Quantum Cryptography and Post-Quantum

MAT364 - Cryptography Course

Instructor: Adil Akhmetov

University: SDU

Week 13

Press Space for next page →

Week 13 Focus

Motivation

- Quantum computers threaten current cryptographic systems
- Shor’s algorithm breaks RSA and ECDSA
- Need quantum-resistant alternatives
- Quantum key distribution offers provable security

Agenda

- Quantum computing fundamentals and threats
- Quantum key distribution (BB84 protocol)
- Post-quantum cryptography families
- NIST PQC standardization
- Migration planning and hybrid approaches
- Lab: Implement a post-quantum signature scheme

Learning Outcomes

1. Understand quantum computing threats to cryptography
2. Explain quantum key distribution (QKD) principles
3. Identify post-quantum cryptographic algorithms
4. Evaluate migration strategies for post-quantum security

The Quantum Threat

Why Quantum Computing Matters

The Threat Timeline

- **Current:** Classical computers (limited threat)
- **5-10 years:** Small-scale quantum computers
- **10-30 years:** Cryptographically relevant quantum computers
- **"Harvest now, decrypt later"** attacks already happening

Shor’s Algorithm Impact

```
Classical: Factor 2048-bit RSA
- Best algorithm: ~10^20 years
- Requires: Classical supercomputer

Quantum: Factor 2048-bit RSA
- Shor's algorithm: ~hours/days
- Requires: ~4000 logical qubits
```

Algorithms at Risk

- **RSA** - Factoring problem (Shor’s algorithm)
- **ECDSA/ECDH** - Discrete log problem (Shor’s algorithm)
- **Diffie-Hellman** - Discrete log problem
- **Symmetric crypto** - Grover’s algorithm (halves key size)

Grover’s Algorithm Impact

- **AES-128** → equivalent to AES-64 security
- **AES-256** → equivalent to AES-128 security
- **Solution:** Use AES-256 for post-quantum security

Critical: Data encrypted today with RSA/ECDSA may be decrypted in 10-20 years. Start planning migration now!

Quantum Computing Basics

Qubits vs Bits

- **Classical bit:** 0 or 1
- **Quantum bit (qubit):** Superposition of 0 and 1
- **Entanglement:** Qubits can be correlated
- **Measurement:** Collapses superposition to 0 or 1

Quantum Advantage

- **Parallelism:** Process many states simultaneously
- **Interference:** Amplify correct answers
- **Entanglement:** Correlate distant qubits
- **Limitations:** Measurement destroys superposition

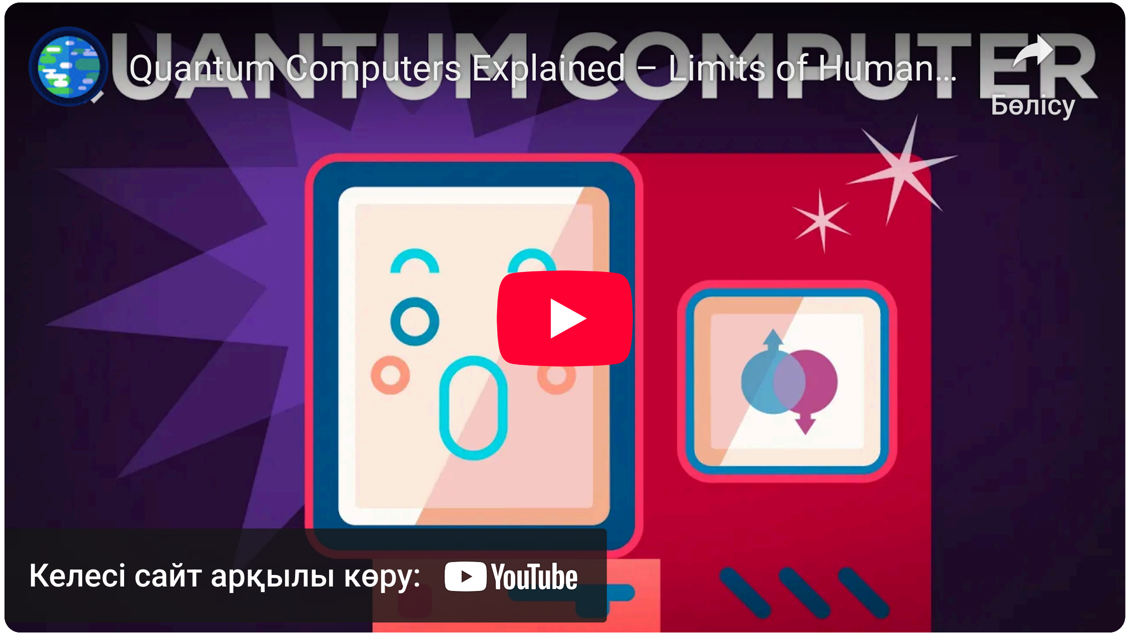
Quantum Gates

- **Hadamard:** Creates superposition
- **CNOT:** Creates entanglement
- **Phase gates:** Manipulate quantum phases
- **Measurement:** Extract classical information

Current State

- **IBM:** 100+ qubit processors
- **Google:** Quantum supremacy demonstrated
- **Challenges:** Error rates, decoherence, scaling
- **Timeline:** Cryptographically relevant QC in 10-30 years

Video: Introduction to Quantum Computing


Quantum Computers Explained – Limits of Human...
QUANTUM COMPUTER
Бөлісу
Келесі сайт арқылы көру: YouTube

Source: Veritasium - Quantum Computing Explained

Quantum Key Distribution (QKD)

BB84 Protocol

Protocol Overview

BB84 (Bennett & Brassard, 1984):

- First practical quantum key distribution
- Uses quantum properties for security
- Provably secure against eavesdropping
- No computational assumptions

Quantum States

- **Basis 0 (Z):** $|0\rangle$, $|1\rangle$
- **Basis 1 (X):** $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$, $|-\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$
- **Random basis choice** for each qubit
- **Measurement** in wrong basis gives random result

Protocol Steps

1. **Alice** sends qubits in random bases
2. **Bob** measures in random bases
3. **Public discussion:** Compare bases (not results)
4. **Key extraction:** Keep bits where bases matched
5. **Error checking:** Detect eavesdropping
6. **Privacy amplification:** Remove leaked information

Security Guarantee

- **Eavesdropping detection:** Any measurement disturbs qubits
- **Information-theoretic security:** No computational assumptions
- **Perfect secrecy:** Even with unlimited computational power

BB84 Implementation Example

Simplified Python Simulation

```python
import random
import numpy as np

class BB84Protocol:
    def __init__(self):
        self.bases = ['Z', 'X']  # Two measurement base

    def alice_prepare_qubit(self, bit, basis):
        """Alice prepares a qubit"""
        # In real QKD, this would be a physical qubit
        # Here we simulate with classical representatio
        return {
            'bit': bit,
            'basis': basis
        }
```

Error Detection

```python
def error_detection(self, key1, key2, sample_size=10):
    """Detect errors (eavesdropping)"""
    # Compare random sample of bits
    sample_indices = random.sample(
        range(min(len(key1), len(key2))),
        sample_size
    )

    errors = 0
    for idx in sample_indices:
        if key1[idx] ≠ key2[idx]:
            errors += 1

    error_rate = errors / sample_size
    return error_rate < 0.1  # Threshold for acceptabl
```

Real-World QKD

- **Distance limits:** ~200km over fiber, longer with repeaters
- **Rate limits:** ~Mbps key generation
- **Cost:** Expensive equipment

Video: Quantum Key Distribution Explained

Source: Quantum Key Distribution

Post-Quantum Cryptography

Post-Quantum Algorithm Families

Lattice-Based

- **Security:** Shortest Vector Problem (SVP)
- **Examples:** CRYSTALS-Kyber, CRYSTALS-Dilithium, Falcon
- **Pros:** Fast, versatile (encryption, signatures, KEM)
- **Cons:** Large public keys/signatures

Code-Based

- **Security:** Decoding random linear codes
- **Examples:** Classic McEliece, BIKE
- **Pros:** Long history, well-studied
- **Cons:** Large public keys

Hash-Based

- **Security:** One-way hash functions
- **Examples:** SPHINCS+, XMSS, LMS
- **Pros:** Mature, conservative security
- **Cons:** Large signatures, stateful schemes

Multivariate

- **Security:** Solving systems of multivariate equations
- **Examples:** Rainbow (broken), GeMSS
- **Pros:** Fast verification
- **Cons:** Large keys, less mature

NIST PQC Standardization: Selected CRYSTALS-Kyber (KEM) and CRYSTALS-Dilithium, FALCON, SPHINCS+ (signatures) in 2024.

NIST Post-Quantum Cryptography Standardization

Timeline

- **2016:** Call for proposals
- **2017:** 69 submissions received
- **2019-2020:** Round 2 and 3 evaluations
- **2022:** Finalists selected
- **2024:** Standards published (FIPS 203, 204, 205)

Selected Algorithms

Key Encapsulation (KEM):

- **CRYSTALS-Kyber** - Primary standard
- **Alternatives:** BIKE, HQC, SIKE (withdrawn)

Digital Signatures:

- **CRYSTALS-Dilithium** - Primary standard
- **FALCON** - For small signatures
- **SPHINCS+** -

Algorithm Comparison

Algorithm	Type	Key Size	Signature Size	Security Level
Kyber-768	KEM	1,568 B	-	Level 3
Dilithium-3	Signature	1,952 B	3,293 B	Level 3
Falcon-512	Signature	897 B	666 B	Level 1
SPHINCS±256f	Signature	64 B	49,856 B	Level 5

Migration Priority

1. **High:** TLS, VPN, email encryption

CRYSTALS-Kyber (KEM)

Overview

- **Type:** Lattice-based key encapsulation
- **Security:** Module-LWE problem
- **Standard:** FIPS 203
- **Use case:** Replace RSA/ECDH key exchange

Key Generation

```python
# Conceptual implementation
class KyberKEM:
    def __init__(self, security_level=3):
        # security_level: 1, 3, or 5
        self.n = 256  # Polynomial degree
        self.q = 3329  # Modulus
        self.k = {1: 2, 3: 3, 5: 4}[security_level]

    def keygen(self):
        """Generate key pair"""
        # Generate matrix A (public parameter)
```

Encapsulation & Decapsulation

```python
    def encapsulate(self, public_key):
        """Encapsulate shared secret"""
        t, A = public_key

        # Generate random vector
        m = self.sample_message()

        # Generate error vectors
        r, e1, e2 = self.sample_errors()

        # Compute ciphertext
        u = A.T * r + e1
        v = t.T * r + e2 + self.encode(m)

        # Derive shared secret
```

Real Implementation

- Use **liboqs** or **pqcrypto** libraries
- Never implement from scratch for production
- Follow NIST specifications exactly

CRYSTALS-Dilithium (Signatures)

Overview

- **Type:** Lattice-based digital signature
- **Security:** Module-LWE and Module-SIS
- **Standard:** FIPS 204
- **Use case:** Replace RSA/ECDSA signatures

Key Generation

```
class DilithiumSignature:
    def __init__(self, security_level=3):
        self.n = 256  # Polynomial degree
        self.q = 8380417  # Modulus
        self.k = {1: 4, 3: 6, 5: 8}[security_level]
        self.l = {1: 4, 3: 5, 5: 7}[security_level]

    def keygen(self):
        """Generate signing key pair"""
        # Generate matrix A
        A = self.generate_matrix()
```

Signing & Verification

```
    def sign(self, message, secret_key):
        """Sign message"""
        A, t1 = self.public_key
        s1, s2, t = secret_key

        # Generate random vector
        y = self.sample_y()

        # Compute challenge
        w1 = self.low_bits(A * y)
        c = self.hash(message, w1)

        # Compute signature
        z = y + c * s1
        h = self.make_hint(-c * t, w1 - c * s2)
```

Video: Post-Quantum Cryptography Overview

Understanding Post-Quantum Cryptography (PQC)
CURRENT CRYPTOGRAPHY MECHANISMS WILL BE AT RISK
Бөлісу
THALES
Келесі сайт арқылы көру: YouTube

Source: Understanding Post-Quantum Cryptography

Practical Implementation

Using Post-Quantum Libraries

liboqs (C/C++)

```
# Python bindings for liboqs
from oqs import KeyEncapsulation, Signature

# Key Encapsulation (Kyber)
kem = KeyEncapsulation('Kyber768')
public_key, secret_key = kem.generate_keypair()

# Encapsulate
ciphertext, shared_secret = kem.encapsulate(public_key
# Decapsulate
shared_secret2 = kem.decapsulate(ciphertext, secret_key
assert shared_secret == shared_secret2

# Digital Signature (Dilithium)
```

Python cryptography Library

```
from cryptography.hazmat.primitives.asymmetric import k
from cryptography.hazmat.primitives import serializatio

# Kyber KEM
private_key = kyber.generate_private_key(kyber.Kyber768
public_key = private_key.public_key()

# Encapsulate
ciphertext, shared_secret = public_key.encapsulate()

# Decapsulate
shared_secret2 = private_key.decapsulate(ciphertext)

# Dilithium Signature
private_key = dilithium.generate_private_key(dilithium
```

Installation

```
# Install liboqs Python bindings
pip install oqs
```

Hybrid Approaches

Why Hybrid?

- **Transition period:** Support both classical and PQ
- **Backward compatibility:** Works with existing systems
- **Risk mitigation:** If one breaks, other still works
- **Gradual migration:** Phase out classical over time

Hybrid TLS

```
TLS 1.3 with hybrid key exchange:
- ECDHE (X25519) + Kyber-768
- Both keys exchanged
- Shared secret = KDF(ECDHE_secret || Kyber_secret)
- Secure if either algorithm is secure
```

Implementation Example

```
from cryptography.hazmat.primitives.asymmetric import x
from cryptography.hazmat.primitives.kdf.hkdf import HKD
from cryptography.hazmat.primitives import hashes

def hybrid_key_exchange():
    # Classical: X25519
    x25519_private = x25519.X25519PrivateKey.generate()
    x25519_public = x25519_private.public_key()

    # Post-quantum: Kyber
    kyber_private = kyber.generate_private_key(kyber.Ky
    kyber_public = kyber_private.public_key()

    # Exchange public keys (simulated)
    #     network exchange
```

Migration Strategy

Migration Planning

Phase 1: Assessment (Now)

- **Inventory:** List all cryptographic systems
- **Risk analysis:** Identify critical data
- **Dependencies:** Map crypto library usage
- **Timeline:** Estimate migration effort

Phase 2: Hybrid Deployment (1-2 years)

- **Enable hybrid:** Support both classical and PQ
- **Test thoroughly:** Validate PQ implementations
- **Monitor performance:** Measure overhead
- **Update standards:** Revise security policies

Phase 3: Full Migration (3-5 years)

- **Remove classical:** Phase out old algorithms
- **Update protocols:** TLS, SSH, etc.
- **Train staff:** Update documentation
- **Audit compliance:** Verify PQ adoption

Phase 4: Post-Migration (Ongoing)

- **Monitor standards:** Watch for new attacks
- **Update algorithms:** Migrate to newer PQ schemes
- **Maintain hybrid:** Keep flexibility

Timeline: Start planning now! Full migration may take 5-10 years, but critical systems should be hybrid-ready within 2-3 years.

Migration Checklist

Technical Tasks

- ☐Audit all cryptographic systems
- ☐Identify RSA/ECDSA usage
- ☐Test PQ libraries in dev environment
- ☐Implement hybrid key exchange
- ☐Update TLS configurations
- ☐Modify certificate infrastructure
- ☐Update code signing workflows
- ☐Test performance impact

Organizational Tasks

- ☐Train development teams
- ☐Update security policies
- ☐Revise compliance documentation
- ☐Plan budget for migration
- ☐Coordinate with vendors
- ☐Establish testing procedures
- ☐Create rollback plans
- ☐Monitor industry standards

Performance Considerations

Metric	Classical	Post-Quantum	Impact
Key Exchange	~1ms	~2-5ms	2-5x slower
Signature Size	64-256 B	666-50k B	10-200x larger
Public Key Si	32-256 B	800-2k B	10-20x l

Lab: Post-Quantum Implementation

🎯 Student Lab Assignment

Scenario

You need to implement a post-quantum secure messaging system that can replace an existing RSA-based system.

Tasks

1. Install and test `liboqs` Python bindings (or similar library)
2. Implement Kyber key exchange between two parties
3. Implement Dilithium signatures for message authentication
4. Create a hybrid system that supports both classical (X25519) and post-quantum (Kyber) key exchange
5. Measure and compare performance (key generation, encryption, signing times)

Deliverables

- Working code demonstrating PQ key exchange and signatures
- Performance comparison table (classical vs PQ vs hybrid)
- Short report on migration challenges and recommendations

✅ Solution Outline

Implementation Structure

```python
from oqs import KeyEncapsulation, Signature
import json
import time

class PostQuantumMessaging:
    def __init__(self):
        # Initialize KEM and signature schemes
        self.kem = KeyEncapsulation('Kyber768')
        self.sig = Signature('Dilithium3')

        # Generate keys
        self.kem_pub, self.kem_priv = self.kem.generate
        self.sig_pub, self.sig_priv = self.sig.generate

    def send_message(self, message, recipient_kem_pub)
```

Performance Benchmarking

```python
def benchmark_pq_algorithms():
    results = {}

    # Benchmark Kyber
    kem = KeyEncapsulation('Kyber768')
    start = time.time()
    pub, priv = kem.generate_keypair()
    results['kyber_keygen'] = time.time() - start

    start = time.time()
    ct, ss = kem.encapsulate(pub)
    results['kyber_encaps'] = time.time() - start

    start = time.time()
    ss2 = kem.decapsulate(ct, priv)
```

Challenges and Considerations

Post-Quantum Challenges

Technical Challenges

- **Large key/signature sizes:** Bandwidth and storage impact
- **Performance overhead:** Slower than classical crypto
- **Library maturity:** Fewer implementations available
- **Standardization:** Still evolving (NIST PQC Round 4)
- **Interoperability:** Need compatible implementations

Operational Challenges

- **Cost:** Hardware/software upgrades
- **Training:** Staff education required
- **Vendor support:** Limited PQ support
- **Compliance:** Regulatory approval needed
- **Timeline:** Long migration periods

Implementation Risks

- **Side-channel attacks:** New attack vectors
- **Implementation bugs:** Less battle-tested code
- **Algorithm selection:** Risk of choosing broken algorithm
- **Migration complexity:** Large codebase changes

Best Practices

- **Use hybrid approach:** Deploy both classical and PQ
- **Follow standards:** Use NIST-approved algorithms
- **Test thoroughly:** Extensive testing before deployment
- **Monitor research:** Stay updated on PQ developments
- **Plan ahead:** Start migration planning early

Video: The Future of Cryptography

How Quantum Computers Break Encryption | Shor'...
Бөлісу
Quantum Computers
Destroy Internet Security
Келесі сайт арқылы көру: YouTube

Source: MinutePhysics - How Quantum Computers Break Encryption | Shor's Algorithm Explained

Summary

- **Quantum computers** threaten current cryptographic systems (RSA, ECDSA)
- **Quantum key distribution** offers provable security but has practical limitations
- **Post-quantum cryptography** provides quantum-resistant alternatives
- **NIST standards** (Kyber, Dilithium, Falcon, SPHINCS+) are now available
- **Hybrid approaches** enable gradual migration while maintaining security
- **Migration planning** should start now for critical systems

Next Week: Practical projects and final project presentations.

Assignment: Complete the post-quantum lab and submit performance analysis report.

Questions?

Thanks for exploring quantum and post-quantum cryptography! ⚛️🔐