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

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. The system must establish secure communication with a host-based DTLS server using quantum-resistant cryptographic algorithms, specifically ML-KEM (Kyber) for key encapsulation.

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 Choices

3.1 ML-KEM-512 (Kyber)

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

- **NIST Standardization:** Selected as the primary KEM in FIPS 203
- **Memory Efficiency:** Smallest variant with 800-byte public keys and 768-byte ciphertexts
- **Security Level:** NIST Level 1 (128-bit classical security)
- **wolfSSL Support:** Native implementation available

Key Sizes:

- Public Key: 800 bytes
- Secret Key: 1,632 bytes
- Ciphertext: 768 bytes
- Shared Secret: 32 bytes

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 Design

4.1 Initialization Sequence

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

Listing 1: Boot sequence

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.

4.3 Memory Layout

Region	Address	Size
ROM	0x00000000	128 KB
SRAM	0x10000000	8 KB
Main RAM	0x40000000	100 MB
ETH MAC	0x80000000	8 KB
CSR	0xF0000000	64 KB
Stack	(top of RAM)	500 KB
Heap	(after BSS)	500 KB

Table 1: Memory regions from linker configuration

5 wolfSSL/wolfCrypt Integration

5.1 Configuration (`user_settings.h`)

Key configuration macros for bare-metal PQC-DTLS operation:

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

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

/* ECC and RSA */
#define HAVE_ECC
#define HAVE_CURVE25519
#define HAVE_ED25519
#define WC_RSA_PSS

/* Crypto primitives */
#define HAVE_AESGCM
#define WOLFSSL_SHA256
#define WOLFSSL_SHA512
#define WOLFSSL_SHA3

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

Listing 2: wolfSSL/wolfCrypt configuration

5.2 Enabled Features

The configuration enables: DTLS 1.3 with fragmentation, ML-KEM post-quantum KEM, ECC (Curve25519, Ed25519), RSA with PSS, 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 Considerations

- **Quantum Resistance:** ML-KEM provides protection against Shor's algorithm attacks on key exchange
- **Forward Secrecy:** Ephemeral ML-KEM keys per session
- **Authenticated Encryption:** AES-GCM provides confidentiality and integrity
- **PRNG Limitation:** Current PRNG is deterministic (demo only); production requires hardware TRNG
- **Certificate Verification:** Disabled for demo; production should enable with Dilithium signatures

8 Performance Metrics

Metric	Value
Memory Layout (from linker map)	
Firmware Size (.text)	55,656 bytes (54 KB)
Read-only Data (.rodata)	4,272 bytes (4 KB)
Initialized Data (.data)	24 bytes
BSS (.bss)	400 bytes
Total Binary (boot.bin)	59,952 bytes (59 KB)
Stack/Heap Allocation	
Stack Size (allocated)	500 KB
Heap Size (allocated)	500 KB
Crypto Operations (estimated)	
ML-KEM-512 KeyGen	∼50 ms
ML-KEM-512 Encaps	∼30 ms
ML-KEM-512 Decaps	∼35 ms

Table 2: Memory and performance on VexRiscv @ 100MHz (simulated)

9 Session Resumption

wolfSSL's DTLS 1.3 implementation supports session resumption via PSK (Pre-Shared Key) mode. Our configuration enables:

- Session ticket storage for abbreviated handshakes
- 0-RTT data support (when enabled)

- Reduced computational overhead on reconnection

Full session resumption requires server-side ticket issuance and client-side caching, which can be enabled via `HAVE_SESSION_TICKET`.

10 Optimizations for Low-Power RISC-V

1. **Small Math Library:** `WOLFSSL_SP_SMALL` reduces code size with minor performance impact
2. **32-bit Word Size:** `SP_WORD_SIZE=32` matches RV32 architecture
3. **Small Tables:** `WOLFSSL_AES_SMALL_TABLES` reduces AES lookup table size
4. **Memory-Optimized ML-KEM:** `MLKEM_SMALL_MEM` variants reduce peak memory
5. **No Dynamic Allocation in SP:** `WOLFSSL_SP_NO_MALLOC` uses stack allocation

11 Entropy Source

The current implementation uses a Linear Congruential Generator (LCG) seeded with a constant (`0xDEADBEEF`) for demonstration purposes. The custom RNG function:

```
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 (e.g., ring oscillator-based) or use LiteX's PRNG peripheral with proper entropy accumulation.

12 Build System

The firmware is built using a custom Makefile integrated with LiteX's build infrastructure:

- **Toolchain:** RISC-V GCC cross-compiler (riscv64-unknown-elf-gcc)
- **C Library:** picolibc with nano-malloc for embedded systems
- **Linker:** Custom linker script with 500KB stack and 500KB heap
- **Libraries:** libliteeth for Ethernet, libbase for UART/console
- **Server:** Companion Linux DTLS server using system wolfSSL

Key compiler flags:

```
-DWOLFSSL_USER_SETTINGS
-DWOLFSSL_SMALL_STACK
-DWOLFSSL_STATIC_MEMORY
```

13 Conclusion

We implemented a full PQC-DTLS 1.3 client on a bare-metal RISC-V platform using LiteX simulation. The system establishes a secure channel with a Linux-based DTLS server using wolfSSL's DTLS 1.3 implementation with ML-KEM-512 post-quantum key exchange. Key achievements include:

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

Repository: `https://github.com/Team94/LP_Constraint_Env_Sim`

References

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

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

[3] Enjoy-Digital, "LiteX: A Migen/MiSoC based SoC builder," `https://github.com/enjoy-digital/litex`

[4] SpinalHDL, "VexRiscv: A FPGA friendly 32-bit RISC-V CPU," `https://github.com/SpinalHDL/VexRiscv`

[5] E. Rescorla, H. Tschofenig, N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3," RFC 9147, 2022.

[6] R. Avanzi et al., "CRYSTALS-Kyber: Algorithm Specifications and Supporting Documentation," NIST PQC Round 3 Submission, 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
|   |   |-- boot.bin                # Compiled binary (output)
|   |   |-- src/                    # wolfSSL TLS source
|   |   |-- wolfssl/                # wolfSSL headers
|   |   |   `-- wolfcrypt/
|   |   |       `-- user_settings.h # wolfSSL configuration
|   |   `-- wolfcrypt/src/          # wolfCrypt source files
|   |-- build/                      # LiteX build output
|   |-- report/                     # Technical report
|   |-- litex/, liteeth/, migen/    # LiteX framework
|   `-- README.md
|
|-- boot/                           # Root-level boot directory
|   |-- main.c                      # Client firmware (copy)
|   `-- server/                     # Linux DTLS server
|       |-- pqc_dtls_server.c       # DTLS 1.3 server
|       |-- Makefile
|       `-- user_settings.h
|
`-- PQC_DTLS_README.md              # Protocol documentation
```

Annexure B: Build Instructions

Prerequisites:

- RISC-V GCC toolchain (riscv64-unknown-elf-gcc)
- LiteX framework with VexRiscv support
- wolfSSL library (v5.6.0 or later with PQC support)
- Python 3.8+ with Migen/LiteX dependencies

Building the Firmware:

```
# Navigate to boot directory
cd LP_Constraint_Env_Sim/boot

# Build the firmware
make clean && make

# Output: boot.bin (load into LiteX simulation)
```

Building the Server:

```
# Navigate to server directory (root level)
cd Team94_L1/boot/server

# Build with wolfSSL
make

# Run the server
./build/pqc_dtls_server
```

Running the Simulation:

```
# Create TAP interface
sudo ip tuntap add tap0 mode tap
sudo ip addr add 192.168.1.100/24 dev tap0
sudo ip link set tap0 up

# Run LiteX simulation
litex_sim --with-ethernet --ethernet-tap tap0
```

Annexure C: Network Configuration

Parameter	Client (LiteX)	Server (Host)
IP Address	192.168.1.50	192.168.1.100
UDP Port	22222	11111
MAC Address	0x10:0xe2:d5:00:00:02	(host default)
Interface	LiteETH	tap0

Table 3: Network configuration for DTLS communication

Annexure D: wolfSSL Configuration Summary

Feature	Configuration Macro
DTLS 1.3	`WOLFSSL_DTLS13`, `WOLFSSL_TLS13`
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`
No Filesystem	`NO_FILESYSTEM`
Custom RNG	`CUSTOM_RAND_GENERATE_SEED`

Table 4: Key wolfSSL/wolfCrypt configuration macros

Annexure E: DTLS 1.3 Handshake Flow

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