International Journal of Computer Techniques–IJCT Volume 13 Issue 3, May-June - 2026

Open Access and Peer Review Journal ISSN 2394-2231 https://ijctjournal.org/

AQCTCS: An Adaptive Quantum–Classical Trust-Based Secure Communication System

Author Details

K. Sivasankar rao | Reg. No: 1522051149 | Dept. of Cyber Security, DSU, Samayapuram-621112, Trichy, India
M. Swagath Kumar | Reg. No: 1522061029 | Dept. of Cyber Security, DSU, Samayapuram-621112, Trichy, India
T. Shasi Kiran | Reg. No: 1522061056 | Dept. of Cyber Security, DSU, Samayapuram-621112, Trichy, India
R. Shanmuka Srinivas | Reg. No: 1522061046 | Dept. of Cyber Security, DSU, Samayapuram-621112, Trichy, India
Ms. D. Sasikala MCA, ME — Assistant Professor, School of Engineering and Technology, DSU

Abstract— *The advancement of quantum computing presents significant risks to conventional cryptographic systems that secure modern communication networks. This paper proposes an Adaptive Quantum–Classical Trust Communication System (AQCTCS), a hybrid secure messaging framework designed to enhance resilience against both classical and quantum threats. The system integrates three security layers: AES-256 for classical encryption, Post-Quantum Cryptography (PQC) for quantum-resistant protection, and a BB84-based Quantum Key Distribution (QKD) simulation for high-risk scenarios. A Trust Engine continuously evaluates security indicators, including anomaly detection parameters and Quantum Bit Error Rate (QBER), to compute a dynamic trust score. Based on this score, an Adaptive Security Controller automatically selects the appropriate encryption mode, balancing performance and security. The platform is implemented as a web application using React for the frontend, FastAPI for backend services, and Qiskit for QKD simulation. AQCTCS demonstrates an adaptive, scalable, and future-ready communication architecture suitable for post-quantum environments.*

Keywords— *Quantum Key Distribution, Post-Quantum Cryptography, AES-256, Trust Engine, BB84 Protocol, Adaptive Encryption, Secure Messaging, QBER.*

I. INTRODUCTION

The rapid proliferation of quantum computing introduces unprecedented threats to conventional cryptographic systems that underpin global digital communication. Algorithms such as RSA and ECC, which rely on the computational difficulty of integer factorization and discrete logarithm problems, are rendered vulnerable by Shor's algorithm running on sufficiently powerful quantum hardware. This necessitates the development of communication architectures that remain secure under both classical and quantum adversarial models.

Traditional cryptographic frameworks adopt a static encryption approach, applying a single algorithm regardless of the dynamic security context. This rigidity fails to account for fluctuating threat levels, session-specific risk factors, and the varying computational costs of different encryption strategies. As communication environments grow increasingly heterogeneous, there is a compelling need for adaptive systems capable of selecting appropriate cryptographic mechanisms in real time.

This paper presents AQCTCS — an Adaptive Quantum–Classical Trust-Based Secure Communication System — which addresses these challenges through a multi-layered, trust-driven encryption framework. AQCTCS integrates AES-256 for high-trust low-risk sessions, Post-Quantum Cryptography (PQC) for medium-risk environments, and a BB84-simulated Quantum Key Distribution (QKD) mechanism for low-trust critical scenarios. A dedicated Trust Engine evaluates continuous security metrics including session anomaly rates, Quantum Bit Error Rate (QBER), key freshness, and session duration to compute a dynamic trust score, which then drives automatic encryption mode switching.

The key contributions of this paper are as follows:

- Design of a three-tier adaptive encryption architecture combining AES-256, PQC, and QKD-based security.
- Development of a real-time Trust Engine that computes dynamic trust scores from multiple security indicators.
- Implementation of an Adaptive Security Controller that performs threshold-based automatic encryption mode switching.
- A full-stack web application deployment using React, FastAPI, and Qiskit.
- Systematic evaluation demonstrating AQCTCS's superiority over static encryption frameworks.

II. LITERATURE SURVEY

Several foundational and contemporary works inform the design of AQCTCS. The following table summarizes the key references, their contributions, limitations, and how AQCTCS builds upon or improves them.

Table I: Literature Survey Summary

Ref / Year	Area	Key Idea	Limitations	How AQCTCS Improves
Joshi et al., 2020	Network QKD	Multi-user QKD network without trusted nodes	Infrastructure-focused; not app-level messaging	AQCTCS uses BB84/QBER concept (simulated) for high-risk sessions
NIST, 2024	PQC Standardization	First finalized PQC standards (FIPS 203/204/205)	Standards don't define adaptive switching	AQCTCS transitions between classical → PQC → quantum-inspired modes
Auerbach et al., 2025	PQ Secure Messaging	PQ messaging overhead and bandwidth constraints	No single design fits all networks; overhead issues	AQCTCS solves via adaptive selection (AES when safe, PQC/QKD when risky)
Adaptive Trust Systems	Trust Security	Dynamic trust affects access/auth decisions	Encryption method remains fixed	AQCTCS uniquely uses trust score to control encryption switching

Joshi et al. (2020) demonstrated multi-user entanglement-based QKD over a metropolitan network, establishing the practical viability of quantum-secured communication. However, their approach targeted network-layer infrastructure without addressing application-level adaptive security. NIST's 2024 finalization of PQC standards (FIPS 203, 204, 205) provides the algorithmic backbone for quantum-resistant systems, though these standards do not define adaptive switching logic. Auerbach et al. (2025) identified bandwidth and overhead constraints in PQ-secure messaging systems, which AQCTCS addresses through selective mode activation driven by trust scoring. Adaptive trust models in access control systems further motivate the AQCTCS Trust Engine design, which extends trust-based decision-making into the domain of encryption selection. The system is implemented using a modern software stack consisting of FastAPI for backend services, React Native (Expo) for cross-platform frontend development, and WebSocket-based real-time communication. The backend handles authentication, trust evaluation, and encryption policy enforcement, while the frontend provides an interactive messaging interface similar to modern chat applications. The database layer stores user identities, encrypted messages, and trust metrics securely.

Feature	Classical Systems	Single-Layer Crypto	Fixed PQC Systems	AQCTCS (Proposed)
Encryption Mode	Static AES only	Single algorithm	PQC only	**Adaptive AES/PQC/QKD**
Quantum Resistance	None	None	Yes	**Yes (Multi-Layer)**
Trust Evaluation	Absent	Absent	Absent	**Dynamic Trust Engine**
Key Management	Manual	Basic rotation	PQC keys	**Adaptive rotation + BB84**

Scalability	Moderate	Moderate	High	**High**
Deployment	Local/Cloud	Cloud	Cloud	**Web App (React + FastAPI)**

III. COMPARISON WITH PREVIOUS METHODOLOGY

Conventional cryptographic communication systems employ static, single-mode encryption. Classical systems rely exclusively on AES or similar symmetric algorithms, which, while computationally efficient, offer no resistance to quantum attacks. PQC-only systems improve quantum resilience but apply a single algorithm uniformly without adapting to real-time threat levels. The table below provides a structured comparison.

Table II: Comparison with Previous Methodologies

The proposed AQCTCS framework differentiates itself through three dimensions: (1) dynamic trust evaluation that continuously assesses session risk, (2) adaptive multi-mode encryption that selects the most appropriate algorithm based on trust score thresholds, and (3) seamless key management integrating classical key generation with BB84-inspired quantum key refreshing. This adaptive behavior ensures that computational overhead from quantum-resistant algorithms is incurred only when justified by the threat level, maintaining performance during low-risk sessions.

IV. PROPOSED SYSTEM

A. System Architecture Overview

AQCTCS is architected as a layered web application comprising the following primary modules: User Authentication Module, Trust Evaluation Module (Trust Engine), Hybrid Encryption Module, Secure Messaging Module (Real-Time Chat Engine), Key Management Module, Backend API Module, Database Management Module, and Mobile Application Interface Module. These modules interact through clearly defined interfaces, ensuring modularity and maintainability.

B. User Authentication Module

The authentication subsystem enforces access control through JWT-based token management. User credentials are stored using bcrypt-hashed passwords via passlib, preventing plain-text storage vulnerabilities. Upon successful login, a signed JWT token is issued, which is validated for all protected API endpoints and WebSocket connections. This establishes a secure perimeter before any cryptographic operations are initiated.

C. Trust Evaluation Module (Trust Engine)

The Trust Engine is the central decision-making component of AQCTCS. Each communication session is initialized with a trust score of 100. The engine continuously updates this score based on a weighted combination of security indicators including: number of messages exchanged, session duration, key freshness (time since last key rotation), anomaly detection flags, and connection instability metrics.

Encryption mode is determined by the following threshold logic:

- Trust $\geq 75 \rightarrow$ AES-256 Mode (High Trust, Low Risk)
- $40 \leq$ Trust $< 75 \rightarrow$ PQC Mode (Medium Trust, Moderate Risk)
- Trust $< 40 \rightarrow$ Quantum Mode / Simulated QKD (Low Trust, High Risk)

The QBER (Quantum Bit Error Rate) is monitored during QKD-mode sessions and fed back into the trust score computation, providing a closed-loop security evaluation mechanism.

D. Hybrid Encryption Module

The encryption module implements all three security tiers. AES-256 (Advanced Encryption Standard with 256-bit keys) is applied during high-trust sessions, providing computationally efficient symmetric encryption suitable for real-time messaging. PQC mode employs simulated post-quantum cryptographic primitives from the lattice-based (Kyber, Dilithium) and hash-based (SPHINCS+) families, representing the NIST 2024 standardized algorithms. Each message is tagged with its encryption mode (AES/PQC/QKD) to enable correct decryption by the receiver.

In Quantum Mode, the BB84 protocol is simulated using Qiskit, generating fresh session keys and performing frequent key refreshes. The BB84 simulation detects eavesdropping via QBER monitoring, triggering immediate key rotation when anomalous error rates are detected.

E. Key Management Module

The Key Management Module maintains cryptographic key lifecycles across all three modes. Symmetric AES keys are generated using a Cryptographically Secure Pseudo-Random Number Generator (CSPRNG). PQC-related key pairs are simulated for the prototype. In Quantum Mode, BB84-derived keys are generated as fresh random bitstrings and distributed to communicating parties via secure system messages. Key rotation is triggered by: fixed time intervals, message count thresholds, or encryption mode transitions (AES→PQC→QKD).

F. Secure Messaging Module

Real-time message delivery is implemented through WebSockets, enabling bidirectional communication with low latency. The module supports one-to-one and group (room-based) chat sessions. Before transmission, each message is processed through the Trust Engine and Encryption Module. The messaging engine ensures that encryption mode transitions are seamless, with no message loss during mode switching.

G. Backend API and Database Modules

FastAPI provides the REST API layer, handling endpoints for user registration (/register), login (/login), room management (/rooms), and user profile retrieval. Pydantic enforces request validation. SQLite is used for the prototype database, with PostgreSQL as the production target. Stored data includes user credentials (hashed), chat room configurations, and optional message metadata (sender, receiver, timestamp, encryption mode).

H. Frontend Interface

The React-based frontend provides a WhatsApp-like messaging interface. Key UI elements include: chat bubble views with encryption mode indicators (AES/PQC/QKD), real-time WebSocket connection management with auto-reconnect, and token-based session persistence using AsyncStorage/SecureStore. The encryption mode indicator provides users with visual feedback on the current security level of their session.

V. ALGORITHMS

A. AES-256 (Classical Encryption)

AES-256 is applied during high-trust sessions. It employs a 256-bit symmetric key across 14 rounds of SubBytes, ShiftRows, MixColumns, and AddRoundKey operations. Its computational efficiency makes it ideal for real-time encrypted chat where latency must be minimized. AES-256 is considered quantum-resistant against Grover's search algorithm to the extent that its effective security remains at 128 bits even under quantum attack — acceptable for current risk levels.

B. Trust Scoring Algorithm

The Trust Scoring Algorithm is a custom rule-based decision engine constituting the core novelty of AQCTCS. Beginning at a baseline score of 100, the algorithm applies weighted decrements for each adverse security event. Thresholds (75 and 40) map the continuous trust score to discrete encryption modes. The algorithm enables adaptive security that applies stronger cryptographic primitives only when the session risk profile warrants the additional computational overhead.

C. PQC Algorithms (Simulated)

The AQCTCS prototype simulates PQC mode to demonstrate mode-switching behavior and message tagging. The conceptual basis references NIST-standardized algorithms: CRYSTALS-Kyber for key encapsulation (lattice-based, ML-KEM per FIPS 203) and CRYSTALS-Dilithium for digital signatures (lattice-based, ML-DSA per FIPS 204). SPHINCS+ (hash-based, SLH-DSA per FIPS 205) is referenced for stateless signature schemes. Production deployment would replace the simulation with actual library implementations.

D. BB84 Protocol (Simulated QKD)

The BB84 quantum key distribution protocol, originally proposed by Bennett and Brassard (1984), generates shared secret keys using quantum-mechanical properties. AQCTCS simulates BB84 via Qiskit by generating random quantum basis choices, encoding key bits, and computing QBER from simulated measurement outcomes. Detection of anomalous QBER (typically > 11% threshold) signals potential eavesdropping and triggers immediate key rotation and trust score reduction.

E. Key Rotation Algorithm

Key rotation is implemented as a scheduled background process that refreshes cryptographic keys based on time intervals, message count thresholds, and mode transitions. This limits the impact of key compromise by reducing the volume of ciphertext encrypted under any single key — a critical security property for long-duration sessions.

VI. SOFTWARE AND HARDWARE COMPONENTS

A. Hardware Requirements

- Development Laptop/Desktop: Minimum 8 GB RAM (16 GB recommended), Intel i5/Ryzen 5 or higher, SSD ≥ 256 GB
- Android Smartphone (for APK testing); iPhone (optional, iOS testing)
- Stable Internet Connection; Cloud Server/Virtual Machine for deployment
- Optional (Future): Dedicated Linux server, Firewall device, Quantum communication hardware for real QKD

B. Software Stack

- Backend: Python 3.10+, FastAPI, Uvicorn, WebSocket support, Pydantic, SQLAlchemy/SQLModel, SQLite/PostgreSQL, PyJWT/python-jose, Passlib, bcrypt, cryptography library, Qiskit
- Frontend: React Native, Expo, React Navigation, WebSocket client, Fetch/Axios, AsyncStorage/expo-secure-store, Gifted Chat
- DevOps: VS Code, Git, GitHub, Postman, Docker, Docker Compose, Nginx, Let's Encrypt SSL, Expo EAS Build
- Deployment: Render, Railway, Fly.io, DigitalOcean, AWS, or GCP

VII. RESULTS AND DISCUSSION

A. Experimental Setup

The prototype AQCTCS system was deployed on a local machine running 64-bit Windows with an Intel Core i5 processor, 16 GB RAM, and a locally configured FastAPI server. The React frontend was tested on both web browser and Android APK environments. WebSocket communication was validated for both one-to-one and group chat scenarios.

B. Encryption Mode Transitions

Controlled experiments demonstrated successful automatic transitions across all three encryption modes. Sessions initialized with trust scores of 100 operated under AES-256. As simulated anomaly events were introduced — including increased message frequency, key aging, and injected QBER variance — the trust score decreased below the defined thresholds, triggering transitions first to PQC mode and subsequently to QKD mode. Mode transitions were logged and visually indicated on the frontend encryption mode indicator.

C. Performance Comparison

Table III: System Performance Comparison

Metric	Classical Encryption	PQC-Only System	AQCTCS (Proposed)
Encryption Layers	1 (AES)	1 (PQC)	**3 (AES + PQC + QKD)**
Trust Score Engine	Absent	Absent	**Dynamic (0–100)**
Adaptive Switching	No	No	**Yes (Threshold-Based)**
Key Freshness	Static	Moderate	**Continuous Rotation**
QBER Monitoring	No	No	**Yes (Simulated)**
Real-Time Messaging	WebSocket/HTTP	WebSocket/HTTP	**WebSocket + Encrypted**
Quantum Threat	None	Partial	**High**

Resistance			

The adaptive encryption framework introduces minor computational overhead during mode transitions but maintains low-latency operation during stable AES sessions. The inclusion of trust-driven mode selection ensures that the more computationally intensive QKD-simulated mode is activated only when session risk metrics justify the additional processing cost.

D. Discussion

AQCTCS demonstrates that adaptive, trust-driven cryptographic selection is architecturally feasible and practically deployable. The Trust Engine's closed-loop feedback mechanism — incorporating QBER, anomaly detection, and key freshness — provides a nuanced security posture that static encryption frameworks cannot achieve. The system's layered design ensures that individual modules (Trust Engine, Encryption, Key Management) can be upgraded or replaced independently as cryptographic standards evolve.

A key limitation of the current prototype is the use of simulated PQC and QKD implementations. Deployment in production environments would require integration with certified PQC libraries (e.g., liboqs) and, for genuine quantum security, hardware quantum key distribution infrastructure. Additionally, CPU-based local inference constrains throughput for high-concurrency deployments.

VIII. CONCLUSION

This paper presented AQCTCS, an Adaptive Quantum–Classical Trust-Based Secure Communication System that addresses the dual challenge of securing communications against both contemporary classical threats and emerging quantum computing capabilities. By integrating AES-256, PQC algorithms, and BB84-based QKD simulation within a unified adaptive framework governed by a dynamic Trust Engine, AQCTCS achieves context-sensitive cryptographic selection without manual intervention.

The system's architecture — comprising modular authentication, trust evaluation, adaptive encryption, real-time messaging, and key management components — demonstrates engineering rigor and deployment feasibility through a fully functional web application built with React and FastAPI. Experimental evaluation confirms that automatic mode switching based on trust score thresholds operates correctly and that the system maintains usability throughout security-level transitions.

AQCTCS establishes a practical foundation for post-quantum secure communication architectures. Future work will focus on integrating certified PQC libraries, exploring hardware QKD interfaces, implementing asynchronous agent processing, and validating the system under high-concurrency production workloads.

REFERENCES

[1] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," Proc. 35th Annual Symposium on Foundations of Computer Science, 1994, pp. 124–134.

[2] C. H. Bennett and G. Brassard, "Quantum cryptography: Public key distribution and coin tossing," Proc. IEEE Int. Conf. on Computers, Systems and Signal Processing, 1984.

[3] D. J. Bernstein, J. Buchmann, and E. Dahmen, Post-Quantum Cryptography, Springer, 2009.

[4] National Institute of Standards and Technology (NIST), "FIPS 197: Advanced Encryption Standard (AES)," 2001.

[5] National Institute of Standards and Technology (NIST), "Post-Quantum Cryptography Standardization Project – FIPS 203, 204, 205," 2024.

[6] S. Pirandola et al., "Advances in quantum cryptography," Advances in Optics and Photonics, vol. 12, no. 4, pp. 1012–1236, 2020.

[7] S. K. Joshi et al., "A trusted node-free eight-user metropolitan quantum communication network," Science Advances, vol. 6, no. 36, 2020.

[8] J. Auerbach et al., "How to compare bandwidth-constrained secure messaging protocols," USENIX Security Symposium, 2025.

[9] T. Cohn-Gordon et al., "A formal security analysis of the Signal messaging protocol," IEEE European Symposium on Security and Privacy, 2017.

[10] H. Abraham et al., "Qiskit: An open-source framework for quantum computing," 2019. [Online]. Available: https://qiskit.org

[11] S. Marsh, "Formalising trust as a computational concept," Ph.D. thesis, University of Stirling, 1994.

[12] H. J. Kimble, "The quantum internet," Nature, vol. 453, pp. 1023–1030, 2008.