Upgraded Quantum-Classical Hybrid Encryption Framework

Upgraded Quantum-Classical Hybrid Encryption Framework

Complete System Documentation with Workflow Diagrams

Table of Contents

Executive Summary

This document details the evolution of a Quantum-Classical Hybrid Encryption Framework that combines:
- BB84 Quantum Key Distribution (QKD) for secure key exchange
- HKDF key derivation for cryptographically separated keys
- Three AEAD cipher modes: AES-GCM, ChaCha20-Poly1305, AES-SIV
- Post-quantum Dilithium5 signatures for authenticity
- Realistic channel modeling with noise, loss, attacks
- Multi-layer tamper detection and fault tolerance

The framework evolved from a basic simulation (AES-CBC + ideal BB84) to a production-grade system with realistic quantum channel simulation, multiple encryption options, and defense-in-depth security.

Original Framework (Baseline)

Components

- Quantum Layer: Simulated BB84 protocol for key exchange between Alice and Bob
- Classical Layer: AES-256-CBC for data encryption
- Integrity: HMAC-SHA256 used separately for message authentication
- Implementation: Python simulation using Qiskit; modular GUI

Limitations

[X] AES-CBC was not authenticated (vulnerable to tampering and padding oracle attacks)
[X] Manual PKCS#7 padding required (error-prone)
[X] BB84 simulation lacked realism (no noise, loss, detector effects, partial attacks)
[X] Single-purpose key derivation (no separation for encryption, authentication, signing)

[X] No post-quantum signatures (vulnerable to quantum adversaries)
[X] Limited metadata protection (filename, version not authenticated)
[X] Single encryption mode (no flexibility)
[X] QBER often 0% (unrealistic for educational/research purposes)

Quantum Layer Improvements (BB84)

2.1 Realistic Channel Simulation

Changes Made:
- Depolarizing noise (p_depolarize): Probabilistically flips measured bits
- Photon loss (p_loss): Simulates detection failures; lost photons removed in sifting
- Dark counts (dark_count): Detector false positives; random clicks or bit flips
- Partial intercept-resend (attack_fraction): Eve attacks only a random slice of qubits (default 8%)
- Multiple measurement shots (shots_per_qubit): Introduces stochastic measurement outcomes

Why Important:
[OK] Makes BB84 simulation closer to real-world quantum channels
[OK] Enables eavesdropping detection and QBER analysis
[OK] Educational value: students/researchers see realistic error rates
[OK] Scientific validation: reproducible noise/attack experiments

2.2 Biased Bases (Efficient BB84)

Configuration:
- p_Z = 0.8 (80% Z-basis, 20% X-basis) for both Alice and Bob
- Reduces basis mismatch losses from 50% to ~36%
- Standard in modern QKD implementations

Why Important:
[OK] More efficient key generation (higher sifted rate)
[OK] Practical optimization without security trade-offs

2.3 BB84 Key Confirmation

Process:
1. Sacrifice a subset of sifted bits (e.g., 20 bits)
2. Alice and Bob compare these bits over public channel
3. Calculate error rate (QBER)
4. Abort encryption if QBER > 15% (eavesdropping detected)

Why Important:
[OK] Detects quantum channel tampering before encryption
[OK] Standard BB84 protocol step (ensures key security)
[OK] Prevents using compromised keys

2.4 GUI Integration with Demo Preset

Feature:
- run_qkd_demo() function with tuned parameters:
- p_depolarize=0.012, p_loss=0.03, dark_count=0.01

- attack="intercept_resend", attack_fraction=0.08, shots_per_qubit=6
- GUI calls demo preset to display realistic QBER (~1-5%) every run

Why Important:
[OK] Users see observable, non-zero QBER consistently
[OK] Demonstrates quantum security principles visually
[OK] Useful for education and demonstrations

Classical Layer Improvements

3.1 AES-GCM Upgrade (AEAD Migration)

What Changed:
```
OLD: AES-256-CBC + HMAC-SHA256
NEW: AES-256-GCM (Authenticated Encryption with Associated Data)
```

Implementation:
- Nonce: 12-byte random nonce per encryption (using os.urandom())
- Key: 32-byte (256-bit) derived via HKDF from BB84 bits
- AAD: Metadata (filename, version, salt) authenticated but not encrypted
- Tag: 16-byte authentication tag ensures integrity
- No padding: GCM is a stream mode (no PKCS#7 needed)

Why Important:
[OK] Eliminates padding oracle attacks (no manual padding)
[OK] Atomic operation: confidentiality + integrity in one step
[OK] NIST-approved standard: used in TLS 1.3, IPsec
[OK] Prevents MAC misuse: no "encrypt-then-MAC" vs "MAC-then-encrypt" confusion
[OK] Tamper detection: any modification triggers InvalidTag error

API Simplification:
```
# Before (error-prone)
cipher = AES.new(key, AES.MODE_CBC, iv)
ciphertext = cipher.encrypt(padded_plaintext)
mac = HMAC.new(auth_key, ciphertext, SHA256).digest()

# After (clean and secure)
cipher = AESGCM(key)
ciphertext = cipher.encrypt(nonce, plaintext, associated_data)
```

3.2 Key Separation with HKDF

What Changed:
- Moved from single-purpose key to three independent keys derived via HKDF:
1. Encryption Key (32 bytes): AES-GCM, ChaCha20-Poly1305, AES-SIV
2. Authentication Key (32 bytes): Optional HMAC or additional verification
3. Signature Key (derived separately): Post-quantum Dilithium signatures

HKDF Process:
```
BB84 bits + salt -> HKDF-Extract -> PRK (pseudorandom key)
PRK + info labels -> HKDF-Expand -> Encryption Key, Auth Key, Signature Key
```

Why Important:
[OK] Prevents key reuse attacks: Each key serves one purpose only

[OK] Cryptographically sound: HKDF is RFC 5869 standard, designed for high-entropy sources
[OK] Future-proof: Easy to add more keys (e.g., for key rotation) without weakening existing keys
[OK] NIST-compliant: Follows SP 800-108 guidelines

Security Advantage:
If one key is compromised (e.g., encryption key leaked), authentication and signature keys remain secure.

3.3 ChaCha20-Poly1305 Integration

What Changed:
- Added ChaCha20-Poly1305 as second AEAD option
- Stream cipher (no block alignment needed)
- Poly1305 MAC for authentication

Implementation:
```
from cryptography.hazmat.primitives.ciphers.aead import ChaCha20Poly1305

key = derive_chacha20_key_from_bb84_bits(bb84_bits, salt)  # HKDF
cipher = ChaCha20Poly1305(key)
ciphertext = cipher.encrypt(nonce, plaintext, aad)
```

ChaCha20 Stream Cipher:
- 20 rounds of ARX operations (Add, Rotate, XOR)
- 256-bit key, 96-bit nonce, 32-bit counter
- Generates pseudorandom keystream; XOR with plaintext

Poly1305 MAC:
- First 32 bytes of ChaCha20 keystream -> MAC key
- Computes 16-byte authentication tag over ciphertext + AAD

Why Important:
[OK] Fast on CPUs without AES-NI (5-15× faster than AES on ARM/mobile)
[OK] Constant-time operations (resistant to timing attacks)
[OK] Modern standard: RFC 8439, used in TLS 1.3, SSH
[OK] Software-optimized: No special hardware instructions needed

Performance Comparison:

Platform	AES-GCM	ChaCha20-Poly1305
Intel/AMD (AES-NI)	4-8 GB/s	500-800 MB/s
ARM/Mobile	50-100 MB/s	500-800 MB/s
Embedded	Very slow	Fast

3.4 AES-SIV Integration (Misuse-Resistant AEAD)

What Changed:
- Added AES-SIV (Synthetic IV mode) as third AEAD option
- No nonce required (deterministic encryption)
- Resistant to nonce reuse (misuse-resistant)

Implementation:
```
from cryptography.hazmat.primitives.ciphers.aead import AESSIV

# Requires 512-bit key (two 256-bit keys concatenated)
key = derive_aes_siv_key_from_bb84_bits(bb84_bits, salt)  # HKDF -> 64 bytes
cipher = AESSIV(key)
ciphertext = cipher.encrypt(plaintext, [aad])  # No nonce!
```

AES-SIV Process:
1. SIV (Synthetic IV): Derives 16-byte IV from plaintext + AAD using CMAC
2. CTR Mode Encryption: Uses derived IV as counter; encrypts plaintext
3. Result: IV || ciphertext (IV serves as authentication tag)

Why Important:
[OK] Misuse-resistant: Safe even if nonce is accidentally reused
[OK] Deterministic: Same plaintext + AAD -> same ciphertext (useful for deduplication)
[OK] No nonce management overhead: Simplifies key handling
[OK] RFC 5297 standard: Approved for high-security applications

Use Cases:
- Research environments where nonce management is difficult
- Systems with strict auditability requirements
- Backup/archive systems requiring deterministic encryption

Trade-offs:
- Slightly slower than AES-GCM (two-pass: CMAC + CTR)
- Requires 512-bit key (vs 256-bit for GCM/ChaCha20)
- Deterministic (may leak if same data encrypted multiple times)

Post-Quantum Signatures (Dilithium5)

4.1 Signature Integration

What Changed:
- Every encrypted package is signed with Dilithium5 (CRYSTALS-Dilithium)
- Signature covers: AAD + ciphertext
- Verify-before-decrypt: Signature checked first, before attempting decryption

Dilithium5 Parameters:
- Security level: NIST Level 5 (highest)
- Public key: ~2592 bytes
- Signature: ~4595 bytes
- Algorithm: Module-Lattice-Based Digital Signature (ML-DSA)

Why Important:
[OK] Quantum-safe authenticity: Dilithium is NIST PQC standard (resistant to Shor's algorithm)
[OK] Layered defense: Even if AEAD fails, signature detects tampering
[OK] Future-proof: Secure against quantum adversaries
[OK] DoS protection: Verify-before-decrypt saves CPU on tampered files

4.2 Verify-Before-Decrypt Flow

Old Flow:
```
Parse package -> Decrypt -> Verify AEAD tag -> Verify signature (if any)
```

New Flow:
```
Parse package -> Verify Dilithium signature -> Decrypt -> Verify AEAD tag
                    v (fail fast)
                 Reject invalid packages immediately
```

Performance Impact:

- Tampered files rejected 5-6× faster (no expensive decryption)
- Valid files: negligible overhead (~1-2 ms signature verification)

Why Important:
[OK] Prevents DoS attacks: Attackers cannot force expensive decryption operations
[OK] Clear security flow: Signature -> decrypt -> AEAD (unambiguous)
[OK] Saves resources: Early rejection of invalid packages

Security Enhancements

5.1 Metadata Authentication (AAD)

What Changed:
- Metadata (filename, version, salt) included as Additional Authenticated Data in AEAD modes
- Authenticated but NOT encrypted (readable, tamper-proof)

AAD Construction:
```
aad = version.encode() + filename.encode() + salt
```

Why Important:
[OK] Prevents metadata tampering: Filename/version changes detected
[OK] No encryption overhead: Metadata remains readable
[OK] Binding: Ciphertext + metadata cryptographically linked

5.2 Multi-Layer Tamper Detection

Layers:
1. Dilithium signature (post-quantum authenticity)
2. AEAD authentication tag (integrity of ciphertext + AAD)
3. Optional HMAC (additional layer if needed)

Rejection Points:
```
Tampered package detected at ANY layer -> Abort immediately
```

Why Important:
[OK] Defense-in-depth: Multiple independent checks
[OK] Early detection: Fail fast at first sign of tampering
[OK] No ambiguity: Clear pass/fail for every package

5.3 Fault Tolerance

Early Rejection Rules:
- Invalid Key B -> reject before decryption
- Corrupted metadata -> reject at AAD verification
- Tampered ciphertext -> reject at AEAD tag check
- Invalid signature -> reject before decryption

Metrics Logging:
- All rejection events logged with timestamps, reasons, file hashes
- JSON export for analysis

Why Important:
[OK] Prevents processing invalid data (security risk)

[OK] Clear audit trail (compliance, forensics)
[OK] No partial states (encryption succeeds or fails atomically)

5.4 Secure Randomness & Nonce Management

Sources:
- os.urandom() for nonces (cryptographically secure)
- secrets module for key generation
- System entropy pool (platform-dependent: /dev/urandom, CryptGenRandom, etc.)

Nonce Rules:
- AES-GCM / ChaCha20: Fresh 12-byte nonce per encryption (never reused)
- AES-SIV: No nonce (deterministic)

Why Important:
[OK] Prevents nonce reuse attacks (catastrophic for GCM/ChaCha20)
[OK] Unpredictable keys (essential for cryptographic security)
[OK] NIST compliance (SP 800-90A/B/C)

Unified System Workflow

6.1 Complete Encryption Workflow (Alice -> Bob)

```
+---------------------------------------------------------------------+
|  STEP 1: Quantum Key Distribution (BB84)                            |
+---------------------------------------------------------------------+
|  1a. Alice generates random bits + biased bases (p_Z=0.8)           |
|  1b. Optional: Eve intercepts fraction of qubits (attack_fraction)  |
|  1c. Transmission through noisy channel:                            |
|       - Depolarizing noise (p_depolarize)                           |
|       - Photon loss (p_loss)                                        |
|       - Dark counts (dark_count)                                    |
|  1d. Bob measures with biased bases + multiple shots                |
|  1e. Basis reconciliation (public channel): keep matching bases     |
|  1f. Sifting: remove lost photons, basis mismatches                 |
|                                                                      |
|  Output: Raw key bits (Key A for Alice, Key B for Bob)            |
+---------------------------------------------------------------------+
                              v
+---------------------------------------------------------------------+
|  STEP 2: BB84 Key Confirmation (Eavesdropping Detection)            |
+---------------------------------------------------------------------+
|  2a. Sacrifice subset of key bits (e.g., 20 bits)                   |
|  2b. Alice and Bob compare bits over public channel                 |
|  2c. Calculate QBER (Quantum Bit Error Rate)                        |
|  2d. Abort if QBER > 15% -> Eavesdropper detected!                   |
|                                                                      |
|  Output: Confirmed secure key bits                                  |
+---------------------------------------------------------------------+
                              v
+---------------------------------------------------------------------+
|  STEP 3: Key Derivation (HKDF)                                      |
+---------------------------------------------------------------------+
|  3a. Generate random 16-byte salt                                   |
|  3b. HKDF-Extract: BB84 bits + salt -> PRK                           |
|  3c. HKDF-Expand: PRK + info labels -> separate keys                 |
|       - Encryption Key (32 bytes): AES-GCM / ChaCha20 / AES-SIV     |
```

```
|       - Authentication Key (32 bytes): Optional HMAC                  |
|       - Signature Key: Dilithium keypair generation                   |
|                                                                        |
|  Output: Separated cryptographic keys                                 |
+-----------------------------------------------------------------------+
                              v
+-----------------------------------------------------------------------+
|  STEP 4: Encryption Mode Selection (GUI)                              |
+-----------------------------------------------------------------------+
|  User selects one of three modes:                                     |
|  +-----------------+-----------------+-----------------+            |
|  |   AES-GCM       |  ChaCha20-Poly  |    AES-SIV      |            |
|  |  (Hardware)     |   (Software)    | (Misuse-Resist) |            |
|  +-----------------+-----------------+-----------------+            |
+-----------------------------------------------------------------------+
                              v
+-----------------------------------------------------------------------+
|  STEP 5: AEAD Encryption + Metadata                                   |
+-----------------------------------------------------------------------+
|  5a. Prepare AAD (Additional Authenticated Data):                     |
|       AAD = version || filename || salt                               |
|                                                                        |
|  5b. Generate nonce (if needed):                                      |
|       - AES-GCM: 12-byte random nonce                                 |
|       - ChaCha20: 12-byte random nonce                                |
|       - AES-SIV: No nonce (deterministic)                             |
|                                                                        |
|  5c. Encrypt:                                                          |
|       ciphertext || tag = AEAD.encrypt(nonce, plaintext, aad, key)  |
|                                                                        |
|  Output: Encrypted package components                                 |
+-----------------------------------------------------------------------+
                              v
+-----------------------------------------------------------------------+
|  STEP 6: Post-Quantum Signature (Dilithium5)                          |
+-----------------------------------------------------------------------+
|  6a. Generate Dilithium5 keypair (pk, sk)                             |
|  6b. Sign: signature = dilithium5.sign(sk, aad || ciphertext)        |
|  6c. Include public key in package                                    |
|                                                                        |
|  Output: Quantum-resistant digital signature                          |
+-----------------------------------------------------------------------+
                              v
+-----------------------------------------------------------------------+
|  STEP 7: Package Assembly                                             |
+-----------------------------------------------------------------------+
|  JSON structure:                                                       |
|  {                                                                     |
|    "ciphertext": base64(ciphertext),                                  |
|    "salt": base64(salt),                                              |
|    "nonce": base64(nonce),        // Optional (not in AES-SIV)       |
|    "version": "AES-GCM-v1" | "ChaCha20-v1" | "AES-SIV-v1",         |
|    "filename": "original.ext",                                        |
|    "pq_signature": base64(signature),                                 |
|    "pq_public_key": base64(pk)                                        |
|  }                                                                     |
|                                                                        |
|  Save as: filename_MODE_E.bb84                                        |
+-----------------------------------------------------------------------+
                              v
+-----------------------------------------------------------------------+
|  STEP 8: Transmission                                                 |
+-----------------------------------------------------------------------+
|  Send .bb84 package to Bob via classical channel (email, USB, etc.) |
```

```
|  Key B transmitted separately via secure out-of-band channel          |
+-------------------------------------------------------------------+
```

6.2 Complete Decryption Workflow (Bob)

```
+-------------------------------------------------------------------+
|  STEP 1: Parse Package                                            |
+-------------------------------------------------------------------+
|  1a. Read .bb84 file                                              |
|  1b. Parse JSON structure                                         |
|  1c. Extract:                                                      |
|       - ciphertext (base64 decode)                                |
|       - salt (base64 decode)                                      |
|       - nonce (base64 decode, if present)                         |
|       - version (auto-detect cipher mode)                         |
|       - filename                                                  |
|       - pq_signature (base64 decode)                              |
|       - pq_public_key (base64 decode)                             |
|                                                                    |
|  Output: Package components ready for verification                |
+-------------------------------------------------------------------+
                              v
+-------------------------------------------------------------------+
|  STEP 2: Verify Dilithium Signature (FIRST!)                      |
+-------------------------------------------------------------------+
|  2a. Reconstruct signed data: aad || ciphertext                   |
|  2b. Verify: valid = dilithium5.verify(pk, signature, data)       |
|  2c. If invalid -> ABORT (reject package immediately)              |
|                                                                    |
|  [OK] Signature valid -> Proceed to decryption                       |
|  [X] Signature invalid -> Reject (tampered package)                |
+-------------------------------------------------------------------+
                              v
+-------------------------------------------------------------------+
|  STEP 3: Key Derivation (Bob's Side)                              |
+-------------------------------------------------------------------+
|  3a. Bob retrieves Key B (from secure out-of-band channel)        |
|  3b. Extract salt from package                                    |
|  3c. HKDF: Key B + salt -> Encryption Key (32 bytes)               |
|                                                                    |
|  Output: Decryption key matching Alice's encryption key           |
+-------------------------------------------------------------------+
                              v
+-------------------------------------------------------------------+
|  STEP 4: Rebuild AAD                                              |
+-------------------------------------------------------------------+
|  4a. Reconstruct: aad = version || filename || salt               |
|  4b. Must match exactly (byte-for-byte) with Alice's AAD          |
|                                                                    |
|  Output: AAD for AEAD verification                                |
+-------------------------------------------------------------------+
                              v
+-------------------------------------------------------------------+
|  STEP 5: AEAD Decryption + Verification                           |
+-------------------------------------------------------------------+
|  Mode-specific decryption:                                        |
|                                                                    |
|  AES-GCM:                                                          |
|    plaintext = AESGCM(key).decrypt(nonce, ciphertext, aad)        |
|                                                                    |
|  ChaCha20-Poly1305:                                                |
|    plaintext = ChaCha20Poly1305(key).decrypt(nonce, ciphertext, aad)|
|                                                                    |
```

```
|  AES-SIV:                                                                   |
|    plaintext = AESSIV(key).decrypt(ciphertext, [aad])                      |
|                                                                             |
|  Verification:                                                              |
|    - AEAD tag checked automatically                                        |
|    - InvalidTag exception raised if tampered                               |
|                                                                             |
|  [OK] Tag valid -> Plaintext recovered                                        |
|  [X] Tag invalid -> Reject (tampered ciphertext or AAD)                      |
+----------------------------------------------------------------------+
                              v
+----------------------------------------------------------------------+
|  STEP 6: Extract Payload                                               |
+----------------------------------------------------------------------+
|  6a. Parse internal JSON payload                                       |
|  6b. Base64 decode file bytes                                          |
|  6c. Restore original filename from metadata                           |
|                                                                         |
|  Output: Original file recovered                                       |
+----------------------------------------------------------------------+
                              v
+----------------------------------------------------------------------+
|  STEP 7: Save Decrypted File                                           |
+----------------------------------------------------------------------+
|  7a. Write bytes to disk with original filename                        |
|  7b. Generate decryption report (PDF/JSON)                             |
|  7c. Log metrics (time, size, hash verification)                       |
|                                                                         |
|  Output: Decrypted file + audit trail                                  |
+----------------------------------------------------------------------+
```

6.3 Security Checkpoints (Decryption)

```
+--------------------------------------------------+
|         DECRYPTION SECURITY CHECKPOINTS          |
+--------------------------------------------------+
|  1. [OK] Dilithium signature valid                |
|  2. [OK] Key B matches (HKDF derivation succeeds) |
|  3. [OK] AAD intact (version, filename, salt)     |
|  4. [OK] AEAD tag valid (ciphertext unmodified)   |
|  5. [OK] No exceptions during decryption          |
+--------------------------------------------------+
|  [X] ANY checkpoint fails -> ABORT immediately    |
|  [X] No partial decryption                       |
|  [X] No ambiguous states                         |
+--------------------------------------------------+
```

Key Benefits Summary

7.1 Security

[OK] Quantum-Resilient: BB84 + Dilithium signatures resist quantum attacks
[OK] AEAD Encryption: Confidentiality + integrity in atomic operations
[OK] Misuse-Resistant Option: AES-SIV safe against nonce reuse
[OK] Layered Integrity: Signature + AEAD + optional HMAC
[OK] Metadata Protection: AAD ensures tamper detection
[OK] Key Separation: HKDF prevents key reuse attacks

7.2 Performance

[OK] Hardware-Accelerated: AES-GCM fast with AES-NI
[OK] Software-Optimized: ChaCha20 fast on ARM/mobile
[OK] Verify-Before-Decrypt: 5-6× faster rejection of tampered files
[OK] No Padding Overhead: AEAD modes eliminate padding

7.3 Usability

[OK] Three Cipher Options: User-selectable in GUI
[OK] Realistic QBER Demo: Educational value with run_qkd_demo()
[OK] Mode-Specific File Naming: Clear identification (filename_AES-GCM_E.bb84)
[OK] Automatic Mode Detection: Decryption auto-selects cipher

7.4 Observability

[OK] Comprehensive Metrics: QBER, entropy, timing, errors
[OK] JSON/PDF Exports: Reproducible scientific validation
[OK] Audit Trails: All rejections logged with reasons

7.5 Standards Compliance

[OK] NIST-Approved: AES-GCM (SP 800-38D), Dilithium (PQC)
[OK] RFC Standards: HKDF (5869), ChaCha20 (8439), AES-SIV (5297)
[OK] TLS 1.3 Compatible: Uses same algorithms as modern protocols

---

Detailed Workflow Diagrams

8.1 BB84 Protocol with Realistic Channel

```
ALICE                          CHANNEL                          BOB
  |                               |                               |
  | 1. Generate bits + bases      |                               |
  |    (biased: p_Z=0.8)          |                               |
  +------------------------------>|                               |
  |                               |                               |
  |                               | 2. Optional: Eve intercepts   |
  |                               |    (attack_fraction=0.08)     |
  |                               |    +---------+                |
  |                               +--->|   EVE   |                |
  |                               |    | Measure |                |
  |                               |    | Resend  |                |
  |                               |<---+         |                |
  |                               |    +---------+                |
  |                               |                               |
  |                               | 3. Channel noise:             |
  |                               |    - Depolarize (0.012)       |
  |                               |    - Photon loss (0.03)       |
  |                               |    - Dark counts (0.01)       |
  |                               |                               |
  |                               | 4. Measurement                |
  |                               +------------------------------>|
  |                               |   (shots_per_qubit=6)         |
  |                               |                               |
```

```
  | 5. Basis reconciliation        |                                |
  |    (public channel)            |                                |
  |<-------------------------------+------------------------------->|
  | "My bases: ZXXZZ..."           |                                |
  |                                |    "My bases: XZXZZ..."        |
  |                                |                                |
  | 6. Sifting (keep matches)      |                                |
  |    ~65% efficiency             |                                |
  |                                |                                |
  | 7. Key confirmation            |                                |
  |    (sacrifice 20 bits)         |                                |
  |<-------------------------------+------------------------------->|
  | "Bits 5,12,23... are X"        |                                |
  |                                |    "Mine match!"               |
  |                                |                                |
  | 8. Calculate QBER              |                                |
  |    QBER = 2.4% [OK]              |                                |
  |    (below 15% threshold)       |                                |
  |                                |                                |
  v                                |                                v
KEY A                              |                             KEY B
(secure)                           |                           (secure)
```

8.2 HKDF Key Derivation

```
+----------------------------------------------------------------+
|                    HKDF KEY DERIVATION                           |
+----------------------------------------------------------------+

Input: BB84 bits (high-entropy quantum randomness)
       Salt (16 bytes, random per encryption)

Step 1: HKDF-Extract
+-----------------+
|  BB84 bits      |
|  (e.g., 4096)   |--+
+-----------------+  |
                     |   +----------------+
+-----------------+  +-->| HMAC-SHA256    |
|  Salt (16 bytes)|--+   |  (Extract)     |
+-----------------+      +-------+--------+
                                 |
                                 v
                         +----------------+
                         | PRK (32 bytes)|
                         | Pseudorandom  |
                         |     Key       |
                         +-------+--------+
                                 |
Step 2: HKDF-Expand              |
                                 |
        +------------------------+------------------------+
        |                        |                        |
        v                        v                        v
+----------------+       +----------------+       +----------------+
| Encryption Key|       |  Auth Key      |       | Signature Key |
|  (32 bytes)   |       |  (32 bytes)    |       |  (derived)     |
|               |       |                |       |                |
| AES-GCM       |       | Optional HMAC |       | Dilithium5     |
| ChaCha20      |       |                |       |                |
| AES-SIV       |       |                |       |                |
+----------------+       +----------------+       +----------------+
```

8.3 Encryption Comparison Matrix

```
+-------------------------------------------------------------------+
|              ENCRYPTION MODE COMPARISON                            |
+--------------+-------------+-------------+-------------------+
|  Feature     |  AES-GCM    |  ChaCha20   |    AES-SIV        |
+--------------+-------------+-------------+-------------------+
|  Type        | Block AEAD  | Stream AEAD | Block AEAD (SIV)  |
|  Key Size    | 256 bits    | 256 bits    | 512 bits (2×256)  |
|  Nonce       | 12 bytes    | 12 bytes    | None (deterministic)|
|  Tag Size    | 16 bytes    | 16 bytes    | 16 bytes (SIV)    |
|  Hardware    | AES-NI      | None needed | AES-NI optional   |
|  Speed (x64) | 4-8 GB/s    | 500-800 MB/s | 2-3 GB/s          |
|  Speed (ARM) | 50-100 MB/s | 500-800 MB/s | 100-200 MB/s      |
|  Misuse Safe | [X] No       | [X] No       | [OK] Yes             |
|  Standard    | NIST 800-38D| RFC 8439    | RFC 5297          |
|  TLS 1.3     | [OK] Yes      | [OK] Yes       | [X] No (niche use)  |
|  Best For    | Servers     | Mobile/IoT  | Research/archives |
+--------------+-------------+-------------+-------------------+
```

8.4 Post-Quantum Signature Verification

```
+---------------------------------------------------------------+
|           DILITHIUM5 SIGNATURE VERIFICATION                    |
+---------------------------------------------------------------+

Encryption (Alice):
+-------------+
| AAD +       |
| Ciphertext  |------+
+-------------+      |
                     |   +----------------+
+-------------+      +-->| Dilithium5     |
| Private Key |------+   | Sign           |
| (Alice's sk)|          +--------+-------+
+-------------+                   |
                                  v
                          +---------------+
                          |  Signature    |
                          |  (4595 bytes) |
                          +---------------+

Decryption (Bob):
+-------------+
| AAD +       |
| Ciphertext  |------+
+-------------+      |
                     |   +----------------+      +----------+
+-------------+      +-->| Dilithium5     |----->| Valid?   |
| Public Key  |------+   | Verify         |      +----------+
| (from pkg)  |          +----------------+      | [OK] Yes   |
+-------------+                                   | [X] No    |
                                                  +----+-----+
                                                       |
                                     +-----------------+-----------------+
                                     |                                   |
                                     v                                   v
                            [OK] Proceed to decrypt            [X] REJECT immediately
                                                                (tampered package)
```

8.5 System Architecture Overview

```
+---------------------------------------------------------------------+
|                     USER INTERFACE (Tkinter GUI)                     |
|  +------------+  +------------+  +------------+  +------------+    |
|  | File Select|  | AES-GCM    |  | ChaCha20   |  |  AES-SIV   |    |
|  +------------+  +------------+  +------------+  +------------+    |
+--------------------------------+------------------------------------+
                                 |
+--------------------------------v------------------------------------+
|                      CONTROLLER LAYER                                |
|  encrypt_file_local() / decrypt_file_local()                         |
|  - Validates inputs                                                  |
|  - Routes to cipher packager                                         |
|  - Generates reports                                                 |
+--------------------------------+------------------------------------+
                                 |
        +------------------------+------------------------+
        |                        |                        |
+-------v---------+   +---------v---------+   +-------v---------+
| BB84 QUANTUM    |   |  KEY DERIVATION   |   |  CIPHER ENGINES |
| - run_qkd()     |-->|  - HKDF-SHA256    |-->|  - AES-GCM      |
| - Channel knobs |   |  - Key separation |   |  - ChaCha20     |
| - Demo preset   |   |  - Salt generation|   |  - AES-SIV      |
| - QBER tracking |   +-------------------+   +-----------------+
+-----------------+                                    |
                                                       |
+------------------------------------------------------v------------+
|                    SECURE PACKAGING LAYER                           |
|  - JSON structure                                                   |
|  - Base64 encoding                                                  |
|  - AAD construction                                                 |
|  - Dilithium5 signatures                                            |
|  - Version tagging                                                  |
+----------------------------------------+----------------------------+
                                         |
+----------------------------------------v----------------------------+
|                      OUTPUT: .bb84 FILE                              |
|  {                                                                   |
|    "ciphertext": "...",     // Encrypted data                       |
|    "salt": "...",           // HKDF salt                            |
|    "nonce": "..." | null,   // Mode-dependent                       |
|    "version": "...",        // Auto-detect cipher                   |
|    "filename": "...",       // Original name                        |
|    "pq_signature": "...",   // Dilithium5                           |
|    "pq_public_key": "..."   // PQ verification key                  |
|  }                                                                   |
+---------------------------------------------------------------------+
```

Appendix: Quick Reference

Cipher Selection Guide

Use AES-GCM if:
- Running on modern x86/x64 CPU with AES-NI
- Need maximum speed (4-8 GB/s)
- Nonce management is handled carefully
- TLS-compatible encryption required

Use ChaCha20-Poly1305 if:

- Running on ARM/mobile/embedded device
- No AES-NI available
- Need constant-time security
- Prefer software-only solution

Use AES-SIV if:
- Research or high-security application
- Nonce management is difficult
- Need deterministic encryption
- Misuse resistance is critical

Security Parameters

Parameter	Value	Purpose
AES key size	256 bits	Quantum-safe (brute force)
ChaCha20 key size	256 bits	Quantum-safe (brute force)
AES-SIV key size	512 bits	Quantum-safe (brute force)
Dilithium level	5 (highest)	Post-quantum signatures
QBER threshold	15%	Eavesdropping detection
p_Z (biased bases)	0.8	Efficient BB84
Salt size	16 bytes	HKDF uniqueness
Nonce size	12 bytes	AEAD uniqueness

File Naming Convention

```
Original: document.pdf
Encrypted (AES-GCM): document_AES-GCM_E.bb84
Encrypted (ChaCha20): document_CHACHA20_E.bb84
Encrypted (AES-SIV): document_AES-SIV_E.bb84
Decrypted: document_AES-GCM_E_decrypted.pdf
```

Document Version: 1.0
Last Updated: December 12, 2025
System Status: [OK] Fully operational with realistic QKD + triple AEAD stack