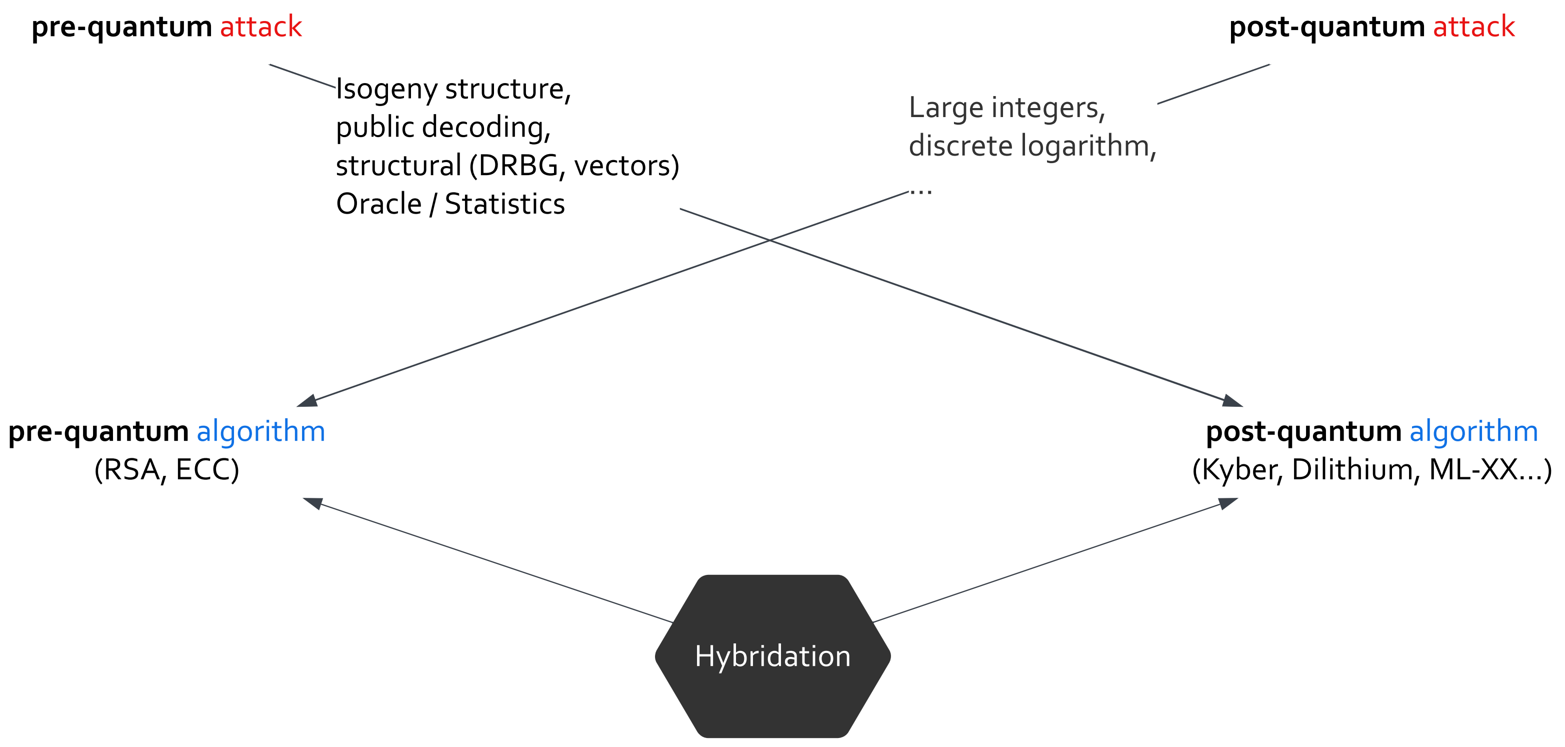
POST-QUANTUM HYBRIDIZATION DESIGNING RESILIENT CRYPTOGRAPHIC TRANSITIONS

Michael VERGOZ
m.vergoz@protonmail.com

WHAT IS AT RISK



SHOR & GROVER

⚡

Shor's Algorithm

- **Goal:** Efficiently factors large integers and solves discrete logarithms **exponentially faster** than classical algorithms.
- **Impact:** It **completely breaks** RSA, Diffie–Hellman, and elliptic-curve cryptography (ECC), since these rely on those hard mathematical problems.

🔍

Grover's Algorithm

- **Goal:** Speeds up unstructured search (e.g., brute force) from **O(N)** to **O(√N)** time.
- **Impact:** It **partially weakens** symmetric algorithms (like AES or SHA-2).
- → Security can be restored by **doubling key sizes** (e.g., AES-256 offers quantum-resistant strength similar to AES-128 today).

🛡

Hash Functions and Grover's Algorithm

- **Grover's algorithm** gives only a *quadratic speed-up* for brute-force search - it doesn't break the underlying structure of hash functions.
- This means existing hashes such as **SHA-256**, **SHA-3**, or **BLAKE2s** remain **secure against quantum attacks**, though with **reduced effective strength**.
- To maintain the same security margin, you can **double the output size**:
    - 256-bit hash → ~128-bit post-quantum security
    - 512-bit hash → ~256-bit post-quantum security

✅ **In practice:** SHA-256 and BLAKE2s are still considered **quantum-resistant enough** for most uses - no need to abandon them.

🔐 Protection through Algorithms

Modern protection against quantum attacks relies on new post-quantum algorithms.

- Examples:
- Kyber → encryption / key encapsulation (KEM)
- Dilithium → digital signature

Both are based on lattice problems, specifically the Learning With Errors (LWE) and Module-LWE problems.

- Other post-quantum algorithms include:
- Falcon (signature, lattice-based)
- SPHINCS+ (signature, hash-based)
- Classic McEliece (encryption, code-based)

⚗️ Hybridization: Combining Both Worlds

To be safe, current recommendations suggest hybrid approaches, combining pre-quantum (classical) and post-quantum algorithms.
Examples:

- 🔑 Hybrid key exchange: combine Curve25519 (ECDH) and Kyber.

→ If one scheme fails, the other still ensures confidentiality.

- ✍️ Hybrid signature: sign each message with two algorithms (e.g., ECDSA + Dilithium).

→ The verifier must check that both signatures are valid.

🧩 Common Feature: New and Tested Over Time

These algorithms are relatively new, and their security relies on mathematical problems different from classical cryptography (like RSA or ECC).
In cryptography, time and analysis are what ultimately prove an algorithm's resilience.

⚠️ Caution: The Challenge Lies in the Link

The difficulty of hybridization is not only in combining algorithms,
but also in how the two worlds are linked -
key management, combined validation, data formats, and implementation details all matter.

⚠️ Technical constraints of PQC usage

When deploying PQC, one must account for larger key/signature/ciphertext sizes and greater overheads. Some representative figures:

- For CRYSTALS-Dilithium (signature scheme): public key size ≈ 1,312 bytes, signature size ≈ 2,420 bytes for the Level 1 set.
- For CRYSTALS-Kyber (key-encapsulation / encryption): public key size ≈ 800 bytes (Kyber512), ciphertext ≈ 768 bytes.

- Overhead example: In a typical classical ECDH exchange, a public key might be ~32 bytes, whereas a Kyber ciphertext at similar security level might be ~768 bytes - i.e., ~24× larger.
- Hence, PQC deployment may require: more bandwidth, more storage for keys/certificates, modifications in protocols (e.g., TLS, certificates) to accommodate size jumps.

HYBRID KEMS AND SIGNATURES REQUIRE STRONGER STRUCTURE

Hybridizing a **KEM** (key encapsulation mechanism) or a **signature scheme** isn't just about using two algorithms side by side. When a *hybrid public key* is received, it actually consists of **two (or more) public keys**, and the **relationship and ordering** between these components must be **explicit and verifiable**.

⚙️ Why This Matters

If the binding between public keys isn't clearly defined, an attacker could:

- **Swap or reorder components,**
- **Mix keys from different sets,**
- **Or inject mismatched pairs to break authentication or key consistency.**

Therefore, a **hybrid key set** must come with a **structured format** that:

1. Groups all component keys,
2. Defines their order,
3. And **binds them together using a cryptographic hash.**

🧩 Binding Example

To ensure that all component keys belong together, we compute a *group hash* such as:

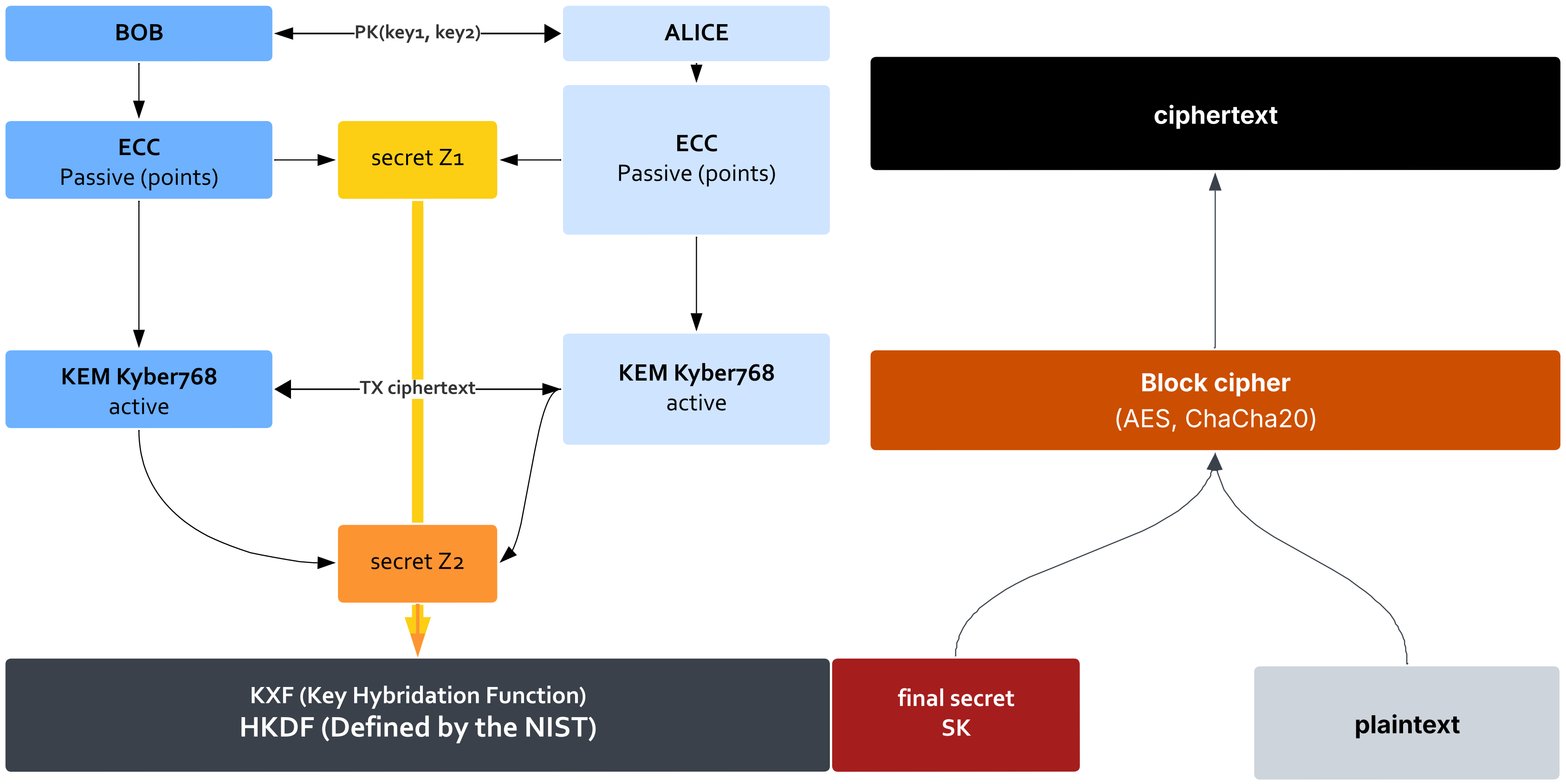
```
group_hash = H("HYBRID-KEY" || version || context || alg1 || PK1 || alg2 || PK2 || ...)
```

This group_hash acts as a **seal of consistency**, preventing reordering or substitution.
It should accompany the hybrid key structure anywhere it's transmitted or certified (e.g., in X.509 extensions or TLS identifiers).

FOR HYBRID KEMS

Each algorithm (classical + PQC) produces its own shared secret (Z1, Z2), which are then **combined** using a secure KDF that includes the group hash:

```
ss = HKDF-Extract(salt = group_hash, IKM = Z1 || Z2)
```



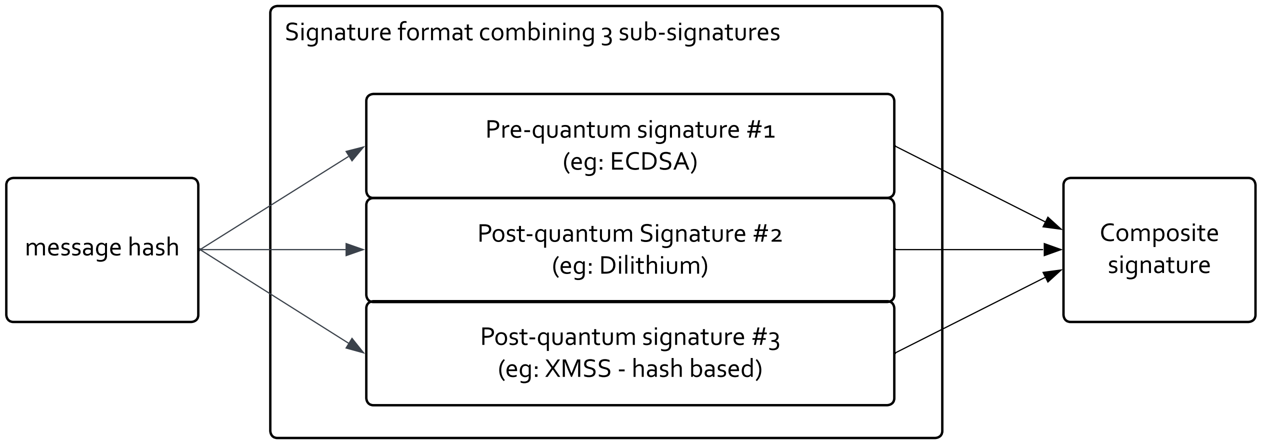
FOR SIGNATURES

Each message is signed twice (once per algorithm), but both signatures are bound to the **same transcript** and the **group hash**:

```
sig1 = Sign1(H(message || group_hash))
sig2 = Sign2(H(message || group_hash))


CompositeSignature = { order: [alg1, alg2], sigs: [sig1, sig2] }
```

This guarantees that both signatures refer to the same hybrid key set, in the same order.



PARTIAL POST-QUANTUM SAFETY IN SOME CLASSICAL CRYPTOSYSTEMS

Some cryptographic systems remain **partially or even fully secure against quantum attacks**, **despite using classical (pre-quantum) algorithms** such as ECDSA or RSA.

🔒 Why?

Because their **public keys are not directly exposed** until they are used.
If an attacker never sees the public key, they have **no entry point** to perform a quantum attack (e.g., no data to factorize or solve a discrete logarithm against).

⚙️ The Case of Bitcoin and Ethereum

This property applies to most **UTXO-based blockchains**, such as **Bitcoin**, and to some extent **Ethereum**.

- In **Bitcoin**, the cryptosystem remains safe **as long as the wallet owner has never spent coins**.
- The **public key is not published** on the blockchain until a spending transaction occurs.
  - The wallet address is derived from the **hash** of the public key:

```
address = RIPEMD-160(SHA-256(pubkey))
```

- This means what's visible on-chain is only a **double hash of the public key**,
- not the key itself — making it **quantum-safe by obscurity** until first use.

⛓️ Why Bitcoin Can't Simply Switch to PQC Signatures

Bitcoin (and similar blockchains) **cannot easily adopt post-quantum signatures** such as **Dilithium** or **SPHINCS+**, because these algorithms have **much larger keys and signatures** (from kilobytes to tens of kilobytes).
That would **drastically increase blockchain size and transaction costs**, breaking scalability.

Moral of the story: Hiding public keys can protect against post-quantum attacks and can be leveraged as a PQC-safe design pattern.

QUANTUM COIN FORWARDER

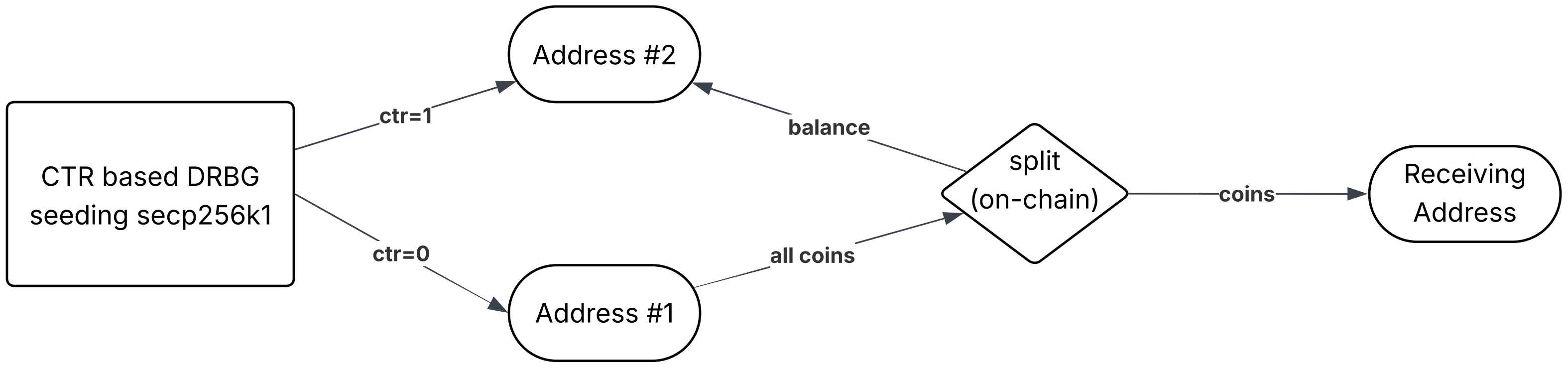
One practical approach to reduce PQC risk while keeping classical cryptography is:

Forwarding remaining coins to a new address immediately after making any transaction.

This pattern ensures:

- Each transaction uses a **fresh keypair and address**.
- Previously exposed public keys become **obsolete ("dead keys")**.
- Long-term funds remain under **unexposed public keys**,
- acting like **long-lived ephemeral wallets**.

This way, the system keeps benefiting from pre-quantum algorithms while minimizing exposure to future quantum threats.


CTR based DRBG seeding secp256k1
ctr=1
ctr=0
Address #2
Address #1
balance
all coins
split (on-chain)
coins
Receiving Address