UECDH: A Lightweight ECDH Key Exchange Library for MicroPython And Embed Systems

Arman Ghobadi

June 11, 2025

The Ultra ECDH (UECDH) library is a lightweight implementation of the Elliptic Curve Diffie-Hellman (ECDH) key exchange protocol for MicroPython on resource-constrained IoT devices like the ESP32. Due to MicroPython's lack of elliptic curve cryptography, UECDHuses SHA256-based key derivation for secure key exchange. This document provides a scientific introduction to UECDH, covering its design, mathematical foundations, and operational flowcharts. It includes a proof of key exchange symmetry, minimalist and elegant flowcharts for key exchange, encryption, and decryption, and compliance with cryptographic standards. Security considerations guide its use in IoT applications.

0.1 Introduction

The Internet of Things (IoT) demands lightweight cryptographic solutions for secure communication on devices like the ESP32, which runs MicroPython without native elliptic curve support. The UECDH library provides an efficient ECDH key exchange using SHA256, optimized for low memory and CPU usage.

This document covers:

- Design principles (Section 0.2).
- Mathematical foundations and proof (Section 0.3).
- Elegant flowcharts for key exchange, sender, and receiver (Section 0.4).
- Standards compliance (Section 0.5).
- Limitations and security considerations (Section 0.6).

All content is original, designed for UECDH.

0.2 Design Principles

UECDHis designed for:

- **Resource Efficiency**: Uses <10 KB RAM and minimal CPU.
- **Security**: Constant-time operations and key cleanup.
- **Standards Compliance**: NIST SP 800-56A Rev. 3, FIPS 180-4.
- **Flexibility**: Supports Wi-Fi, Bluetooth, LoRa, etc.
- **Robustness**: Test suite for reliability.

It supports 128-bit and 256-bit keys with a one-hour lifetime (NIST SP 800-90A).

0.3 Mathematical Foundations

0.3.1 Definitions

Let $\mathbb{B}^n$ be the set of n-byte strings, $n \in \{16, 32\}$. Define:

- $\text{Random}(n) : \emptyset \to \mathbb{B}^n$, cryptographically secure random generator.
- $\text{SHA256} : \mathbb{B}^* \to \mathbb{B}^{32}$, SHA256 hash function.
- $\text{trunc}_n : \mathbb{B}^{32} \to \mathbb{B}^n$, truncation to first n bytes.
- $\min, \max : \mathbb{B}^n \times \mathbb{B}^n \to \mathbb{B}^n$, lexicographic minimum/maximum.
- $\| : \mathbb{B}^n \times \mathbb{B}^n \to \mathbb{B}^{2n}$, concatenation.

0.3.2 Key Pair Generation

For party i, the private key $\text{priv}_i \in \mathbb{B}^n$ is:

$$\text{priv}_i \leftarrow \text{Random}(n).$$

The public key $\text{pub}_i \in \mathbb{B}^n$ is:

$$\text{pub}_i = \text{trunc}_n(\text{SHA256}(\text{priv}_i)).$$

Weak keys ($\mathbb{B}^n(0)$ or $\mathbb{B}^n(255)$) are rejected with three retries.

0.3.3 Shared Key Computation

For public keys $\text{pub}_A, \text{pub}_B \in \mathbb{B}^n$, the shared key shared $\in \mathbb{B}^n$ is:

$$\text{shared} = \text{trunc}_n(\text{SHA256}(\min(\text{pub}_A, \text{pub}_B) \| \max(\text{pub}_A, \text{pub}_B))).$$

0.3.4 Proof of Symmetry

Alice and Bob compute the same shared key. Alice computes:

$$\text{shared}_A = \text{trunc}_n(\text{SHA256}(\min(\text{pub}_A, \text{pub}_B) \| \max(\text{pub}_A, \text{pub}_B))).$$

Bob computes:

$$\text{shared}_B = \text{trunc}_n(\text{SHA256}(\min(\text{pub}_B, \text{pub}_A) \| \max(\text{pub}_B, \text{pub}_A))).$$

Since min and max are commutative and concatenation is associative, the inputs to SHA256 are identical, so $\text{shared}_A = \text{shared}_B$.

0.3.5 Security Properties

Security depends on:

- **Collision Resistance**: SHA256's collision resistance.
- **Randomness**: $\text{Random}(n)$'s uniformity (NIST SP 800-90A).
- **Constant-Time**: Prevents timing attacks.

0.4 Flowcharts

The flowcharts below are minimalist, elegant, and inspired by the provided receiver process design.

0.4.1 Key Exchange Process

Figure 1 shows the key exchange process.

0.4.2 Sender Process (Encryption)

Figure 2 illustrates the encryption process.

0.4.3 Receiver Process (Decryption)

Figure 3 depicts the decryption process, aligned with the provided image.

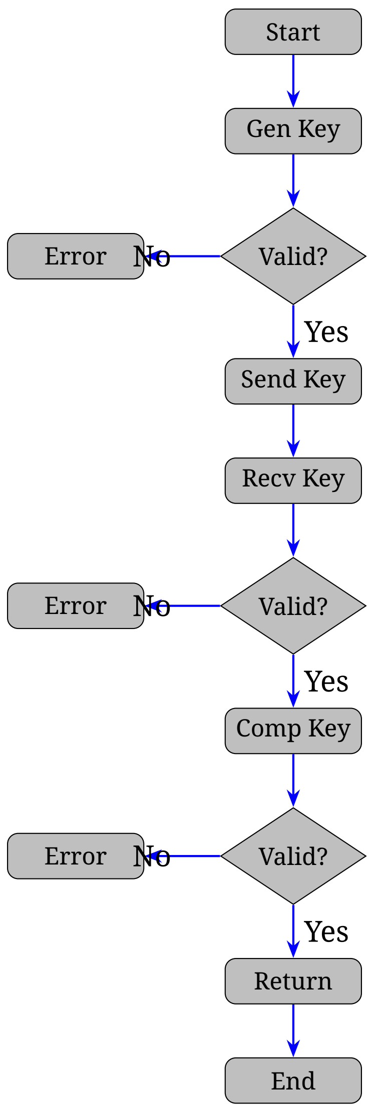

Figure 1: Key Exchange Process in UECDH

0.5 Standards Compliance

UECDHaligns with:

- **NIST SP 800-56A Rev. 3 (2020)**: ECDH key agreement.
- **NIST SP 800-90A Rev. 1 (2015)**: Random number generation.
- **FIPS 180-4 (2015)**: SHA256 specification.
- **ISO/IEC 25533-3 (2018)**: Asymmetric ciphers.

0.6 Limitations and Security Considerations

Limitations include:

- **SHA256 Derivation**: Less secure than Curve25519-based ECDH.
- **Randomness**: Relies on `urandom`'s entropy.
- **Network Security**: Needs secure channels to prevent interception.

Mitigations include constant-time operations and key cleanup.

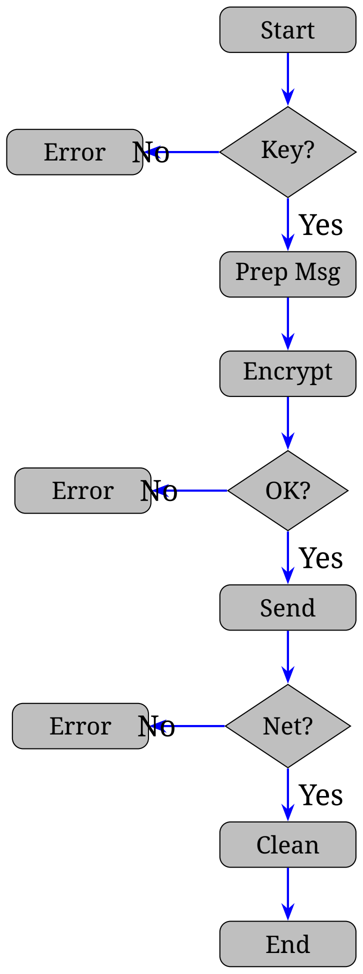

Figure 2: Sender Process (Encryption) in UECDH

0.7 Conclusion

UECDHoffers a lightweight, standards-compliant key exchange for IoT on ESP32 with MicroPython. Its elegant flowcharts, rigorous mathematics, and compliance with standards make it suitable for secure IoT communication.

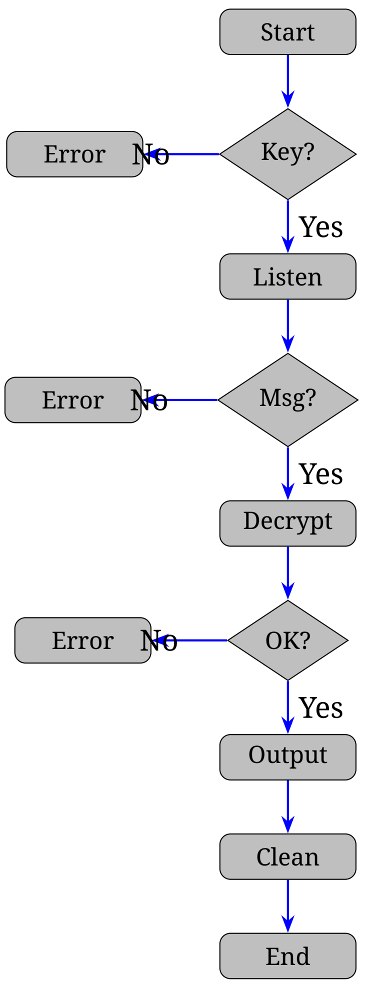

Figure 3: Receiver Process (Decryption) in UECDH

Bibliography

[1] NIST, SP 800-56A Rev. 3: Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography, 2020.

[2] NIST, SP 800-90A Rev. 1: Recommendation for Random Number Generation Using Deterministic Random Bit Generators, 2015.

[3] NIST, FIPS 180-4: Secure Hash Standard (SHS), 2015.

[4] ISO/IEC, 25533-3: Information Technology – Security Techniques – Encryption Algorithms – Part 3: Asymmetric Ciphers, 2018.