QSEC2

Quantum-Secured Encrypted Chat

Full Flow Architecture

v2.4.0 | BB84 / AES-256-GCM | Feb 2026

Zero-Knowledge Server Architecture

End-to-End Encrypted Communication

1. Overview

QSEC2 is a real-time encrypted chat application that uses a simulated BB84 Quantum Key Distribution protocol to establish session keys between peers, then encrypts all messages with AES-256-GCM. The core design principle is a zero-knowledge server - the server never sees plaintext messages or cryptographic key material.

> DESIGN PRINCIPLE: The server acts purely as a message relay. All cryptographic operations - key generation, BB84 simulation, AES encryption/decryption - happen exclusively on the client side. If the server is compromised, the attacker gains nothing useful.

Aspect	Technology
Backend	Python / Flask + Flask-SocketIO
Transport	WebSocket (Socket.IO)
Database	SQLite (metadata/logs only)
Key Exchange	Simulated BB84 (128 qubits)
Symmetric Encryption	AES-256-GCM (Web Crypto API)
Frontend	Vanilla HTML/CSS/JS + Three.js + Chart.js

2. File Structure

`server.py` *Flask + SocketIO entry point*
`chatcontainer.py` *All Socket.IO event handlers (600 lines)*
`db.py` *SQLite database layer*
`qber_analysis.py` *QBER computation helpers*
`room_logs.py` *CLI tool to view/follow room logs*
`init_db.py` *DB initialization script*
`data/qsec2.db` *SQLite database (auto-created)*
`static/client.html` *Landing page (join/create room)*
`static/room.html` *Chat room UI (1000+ lines)*
`static/room.js` *Client-side BB84 + AES + UI (1400+ lines)*
`static/style.css` *Global design system*

3. Complete User Flow

Step 1 - Landing Page

User opens / -> client.html. Enters Room ID, Username, Expected Capacity. Clicks "INITIALIZE LINK" -> redirects to /room/<room_id>?username=X&expected=N.

Step 2 - Room Page Loads

room.html + room.js initialize. Socket.IO connects and emits "join" with room, username, expected.

Step 3 - Server Handles Join

Server validates capacity, adds user to ROOM_MEMBERS/ROOM_ORDER/SID_MAP. Broadcasts user_list to all room

members. Sends existing logs + pubkeys to new user.

Step 4 - BB84 Triggers

Leader (first joiner) generates AES-256 Room Key and starts BB84 with each peer. Subsequent users wait for BB84 initiation.

Step 5 - BB84 Key Exchange

6-step quantum key exchange completes. Session key derived from sifted bits via SHA-256. Leader encrypts Room Key with Session Key and sends to peer.

Step 6 - Secure Channel Active

Peer decrypts Room Key. Send button activates. All messages encrypted with AES-256-GCM. Server relays ciphertext blindly.

4. Server Side Architecture

4.1 - server.py (Entry Point)

- Creates Flask app + SocketIO instance (CORS: *, threading mode, 50MB payload limit)
- Initializes SQLite database via db.init_db()
- Maintains in-memory tracking: ROOM_MEMBERS, SID_MAP, ROOM_KEYS
- Routes: / -> client.html, /room/<id> -> room.html
- Delegates all event handling to chatcontainer.py

4.2 - chatcontainer.py (Event Handlers)

Single function register_chat_handlers() registers 20+ Socket.IO event handlers:

Event	Purpose
join	Room membership, capacity check, log events
leave / disconnect	Remove from room, clean up keys, log termination
relay	Generic event relay (inspects type field, broadcasts)
bb84_relay	Forward BB84 messages between specific peers
bb84_meta	Log BB84 sifting completion metadata
bb84_session	Log BB84 session key derivation
announce_pubkey	RSA public key distribution to room
roomkey_share	Encrypted room key delivery to peer
roomkey_generated	Log room key generation event
roomkey_decrypted	Log successful room key decryption
group_message	Relay AES-encrypted chat messages
encrypted_message	Forward encrypted payload
store_encrypted	Store + broadcast encrypted message
plain_message	Plaintext message relay (fallback)
fetch_room_logs	Client polls for audit logs
msg_decrypted	Log successful message decryption

4.3 - In-Memory State

Variable	Type	Purpose
ROOM_MEMBERS	dict[str, set]	room_id -> set of usernames
SID_MAP	dict[str, tuple]	socket_sid -> (room_id, username)
ROOM_ORDER	dict[str, list]	room_id -> ordered list (first = leader)
ROOM_EXPECTED	dict[str, int]	room_id -> expected capacity
PUBKEYS	dict[str, dict]	room_id -> {username: RSA_pubkey_b64}

5. Client Side Architecture

5.1 - room.js Module Breakdown

Module	Lines	Purpose
Config + State	1-30	Colors, BB84 qubit count (128), global state
DOM References	31-55	All UI element references
Logging (sysLog)	73-103	Dual-write to sidebar + popup log
AES-GCM Helpers	88-115	generateRoomKey, aesEncrypt, aesDecrypt
BB84 Protocol	117-261	startBB84, handleBB84 (6-step flow)
UI Helpers	292-332	enableSendButton, setHeaderStatus
Socket Events	334-403	user_list, bb84_signal, connect/join
Chat	405-595	Send/receive (text, image, audio, video, file)
BlochSphere (2D)	597-751	Canvas-based qubit state visualization
ChannelMonitor	753-916	Oscilloscope eavesdropper detection
QBER Analysis	920-955	Stat boxes + verdict display
BlochSphere3D	958-1332	Three.js interactive popup
EventLogPopup	1335-1440	Maximized log viewer

5.2 - room.html Layout

Column	Width	Contents
Left Sidebar	280px	User list, key status, protocol info, session timer
Center	flex: 1	Chat messages + input area
Right Panel	320px	Telemetry: Channel Monitor, Bloch, QBER, Log

6. BB84 Protocol Deep Dive

The BB84 Quantum Key Distribution protocol is simulated entirely on the client side, using the server only as a relay. Here is the step-by-step flow:

Step	Action	Who	Details
1	Generate Qubits	Leader	Generate 128 random bits + 128 random bases
2	Send Encoded Qubits	Leader->Peer	Qubits sent via server relay (bb84_signal)
3	Measure Qubits	Peer	Measure with own random bases, visualize on Bloch
4	Share Bases	Peer->Leader	Bases sent back (public). Qubits NOT revealed
5	Sifting	Both	Keep matching bases, discard rest. Compute QBER
6	Key Derivation	Both	Sifted bits -> SHA-256 -> AES-256 Session Key

QBER (Quantum Bit Error Rate)

- QBER < 11% - Channel is secure, proceed with key exchange

- QBER >= 11% - Potential eavesdropper detected, may abort

Bloch Sphere Visualization

Basis	Bit = 0	Bit = 1	Color
Rectilinear	|0> (north)	|1> (south)	Teal
Diagonal	|+> (equator +X)	|-> (equator -X)	Amber

7. Security Model

> ZERO-KNOWLEDGE SERVER: The server NEVER sees plaintext messages or key material. It only relays encrypted payloads. If compromised, attackers gain only ciphertext.

Key Hierarchy

```
CLIENT A                  SERVER                  CLIENT B
  ---------                 ------                  ---------
  Room Key (generated)          X (never seen)        X (not yet)
       |                                                  |
       +-- BB84 exchange --> relay only  --> BB84 response    |
       |                                                  |
  Session Key (derived)         X                   Session Key (derived)
       |                                                  |
       +-- Encrypt Room Key --> relay --> Decrypt Room Key    |
       |                                                  |
  Room Key ------------------ X ------------------ Room Key
       |                                                  |
       +-- AES-256 Encrypt --> relay --> AES-256 Decrypt ----+
       |    (ciphertext only)  forwarded (plaintext local)   |
```

Encryption Details

Layer	Algorithm	Key Size	Purpose
Session Key	SHA-256 of sifted BB84 bits	256-bit	Encrypt Room Key during transfer
Room Key	AES-256-GCM	256-bit (32B)	Encrypt/decrypt all chat messages
Per-Message	AES-256-GCM	96-bit IV	Authenticated encryption per message

8. Visualization Layer

Widget	Technology	What It Shows
Bloch Sphere (2D)	Canvas 2D	Current qubit state during BB84
Bloch Sphere (3D)	Three.js WebGL	Interactive: drag to rotate, scroll zoom
Channel Monitor	Canvas 2D	Oscilloscope: flat=secure, noise=eavesdropper
QBER Analysis	HTML + JS	Qubits matched/discarded, error rate, verdict
Event Log	HTML (popup)	Timestamped events with clear/maximize
Key Status	HTML badge	NEGOTIATING -> KEY GENERATED
Session Timer	JS interval	HH:MM:SS since join

Bloch Sphere States

State	Visual	When
IDLE	Arrow drifts near |0>	Before BB84 starts
TRANSMIT	Arrow jumps per qubit	During BB84 processing

SECURE	Arrow orbits smoothly	After key exchange complete

9. Database Schema

SQLite database at data/qsec2.db with three tables:

Table	Columns	Purpose
rooms	room_id TEXT PK, key BLOB	Room registry
room_users	room_id TEXT, username TEXT, UNIQUE	User-room associations
room_logs	id INT PK, room_id, message, created_at	Structured audit trail

Structured Log Tags

Tag	Example
[ROOM_CREATED]	room=testroom leader=alice expected=2
[USER_JOIN]	room=testroom user=bob
[ROOM_READY]	room=testroom members=2
[BB84_INIT]	from=alice to=bob length=128
[BB84_SIFTING_COMPLETE]	matched_bits=64
[ROOMKEY_GENERATED]	by=alice size=32bytes
[ROOMKEY_ENCRYPTED]	to=bob method=AES-GCM(session_hash)
[ROOMKEY_DECRYPTED]	user=bob
[MSG_ENCRYPTED]	user=alice iv_len=12
[MSG_RELAYED]	room=testroom sender=alice
[KEY_DESTROYED]	user=alice
[ROOM_TERMINATED]	room=testroom

10. Utility Scripts

Script	Usage	Purpose
room_logs.py	python room_logs.py --room X --follow	Tail room logs in real-time from SQLite
qber_analysis.py	compute_qber(alice, bob, sifted, 10)	Compute mismatches and QBER percentage
init_db.py	python init_db.py	Create/initialize SQLite database tables