Group10 : Feng Ruhui / Zhao Jinghao

TLS Handshake Process
Complete Demonstration

Cryptography Implementation and Analysis Based on Java

Java Implementation

Security Analysis

7-Stage Protocol

Course

SC6104 Introduction to Cryptography

Date

February 2026

Introduction

Project Overview

Project Objectives

Complete simulation of **7 key stages** of TLS handshake through Java code, providing an educational platform for understanding modern cryptographic protocols.

Demonstration Value

- Visualize TLS handshake process with step-by-step execution
- Deep understanding of asymmetric and symmetric cryptography
- Showcase practical applications in real-world secure communications

Core Demonstration Modules

1 RSA Key Exchange

Asymmetric encryption for secure Pre-Master Secret transmission

2 Certificate Verification

CA signature validation to prevent MITM attacks

3 AES-GCM Cipher Suite

Authenticated encryption for secure data transmission

4 Complete Handshake

Full TLS handshake simulation integrating all modules

Presentation Duration

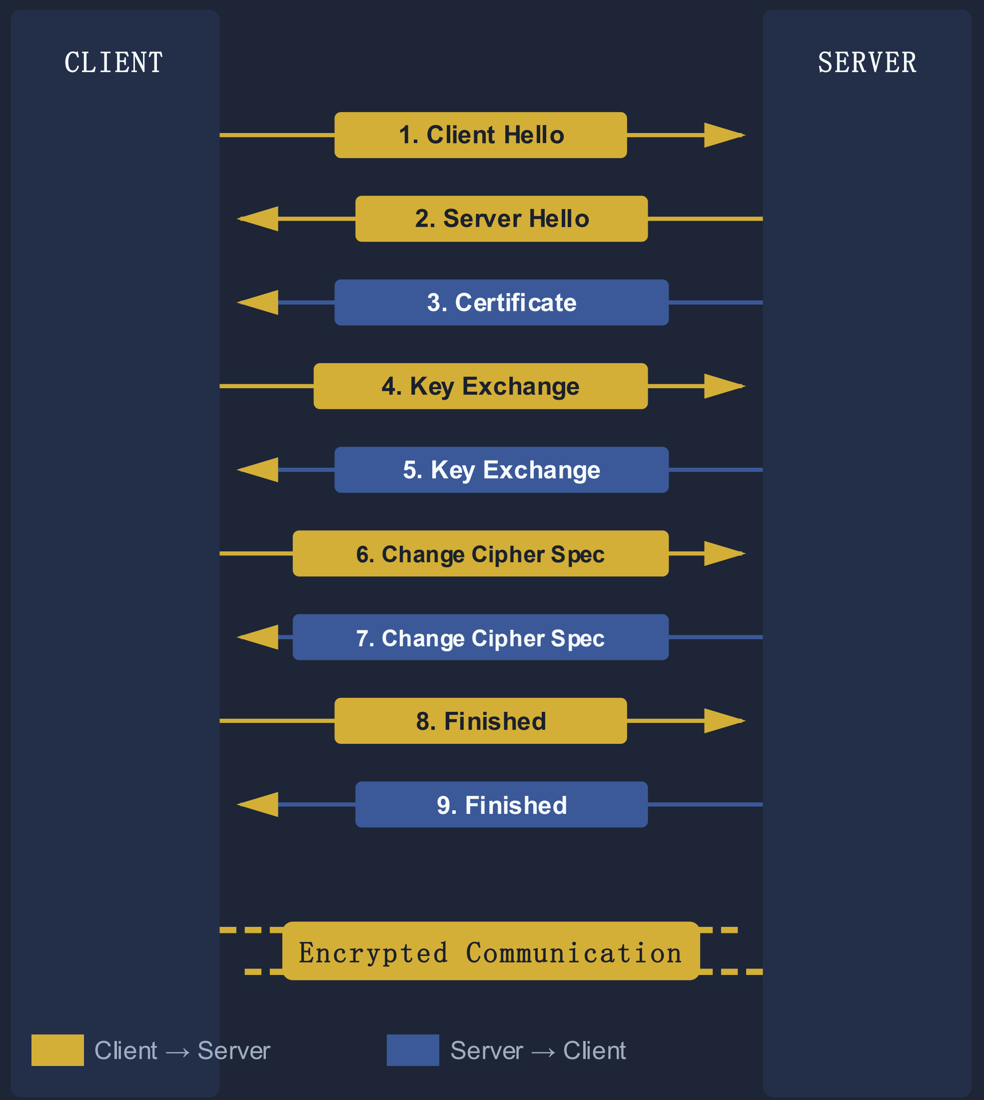
Approximately 6 minutes with live demonstrations

4 Modules

Protocol Flow

TLS Handshake 7-Stage Overview

Complete Handshake Sequence



1. **Negotiation Phase**
   Client/Server Hello: Algorithm negotiation, random number exchange (R1, R2)

2. **Identity Verification**
   Server presents X.509 certificate with CA signature for authentication

3. **Key Exchange**
   Pre-Master Secret (PMS) encrypted with server public key, transmitted securely

4. **Session Key Generation**
   Both parties derive identical session key from R1, R2, PMS using PRF

5. **Encrypted Communication**
   AES-GCM symmetric encryption for all subsequent data transmission

Key Insight: Asymmetric encryption secures key exchange; symmetric encryption ensures efficient data transmission

Core Module 1

RSA Key Exchange

Mathematical Principles

Encryption Formula

$C \equiv PMS^e \pmod{n}$

Decryption Formula

$PMS \equiv C^d \pmod{n}$

Security: Only server with private key d can decrypt PMS

Key Security Points

- Asymmetric encryption secures symmetric key exchange
- Only server can decrypt PMS, ensuring confidentiality

Demonstration Flow

1. **Server Generates RSA Key Pair**

   2048-bit RSA key pair generation

   ```
   KeyPairGenerator.getInstance("RSA")
   ```

2. **Client Generates PMS**

   48-byte random number: TLS version (2 bytes) + random data (46 bytes)

3. **Client Encrypts PMS**

   Encrypt using server public key → Output: Ciphertext (Base64)

4. **Server Decrypts PMS**

   Decrypt using private key → Verify: PMS matches original

5. **Generate Session Key**

   Both parties compute:

   ```
   Session Key = SHA-256(R1 || R2 || PMS)
   ```

Core Module 2

Digital Certificate & CA Signature

Core Mechanisms

CA Signature Process

Sign = RSA_Encrypt(SHA-256(Cert), CA_Priv)

Client Verification

Verify(Sign, Hash(Cert), CA_Pub)

MITM Attack Prevention

Valid CA Signature
Verification passes ✓

Attacker Forged Signature
Verification fails ✗

Tampered Certificate
Hash mismatch detected ✗

Verification Flow

1 CA Generates Key Pair

RSA 2048-bit key pair: CA private key (secret) for signing, CA public key (public) for verification

2 CA Signs Certificate

- Compute certificate hash (SHA-256)
- Sign hash with CA private key
- Output: Digital signature (Base64)

3 Client Verifies Signature

- Recompute certificate hash
- Verify using CA public key

✓ Server identity verified

Security: Attackers cannot forge CA signature without CA private key; certificate tampering is detected via hash verification

Core Module 3

AES-GCM Symmetric Encryption

Algorithm Characteristics

Algorithm	AES-256-GCM
Mode	Galois/Counter Mode

Key Advantage:

Authenticated Encryption with Associated Data (AEAD) — provides both confidentiality and integrity

Key Security Points

- Symmetric encryption is **much faster** than asymmetric
- GCM mode provides **encryption + authentication**
- Unique IV per encryption: same plaintext → different ciphertext

Encryption & Decryption Flow

1 Derive AES Key from Session Key

Extract 32 bytes from session key as AES-256 key

2 Client Encrypts HTTP Message

- Input: HTTP request (plaintext)
- Generate random IV (12 bytes)
- AES-GCM encryption
- Output: Ciphertext + IV (Base64)

3 Server Decrypts HTTP Message

- Input: Ciphertext + IV
- AES-GCM decryption
- Output: Original HTTP message

4 Tampering Detection Demo

- Scenario: Modify 1 byte in ciphertext

✗ Decryption fails, exception thrown

- GCM mode automatically verifies data integrity

Integration

Complete TLS Handshake Flow

1 Client Hello & Server Hello

- Negotiate TLS version and cipher suite
- Exchange random numbers R1 and R2
- Output: Negotiation results, random numbers (Base64)

2 Server Certificate

- Server presents certificate information
- Display: Subject, Issuer, validity period, etc.

3 Certificate Verification

- Client verifies certificate authenticity
- CA signature validation process

✓ Certificate verification passed

4 Key Exchange

- Client generates PMS
- Encrypt PMS with server public key
- Server decrypts PMS with private key

✓ PMS match successful

5 Session Key Generation

- Generate session key using R1, R2, PMS
- Output: Session key (Base64)

6 Change Cipher Spec & Finished

- Both parties switch to encrypted mode
- Exchange Finished messages to confirm handshake completion

7 Encrypted Communication

- Encrypt HTTP messages using session key
- Server decrypts HTTP messages

✓ Data transmitted completely and securely

Execution Command

```
java tls.demo.TLSHandshakeSimulator
```

Key Takeaways

- Asymmetric encryption (RSA) securely exchanges symmetric keys
- Digital certificates and CA signatures prevent MITM attacks
- Symmetric encryption (AES-GCM) efficiently encrypts actual data
- Session keys generated from both random numbers and PMS ensure uniqueness

Security Assessment

Security Analysis & Protection

Key Exchange Security

RSA Foundation:

Security based on large integer factorization problem — computationally infeasible to derive private key from public key

Exclusive Decryption:

Only server with private key d can decrypt PMS

Eavesdropping Protection:

Even if attacker intercepts ciphertext, PMS cannot be recovered without private key

Identity Verification

CA Signature:

Prevents certificate forgery — attackers cannot create valid signatures without CA private key

Tamper Detection:

Any modification to certificate changes hash value, causing verification failure

MITM Prevention:

Attackers cannot impersonate legitimate servers without valid CA-signed certificate

Data Encryption Security

AES-256 Strength:

256-bit key provides computational security — brute force attack infeasible

GCM Authentication:

Provides both confidentiality and integrity — any tampering detected immediately

Replay Attack Prevention:

Unique IV for each encryption ensures same plaintext produces different ciphertext

Limitations

- ⚠ This project demonstrates RSA key exchange (TLS 1.2 method)
- ⚠ TLS 1.3 has removed RSA key exchange, replaced with ECDHE
- ⚠ ECDHE provides forward secrecy — compromised server key doesn't expose past sessions

Extended Topics

- → **TLS 1.3 Improvements:** 0-RTT handshake, faster speed, enhanced security
- → **Forward Secrecy:** Even if server private key leaked, historical communications remain secure

Implementation

Technical Implementation

Implementation Stack

Programming Language
Java 8+

Cryptography Library
Java Cryptography Extension (JCE)

Core Algorithms

Asymmetric Encryption	RSA 2048-bit
Symmetric Encryption	AES-256-GCM
Hash Algorithm	SHA-256
Signature Algorithm	SHA256withRSA

Key Lengths

RSA Key	Pre-Master Secret	AES Key	Client/Server Random
2048 bits	48 bytes	256 bits	32 bytes each

Code Structure

KeyExchangeDemo.java
RSA key exchange demonstration

```
java tls.demo.KeyExchangeDemo
```

SignatureVerify.java
CA signature verification

```
java tls.demo.SignatureVerify
```

CipherSuite.java
AES-GCM symmetric encryption

```
java tls.demo.CipherSuite
```

TLSHandshakeSimulator.java
Complete handshake integration

```
java tls.demo.TLSHandshakeSimulator
```

Design: Each module runs independently with detailed console output, mathematical formulas, and security analysis

Conclusion

Project Summary & Future

Core Achievements

- Complete implementation of **7 TLS handshake stages**
- In-depth demonstration of **core cryptography principles**
- Practical application showcase with **real-world scenarios**

Learning Outcomes

- → Understanding **asymmetric-symmetric encryption coordination**
- → Mastering **digital certificate and CA signature mechanisms**
- → Comprehending **TLS security guarantee mechanisms**

Future Directions

1. Implement **TLS 1.3** with ECDHE key exchange
2. Add **forward secrecy** demonstration
3. **Performance optimization** and extended scenarios

Thank You!