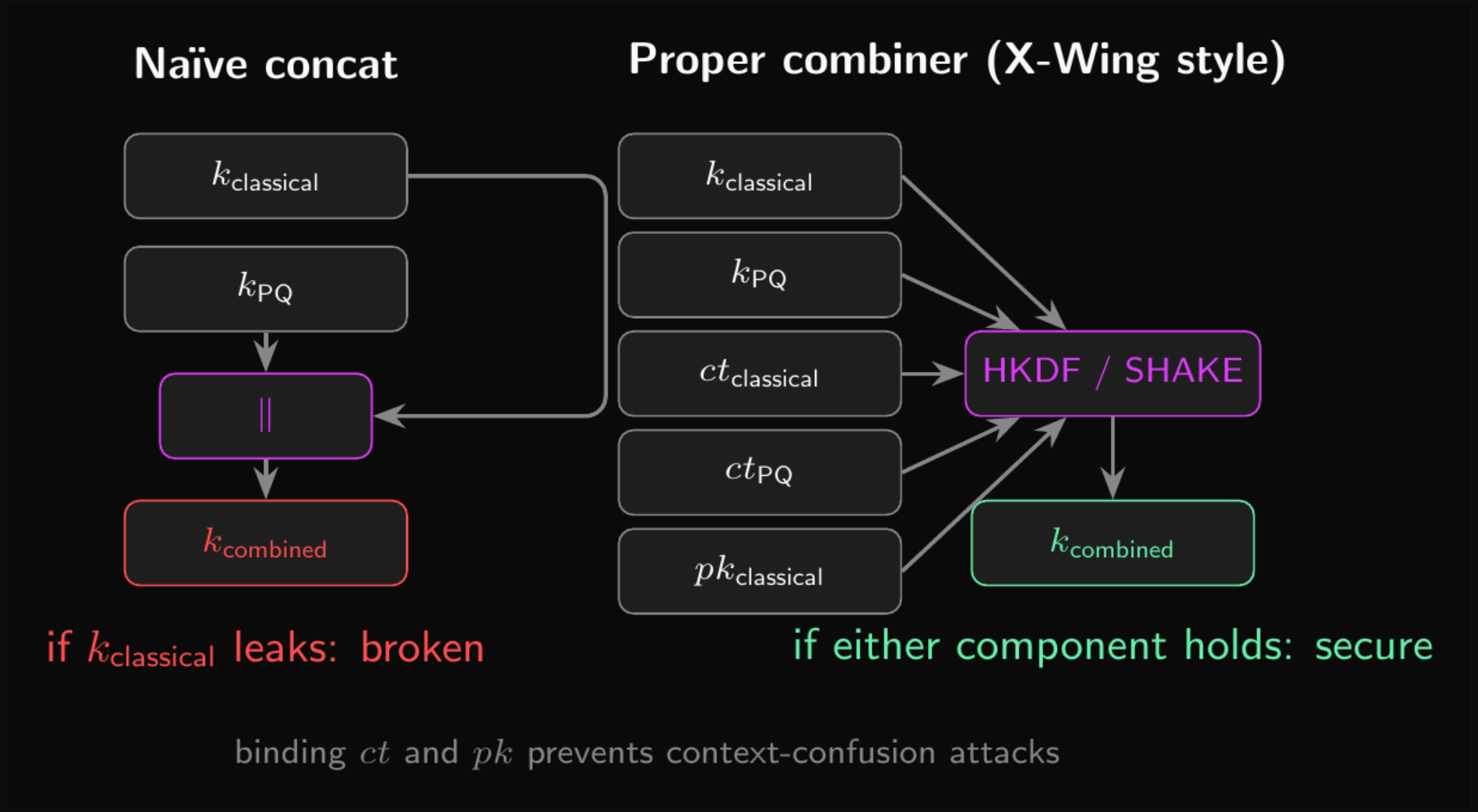
X25519MLKEM768: The Deployed Hybrid

- Specified in the IETF hybrid KEM framework, assigned TLS 1.3 group codepoint
- Structure: concatenate the X25519 and ML-KEM-768 shared secrets and feed through HKDF
  - $k \leftarrow \mathrm{HKDF}(k_{\mathrm{X25519}} \| k_{\mathrm{ML\text{-}KEM768}} \| ct_{\mathrm{X25519}} \| ct_{\mathrm{ML\text{-}KEM768}})$
- Deployed in production since 2024:
  - Chrome (Google switched from Kyber768 draft to ML-KEM-768 hybrid in Chrome 131)
  - Cloudflare, AWS KMS, Fastly
  - OpenSSH (MLKEM768X25519 supported in 9.9+)

X-Wing: A Cleaner Construction

- X-Wing (2024, IACR ePrint 2024/039)
  - A single fixed ciphersuite: X25519 + ML-KEM-768, no parameter negotiation
  - Combiner: $k \leftarrow \mathrm{SHA3\text{-}256}(\mathrm{label} \| k_{\mathrm{X25519}} \| k_{\mathrm{ML\text{-}KEM768}} \| ct_{\mathrm{X25519}} \| pk_{\mathrm{X25519}})$
  - Proven IND-CCA2 under hybrid assumptions with a tight reduction
- **One hybrid KEM you deploy**
  - Unlike X25519MLKEM768, X-Wing is designed as a named primitive with a standalone security proof
  - IETF draft in progress (draft-connolly-cfrg-xwing-kem)

Naïve Concat vs Proper Combiner



Libraries, Benchmarks, and EPIC Pathways

What does this look like in code, and is it realistic to try?

Library Tour

- `liboqs` (Open Quantum Safe project): C library, multi-algorithm, the standard sandbox
  - ML-KEM-512/768/1024, ML-DSA-44/65/87, SLH-DSA variants, and more
  - Python bindings: `liboqs-python` (three lines for keygen/encaps/decaps)
  - Rust bindings: `oqs` crate
- **PQClean**: clean, portable reference implementations, used as upstream by many language bindings
- `pq-crystals` **reference C implementations**: authoritative implementations from the ML-KEM and ML-DSA authors (Bos, Ducas et al.)
- **Python** `cryptography` **package**: ML-KEM landing in mainline post-v43, check current version
- **Recommendation:** start with `liboqs-python`, which covers all NIST schemes and has a stable Python API

Live Benchmark

```python
# liboqs-python: ML-KEM-768 in three lines
import oqs

kem = oqs.KeyEncapsulation("ML-KEM-768")
pk = kem.generate_keypair()
ciphertext, shared_secret_enc = kem.encap_secret(pk)
shared_secret_dec = kem.decap_secret(ciphertext)
assert shared_secret_enc == shared_secret_dec
```

Benchmark output (pre-recorded, modern x86_64):

	keygen	encaps	decaps	pk	ct
X25519	18 µs	55 µs	54 µs	32 B	32 B
ML-KEM-768	20 µs	25 µs	28 µs	1184 B	1088 B
X25519+ML-KEM-768	39 µs	82 µs	83 µs	1216 B	1120 B

The CPU cost is negligible. Bandwidth is the cost.

A1 EPIC Pathway

A credible A1 PQ attempt is *not* "implement Kyber from scratch." It is:

1. **Swap the KEM** in your HPKE-based messaging app for a hybrid (X25519MLKEM768 via `liboqs-python`)
2. **Measure** end-to-end latency and bandwidth impact on the wire
3. **Document** the design trade-off in your Cryptographic Design Document (criterion 4 of the rubric)
4. **Defend** the choice of hybrid vs pure-PQ at interview: why not just ML-KEM-768? (criterion 5)

Prepare to answer: - Why hybrid rather than pure ML-KEM? - What does the bandwidth cost look like, and is it acceptable? - Why did the HPKE interface stay the same? - What security assumptions does your design rest on?

Crypto Agility: The Transferable Principle

- The lesson of post-quantum cryptography is not “use ML-KEM”
- It is: **design systems that can change which KEM they use**
- HPKE’s algorithm-agility design (RFC 9180) was not theoretical cleanliness
  - It was preparation for exactly this moment
  - The same framework absorbs X25519, ML-KEM-768, and hybrids without changing the KDF or AEAD layers
- Every secure messaging protocol, every TLS stack, every key management system built with a fixed-primitive assumption is paying a migration tax right now
- Systems built with the KEM/KDF/AEAD separation absorb the transition cheaply
- **Write code that does not care which KEM is at the KEM slot**

Summary

The Two Things Worth Remembering

- **Hardness is a defended position, not a mathematical fact**
  - Shor changed the adversary model; the math did not change
  - We moved on from RSA because the threat model evolved, not because RSA was wrong
  - Lattice cryptanalysis is younger than RSA cryptanalysis: hold both ML-KEM and SLH-DSA, not just the faster one
- **Design for algorithm agility**
  - HPKE's KEM slot is the architectural pattern: swap the primitive, keep the protocol
  - The migration from classical to PQ asymmetric cryptography is the largest cryptographic transition since DES was retired
  - Systems with clean abstraction boundaries survive this; systems with hard-coded primitives do not

References