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:

- Implement DTLS 1.3 client on LiteX-simulated VexRiscv SoC
- Integrate ML-KEM-512 post-quantum key exchange
- Use wolfSSL/wolfCrypt for cryptographic operations
- Establish network communication via LiteETH
- Optimize for embedded constraints (memory, compute)

2 Architecture and Design

2.1 System Overview

The architecture consists of two components connected via a virtual TAP network interface:

Host Machine (Linux): Runs the PQC-DTLS 1.3 server with wolfSSL, performing ML-KEM encapsulation, AES-GCM encryption, and SHA-256 key derivation over UDP sockets at `192.168.1.100:11111`.

LiteX Simulation: Executes the bare-metal DTLS client on a VexRiscv RISC-V softcore (32-bit RV32IM, ∼100MHz) with LiteETH MAC at `192.168.1.50:22222`.

2.2 Software Stack

1. **Application Layer:** PQC-DTLS 1.3 client (`main.c`)
2. **TLS Layer:** wolfSSL with custom I/O callbacks
3. **Crypto Layer:** wolfCrypt (ML-KEM, AES-GCM, SHA-256)
4. **Network Layer:** LiteETH UDP API with ring buffer
5. **HAL:** LiteX CSR-based peripheral access

3 PQC Algorithm Selection

3.1 ML-KEM-512 (Kyber)

We selected **ML-KEM-512** (formerly Kyber-512) as the post-quantum Key Encapsulation Mechanism:

- **NIST Standardization:** Selected as primary KEM in FIPS 203
- **Memory Efficiency:** Smallest variant suitable for embedded
- **Security Level:** NIST Level 1 (128-bit classical security)
- **wolfSSL Support:** Native implementation available

Parameter	ML-KEM-512
Public Key	800 bytes
Secret Key	1,632 bytes
Ciphertext	768 bytes
Shared Secret	32 bytes

Table 1: ML-KEM-512 key sizes

3.2 Symmetric Cryptography

- **AES-128-GCM:** Authenticated encryption for record protection
- **SHA-256:** Key derivation and HKDF operations
- **SHA3/SHAKE:** Required internally by ML-KEM

4 Firmware Implementation

4.1 Initialization Sequence

1. IRQ setup and UART initialization
2. LiteETH PHY initialization
3. UDP stack startup with MAC/IP configuration
4. ARP resolution for server address
5. wolfSSL library initialization
6. DTLS 1.3 context creation
7. Handshake execution

4.2 Custom I/O Callbacks

wolfSSL's socket-based I/O is replaced with LiteETH-specific callbacks:

Send Callback: Copies data to LiteETH TX buffer and triggers UDP transmission via `udp_send()`.

Receive Callback: Polls a ring buffer (8 entries) populated by the `udp_rx_callback` ISR, with configurable timeout for retransmission handling.

4.3 Memory Layout

Region	Address	Size
ROM	0x00000000	128 KB
SRAM	0x10000000	8 KB
Main RAM	0x40000000	100 MB
Stack	(top of RAM)	500 KB
Heap	(after BSS)	500 KB

Table 2: Memory regions from linker configuration

5 wolfSSL/wolfCrypt Configuration

Key configuration macros in `user_settings.h`:

```c
/* DTLS 1.3 Support */
#define WOLFSSL_DTLS13
#define WOLFSSL_TLS13
#define WOLFSSL_DTLS_CH_FRAG

/* ML-KEM (Kyber) PQC */
#define WOLFSSL_HAVE_MLKEM
#define WOLFSSL_WC_MLKEM

/* Crypto primitives */
#define HAVE_AESGCM
#define WOLFSSL_SHA256
#define WOLFSSL_SHA3

/* Embedded optimizations */
#define WOLFSSL_SMALL_STACK
#define WOLFSSL_SP_MATH
#define NO_FILESYSTEM
```

Enabled Features: DTLS 1.3 with fragmentation, ML-KEM post-quantum KEM, ECC (Curve25519, Ed25519), AES-GCM, SHA-256/512, SHA3/SHAKE, and HKDF key derivation.

6 Challenges and Solutions

1. **Memory Constraints:** ML-KEM and DTLS require significant stack space. Solution: Allocated 500KB stack and 500KB heap, enabled `WOLFSSL_SMALL_STACK`.
2. **No OS/Socket Layer:** Standard BSD sockets unavailable. Solution: Implemented custom wolfSSL I/O callbacks wrapping LiteETH UDP API.
3. **Timing/RNG:** No hardware RNG or RTC. Solution: Implemented PRNG with `CUSTOM_RAND_GENERATE_SEED` and timer-based `XTIME()`.
4. **Network Synchronization:** UDP packet loss during handshake. Solution: Ring buffer with 8-slot queue and timeout-based polling.
5. **Build Complexity:** Cross-compilation with wolfSSL. Solution: Custom Makefile integrating wolfCrypt sources with LiteX build system.

7 Security Analysis

Aspect	Status	Production
RNG (Client)	SW PRNG	Use HW TRNG
RNG (Server)	/dev/urandom	Acceptable
Key Storage	RAM only	Secure element
Replay Protection	None	Add sequence #
Forward Secrecy	Per-session	Implemented

Table 3: Security implementation status

Quantum Security: ML-KEM-512 provides protection against Shor's algorithm attacks. Key recovery requires 2^{143} classical or 2^{107} quantum operations.

8 Performance Metrics

Optimizations Applied:

- `WOLFSSL_SP_SMALL`: Reduces code size
- `SP_WORD_SIZE=32`: Matches RV32 architecture
- `WOLFSSL_AES_SMALL_TABLES`: Reduces AES LUT size
- `WOLFSSL_SP_NO_MALLOC`: Stack-based allocation

Metric	Value
Firmware Size (.text)	55,656 bytes (54 KB)
Read-only Data (.rodata)	4,272 bytes (4 KB)
Total Binary (boot.bin)	59,952 bytes (59 KB)
Stack Allocation	500 KB
Heap Allocation	500 KB
ML-KEM-512 KeyGen	∼50 ms
ML-KEM-512 Encaps	∼30 ms
ML-KEM-512 Decaps	∼35 ms

Table 4: Memory and performance on VexRiscv @ 100MHz

9 Session Resumption & Entropy

Session Resumption: wolfSSL's DTLS 1.3 supports PSK-based session resumption via `HAVE_SESSION_TICKET`, enabling abbreviated handshakes and reduced computational overhead on reconnection.

Entropy Source: Current implementation uses an LCG-based PRNG seeded with `0xDEADBEEF` for demonstration. Production systems should integrate hardware TRNG (ring oscillator-based) or LiteX's PRNG peripheral with proper entropy accumulation.

10 Build System

The firmware uses a custom Makefile integrated with LiteX:

- **Toolchain:** riscv64-unknown-elf-gcc cross-compiler
- **C Library:** picolibc with nano-malloc
- **Linker:** Custom script (500KB stack/heap)
- **Libraries:** libliteeth, libbase
- **Flags:** `-DWOLFSSL_USER_SETTINGS -DWOLFSSL_SMALL_STACK`

11 Protocol Flow

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

Key Derivation: After ML-KEM key exchange produces a 32-byte shared secret, AES session keys are derived using SHA-256:

```
hash = SHA256(shared_secret || "client_key")
aes_key = hash[0:15]    // 16 bytes
aes_iv  = hash[16:27]   // 12 bytes
```

12 Conclusion

We successfully implemented a complete PQC-DTLS 1.3 client on a bare-metal RISC-V platform using LiteX simulation. The system establishes quantum-resistant secure channels with a Linux-based DTLS server using ML-KEM-512 key exchange. Key achievements:

- Complete DTLS 1.3 handshake with ML-KEM
- Custom LiteETH I/O callbacks for UDP networking
- Ring buffer-based packet handling with timeout
- Companion Linux server (`pqc_dtls_server.c`)
- Compact firmware footprint (∼59 KB binary)

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

References

[1] NIST, "FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard," 2024.

[2] wolfSSL Inc., "wolfSSL Embedded SSL/TLS Library," 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.

[5] R. Avanzi et al., "CRYSTALS-Kyber Algorithm Specifications," NIST PQC, 2021.

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
|   |   |-- linker.ld               # Memory layout
|   |   |-- wolfssl/                # wolfSSL headers
|   |   `-- wolfcrypt/src/          # wolfCrypt source (104 files)
|   |-- build/                      # LiteX build output
|   |-- report/                     # Technical report
|   `-- litex/, liteeth/, migen/    # LiteX framework
|
|-- boot/                           # Root-level boot (with server)
|   |-- main.c                      # Client firmware
|   `-- server/                     # Linux DTLS server
|       |-- pqc_dtls_server.c       # DTLS 1.3 server
|       `-- Makefile
|
`-- README.md                       # Project documentation
```

Annexure B: Build Instructions

```
# Prerequisites: RISC-V toolchain, LiteX, wolfSSL, Python 3.8+

# Build firmware
cd LP_Constraint_Env_Sim/boot
make clean && make
# Output: boot.bin, boot.elf

# Build server (from repo root)
cd boot/server
make dtls13
# Output: build/pqc_dtls_server

# 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 ------->|
     |                                       |
     |<------- HelloRetryRequest -----------|  (cookie exchange)
     |                                       |
     |-------- ClientHello (retry) -------->|
     |                                       |
     |<------- ServerHello + ML-KEM --------|
     |<------- EncryptedExtensions ---------|
     |<------- Certificate -----------------|  (optional)
     |<------- CertificateVerify -----------|  (optional)
     |<------- 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
MAC Address	10:e2:d5:00:00:02	(host default)
Interface	LiteETH	tap0

Table 5: Network configuration for DTLS communication

Annexure E: Custom RNG Implementation

```c
/* Demo PRNG - NOT for production use */
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 recommendation:
 * - Integrate hardware TRNG (ring oscillator-based)
 * - Use LiteX PRNG peripheral with entropy accumulation
 * - Implement proper seed management and reseeding
 */
```

Annexure F: wolfSSL Configuration Summary

Feature	Configuration Macro
DTLS 1.3	`WOLFSSL_DTLS13`, `WOLFSSL_TLS13`
ClientHello Fragmentation	`WOLFSSL_DTLS_CH_FRAG`
ML-KEM (Kyber)	`WOLFSSL_HAVE_MLKEM`, `WOLFSSL_WC_MLKEM`
AES-GCM	`HAVE_AESGCM`
SHA-256/512	`WOLFSSL_SHA256`, `WOLFSSL_SHA512`
SHA3/SHAKE	`WOLFSSL_SHA3`
ECC Support	`HAVE_ECC`, `HAVE_CURVE25519`
Small Stack	`WOLFSSL_SMALL_STACK`
SP Math	`WOLFSSL_SP_MATH`, `SP_WORD_SIZE=32`
No Filesystem	`NO_FILESYSTEM`
Custom RNG	`CUSTOM_RAND_GENERATE_SEED`

Table 6: Key wolfSSL/wolfCrypt configuration macros