Security in Futuristic Technologies

Lightweight Post-Quantum Messaging

Proposed Solution: Amortized Quantum Messaging

CryptIndo Labs

1 Problem Statement

The modern day messaging architecture depends solely upon the TLS 1.3 handshake protocol, which constructs the security using Classical Cryptographic Suites. It mainly uses Elliptic Curve Diffie-Hellman(ECDHE) for key exchange and RSA/ECDSA for digital authentication. The mentioned classical cryptographic suites are subsets of Integer Factorization and Discrete Logarithm Problem, both of which are fundamentally vulnerable to Shor's Algorithm. A quantum computer can very easily break the encryption and recover the session keys from the recorded traffic. To combat this problem we can adapt to NIST-standardized Post-Quantum Cryptography (PQC) Algorithm, example given Kyber for key exchange and Dilithium for digital authentication. However, a direct substitution would create a colossal performance penalty. A complete PQC handshake will demand an exchange over on average of 5KB of data, whereas classical algorithm requires less than 100 bytes data.

2 Proposed Solution

Keeping these two critical constraints to consideration, we propose Amortized Quantum Messaging(AQM), where instead of performing an expensive PQC handshake for each and every connection, we do the expensive handshake once to establish the master secret and then later extract the lightweight symmetric keys for following sessions. We are "amortizing" the expensive initial cost over a longer period of secure communication. This enforces the quantum resistance while maintaining the low latency and efficiency demand for modern day's huge mobile and IoT ecosystem.

3 Technical Approach

3.1 Blind Courier

In comparison to TLS 1.3 based messaging where the server decrypts and re-encrypts traffic AQM establishes a Blind Courier Architecture. AQM topology is Client-Server-Client. Here the server acts meticulously as an asynchronous mailbox. It stores Public Pre-Keys and navigated the encrypted parcels. The server has no access to private keys

thus it cannot decrypt any payload, ensuring a End-to-End Encryption. Here we also separate the Cryptographic Handshake from the time of data transmission producing a time-decoupled security. The handshake is done in advance, allowing the real messaging happen in Zero Round-Trip Time.

3.2 Coin Minting

To address the constraint of running an expensive NIST-algorithm on 2G/LoRa network we resort to Coin Minting Protocol (CMP). The device judges it resource state and selects the highest security coin that the resource condition can support.

Coins	**Composition**	**Payload Size**	**Use Case**
GOLD COIN (Standard)	Kyber-768 + Dilithium	∼4.0 KB	WiFi / 4G: Full NIST FIPS Compliance. Used for session initialization when bandwidth is abundant.
SILVER COIN (Hybrid)	Kyber-768 + Ed25519	∼1.2 KB	2G / Weak Signal: Maintains full Post-Quantum Confidentiality (Kyber) but uses lightweight Classical Authentication to save 70% bandwidth.
BRONZE COIN (Fallback)	X25519 + Ed25519	∼0.1 KB	Emergency (<5% Battery): Classical fallback. Not Quantum Safe. Used only for "Panic" messages to guarantee delivery when PQC is impossible.

Table 1: CMP and Bandwidth Usage

3.3 Saving Contacts

The system classifies the contacted devices largely into three categories based on message frequency. The local database divides it to the following categories:

Category	Cache Policy (Local Storage)	Est. Storage	User Experience
Bestie (Top 5 Contacts / Daily msgs)	**The "Premium" Stockpile** • 5 Gold Coins (Full PQC) • 4 Silver Coins (Hybrid) • 1 Bronze Coin (Emergency)	∼25.8 KB per person	**Instant & Secure.** Messages fly instantly regardless of whether you are on 4G, 2G, or 1% battery.
Mate (Weekly interactions)	**The "Economy" Pack** • 0 Gold Coins (Too heavy) • 6 Silver Coins (Reliable) • 4 Bronze Coins (Emergency)	∼9.2 KB per person	**Reliable.** Optimized for 2G/Mobile Data. If you are on WiFi (Gold), the app fetches a key on-demand.
Strangers (New or Monthly interactions)	**Zero Inventory** • No pre-fetched keys. • Fetch-on-Demand only.	0 KB	**Efficient.** No storage wasted. First message takes ∼1s to fetch keys, subsequent messages are instant.

Table 2: Saving Contacts

3.4 Workflow

Let's assume that Alice (the sender) and Bob (the receiver).

- **Phase 1: Asynchronous Minting**
  Bob pre-calculates keys when its resources are inexpensive, instead of calculating keys during a chat which takes a lot of battery charge.
  - **Trigger:** Device charging or connected wi-fi
  - **Action:** Bob's device generates a stack of Gold, Silver, and Bronze Coins.
  - **Update:** Public keys are uploaded to the Server's Public Key Directory
  - **Storage:** Private keys are encrypted and stored locally in Bob's Secure Vault

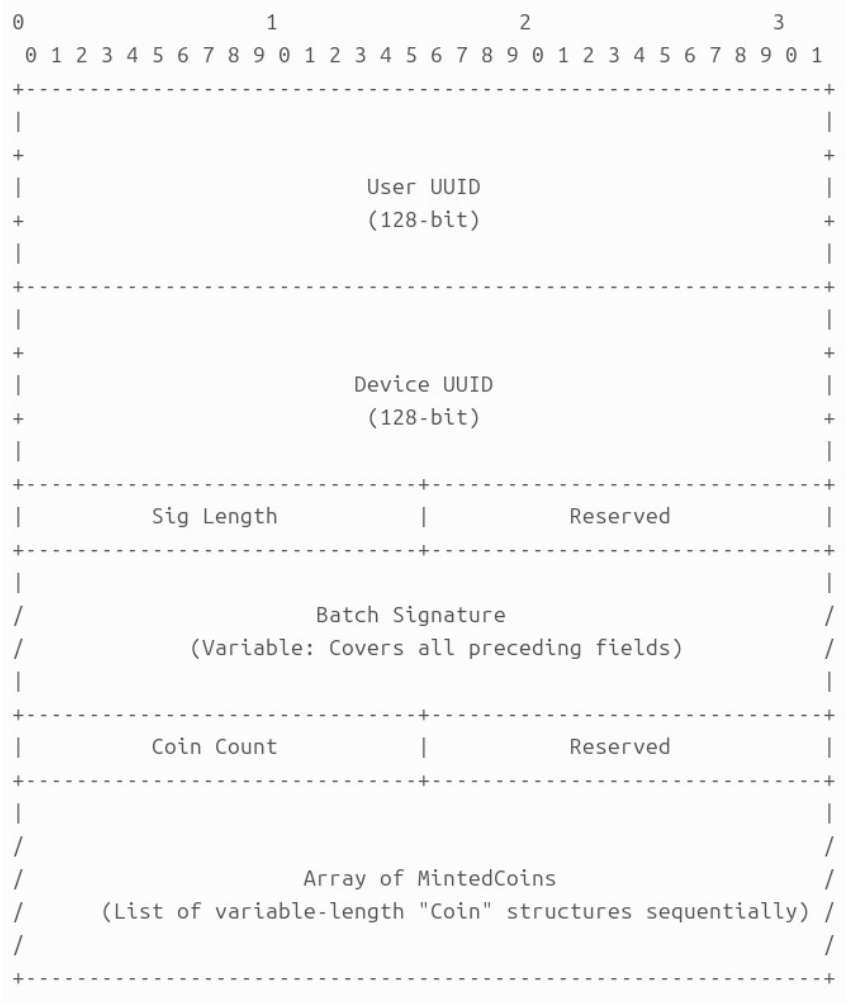

Figure 1: Architecture of Phase 1

- **Phase 2: Smart Pre-fetching**
  Alice predicts all her future conversations with the help of Saving Contacts database.
  - The system marks Bestie devices and Mate devices so Alice prefetches into her smart inventory:-
    * 5 Gold Coins 4 Silver Coins and 1 Bronze Coin for each Bestie devices.
    * 6 Silver Coins and 4 Bronze Coins for each Mate devices.
  - Alice has access to destination keys before even starting the secure communication.

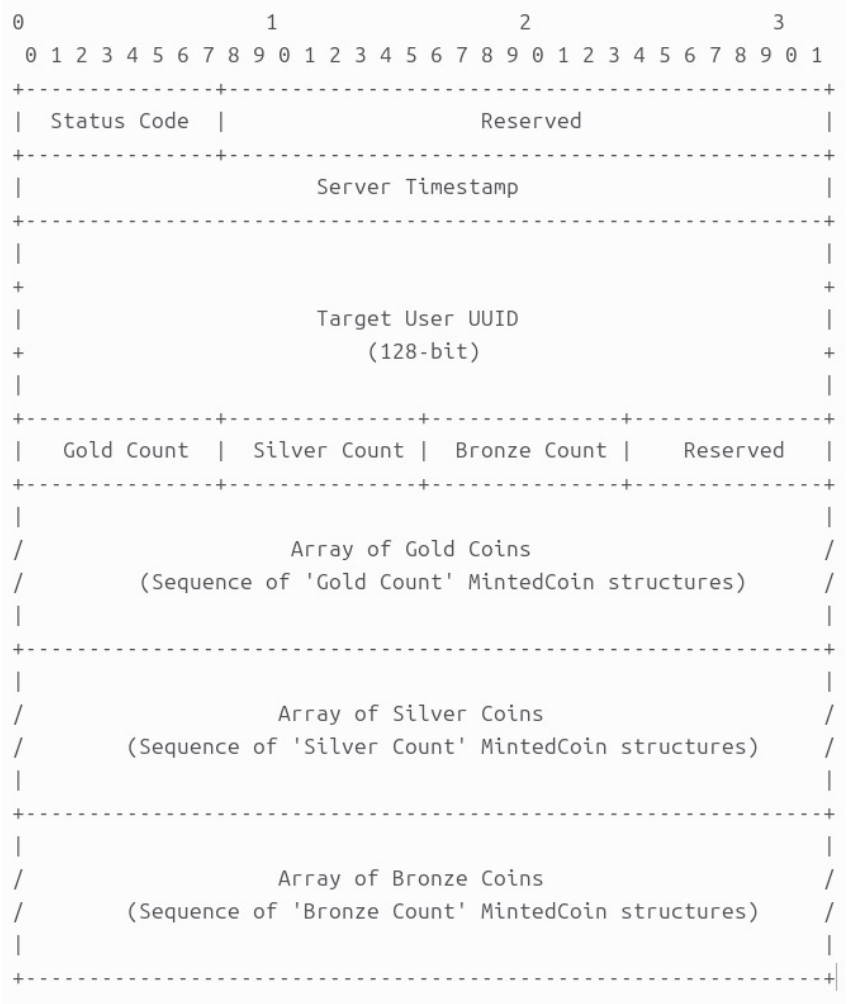

Figure 2: Architecture of Phase 2

- **Phase 3: Silent Messaging**
  When Alice actually starts messaging, the network negotiation step never occurs.
  - Alice's device monitors it's charge and network constraints
  - The system systematically selects a cached Silver Coin from the smart inventory.
  - Alice executes the Kyber Encapsulation locally against the cached key to calculate the shared secret.
  - Alice concatenates the encapsulated key and the AES-Encrypted Message into one parcel.
  - The parcel is sent to Bob and the key is dropped from Alice's smart inventory. The key is within the parcel, so Bob can decrypt it very quickly and deletes the key from its Secure Vault.

Figure 3: Architecture of Phase 3

- **Phase 4: Session Accounting**
  To enforce the amortizing cost to be a constant over a longer period of time, we managed not to spend expensive PQC token on every message.
  - One coin has been used to establish the Master Secret.
  - The following 100 messages uses AES-256 Ratchet Keys calculated from the Master secret, which helps to cut the cost by 50 bytes/message.
  - After 100 message the app auto refreshes and consumes a new PQC coin to rotate the keys thus maintaining Forward Secrecy.

3.5 Databases

Bob's Secure Vault → Private Keys
This database is in the recipient device. It stores all the private keys securely until a message arrives.

Field Name	Description	Why we need it
Key ID	A unique serial number (e.g., #10405).	To match the "Silent Parcel" Alice sends. When a packet arrives saying "Use Key #10405," Bob looks up this row.
Coin Category	Gold, Silver, or Bronze.	Tells the app which algorithm to use (Kyber for Gold/Silver, X25519 for Bronze) to decrypt the key.
The Encrypted Blob	**The most critical field.** This is the actual Private Key, but turned into "gibberish" ciphertext using the Hardware Key.	If a hacker copies the database file, this field remains unreadable because they don't have Bob's physical phone chip to decrypt it.
Encryption IV & Tag	Random data used during the hardware encryption process.	Needed by the AES algorithm to unlock the "Encrypted Blob" above.
Status	"Active" or "Burned".	Prevents Replay Attacks. Once Bob uses a key to read a message, this flag flips to "Burned" (or the row is deleted).

Table 3: Bob's Secure Vault Database Structure

Alice's Smart Inventory → Public Keys
This database is in the sender device. It stores the stockpile of all the public keys for her friends so she can message them even when offline.

Field Name	Description	Why we need it
Contact ID	The person the key belongs to (e.g., "Bob").	So when Alice types "Hello Bob," the app knows where to look.
Key ID	The serial number (e.g., #10405).	Alice sends this number in the packet so Bob knows which key to grab from his vault.
Coin Category	Gold, Silver, or Bronze.	Tells Alice which packet type to build (e.g., if she picks a "Silver" row, she builds a Silver packet).
The Public Key	The "Lock" (Kyber or X25519 Public Key).	Alice uses this to perform the math that encrypts the initial secret.
The Signature	The "Proof" (Dilithium or Ed25519).	Proves that this key actually came from Bob and not a hacker spoofing him.
Contact Priority	"Bestie", "Acquaintance", or "Stranger".	**Crucial for Storage Management.** If the database gets full, the app looks at this field to decide what to keep (Besties) and what to delete (Strangers).
Last Usage Date	A timestamp (e.g., "Jan 12, 10:00 AM").	Used for the "Garbage Collector." If Alice hasn't spoken to someone in 30 days, their keys are the first to be deleted to free up space.

Table 4: Alice's Smart Inventory Database Structure

Server's Coin Inventory

Field Name	Description	Technical Purpose
Record ID	Unique system ID (e.g., `4591`).	**Internal Indexing:** Used by the server to identify and delete the specific row once Alice downloads it.
User / Owner	The User ID (e.g., `Bob_555`).	**Routing:** Identifies who "minted" this key. When Alice asks for "Keys for Bob," the server filters by this column.
Key ID	Client-Side Serial (e.g., `#10405`).	**Client Sync:** A number assigned by Bob's phone. Alice sends this number back in the message packet so Bob knows which private key to unlock.
Coin	`GOLD`, `SILVER`, or `BRONZE`.	**Filtering:** Allows Alice to request specific security levels based on her network context (e.g., "Give me Silver because I'm on 2G").
Public Key Blob	The "Lock" (Kyber or X25519).	**The Payload:** The actual mathematical key Alice needs to perform the encryption.
Signature Blob	The "Proof" (Dilithium or Ed25519).	**Trust:** A digital signature proving the key really came from Bob and wasn't forged by the server or a hacker.
Uploaded At	Date & Time.	**Hygiene:** Used to find and delete "expired" keys that have been sitting on the server too long (e.g., > 30 days).

Table 5: Server Coin Inventory Database Structure

4 Architecture

4.1 Network Architecture → how devices talk

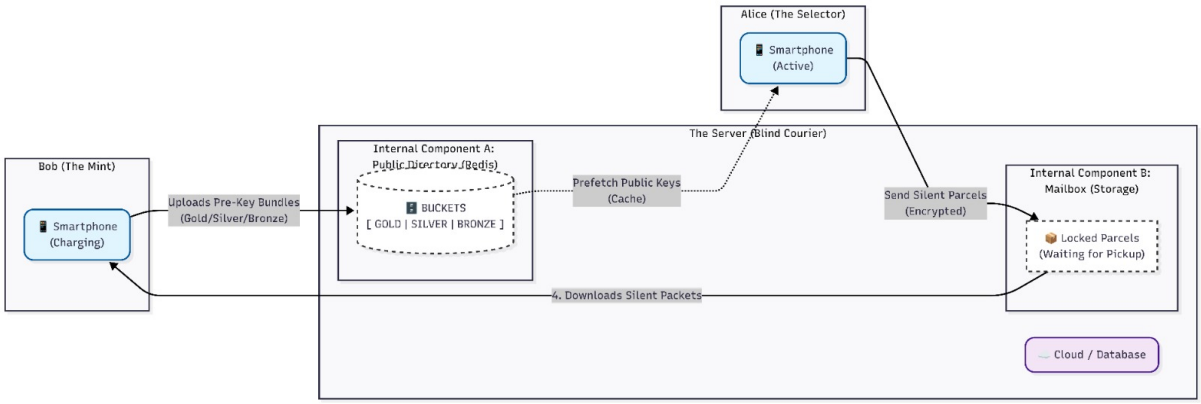

Figure 4: Network Architecture

4.2 Device Software Architecture → the bird view of the system

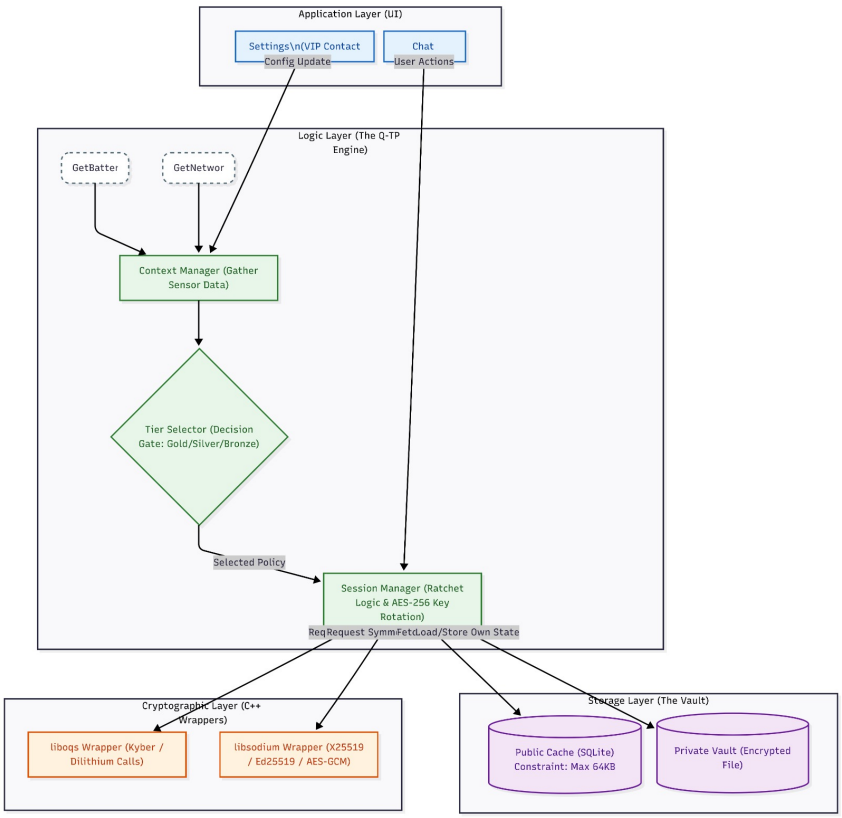

Figure 5: Device Software Architecture

4.3 Data Architecture → whats inside the packet

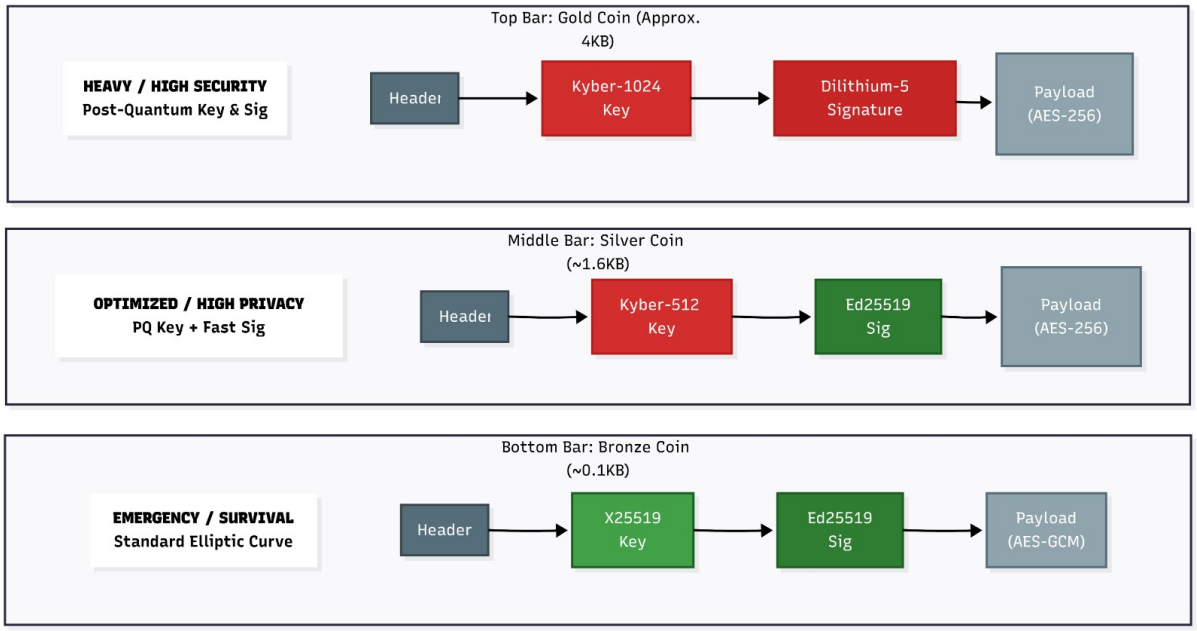

Figure 6: Data Architecture

5 Probable Outcomes

Whent the AQM is tested against the backdrop of unfavorable conditions like:

- 2G Crunch Test
  AQM system detects weak signal switches to silver coin and sends 1.2 KB packets. Small enought to fit in fewer frames with succesful message delivery.

- Dead Battery Test
  AQM detects $< 5\%$ switches to Bronze Coin. Uses X25519. Sends small 100-byte packet with successful message delivery.

- Latency Test
  AQM facilitates 0-RTT. Client(with Pre-fetched Key)→Message which ends in instant transmission.

6 Unique value proposition

- AQM addressed the bandwidth bottleneck of PQC by pairing Kyber Encryption with classical authentication through Silver Coin. This delivers 100% protection against quantum decryption a while minimizing the handshake data size by 75% making messaging very secure feasible even on slow 2G network.

- AQM prioritizes human safety over theoretical purity. If any device has critical battey or poor signal, the system downgrades to Bronze Coin which makes sure

a distress signal always leaves the device preventing the DOS that heavy PQC algorithm cause in emergency.

- AQM decouple the energy cost of cryptography from the act of messaging. Keys are minted only when the device is in highly ideal condition (like charging/Wifi) and prefetched by receivers. This makes real messaging instant and battery neutral, eliminating the delay found in standard secure chats.

7 Road Map

Phase	Focus Area	Key Deliverables
Phase 1	**The Foundation**	• Integration of `liboqs` (Kyber/Dilithium). • Basic Client-Server "Blind Courier" architecture. • Gold Coin (Standard PQC) message delivery.
Phase 2	**The Innovation Engine**	• **Silver Coin Implementation** (Hybrid PQC). • Context Manager (Battery/Signal detection logic). • Packet serialization optimization.
Phase 3	**Storage & Efficiency**	• **Silicon-Bound Vault** . • Smart Inventory System (LRU Caching logic). • Server "Delete-on-Fetch" privacy mechanics.
Phase 4	**Future Vision**	• Mesh Networking (Device-to-Device via Bronze Coin). • Group Chat (MLS-PQC) adaptation. • Voice-over-PQC compression.

Table 6: AQM Development Roadmap