

p-pqc

Hướng dẫn sử dụng thư viện đa nền tảng

Swift / iOS | Kotlin / Android | Flutter

WASM / Browser | Go

Thuật toán	ML-KEM-768 (FIPS 203) + AES-256-GCM
Chuẩn	NIST FIPS 203 (ML-KEM) — Level 3
Ngôn ngữ lõi	Rust — Pure Rust, không native lib
Nền tảng	iOS · Android · Flutter · WASM · Go · Windows · macOS
Liên lạc	letanphuoc@gmail.com

Phiên bản 02/11/2026

p-pqc — Hướng dẫn sử dụng thư viện

Giới thiệu

p-pqc là thư viện mã hoá hậu lượng tử (Post-Quantum Cryptography) sẵn sàng nhúng vào ứng dụng. Dev chỉ cần lấy file thư viện đã build và gọi API — không cần cài Rust hay biên dịch bất cứ thứ gì.

Thuật toán	Mô tả
CRYSTALS-Kyber-768	Key Encapsulation Mechanism — NIST FIPS 203 Level 3
AES-256-GCM	Authenticated Encryption (AEAD) — NIST SP 800-38D
Kết hợp	ML-KEM sinh shared secret, AES-256-GCM mã hoá dữ liệu; số ngẫu nhiên từ HMAC_DRBG SHA-512 (SP 800-90A)

Tính chất bảo mật:

- Chống tấn công máy tính lượng tử (Grover, Shor algorithm)
- Xác thực toàn vẹn — phát hiện giả mạo dữ liệu (AEAD tag 16 bytes)
- Forward secrecy — mỗi lần mã hoá tạo shared secret hoàn toàn mới
- Tương thích chéo — ciphertext tạo bởi Swift đọc được bởi Kotlin, Go, Flutter

---

Danh sách file thư viện

Nhận bộ file sau từ team backend / release:

Nền tảng	File cần dùng	Ghi chú
Windows x64	`p_pqc.dll`, `p_pqc.lib`, `p_pqc.h`	Dynamic library
macOS	`libp_pqc.a`, `p_pqc.h`	Static — universal arm64 + x86_64
macOS (dynamic)	`libp_pqc.dylib`, `p_pqc.h`	Dynamic library
iOS	`PPQC.xcframework/`, `p_pqc.h`	XCFramework — device + simulator
Android	`jniLibs/arm64-v8a/libp_pqc.so`	Thiết bị thực tế 64-bit
Android	`jniLibs/armeabi-v7a/libp_pqc.so`	Thiết bị thực tế 32-bit
Android	`jniLibs/x86_64/libp_pqc.so`	Emulator 64-bit

Nền tảng	File cần dùng	Ghi chú
Android	`jniLibs/x86/libp_pqc.so`	Emulator 32-bit
Flutter Android	`jniLibs/*/libp_pqc.so`	C ABI — dart:ffi, không JNI
Flutter iOS	`PPQC.xcframework/`	Giống iOS native
Go	`libp_pqc.a` , `p_pqc.h`	CGo static linking

Header file `p_pqc.h` — API tóm tắt:

```
// Kích thước key (bytes)
#define PPQC_PK_LEN  1184   // Public key
#define PPQC_SK_LEN  2400   // Secret key
#define PPQC_CT_LEN  1088   // KEM ciphertext
#define PPQC_SS_LEN  32     // Shared secret

// Sinh cặp khoá — return 0=OK, -1=lỗi
int ppqc_keygen(uint8_t *pk_out, uint8_t *sk_out);

// Tính kích thước buffer output trước khi gọi ppqc_encrypt
size_t ppqc_encrypt_output_len(size_t plaintext_len);

// Mã hoá — return 0=OK, -1=lỗi
int ppqc_encrypt(const uint8_t *pk,
                 const uint8_t *plaintext, size_t plaintext_len,
                 uint8_t *out, size_t *out_len);

// Giải mã — return 0=OK, -1=lỗi (sai key hoặc bị giả mạo)
int ppqc_decrypt(const uint8_t *sk,
                 const uint8_t *ciphertext, size_t ciphertext_len,
                 uint8_t *out, size_t *out_len);

// KEM raw (tuỳ chọn — tự quản lý symmetric encryption)
int ppqc_kem_encapsulate(const uint8_t *pk,
                         uint8_t *ct_out, uint8_t *ss_out);
int ppqc_kem_decapsulate(const uint8_t *sk,
                         const uint8_t *ct, uint8_t *ss_out);
```

Windows — C / C++

Chuẩn bị

```
MyProject/
├── main.c
├── p_pqc.h       ← copy vào đây
├── p_pqc.lib     ← import library cho MSVC
└── p_pqc.dll     ← copy vào thư mục chạy exe
```

Visual Studio

Project Properties → Linker → Input → Additional Dependencies:

```
p_pqc.lib
```

Project Properties → C/C++ → General → Additional Include Directories:

```
$(ProjectDir)
```

Code mẫu C

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "p_pqc.h"

int main(void) {
    // 1. Sinh cặp khoá
    uint8_t pk[PPQC_PK_LEN];
    uint8_t sk[PPQC_SK_LEN];
    if (ppqc_keygen(pk, sk) ≠ 0) {
        fprintf(stderr, "Keygen thất bại\n");
        return 1;
    }
    printf("Public key: %d bytes\n", PPQC_PK_LEN);

    // 2. Mã hoá
    const char *message = "Xin chào p-pqc!";
    size_t pt_len  = strlen(message);
    size_t buf_len = ppqc_encrypt_output_len(pt_len);
    uint8_t *blob  = malloc(buf_len);

    size_t out_len = buf_len;
    if (ppqc_encrypt(pk, (const uint8_t *)message, pt_len, blob, &out_len) ≠ 0) {
        fprintf(stderr, "Mã hoá thất bại\n");
        free(blob); return 1;
    }
    printf("Ciphertext: %zu bytes\n", out_len);

    // 3. Giải mã
    uint8_t *plain = malloc(out_len);
    size_t   plain_len = out_len;
    if (ppqc_decrypt(sk, blob, out_len, plain, &plain_len) ≠ 0) {
        fprintf(stderr, "Giải mã thất bại\n");
        free(blob); free(plain); return 1;
    }
    printf("Decrypted: %.*s\n", (int)plain_len, plain);

    free(blob); free(plain);
    return 0;
}
```

Windows — .NET / C# (P/Invoke)

```
using System;
using System.Runtime.InteropServices;
using System.Text;

public static class PPQC
{
    private const string Dll = "p_pqc";

    public const int PK_LEN = 1184;
    public const int SK_LEN = 2400;
    public const int CT_LEN = 1088;
    public const int SS_LEN = 32;

    [DllImport(Dll)] static extern int    ppqc_keygen(byte[] pk, byte[] sk);
    [DllImport(Dll)] static extern IntPtr ppqc_encrypt_output_len(IntPtr ptLen);
    [DllImport(Dll)] static extern int    ppqc_encrypt(
        byte[] pk, byte[] pt, IntPtr ptLen, byte[] output, ref IntPtr outLen);
    [DllImport(Dll)] static extern int    ppqc_decrypt(
        byte[] sk, byte[] ct, IntPtr ctLen, byte[] output, ref IntPtr outLen);

    public static (byte[] pk, byte[] sk) GenerateKeyPair()
    {
        var pk = new byte[PK_LEN];
        var sk = new byte[SK_LEN];
        if (ppqc_keygen(pk, sk) ≠ 0)
            throw new Exception("Sinh khoá thất bại");
        return (pk, sk);
    }

    public static byte[] Encrypt(byte[] pk, string plaintext)
    {
        var pt     = Encoding.UTF8.GetBytes(plaintext);
        var bufLen = ppqc_encrypt_output_len((IntPtr)pt.Length).ToInt64();
        var output = new byte[bufLen];
        var outLen = (IntPtr)bufLen;
        if (ppqc_encrypt(pk, pt, (IntPtr)pt.Length, output, ref outLen) ≠ 0)
            throw new Exception("Mã hoá thất bại");
        var result = new byte[outLen.ToInt64()];
        Buffer.BlockCopy(output, 0, result, 0, result.Length);
        return result;
    }

    public static string Decrypt(byte[] sk, byte[] ciphertext)
    {
        var output = new byte[ciphertext.Length + 64];
        var outLen = (IntPtr)output.Length;
        if (ppqc_decrypt(sk, ciphertext, (IntPtr)ciphertext.Length, output, ref
outLen) ≠ 0)
            throw new Exception("Giải mã thất bại — sai khoá hoặc dữ liệu bị giả
mạo");
        return Encoding.UTF8.GetString(output, 0, outLen.ToInt32());
    }
}

// Sử dụng
class Program
```

```
{
    static void Main()
    {
        var (pk, sk) = PPQC.GenerateKeyPair();
        Console.WriteLine($"PK: {pk.Length} bytes, SK: {sk.Length} bytes");

        byte[] ct  = PPQC.Encrypt(pk, "Xin chào từ C#!");
        string msg = PPQC.Decrypt(sk, ct);
        Console.WriteLine($"Decrypted: {msg}");
    }
}
```

macOS — Swift

Chuẩn bị

```
MyMacApp/
├── MyMacApp.xcodeproj
├── MyMacApp/
│   ├── p_pqc.h       ← copy vào đây
│   ├── PPQC.swift    ← file wrapper
│   └── MyApp-Bridging-Header.h
└── Libs/
    └── libp_pqc.a    ← copy vào đây
```

Xcode — Cấu hình

Build Settings → Other Linker Flags:

```
-L$(PROJECT_DIR)/Libs  -lp_pqc
```

Build Settings → Header Search Paths:

```
$(PROJECT_DIR)/MyMacApp
```

Build Settings → Objective-C Bridging Header:

```
MyMacApp/MyApp-Bridging-Header.h
```

Nội dung `MyApp-Bridging-Header.h`:

```
#include "p_pqc.h"
```

macOS — Code Swift

```
import Foundation

// ── Sinh khoá ──────────────────────────────────────────────────────
var pk = Data(count: Int(ppqc_pk_len()))
var sk = Data(count: Int(ppqc_sk_len()))
```

```
let ret = pk.withUnsafeMutableBytes { pkPtr in
    sk.withUnsafeMutableBytes { skPtr in
        ppqc_keygen(
            pkPtr.bindMemory(to: UInt8.self).baseAddress,
            skPtr.bindMemory(to: UInt8.self).baseAddress
        )
    }
}
guard ret == 0 else { fatalError("Sinh khoá thất bại") }
print("PK: \(pk.count) bytes | SK: \(sk.count) bytes")

// ── Mã hoá ──────────────────────────────────────────────────────
let message  = "Tin nhắn bí mật từ macOS"
let ptData   = message.data(using: .utf8)!
let capacity = Int(ppqc_encrypt_output_len(ptData.count))
var output   = Data(count: capacity)
var outLen   = capacity

let encRet = pk.withUnsafeBytes { pkPtr in
    ptData.withUnsafeBytes { ptPtr in
        output.withUnsafeMutableBytes { outPtr in
            ppqc_encrypt(
                pkPtr.bindMemory(to: UInt8.self).baseAddress,
                ptPtr.bindMemory(to: UInt8.self).baseAddress,
                ptData.count,
                outPtr.bindMemory(to: UInt8.self).baseAddress,
                &outLen
            )
        }
    }
}
guard encRet == 0 else { fatalError("Mã hoá thất bại") }
let blob = output.prefix(outLen)
print("Ciphertext (base64): \(blob.base64EncodedString().prefix(60))...")

// ── Giải mã ─────────────────────────────────────────────────────
var plain    = Data(count: blob.count + 64)
var plainLen = plain.count

let decRet = sk.withUnsafeBytes { skPtr in
    blob.withUnsafeBytes { ctPtr in
        plain.withUnsafeMutableBytes { outPtr in
            ppqc_decrypt(
                skPtr.bindMemory(to: UInt8.self).baseAddress,
                ctPtr.bindMemory(to: UInt8.self).baseAddress,
                blob.count,
                outPtr.bindMemory(to: UInt8.self).baseAddress,
                &plainLen
            )
        }
    }
}
guard decRet == 0 else { fatalError("Giải mã thất bại") }
print("Decrypted: \(String(data: plain.prefix(plainLen), encoding: .utf8)!)")
```

macOS — Go (CGo)

```
myapp/
├── go.mod
├── main.go
└── ppqc/
    ├── pqc.go          ← CGo wrapper
    ├── p_pqc.h     ← copy vào đây
    └── libp_pqc.a  ← copy vào đây
```

Nội dung `ppqc/pqc.go` — phần CGo directive:

```
/*
#cgo CFLAGS: -I.
#cgo darwin LDFLAGS: ${SRCDIR}/libp_pqc.a -ldl -lpthread \
    -framework Security -framework Foundation
#include "p_pqc.h"
#include <stdlib.h>
*/
import "C"
```

Sử dụng trong `main.go`:

```
package main

import (
    "encoding/base64"
    "fmt"
    "log"
    "myapp/ppqc"
)

func main() {
    kp, err := ppqc.GenerateKeyPair()
    if err ≠ nil { log.Fatal(err) }
    fmt.Printf("PK: %d bytes  SK: %d bytes\n",
        len(kp.PublicKey), len(kp.SecretKey))

    ct, err := ppqc.EncryptString(kp.PublicKey, "Xin chào từ Go!")
    if err ≠ nil { log.Fatal(err) }
    fmt.Println("Ciphertext:", ct[:60]+"...")

    msg, err := ppqc.DecryptString(kp.SecretKey, ct)
    if err ≠ nil { log.Fatal(err) }
    fmt.Println("Decrypted:", msg)
    _ = base64.StdEncoding // unused import guard
}
```

```
go build && ./myapp
```

Android — Kotlin (JNI)

Chuẩn bị

```
app/src/main/
├── java/com/yourapp/
│   └── PPQC.kt       ← file wrapper
└── jniLibs/              ← copy toàn bộ từ bộ thư viện
    ├── arm64-v8a/
    │   └── libp_pqc.so
    ├── armeabi-v7a/
    │   └── libp_pqc.so
    ├── x86_64/
    │   └── libp_pqc.so
    └── x86/
        └── libp_pqc.so
```

> Android Studio tự chọn đúng ABI khi cài lên thiết bị — không cần cấu hình thêm.

PPQC.kt — Wrapper

```
package com.yourapp

import android.util.Base64

object PPQC {

    init { System.loadLibrary("p_pqc") }

    const val PK_LEN = 1184
    const val SK_LEN = 2400
    const val CT_LEN = 1088
    const val SS_LEN = 32

    data class KeyPair(val publicKey: ByteArray, val secretKey: ByteArray) {
        val publicKeyBase64 get() = Base64.encodeToString(publicKey, Base64.NO_WRAP)
        val secretKeyBase64 get() = Base64.encodeToString(secretKey, Base64.NO_WRAP)
    }

    class PQCException(msg: String) : Exception(msg)

    // ── Sinh khoá ──────────────────────────────────────────────
    fun generateKeyPair(): KeyPair {
        val raw = nativeKeygen()
        return KeyPair(
            publicKey = raw.copyOfRange(0, PK_LEN),
            secretKey = raw.copyOfRange(PK_LEN, raw.size)
        )
    }

    // ── Mã hoá ─────────────────────────────────────────────────
    fun encryptBytes(plaintext: ByteArray, recipientPublicKey: ByteArray): ByteArray {
        require(recipientPublicKey.size == PK_LEN) { "PK phải là $PK_LEN bytes" }
        return nativeEncryptBytes(recipientPublicKey, plaintext) ?: throw
PQCException("Mã hoá thất bại")
```

```
    }

    fun encrypt(plaintext: String, recipientPublicKey: ByteArray): String =
        Base64.encodeToString(encryptBytes(plaintext.toByteArray(Charsets.UTF_8),
recipientPublicKey), Base64.NO_WRAP)

    // ── Giải mã ──────────────────────────────────────────────────────────
    fun decryptBytes(blob: ByteArray, secretKey: ByteArray): ByteArray {
        require(secretKey.size == SK_LEN) { "SK phải là $SK_LEN bytes" }
        return nativeDecryptBytes(secretKey, blob) ?: throw PQCException("Giải mã thất
bại")
    }

    fun decrypt(ciphertextBase64: String, secretKey: ByteArray): String =
        String(decryptBytes(Base64.decode(ciphertextBase64, Base64.NO_WRAP),
secretKey), Charsets.UTF_8)

    // ── KEM ──────────────────────────────────────────────────────────────
    fun kemEncapsulate(publicKey: ByteArray): Pair<ByteArray, ByteArray> {
        val raw = nativeKemEncapsulate(publicKey)
        return raw.copyOfRange(0, CT_LEN) to raw.copyOfRange(CT_LEN, raw.size)
    }

    fun kemDecapsulate(secretKey: ByteArray, ct: ByteArray): ByteArray =
        nativeKemDecapsulate(secretKey, ct)

    // ── Phục hồi khoá từ base64 ──────────────────────────────────────────
    fun keyPairFromBase64(pkB64: String, skB64: String) = KeyPair(
        publicKey = Base64.decode(pkB64, Base64.NO_WRAP),
        secretKey = Base64.decode(skB64, Base64.NO_WRAP)
    )

    private external fun nativeKeygen(): ByteArray
    private external fun nativeEncryptBytes(pk: ByteArray, plaintext: ByteArray):
ByteArray?
    private external fun nativeDecryptBytes(sk: ByteArray, blob: ByteArray):
ByteArray?
    private external fun nativeKemEncapsulate(pk: ByteArray): ByteArray
    private external fun nativeKemDecapsulate(sk: ByteArray, ct: ByteArray): ByteArray
}
```

Sử dụng trong Activity / ViewModel

```
import com.yourapp.PPQC

// Sinh khoá (Bob — người nhận)
val kp = PPQC.generateKeyPair()

// Chia sẻ publicKey lên server cho Alice
uploadToServer(kp.publicKeyBase64)

// Alice mã hoá gửi Bob
val bobPK   = Base64.decode(getPKFromServer(), Base64.NO_WRAP)
val cipher  = PPQC.encrypt("Nội dung bí mật", bobPK)
sendToServer(cipher)

// Bob giải mã
```

```
val message = PPQC.decrypt(receivedCipher, kp.secretKey)
println("Bob nhận: $message")
```

Lưu khoá an toàn — Android Keystore

```
import android.security.keystore.KeyGenParameterSpec
import android.security.keystore.KeyProperties
import javax.crypto.Cipher
import javax.crypto.KeyGenerator
import javax.crypto.spec.GCMParameterSpec

// Wrap secret key bằng AES-GCM trong Android Keystore
fun saveSecretKey(context: Context, secretKey: ByteArray) {
    val kg = KeyGenerator.getInstance(KeyProperties.KEY_ALGORITHM_AES,
"AndroidKeyStore")
    kg.init(KeyGenParameterSpec.Builder("p_pqc_key",
        KeyProperties.PURPOSE_ENCRYPT or KeyProperties.PURPOSE_DECRYPT)
        .setBlockModes(KeyProperties.BLOCK_MODE_GCM)
        .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_NONE)
        .build())
    val wrapKey = kg.generateKey()

    val cipher = Cipher.getInstance("AES/GCM/NoPadding")
    cipher.init(Cipher.ENCRYPT_MODE, wrapKey)
    val encrypted = cipher.doFinal(secretKey)

    context.getSharedPreferences("pqc", 0).edit()
        .putString("sk", Base64.encodeToString(encrypted, Base64.NO_WRAP))
        .putString("iv", Base64.encodeToString(cipher.iv,  Base64.NO_WRAP))
        .apply()
}
```

---

iOS — Swift Native

Chuẩn bị

```
MyiOSApp.xcodeproj
├── MyiOSApp/
│   ├── MyApp-Bridging-Header.h
│   ├── PPQC.swift       ← file wrapper
│   └── ...
└── Frameworks/
    └── PPQC.xcframework ← kéo vào đây
```

Xcode — Cấu hình

1. Kéo `PPQC.xcframework` vào thư mục **Frameworks** trong Xcode navigator
2. **Target → General → Frameworks, Libraries → Add → PPQC.xcframework**
3. Chọn **"Do Not Embed"** (static library)
4. **Build Settings → Objective-C Bridging Header:** `MyiOSApp/MyApp-Bridging-Header.h`

Nội dung `MyApp-Bridging-Header.h`:

```
#include <p_pqc.h>
```

iOS — Wrapper Swift

```
import Foundation

public enum PPQC {

    public static let pkLen = Int(ppqc_pk_len())
    public static let skLen = Int(ppqc_sk_len())

    // ── Sinh khoá ──────────────────────────────────────────────────────
    public static func generateKeyPair() throws → (pk: Data, sk: Data) {
        var pk = Data(count: pkLen)
        var sk = Data(count: skLen)
        let r = pk.withUnsafeMutableBytes { p in
            sk.withUnsafeMutableBytes { s in
                ppqc_keygen(p.baseAddress?.assumingMemoryBound(to: UInt8.self),
                            s.baseAddress?.assumingMemoryBound(to: UInt8.self))
            }
        }
        guard r == 0 else { throw PpqcError.keygenFailed }
        return (pk, sk)
    }

    // ── Mã hoá → base64 ────────────────────────────────────────────────
    public static func encrypt(_ text: String, publicKey pk: Data) throws → String {
        guard let pt = text.data(using: .utf8) else {
            throw PpqcError.encryptFailed
        }
        let cap   = Int(ppqc_encrypt_output_len(pt.count))
        var out   = Data(count: cap)
        var outLen = cap
        let r = pk.withUnsafeBytes { pkPtr in
            pt.withUnsafeBytes { ptPtr in
                out.withUnsafeMutableBytes { outPtr in
                    ppqc_encrypt(
                        pkPtr.baseAddress?.assumingMemoryBound(to: UInt8.self),
                        ptPtr.baseAddress?.assumingMemoryBound(to: UInt8.self),
                        pt.count,
                        outPtr.baseAddress?.assumingMemoryBound(to: UInt8.self),
                        &outLen)
                }
            }
        }
        guard r == 0 else { throw PpqcError.encryptFailed }
        return out.prefix(outLen).base64EncodedString()
    }

    // ── Giải mã ────────────────────────────────────────────────────────
    public static func decrypt(_ b64: String, secretKey sk: Data) throws → String {
        guard let blob = Data(base64Encoded: b64) else {
            throw PpqcError.invalidInput
        }
```

```
        var out    = Data(count: blob.count + 64)
        var outLen = out.count
        let r = sk.withUnsafeBytes { skPtr in
            blob.withUnsafeBytes { ctPtr in
                out.withUnsafeMutableBytes { outPtr in
                    ppqc_decrypt(
                        skPtr.baseAddress?.assumingMemoryBound(to: UInt8.self),
                        ctPtr.baseAddress?.assumingMemoryBound(to: UInt8.self),
                        blob.count,
                        outPtr.baseAddress?.assumingMemoryBound(to: UInt8.self),
                        &outLen)
                }
            }
        }
        guard r == 0 else { throw PpqcError.decryptFailed }
        return String(data: out.prefix(outLen), encoding: .utf8) ?? ""
    }

    public enum PpqcError: Error {
        case keygenFailed, encryptFailed, decryptFailed, invalidInput
    }
}
```

Sử dụng trong ViewController / SwiftUI

```
// Sinh khoá
let (pk, sk) = try PPQC.generateKeyPair()
print("PK: \(pk.count)B  SK: \(sk.count)B")

// Mã hoá
let cipher = try PPQC.encrypt("Xin chào iOS!", publicKey: pk)

// Giải mã
let plain  = try PPQC.decrypt(cipher, secretKey: sk)
print(plain)  // → Xin chào iOS!
```

Lưu khoá — iOS Keychain

```
import Security

func saveToKeychain(_ data: Data, key: String) {
    let q: [CFString: Any] = [
        kSecClass:       kSecClassGenericPassword,
        kSecAttrAccount: key,
        kSecValueData:   data,
        kSecAttrAccessible: kSecAttrAccessibleWhenUnlockedThisDeviceOnly
    ]
    SecItemDelete(q as CFDictionary)
    SecItemAdd(q as CFDictionary, nil)
}

func loadFromKeychain(key: String) -> Data? {
    let q: [CFString: Any] = [
        kSecClass:       kSecClassGenericPassword,
        kSecAttrAccount: key,
        kSecReturnData:  true,
```

```
        kSecMatchLimit:  kSecMatchLimitOne
    ]
    var result: AnyObject?
    SecItemCopyMatching(q as CFDictionary, &result)
    return result as? Data
}

// Lưu
saveToKeychain(sk, key: "ppqc.pqc.secretkey")

// Load
let secretKey = loadFromKeychain(key: "ppqc.pqc.secretkey")!
```

Flutter — dart:ffi (iOS + Android)

Flutter sử dụng `dart:ffi` gọi trực tiếp C API — không qua JNI.

Chuẩn bị

iOS: Copy `PPQC.xcframework` vào `ios/Frameworks/`, mở `ios/Runner.xcworkspace`:
- Kéo framework vào **Frameworks group**
- **Build Phases → Link Binary → Add → PPQC.xcframework**
- Embed: **Do Not Embed**

Android: Copy toàn bộ `jniLibs/` vào `android/app/src/main/jniLibs/`

pubspec.yaml:

```
dependencies:
  ffi: ^2.1.0
```

```
flutter pub get
```

p_pqc.dart — Wrapper dart:ffi

```
import 'dart:ffi';
import 'dart:io';
import 'dart:convert';
import 'dart:typed_data';

// ── Load thư viện theo platform ──────────────────────────────
final _lib = () {
  if (Platform.isAndroid) return DynamicLibrary.open('libp_pqc.so');
  if (Platform.isIOS)     return DynamicLibrary.process();
  if (Platform.isMacOS)   return DynamicLibrary.open('libp_pqc.dylib');
  if (Platform.isWindows) return DynamicLibrary.open('p_pqc.dll');
  throw UnsupportedError(Platform.operatingSystem);
}();

// ── FFI bindings ─────────────────────────────────────────────
final _keygen  = _lib.lookupFunction<
    Int32 Function(Pointer<Uint8>, Pointer<Uint8>),
```

```
    int   Function(Pointer<Uint8>, Pointer<Uint8>)>('ppqc_keygen');

final _encLen  = _lib.lookupFunction<
    Size Function(Size), int Function(int)>('ppqc_encrypt_output_len');

final _encrypt = _lib.lookupFunction<
    Int32 Function(Pointer<Uint8>, Pointer<Uint8>, Size, Pointer<Uint8>,
Pointer<Size>),
    int   Function(Pointer<Uint8>, Pointer<Uint8>, int,  Pointer<Uint8>,
Pointer<Size>)
>('ppqc_encrypt');

final _decrypt = _lib.lookupFunction<
    Int32 Function(Pointer<Uint8>, Pointer<Uint8>, Size, Pointer<Uint8>,
Pointer<Size>),
    int   Function(Pointer<Uint8>, Pointer<Uint8>, int,  Pointer<Uint8>,
Pointer<Size>)
>('ppqc_decrypt');

// ── Constants ──────────────────────────────────────────────────────────
const pkLen = 1184;
const skLen = 2400;

// ── PPQC API ───────────────────────────────────────────────────────────
class PPQC {

  /// Sinh cặp khoá Kyber-768
  static ({Uint8List pk, Uint8List sk}) generateKeyPair() {
    final pkPtr = calloc<Uint8>(pkLen);
    final skPtr = calloc<Uint8>(skLen);
    try {
      if (_keygen(pkPtr, skPtr) != 0) throw Exception('Keygen thất bại');
      return (
        pk: Uint8List.fromList(pkPtr.asTypedList(pkLen)),
        sk: Uint8List.fromList(skPtr.asTypedList(skLen)),
      );
    } finally {
      calloc.free(pkPtr); calloc.free(skPtr);
    }
  }

  /// Mã hoá — trả về base64 blob
  static String encrypt(String plaintext, Uint8List publicKey) {
    final pt    = utf8.encode(plaintext);
    final cap   = _encLen(pt.length);
    final pkPtr = calloc<Uint8>(pkLen);
    final ptPtr = calloc<Uint8>(pt.length);
    final outPtr= calloc<Uint8>(cap);
    final lenPtr= calloc<Size>();
    try {
      pkPtr.asTypedList(pkLen).setAll(0, publicKey);
      ptPtr.asTypedList(pt.length).setAll(0, pt);
      lenPtr.value = cap;
      if (_encrypt(pkPtr, ptPtr, pt.length, outPtr, lenPtr) != 0)
        throw Exception('Mã hoá thất bại');
      return base64.encode(outPtr.asTypedList(lenPtr.value));
    } finally {
      calloc.free(pkPtr); calloc.free(ptPtr);
      calloc.free(outPtr); calloc.free(lenPtr);
```

```
    }
  }

  /// Giải mã — nhận base64 blob, trả về string
  static String decrypt(String ciphertextB64, Uint8List secretKey) {
    final ct    = base64.decode(ciphertextB64);
    final skPtr = calloc<Uint8>(skLen);
    final ctPtr = calloc<Uint8>(ct.length);
    final outPtr= calloc<Uint8>(ct.length + 64);
    final lenPtr= calloc<Size>();
    try {
      skPtr.asTypedList(skLen).setAll(0, secretKey);
      ctPtr.asTypedList(ct.length).setAll(0, ct);
      lenPtr.value = ct.length + 64;
      if (_decrypt(skPtr, ctPtr, ct.length, outPtr, lenPtr) != 0)
        throw Exception('Giải mã thất bại — sai khoá hoặc dữ liệu bị thay đổi');
      return utf8.decode(outPtr.asTypedList(lenPtr.value));
    } finally {
      calloc.free(skPtr); calloc.free(ctPtr);
      calloc.free(outPtr); calloc.free(lenPtr);
    }
  }
}
```

Sử dụng trong Widget

```
final kp    = PPQC.generateKeyPair();
final ct    = PPQC.encrypt('Xin chào Flutter!', kp.pk);
final plain = PPQC.decrypt(ct, kp.sk);
print(plain);  // → Xin chào Flutter!
```

React Native — JSI / NativeModule

Cho React Native, tạo Native Module gọi C API:

Android — `PpqcModule.kt`:

```
class PpqcModule(reactContext: ReactApplicationContext)
    : ReactContextBaseJavaModule(reactContext) {

    override fun getName() = "PPQC"

    @ReactMethod
    fun generateKeyPair(promise: Promise) {
        val kp = PPQC.generateKeyPair()
        promise.resolve(Arguments.makeNativeMap(mapOf(
            "pk" to kp.publicKeyBase64,
            "sk" to kp.secretKeyBase64
        )))
    }

    @ReactMethod
    fun encrypt(plaintext: String, pkB64: String, promise: Promise) {
        val pk = Base64.decode(pkB64, Base64.NO_WRAP)
        promise.resolve(PPQC.encrypt(plaintext, pk))
    }
```

```
    @ReactMethod
    fun decrypt(ciphertext: String, skB64: String, promise: Promise) {
        val sk = Base64.decode(skB64, Base64.NO_WRAP)
        promise.resolve(PPQC.decrypt(ciphertext, sk))
    }
}
```

JavaScript / TypeScript:

```
import { NativeModules } from 'react-native';
const { PPQC } = NativeModules;

const { pk, sk } = await PPQC.generateKeyPair();
const cipher      = await PPQC.encrypt('Hello PQC!', pk);
const plain       = await PPQC.decrypt(cipher, sk);
console.log(plain);
```

WASM — Trình duyệt / JavaScript / TypeScript

p-pqc chạy trực tiếp trong trình duyệt qua WebAssembly — không cần backend, không cần cài thêm gì.

Build thư viện WASM

```
# Cài wasm-pack (một lần)
brew install wasm-pack        # macOS
# hoặc: cargo install wasm-pack

# Build → dist/web/
./build.sh wasm
```

Output trong `dist/web/`:

```
dist/web/
├── p_pqc.js          ← ES module loader + glue code
├── p_pqc_bg.wasm     ← binary (~80 KB)
├── p_pqc.d.ts        ← TypeScript type definitions
├── p_pqc_bg.wasm.d.ts
└── example.html          ← demo chạy được
```

> **Quan trọng:** Phải serve qua HTTP (không mở file:// trực tiếp). Lý do: trình duyệt chặn `fetch()` với `file://` protocol khi load WASM.

```
# Chạy demo nhanh
cd dist/web && python3 -m http.server 8080
# Mở: http://localhost:8080/example.html
```

API WASM — Tổng quan

```
import init, {
  keygen,          // sinh cặp khoá Kyber-768
  encrypt,         // mã hoá bằng public key → wire-format blob
  decrypt,         // giải mã bằng secret key
  kem_encapsulate, // KEM: sinh shared secret từ public key
  kem_decapsulate, // KEM: khôi phục shared secret
  pk_len,          // 1184
  sk_len,          // 2400
  ct_len,          // 1088
  ss_len,          // 32
} from "./p_pqc.js";

await init(); // BẮT BUỘC gọi một lần trước khi dùng bất kỳ hàm nào
```

Kích thước và kiểu dữ liệu

Hàm / Thuộc tính	Kiểu trả về	Kích thước
`keygen().publicKey`	`Uint8Array`	1184 bytes
`keygen().secretKey`	`Uint8Array`	2400 bytes
`encrypt(pk, pt)`	`Uint8Array`	pt.length + 1121 bytes
`decrypt(sk, blob)`	`Uint8Array`	plaintext gốc
`kem_encapsulate(pk).ciphertext`	`Uint8Array`	1088 bytes
`kem_encapsulate(pk).sharedSecret`	`Uint8Array`	32 bytes
`kem_decapsulate(sk, ct)`	`Uint8Array`	32 bytes

Ví dụ — HTML thuần (ES Module)

```
<!DOCTYPE html>
<html>
<head><meta charset="utf-8"><title>p-pqc demo</title></head>
<body>
<script type="module">
import init, { keygen, encrypt, decrypt } from "./p_pqc.js";

await init();  // load WASM

const enc = new TextEncoder();
const dec = new TextDecoder();

// 1. Sinh cặp khoá
const kp = keygen();
console.log("PK:", kp.publicKey.length, "bytes");
console.log("SK:", kp.secretKey.length, "bytes");

// 2. Mã hoá
const plaintext = enc.encode("Xin chào từ p-pqc WASM!");
```

```
const blob = encrypt(kp.publicKey, plaintext);
console.log("Ciphertext:", blob.length, "bytes");

// 3. Giải mã
const recovered = decrypt(kp.secretKey, blob);
console.log("Decrypted:", dec.decode(recovered));
</script>
</body>
</html>
```

Ví dụ — React / Vite / Webpack

Cài package:

```
npm install ./dist/web   # hoặc copy dist/web vào src/lib/p-pqc/
```

React component:

```
import { useEffect, useState } from "react";

// Lazy-load WASM module
let _ppqc: typeof import("p-pqc") | null = null;
async function loadPpqc() {
  if (_ppqc) return _ppqc;
  const mod = await import("p-pqc");
  await mod.default();   // init()
  _ppqc = mod;
  return mod;
}

export function usePQC() {
  const [ready, setReady] = useState(false);

  useEffect(() => {
    loadPpqc().then(() => setReady(true));
  }, []);

  const generateKeyPair = async () => {
    const mod = await loadPpqc();
    const kp  = mod.keygen();
    return {
      publicKey:  kp.publicKey,   // Uint8Array 1184 bytes
      secretKey:  kp.secretKey,   // Uint8Array 2400 bytes
    };
  };

  const encryptMsg = async (message: string, publicKey: Uint8Array) => {
    const mod = await loadPpqc();
    const pt  = new TextEncoder().encode(message);
    return mod.encrypt(publicKey, pt);  // Uint8Array blob
  };

  const decryptMsg = async (blob: Uint8Array, secretKey: Uint8Array) => {
    const mod      = await loadPpqc();
    const recovered = mod.decrypt(secretKey, blob);
    return new TextDecoder().decode(recovered);
  };
```

```
  return { ready, generateKeyPair, encryptMsg, decryptMsg };
}
```

Sử dụng trong component:

```
function EncryptDemo() {
  const { ready, generateKeyPair, encryptMsg, decryptMsg } = usePQC();

  const handleDemo = async () => {
    const kp      = await generateKeyPair();
    const blob    = await encryptMsg("Tin nhắn bí mật", kp.publicKey);
    const message = await decryptMsg(blob, kp.secretKey);
    console.log("Roundtrip OK:", message);
  };

  return <button disabled={!ready} onClick={handleDemo}>Thử PQC</button>;
}
```

Ví dụ — KEM thuần (tự quản lý symmetric key)

```
import init, { keygen, kem_encapsulate, kem_decapsulate } from "./p_pqc.js";

await init();

// Alice sinh khoá, chia sẻ publicKey lên server
const alice = keygen();

// Bob lấy publicKey của Alice, sinh shared secret
const enc = kem_encapsulate(alice.publicKey);
//   enc.ciphertext    → 1088 bytes (gửi cho Alice)
//   enc.sharedSecret  → 32 bytes   (Bob giữ bí mật — dùng làm AES/ChaCha key)

// Alice khôi phục shared secret
const ss = kem_decapsulate(alice.secretKey, enc.ciphertext);
//   ss → 32 bytes, giống hệt enc.sharedSecret

const match = enc.sharedSecret.every((b, i) => b === ss[i]);
console.log("Shared secret khớp:", match);  // → true

// Cả hai bây giờ có cùng 32-byte secret để dùng với AES-GCM hoặc ChaCha20
```

Vite — cấu hình tối ưu

vite.config.ts:

```
import { defineConfig } from "vite";

export default defineConfig({
  optimizeDeps: {
    exclude: ["p-pqc"],  // WASM module không transpile
  },
  server: {
    headers: {
      // SharedArrayBuffer cần COOP + COEP (cho Atomics nếu dùng Worker)
```

```
      "Cross-Origin-Opener-Policy":   "same-origin",
      "Cross-Origin-Embedder-Policy": "require-corp",
    },
  },
});
```

Next.js — Dynamic Import

```
// lib/ppqc.ts — chỉ chạy client-side
export async function getPpqc() {
  if (typeof window === "undefined") {
    throw new Error("p-pqc chỉ chạy ở client side");
  }
  const mod = await import("p-pqc");
  await mod.default();  // init WASM
  return mod;
}

// Trong component
const { keygen, encrypt, decrypt } = await getPpqc();
```

Node.js

```
import { readFile } from "fs/promises";
import { fileURLToPath } from "url";
import { dirname, join } from "path";

// Node 18+ hỗ trợ WASM natively
const __dir   = dirname(fileURLToPath(import.meta.url));
const wasmBuf = await readFile(join(__dir, "p_pqc_bg.wasm"));

import initSync, { keygen, encrypt, decrypt } from "./p_pqc.js";
await initSync({ module: new WebAssembly.Module(wasmBuf) });

const kp      = keygen();
const pt      = Buffer.from("Hello from Node.js");
const blob    = encrypt(kp.publicKey, pt);
const plain   = decrypt(kp.secretKey, blob);
console.log(Buffer.from(plain).toString("utf8"));
```

Lưu khoá — Browser

```
// Lưu vào IndexedDB (không dùng localStorage — không đủ bảo mật)
async function saveKeyPair(pk, sk) {
  const db = await openDB("ppqc-keys", 1, {
    upgrade(db) { db.createObjectStore("keys"); }
  });
  // pk có thể lưu tự do; sk nên mã hoá thêm bằng Web Crypto API
  await db.put("keys", pk, "publicKey");

  // Mã hoá sk bằng Web Crypto trước khi lưu
  const wrappingKey = await crypto.subtle.generateKey(
    { name: "AES-GCM", length: 256 }, false, ["wrapKey", "unwrapKey"]
  );
```

```
  // ... (xem Web Crypto API docs để wrap/unwrap key)
  await db.put("keys", sk, "secretKey_encrypted");
}
```

Hiệu năng WASM (đo trên Chrome M3 MacBook)

Thao tác	Thời gian	Ghi chú
Load WASM (`init()`)	~5 ms	Chỉ gọi một lần
`keygen()`	~1.2 ms	Sinh Kyber-768 keypair
`encrypt()` (1KB)	~1.5 ms	KEM + AES-256-GCM
`decrypt()` (1KB)	~1.0 ms	Decap + AEAD verify
`kem_encapsulate()`	~1.0 ms	KEM only
`kem_decapsulate()`	~0.8 ms	KEM decap only

> WASM chậm hơn native ~2–3× nhưng vẫn đủ nhanh cho mọi ứng dụng thực tế.

Wire Format — Tương thích chéo nền tảng

Ciphertext tạo bởi **bất kỳ nền tảng nào** đều giải mã được bởi nền tảng khác.

```
Offset    Kích thước    Nội dung
──────    ──────────    ──────────────────────────────────────────
0         1 byte        Version: 0x06 (0x05, 0x04 cũ — chỉ đọc)
1-4       4 bytes       Độ dài KEM ciphertext (little-endian) = 1088
5-1092    1088 bytes    ML-KEM-768 KEM ciphertext
1093-1104 12 bytes      Nonce ngẫu nhiên 96 bit (HMAC_DRBG)
1105-N    N bytes       Ciphertext (dữ liệu mã hoá)
N+1-N+16  16 bytes      Authentication tag (GCM; header là AAD)
```

Overhead cố định: 1121 bytes cho mọi message.

Kích thước plaintext	Kích thước blob	Base64
10 bytes	1131 bytes	1508 ký tự
100 bytes	1221 bytes	1628 ký tự
1000 bytes	2121 bytes	2828 ký tự
10000 bytes	11121 bytes	14828 ký tự

Kích thước key và hiệu năng

Thông số	Giá trị	Ghi chú
Public key	1184 bytes	Chia sẻ tự do
Secret key	2400 bytes	Phải lưu bảo mật
KEM ciphertext	1088 bytes	Trong wire format
Shared secret	32 bytes	Dùng làm AES/ChaCha key
Keygen	~0.5 ms	ARM64
Encrypt	~0.8 ms	ARM64, 1KB payload
Decrypt	~0.6 ms	ARM64, 1KB payload

Bảo mật — Lưu trữ khoá đúng cách

Public key — có thể lưu tự do:

```
SharedPreferences / UserDefaults / Database / Server
```

Secret key — bắt buộc dùng secure storage:

Nền tảng	Nơi lưu bắt buộc
iOS	Keychain — `kSecAttrAccessibleWhenUnlockedThisDeviceOnly`
Android	Android Keystore — wrap bằng AES-GCM
Flutter	flutter_secure_storage package
Windows	Windows DPAPI hoặc Windows Credential Manager
macOS	Keychain Services
Go / Server	Vault, AWS KMS, Azure Key Vault, hoặc HSM

Tuyệt đối không:

```
✗  Lưu secret key vào SharedPreferences / UserDefaults dạng plain text
✗  Log secret key ra console
✗  Gửi secret key qua mạng
✗  Nhúng secret key cứng trong source code
```

Tăng cường bảo mật (Security Hardening)

Bản cập nhật **2026-10-15** sửa hai phát hiện của báo cáo đánh giá kỹ thuật (không đổi API, không đổi định dạng blob):

- **Không còn lệnh chia trên dữ liệu bí mật (lớp lỗi KyberSlash):** `qmod` (`a % Q`), bước nén ciphertext (`(x·2^d + Q/2) / Q`) và bước giải mã thông điệp nay là phép nhân-dịch viết tay (Barrett; hằng 80635 / 1290167 như bản sửa của pq-crystals), bit-đúng với công thức cũ (test vét cạn). Trước đây chỉ bản native release được LLVM tự đổi; bản **WASM** còn `i32.div_s` / `i32.rem_s` mà trình duyệt có thể JIT ra lệnh chia phần cứng có thời gian phụ thuộc giá trị, và bản debug còn `sdiv`. `tests/ct/no-div.sh` rã mã máy WASM release + debug của máy đang chạy và báo lỗi nếu còn lệnh chia trong bất kỳ hàm `p_pqc::*` nào (ngoại lệ duy nhất `p_pqc::keccak`, chỉ chia độ dài/chỉ số công khai).
- **DRBG an toàn khi `fork()`:** trên unix, HMAC_DRBG của mỗi luồng ghi PID lúc seed; khi PID đổi (đang ở tiến trình con) nó reseed bằng entropy mới của hệ điều hành trước khi sinh, nên tiến trình con không còn lặp lại `m` của encaps hay nonce GCM của tiến trình cha. Test `drbg_reseeds_after_fork` fork thật và so byte cha/con.

Bản cập nhật **2026-06-21** áp dụng các fix bảo mật vào lõi thư viện (không đổi API):

- **Constant-time FO transform:** phép so sánh re-encryption trong Fujisaki-Okamoto khi giải nang (decapsulate) nay dùng `subtle::ConstantTimeEq` + bitwise select không rẽ nhánh, thay cho `if` phụ thuộc dữ liệu — loại bỏ timing oracle có thể phá vỡ IND-CCA2.
- **Chống DoS:** `pqc_decrypt` validate độ dài KEM ciphertext nhúng (`kem_len == CT_BYTES`) trước khi giải nang — từ chối blob giả mạo từng gây panic.
- **Xoá bộ nhớ (zeroize):** `KeyPair` và `Encapsulation` derive `Zeroize` / `zeroize(drop)`; mọi giá trị bí mật trung gian (seed keygen d/z, message KEM m, buffer hash g, m_prime giải mã, secret key trung gian) được xoá khỏi RAM sau khi dùng.
- **Validate biên JNI:** cầu nối JNI Android nay kiểm tra độ dài public key / secret key / ciphertext trước khi gọi vào lõi Kyber (ByteArray ngắn từ Java từng gây panic làm **abort cả app** trên Rust 1.81+); thay các `.unwrap()` bằng trả null an toàn để tránh abort khi plaintext giải mã chứa NUL byte hoặc JVM OOM.

> Lõi crypto (Keccak/SHA-3, SHAKE, NTT, ChaCha20-Poly1305 AEAD) đã được kiểm định **đúng** và không thay đổi. Chữ ký C ABI / API công khai **không đổi** — không cần sửa code tích hợp.

Liên lạc

Email: letanphuoc@gmail.com

Thư viện: `p-pqc`
Header: `p_pqc.h`
Ví dụ đầy đủ: `examples/swift/` · `examples/kotlin/` · `examples/flutter/`