PQC-DTLS 1.3 Implementation on RISC-V Bare-Metal

Inter IIT Tech Meet 14.0 – Qtrino Labs Challenge

Team 94

1 Problem Understanding

The challenge requires implementing a Post-Quantum Cryptography (PQC) enabled DTLS 1.3 client on a resource-constrained RISC-V bare-metal environment, establishing secure communication with a host DTLS server using ML-KEM (Kyber) for quantum-resistant key exchange.

Key Objectives: (1) DTLS 1.3 client on LiteX VexRiscv SoC, (2) ML-KEM-512 post-quantum key exchange, (3) wolfSSL/wolfCrypt integration, (4) LiteETH networking, (5) Embedded optimization.

2 Architecture and Design

The system comprises two components via TAP virtual network:

Host (Linux): PQC-DTLS 1.3 server with wolfSSL at `192.168.1.100:11111`, performing ML-KEM encapsulation and AES-GCM encryption.

LiteX Simulation: Bare-metal DTLS client on VexRiscv (32-bit RV32IM, ∼100MHz) with LiteETH at `192.168.1.50:22222`.

Software Stack: Application (main.c) → wolfSSL with custom I/O → wolfCrypt (ML-KEM, AES-GCM, SHA-256) → LiteETH UDP → LiteX CSR HAL.

3 PQC Algorithm Selection

ML-KEM-512 (formerly Kyber-512) selected as KEM for: NIST FIPS 203 standardization, memory efficiency (800B pubkey, 768B ciphertext), Level 1 security (128-bit classical), and native wolfSSL support.

Symmetric: AES-128-GCM for authenticated encryption, SHA-256 for key derivation, SHA3/SHAKE for ML-KEM internals.

4 Firmware Implementation

4.1 Boot Sequence

IRQ/UART init → LiteETH PHY init → UDP stack with MAC/IP → ARP resolution → wolfSSL init → DTLS 1.3 context → Handshake.

4.2 Custom I/O Callbacks

wolfSSL's socket I/O replaced with LiteETH callbacks: **Send** copies to TX buffer via `udp_send()`; **Receive** polls ring buffer (8 slots) from `udp_rx_callback` ISR with timeout.

Region	Address	Size
Main RAM	0x40000000	100 MB
Stack	(top of RAM)	500 KB
Heap	(after BSS)	500 KB

4.3 Memory Layout

5 wolfSSL Configuration

Key `user_settings.h` macros:

```
/* DTLS 1.3 + PQC */
#define WOLFSSL_DTLS13
#define WOLFSSL_HAVE_MLKEM
#define WOLFSSL_WC_MLKEM
/* Crypto */
#define HAVE_AESGCM
#define WOLFSSL_SHA3
/* Embedded */
#define WOLFSSL_SMALL_STACK
#define NO_FILESYSTEM
```

Enabled: DTLS 1.3 with fragmentation, ML-KEM, ECC (Curve25519), AES-GCM, SHA-256/512, SHA3/SHAKE, HKDF.

6 Challenges and Solutions

1. **Memory:** 500KB stack/heap, `WOLFSSL_SMALL_STACK`
2. **No Sockets:** Custom wolfSSL I/O wrapping LiteETH UDP
3. **No HW RNG:** PRNG with `CUSTOM_RAND_GENERATE_SEED`
4. **Packet Loss:** 8-slot ring buffer with timeout polling
5. **Build:** Custom Makefile integrating wolfCrypt with LiteX

7 Security Analysis

- **Quantum Resistance:** ML-KEM protects against Shor's algorithm
- **Forward Secrecy:** Ephemeral ML-KEM keys per session
- **AEAD:** AES-GCM provides confidentiality + integrity
- **Limitations:** Software PRNG (demo); production needs HW TRNG

8 Performance

Metric	Value
Firmware (.text)	54 KB
Total Binary	59 KB
ML-KEM KeyGen	∼50 ms
ML-KEM Encaps/Decaps	∼30-35 ms

Optimizations: `WOLFSSL_SP_SMALL` for code size, `SP_WORD_SIZE=32` for RV32, `WOLFSSL_AES_SMALL_TABLES`, `WOLFSSL_SP_NO_MALLOC`.

9 Conclusion

We implemented a complete PQC-DTLS 1.3 client on bare-metal RISC-V using LiteX simulation. The system establishes quantum-resistant secure channels with a Linux DTLS server using ML-KEM-512 key exchange. Achievements: full DTLS 1.3 handshake, custom LiteETH I/O, ring buffer packet handling, and compact 59KB firmware.

Repository: `https://github.com/SreejitaChatterjee/Team94_L1`

References

[1] NIST, "FIPS 203: ML-KEM Standard," 2024.

[2] wolfSSL Inc., "wolfSSL Embedded SSL/TLS," `https://www.wolfssl.com/`

[3] Enjoy-Digital, "LiteX SoC Builder," `https://github.com/enjoy-digital/litex`

[4] E. Rescorla et al., "DTLS 1.3," RFC 9147, 2022.

Annexures

Annexure A: Directory Structure

```
Team94_L1/
|-- LP_Constraint_Env_Sim/          # LiteX simulation environment
|   |-- boot/                       # RISC-V client firmware
|   |   |-- main.c                  # PQC-DTLS 1.3 client
|   |   |-- Makefile                # Build configuration
|   |   |-- server/                 # Linux DTLS server
|   |   |   `-- pqc_dtls_server.c
|   |   |-- wolfssl/                # wolfSSL headers
|   |   `-- wolfcrypt/src/          # wolfCrypt source files
|   |-- build/                      # LiteX build output
|   |-- report/                     # Technical report
|   `-- litex/, liteeth/, migen/    # LiteX framework
|
`-- README.md                       # Project documentation
```

Annexure B: Build Instructions

```
# Build firmware
cd LP_Constraint_Env_Sim/boot && make clean && make

# Build server
cd boot/server && make dtls13

# Setup TAP interface
sudo ip tuntap add tap0 mode tap user $USER
sudo ip addr add 192.168.1.100/24 dev tap0
sudo ip link set tap0 up

# Run simulation
litex_sim --with-ethernet --ethernet-tap tap0 --ram-init=boot/boot.bin
```

Annexure C: DTLS 1.3 Handshake Flow

```
Client (LiteX)                       Server (Linux)
     |                                      |
     |-------- ClientHello + ML-KEM ------->|
     |<------- ServerHello + ML-KEM --------|
     |<------- EncryptedExtensions ---------|
     |<------- Finished --------------------|
     |-------- Finished ------------------->|
     |<======= Application Data ==========>|
     |        (AES-128-GCM encrypted)       |
```

Annexure D: Network Configuration

Parameter	Client (LiteX)	Server (Host)
IP Address	192.168.1.50	192.168.1.100
UDP Port	22222	11111
Interface	LiteETH	tap0

Annexure E: Entropy Source (Demo)

```
int CustomRngGenerateBlock(unsigned char *output, unsigned int sz) {
    static unsigned int seed = 0xDEADBEEF;
    for (unsigned int i = 0; i < sz; i++) {
```

```
        seed = seed * 1103515245 + 12345;
        output[i] = (unsigned char)(seed >> 16);
    }
    return 0;
}
// Production: Use hardware TRNG (ring oscillator-based)
```