Post-Quantum Cryptography Developer Quick Start Guide

Implementing NIST PQC Standards with Open Quantum Safe

Overview

This guide covers practical implementation of NIST post-quantum cryptography standards using the Open Quantum Safe (OQS) project. Focus on ML-KEM (FIPS 203) for key encapsulation and ML-DSA (FIPS 204) for digital signatures. For production, consult vendor documentation.

NIST PQC Standards Quick Reference

FIPS	Algorithm	Purpose	OQS Name	Security Levels
203	**ML-KEM**	Key Encapsulation	`ML-KEM-*`	512, 768, 1024
204	**ML-DSA**	Digital Signatures	`ML-DSA-*`	44, 65, 87
205	**SLH-DSA**	Stateless Signatures	`SLH-DSA-*`	Multiple variants

1. Environment Setup

Prerequisites (Ubuntu/Debian)

```
sudo apt install cmake gcc ninja-build libssl-dev \
  python3-pytest unzip doxygen graphviz
```

Install liboqs

```
git clone https://github.com/open-quantum-safe/liboqs.git
cd liboqs && mkdir build && cd build
cmake -GNinja -DCMAKE_INSTALL_PREFIX=/usr/local ..
ninja && sudo ninja install
```

Install OQS-OpenSSL Provider (for TLS)

```
git clone https://github.com/open-quantum-safe/oqs-provider.git
cd oqs-provider && mkdir build && cd build
cmake -GNinja -DOPENSSL_ROOT_DIR=/usr/local ..
ninja && sudo ninja install
```

2. ML-KEM Key Encapsulation (FIPS 203)

ML-KEM replaces RSA/ECDH for key exchange. Example using liboqs C API:

```
#include <oqs/oqs.h>

OQS_KEM *kem = OQS_KEM_new(OQS_KEM_alg_ml_kem_768);
uint8_t *public_key = malloc(kem->length_public_key);
uint8_t *secret_key = malloc(kem->length_secret_key);
uint8_t *ciphertext = malloc(kem->length_ciphertext);
uint8_t *shared_secret = malloc(kem->length_shared_secret);

// Generate keypair
OQS_KEM_keypair(kem, public_key, secret_key);

// Encapsulate (sender)
OQS_KEM_encaps(kem, ciphertext, shared_secret, public_key);

// Decapsulate (receiver)
OQS_KEM_decaps(kem, shared_secret, ciphertext, secret_key);

OQS_KEM_free(kem);
```

3. ML-DSA Digital Signatures (FIPS 204)

ML-DSA replaces RSA/ECDSA for signing. Example using liboqs C API:

```
#include <oqs/oqs.h>

OQS_SIG *sig = OQS_SIG_new(OQS_SIG_alg_ml_dsa_65);
uint8_t *public_key = malloc(sig->length_public_key);
uint8_t *secret_key = malloc(sig->length_secret_key);
uint8_t *signature = malloc(sig->length_signature);
size_t sig_len;

// Generate keypair
OQS_SIG_keypair(sig, public_key, secret_key);

// Sign message
OQS_SIG_sign(sig, signature, &sig_len, message, msg_len, secret_key);

// Verify signature
OQS_SIG_verify(sig, message, msg_len, signature, sig_len, public_key);

OQS_SIG_free(sig);
```

4. Python Integration

Using liboqs-python wrapper:

```
pip install liboqs-python

import oqs

# Key Encapsulation
kem = oqs.KeyEncapsulation("ML-KEM-768")
public_key = kem.generate_keypair()
ciphertext, shared_secret = kem.encap_secret(public_key)

# Digital Signatures
sig = oqs.Signature("ML-DSA-65")
public_key = sig.generate_keypair()
signature = sig.sign(message)
is_valid = sig.verify(message, signature, public_key)
```

5. Post-Quantum TLS Configuration

Configure OpenSSL 3.x with oqs-provider for PQC TLS:

```
# openssl.cnf addition
[openssl_init]
providers = provider_sect

[provider_sect]
default = default_sect
oqsprovider = oqsprovider_sect

[oqsprovider_sect]
activate = 1
```

Test PQC Key Exchange

```
# Generate PQC certificate
openssl req -x509 -new -newkey mldsa65 -keyout server.key \
  -out server.crt -nodes -subj "/CN=localhost" -days 365

# Start server with hybrid key exchange
openssl s_server -cert server.crt -key server.key \
  -groups x25519_mlkem768 -www

# Test client connection
openssl s_client -groups x25519_mlkem768 -connect localhost:4433
```

6. Implementation Best Practices

Use Hybrid Mode:	Combine classical + PQC algorithms during transition (e.g., x25519_mlkem768)
Security Level Selection:	ML-KEM-768 / ML-DSA-65 for most use cases (128-bit equivalent)
Key Rotation:	PQC keys should follow same rotation policies as classical crypto
Performance Testing:	PQC operations are larger; benchmark before production deployment
Library Updates:	Stay current with liboqs releases for security patches

Developer Resources

Open Quantum Safe:	openquantumsafe.org
liboqs GitHub:	github.com/open-quantum-safe/liboqs
oqs-provider:	github.com/open-quantum-safe/oqs-provider
NIST PQC Standards:	csrc.nist.gov/projects/post-quantum-cryptography
IBM Quantum Safe:	developer.ibm.com/tutorials/awb-quantum-safe-openssl

Important Note

OQS libraries are for research and prototyping. For production systems, use vendor-supported implementations from your HSM, cloud provider, or security vendor once NIST-certified.