Six things I would do differently if I were to design an algorithm to run a quant fund today

Jacqueline Koay

31 May 2026

1. Design the whole stack as if Q-Day is *within* the investment horizon	
Back then: I assumed that cryptography (TLS, VPNs, PKI, HSMs) was "just infrastructure" and effectively unbreakable within any relevant horizon.	**Now:** I'd explicitly model three time scales: • x: Security shelf-life of my data and IP (how long my signals, models and trades remain valuable if stolen). • y: Time required to migrate my whole stack (data, comms, keys, archives) to quantum-safe primitives. • z: Time until credible quantum adversaries exist (collapse time / Q-Day). I would enforce the inequality: $$z > x + y$$ If I can't make that inequality plausible, I treat my current cryptography as already deprecated and design accordingly.

2. Make the algorithm *crypto-agile* by design	

Back then:
I hard-wired assumptions: RSA for key exchange, ECC for signatures, standard TLS, and maybe proprietary obfuscation around the optimiser.
	Now:
	I would build crypto-agility into the architecture:
	• Abstract cryptographic primitives behind interfaces (e.g., KeyExchange, Signature, KEM) so I can swap RSA/ECC → lattice-based / code-based / hash-based without rewriting the optimiser.
	• Use post-quantum key encapsulation mechanisms (KEMs) and signatures for:
	  ○ model deployment,
	  ○ configuration signing,
	  ○ trade instruction integrity,
	  ○ internal service authentication.
	• Treat all long-term secrets (model parameters, alpha signals, proprietary data) as if they will be harvested now and decrypted later.
	My optimiser becomes not just a mathematical object, but a cryptographic citizen of a quantum-aware ecosystem.

3. Protect *alpha* as if it were a cryptographic key	
Back then: Our edge came from model secrecy, data asymmetry and execution. We relied on obscurity plus access control.	**Now:** I would treat: • Signal definitions, • Factor loadings, • Optimiser constraints and penalty structure, • Execution logic as high-value keys, and**:** • Store them in HSMs or secure enclaves with quantum-safe key management. • Use PQ-secure channels between research, production, and execution environments. • Log access with post-quantum signatures so I can prove integrity and detect tampering even in a post-Q-Day world. In other words, I would stop thinking "this is just IP" and start thinking "this is a cryptographic secret with a half-life." That's building resilience into the system.

4. Architect for quantum *use* without quantum *dependence*
If I were building today, I'd assume: • **Near-term:** Access to small, noisy quantum devices or cloud-based quantum services. • **Long-term:** Potential access to fault-tolerant quantum resources. I would design my stack so that: • The core alpha engine remains classical, robust, and profitable without quantum hardware. • Certain subproblems are optionally quantum-accelerated, for example: a. large-scale optimisation (QAOA-style formulations), b. Monte Carlo variance reduction, c. certain combinatorial allocation problems. But I would **never** make the fund's viability *depend* on quantum hardware availability. Quantum becomes a performance enhancer, not a single point of failure. This is operational resilience.

5. Explicitly model quantum-enabled adversaries

Back then:	**Now:**

My adversary model was:
• other quant funds,
• HFTs,
• information leakage,
• maybe state actors - but all classical.
	I would assume:
	• **State-level actors** with future quantum capability can harvest and store our encrypted traffic *today* and decrypt it later.
	• **Competitors** may use quantum resources to:
	  ○ accelerate their own optimisation,
	  ○ perform more powerful inference on your behaviour,
	  ○ attack weak cryptographic links in your infrastructure.
	So I would:
	• Minimise data exhaust (what my trades reveal about my model).
	• Use differential privacy–style thinking for public signals my activity generates.
	• Treat "harvest now, decrypt later" as a real threat to my historical archives, research logs, and model evolution.

6. Governance: align model lifecycle with cryptographic lifecycle

Finally, I would align:

- Model half-life (how long a signal remains predictive)
- Data half-life (how long historical data remains sensitive)
- Crypto half-life (how long a given scheme is considered safe against quantum attack)

I would build processes so that:

- When I retire a model, I also review how its data and parameters are stored and encrypted.
- When a crypto primitive is downgraded (e.g., NIST guidance changes), I trigger a model-and-data re-encryption cycle.
- My risk reports include not just VaR and tracking error, but a "quantum exposure" metric: how much value is at risk if Q-Day arrives earlier than expected.