Distributed Systems

Security in Distributed Systems

WEEK 15: Cryptography, Authentication, Authorization, Secure Communication, Trust Models.

Online Lecture Series - 15

Felix Edesa

Addis Ababa Science and Technology University

Course Outline: Security in Distributed Systems (Lecture 15)

Topics We Will Cover

- **Cryptography Basics** — Symmetric/asymmetric encryption, hashing
- **Authentication** — Passwords, multi-factor, tokens
- **Authorization** — Role-based and attribute-based access control
- **Secure Communication** — SSL/TLS, secure channels
- **Trust Models** — PKI, certificate authorities
- **Security Threats** — Common attacks in distributed systems
- **Practical Applications** — Blockchain, microservices, cloud security

Cryptography Basics — Introduction

What is Cryptography?

- Cryptography is the science of securing information.
- Protects confidentiality, integrity, and authenticity of data.
- Supports secure communication in distributed systems and industrial applications.
- Enables authentication of users, devices, and systems.
- Forms the foundation for secure protocols like TLS, SSH, and VPNs.

Why it matters

- Prevents unauthorized access to sensitive data.
- Essential for secure payments, messaging, and IoT devices.

Symmetric Encryption — Introduction

Key Concepts

- Same key is used for both encryption and decryption.
- Fast and efficient, ideal for bulk data encryption.
- Examples include AES (Advanced Encryption Standard), DES, 3DES.
- Security depends on keeping the key secret between parties.
- Vulnerable if key distribution is not secure.

Industrial Use Cases

- Disk and file encryption (e.g., BitLocker, VeraCrypt).
- Secure communication within corporate or cloud networks.
- Encrypting payment data in financial systems.

Symmetric Encryption — Example

Mathematical Example (Caesar Cipher)

- Plaintext: HELLO
- Key: 3
- Ciphertext: KHOOR (Shift each letter by key)
- Decryption: Shift back by 3 to get HELLO
- Each letter is shifted uniformly by the key value.
- Only someone with the correct key can decrypt.
- Demonstrates confidentiality using symmetric encryption.

Practical Note

- Modern symmetric encryption uses complex algorithms (AES-256) instead of simple shifts.
- Ensures industrial-grade security for large-scale systems.

Asymmetric Encryption — Introduction

Key Concepts

- Uses a public-private key pair.
- Public key encrypts, private key decrypts.
- Example algorithms: RSA, ECC.

Industrial Use Cases

- Secure email (PGP/GPG)
- Blockchain transactions and smart contracts
- TLS/SSL certificates for web security
- Digital signatures to verify authenticity

Asymmetric Encryption — Example

Mathematical Example (RSA Small Numbers)

- Public key: $(e, n) = (3, 33)$
- Private key: $d = 7$
- Plaintext: $m = 4$
- Encryption: $c = m^e \mod n = 4^3 \mod 33 = 31$
- Decryption: $m = c^d \mod n = 31^7 \mod 33 = 4$

Practical Note

- Real systems use very large primes for security (2048+ bits).
- Common in digital signatures and blockchain verification.

Hashing — Introduction

Key Concepts

- Converts data into a fixed-size string (hash value)
- One-way function: cannot reverse to original data
- Ensures data integrity and tamper detection
- Fast computation for large datasets
- Collision-resistant: different data rarely produce same hash

Industrial Use Cases

- Password storage (bcrypt, SHA-256)
- File and software integrity checks
- Blockchain transaction verification
- Digital signatures validation

Hashing — Example

Example (SHA-256)

- Input: "Distributed"
- SHA-256 Hash: e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
- Any small change in input drastically changes hash

Practical Note

- Integrity verification in software updates, distributed systems, and microservices.

Symmetric vs Asymmetric Encryption

Key Differences

- Symmetric: Same key, fast, suitable for large data.
- Asymmetric: Public-private pair, slower, used for secure key exchange.
- Often combined: Asymmetric for key exchange, symmetric for message encryption.

Industrial Example

- TLS: RSA or ECC for key exchange, AES for data encryption.
- Payment gateways and online banking.

Practical Scenario — Secure Microservices Communication

Example

- Service A sends encrypted data to Service B.
- Symmetric encryption (AES) used for fast data transfer.
- Encryption keys exchanged securely via asymmetric methods (RSA or ECC).
- End-to-end encryption ensures confidentiality and integrity.
- Supports high-performance, scalable microservices communication.

Lesson

- Secure and low-latency communication is essential in distributed systems.
- Combining symmetric and asymmetric encryption provides both speed and security.

Practical Scenario — Blockchain Transactions

Example

- User signs a transaction with their private key (e.g., sending cryptocurrency).
- Transaction broadcast to network nodes; verified using the sender's public key.
- Hashing links blocks (e.g., SHA-256 in Bitcoin) to ensure immutability.
- Smart contracts execute automatically with digital signatures for authorization.
- Real-world examples: Bitcoin, Ethereum, supply chain tracking.

Lesson

- Ensures trustless verification, tamper-proof records, and secure automated execution.
- Demonstrates the combination of cryptography, hashing, and digital signatures in distributed systems.

Authentication — Introduction

Key Concepts

- Authentication verifies the identity of a user, device, or system.
- Forms the first line of defense in cybersecurity.
- Types: passwords, multi-factor, token-based authentication.
- Industrial relevance: banking apps, corporate networks, cloud services.
- Related standards: OAuth 2.0, SAML, OpenID Connect.
- Challenges: phishing, credential theft, replay attacks.

Why it matters

- Protects sensitive data and prevents unauthorized access.
- Reduces risk of fraud, data breaches, and cyberattacks.
- Supports regulatory compliance (GDPR, PCI-DSS, HIPAA).

Password-Based Authentication

Concepts

- User provides a secret password to verify identity.
- Stored securely using hashing algorithms (bcrypt, Argon2).
- Weak passwords are vulnerable to brute-force and dictionary attacks.
- Password leaks can occur through phishing, malware, or data breaches.
- Must be combined with additional security measures in sensitive systems.

Example / Problem

- Problem: User chooses "123456" $\rightarrow$ easily guessed.
- Solution: Enforce strong password policies and hashing.
- Real-world example: LinkedIn 2012 breach $\rightarrow$ exposed millions of weak passwords.

Password Best Practices

Recommendations

- Minimum length: 12+ characters.
- Use mix of uppercase, lowercase, numbers, symbols.
- Avoid dictionary words, repeated patterns, or personal info.
- Periodic password updates in high-risk environments.
- Use password managers to generate and store complex passwords securely.

Industrial Example

- Banking apps enforce password strength and expiration.
- Enterprise VPNs require complex passwords with periodic rotation.
- Cloud services integrate password managers for internal team accounts.

Multi-Factor Authentication (MFA)

Concepts

- Combines two or more factors for identity verification:
  - Something you know (password)
  - Something you have (hardware token, mobile device)
  - Something you are (biometrics: fingerprint, face scan)
- Reduces impact of stolen credentials.
- Can include adaptive authentication based on risk scoring.

Example

- Email login: password + OTP sent to phone.
- Enterprise login: password + fingerprint scan.
- Banking transactions: password + hardware token + SMS OTP.

Password-Only Authentication Risks

Common Problems

- Phishing attacks capture passwords.
- Password reuse across multiple accounts increases compromise risk.
- Brute-force or dictionary attacks.
- Insider threats with access to credentials.
- Credential stuffing attacks on popular services.

Solution

- Implement MFA to add layers of security.
- Monitor login attempts and enforce rate limiting.
- Educate users on phishing awareness and safe practices.

Token-Based Authentication

Concepts

- Authentication via tokens rather than sending passwords repeatedly.
- Tokens can be JWT (JSON Web Token), OAuth access tokens, or session tokens.
- Enables stateless, scalable authentication in distributed systems.
- Tokens often contain claims: user ID, roles, permissions, expiration.
- Secure token storage and verification is essential.

Example

- Web app issues JWT after successful login.
- API requests include token in header; server validates signature.
- Expired tokens force re-authentication, improving security.

Practical Scenario — MFA in Banking

Example

- User logs into bank app with username/password.
- OTP sent via SMS or authenticator app.
- High-value transactions require second factor approval.
- Mobile push notifications alert user of suspicious login.

Lesson

- MFA prevents unauthorized access even if password is compromised.
- Enhances user confidence and reduces fraud risk.

Practical Scenario — Token Authentication in Microservices

Example

- Microservice A requests data from Service B using JWT.
- Token contains user ID, expiration, and permission claims.
- Service B verifies token signature and expiration before granting access.
- Supports horizontal scaling without maintaining server sessions.

Lesson

- Tokens enable secure, stateless, and scalable authentication.
- Reduces overhead of managing centralized session stores.

Authentication Methods Comparison

Summary

- Passwords: simple, but vulnerable if alone.
- MFA: stronger security with additional user effort.
- Tokens: stateless, ideal for distributed and microservices architectures.
- Best practice: combine methods based on system criticality.
- Risk-based authentication: adapt security depending on context.

Example

- Banking: MFA with password + OTP.
- Cloud APIs: short-lived token-based authentication.
- Enterprise apps: password + device-based token for sensitive workflows.

Practical Tips for Authentication

Recommendations

- Use strong, unique passwords and enforce policies.
- Enable MFA for all critical systems.
- Use short-lived tokens for APIs and microservices.
- Educate users about phishing, social engineering, and device security.
- Monitor login activity and detect anomalies in real-time.

Industrial Examples

- Cloud platforms: AWS, Azure, GCP use MFA and token-based authentication.
- Banking apps: OTP + biometric verification.
- Microservices: JWT with expiration, signature verification, and claim-based access.

Authorization — Introduction

Key Concepts

- Authorization determines what an authenticated user is allowed to do.
- Complements authentication: verifies permissions rather than identity.
- Methods: Role-Based Access Control (RBAC), Attribute-Based Access Control (ABAC).
- Industrial relevance: enterprise apps, cloud platforms, microservices, IoT systems.
- Supports regulatory compliance: GDPR, HIPAA, PCI-DSS.

Why it matters

- Prevents unauthorized actions even after login.
- Reduces risk of insider threats and privilege abuse.
- Ensures fine-grained access management in large systems.

Role-Based Access Control (RBAC)

Key Concepts

- Access is granted based on user roles.
- Roles represent a set of permissions (e.g., Admin, Manager, Employee).
- Simplifies management of large user groups.
- Easier to audit than individual permissions.

Example

- HR system: Admin can create/delete users, Employee can only view own profile.
- Cloud platform: Manager can approve requests, Developer can deploy apps but not manage users.

RBAC — Practical Example

Scenario

- Employee logs into corporate portal.
- System checks assigned role: "HR Manager".
- Permissions include adding new employees and viewing payroll.
- Unauthorized action (e.g., delete financial report) is blocked.

Lesson

- RBAC enforces consistent access policies.
- Easy to update roles when responsibilities change.

Attribute-Based Access Control (ABAC)

Key Concepts

- Access is based on user attributes, resource attributes, and environment conditions.
- Allows fine-grained, context-aware access decisions.
- Example attributes: role, department, location, time, device type.
- More flexible than RBAC for dynamic environments.

Example

- Employee in "Finance" department accessing payroll only during office hours.
- Cloud API allows access only from company VPN or trusted device.

ABAC — Practical Example

Scenario

- User requests access to sensitive document.
- Policy checks attributes: user role, clearance level, location, request time.
- If all conditions met, access is granted; otherwise denied.
- Supports dynamic, conditional access decisions in real-time.

Lesson

- ABAC enables flexible and context-aware authorization.
- Useful in microservices, cloud, and hybrid environments.

RBAC vs ABAC — Comparison

Comparison

- RBAC: simpler, role-centric, easier to manage in static environments.
- ABAC: more flexible, attribute-centric, supports dynamic conditions.
- RBAC suitable for predictable workflows, ABAC for conditional and fine-grained control.
- Can combine RBAC + ABAC for hybrid access management.

Industrial Example

- Cloud IAM: RBAC for roles, ABAC for resource tags and context.
- Enterprise apps: RBAC for departments, ABAC for location/time-based restrictions.

Authorization Challenges

Key Issues

- Role explosion in large RBAC systems.
- Policy complexity in ABAC (attribute combinations can grow exponentially).
- Conflicts between roles and attributes.
- Auditing and compliance tracking.

Solution / Example

- Use policy management tools and automated auditing.
- Implement hybrid RBAC+ABAC to reduce complexity.
- Example: Cloud IAM dashboards for RBAC + ABAC enforcement.

Authorization in Microservices

Key Concepts

- Each service validates access independently.
- Centralized policy servers or distributed token claims can be used.
- Ensures least privilege principle across services.
- Example: JWT claims include roles/permissions for each microservice.

Industrial Example

- Netflix microservices: tokens carry roles/permissions for API access.
- Enterprise SaaS: RBAC + ABAC policies enforced per service endpoint.

Practical Tips for Authorization

Recommendations

- Keep role and attribute definitions clear and minimal.
- Regularly audit permissions and policies.
- Apply the principle of least privilege.
- Use centralized logging and monitoring for access events.
- Combine RBAC and ABAC for dynamic, scalable systems.

Industrial Examples

- Cloud IAM (AWS, Azure, GCP) integrates RBAC + ABAC policies.
- Banking: role-based access + conditional approval workflows.
- Enterprise SaaS: attribute-driven dynamic access control.

Authorization Summary

Key Takeaways

- Authorization ensures correct access after authentication.
- RBAC: role-based, simple, predictable.
- ABAC: attribute-based, flexible, context-aware.
- Industrial systems often combine RBAC + ABAC for optimal security.
- Proper authorization reduces insider threats and enforces least privilege.

Practical Examples

- Cloud platforms: conditional access via roles and attributes.
- Microservices: token-based permissions with claims validation.
- Enterprise apps: dynamic approval workflows based on policies.

Secure Communication — Introduction

Key Concepts

- Secure communication ensures data confidentiality, integrity, and authenticity over networks.
- Protects against eavesdropping, tampering, and impersonation.
- Methods: SSL/TLS protocols, VPNs, encrypted tunnels.
- Industrial relevance: web applications, APIs, IoT, microservices.

Why it matters

- Prevents data leaks and cyberattacks during transmission.
- Ensures compliance with regulations (PCI-DSS, HIPAA).
- Maintains trust in online services and enterprise systems.

SSL/TLS — Fundamentals

Key Concepts

- SSL (Secure Sockets Layer) and TLS (Transport Layer Security) encrypt data between client and server.
- TLS is the modern, secure successor to SSL.
- Provides confidentiality, integrity, and authentication.
- Uses asymmetric cryptography for key exchange, symmetric encryption for data, and message authentication codes (MACs) for integrity.

Example / Practical Use

- HTTPS websites encrypt browser-server communication.
- Email servers use TLS to secure SMTP, IMAP, and POP3 connections.

TLS Handshake — How It Works

Handshake Steps

- Client Hello: client proposes TLS version, cipher suites, and sends a random value.
- Server Hello: server selects version, cipher, sends its certificate.
- Key Exchange: client and server agree on a session key (via RSA/ECDHE).
- Session Established: symmetric encryption starts for data transmission.

Practical Example

- Browsing https://bank.com triggers TLS handshake.
- Certificate verification ensures you are connecting to the legitimate server.

Certificates and Trust

Key Concepts

- Certificates issued by Certificate Authorities (CAs) verify server identity.
- Public key in certificate used for asymmetric encryption during handshake.
- Certificate chain ensures trust from root CA to server certificate.
- Self-signed certificates used for testing; not trusted publicly.

Practical Example

- Browsers validate HTTPS certificates.
- Invalid or expired certificates trigger warnings to users.

Symmetric Encryption in TLS

Key Concepts

- Once handshake completes, symmetric key used for data encryption.
- Faster than asymmetric encryption; suitable for large payloads.
- Common algorithms: AES-128, AES-256, ChaCha20.
- Ensures data confidentiality and performance.

Practical Example

- File transfer between client-server over TLS is encrypted using AES.
- Online banking transactions are protected this way.

Integrity in TLS

Key Concepts

- Message Authentication Codes (MACs) detect tampering.
- HMAC (Hash-based MAC) is commonly used.
- Protects against man-in-the-middle (MITM) attacks.
- Ensures data received is exactly what was sent.

Example / Practical Scenario

- Online payment API verifies HMAC signature for request integrity.
- Blockchain nodes verify integrity of transmitted blocks using hashing and MACs.

Secure Channels Beyond TLS

Concepts

- VPNs create secure tunnels over public networks.
- SSH provides encrypted remote access to servers.
- IPSec secures IP communication between endpoints.
- Application-level encryption: end-to-end encryption in messaging apps.

Examples

- Corporate VPN for remote workers.
- Secure file transfer via SFTP (SSH File Transfer Protocol).
- Messaging apps like Signal and WhatsApp using end-to-end encryption.

Common Issues in Secure Communication

Challenges

- Expired or misconfigured certificates.
- Weak cipher suites or deprecated protocols.
- MITM attacks if certificate validation skipped.
- Key compromise or leakage.

Solutions / Best Practices

- Enforce TLS 1.2 or 1.3 and strong cipher suites.
- Regular certificate renewal and validation.
- Implement HSTS and certificate.
- Secure key storage and rotation policies.

Practical Scenario — Web Application

Example

- User logs into e-commerce site over HTTPS.
- TLS handshake establishes secure channel.
- Sensitive data (password, payment info) encrypted with AES.
- HMAC ensures integrity; server authenticates with certificate.

Lesson

- TLS ensures confidentiality, integrity, and trust in online transactions.

Secure Communication Summary

Key Takeaways

- TLS/SSL encrypts data and ensures integrity between endpoints.
- Certificates establish trust.
- Symmetric encryption used for data, asymmetric for key exchange.
- Secure channels extend TLS with VPNs, SSH, IPSec, and application-level encryption.
- Proper configuration prevents common attacks and ensures regulatory compliance.

Industrial Examples

- HTTPS websites, banking apps, cloud APIs.
- Remote access via VPN or SSH.
- Messaging platforms with end-to-end encryption.

Trust Models in Distributed Systems

Key Concepts

- Trust models define how entities verify identities and establish confidence.
- PKI (Public Key Infrastructure) provides a framework for key management and verification.
- Certificate Authorities (CAs) issue, revoke, and validate certificates.
- Trust chains ensure that certificates are valid and signed by recognized authorities.

Practical Examples

- HTTPS websites rely on PKI to establish secure connections.
- Enterprise systems use internal CAs for internal service authentication.

Security Threats in Distributed Systems

Common Threats

- Man-in-the-middle (MITM) attacks on insecure channels.
- Replay attacks: resending captured messages.
- Insider attacks: malicious or compromised nodes.
- Denial of Service (DoS) targeting services or nodes.

Practical Examples

- Blockchains use consensus to prevent double-spending (replay attacks).
- Microservices validate tokens to avoid impersonation.
- Cloud services monitor unusual traffic patterns to detect DoS attacks.

PKI and Certificate Authorities in Practice

Concepts

- Public/private key pairs enable secure authentication and encryption.
- Certificates bind public keys to entity identities.
- CAs sign certificates; trust is propagated through chains.

Practical Examples

- Email servers (SMTP over TLS) verify certificates for secure delivery.
- Cloud APIs use client certificates for mutual TLS authentication.
- Internal microservices authenticate each other via PKI-issued certificates.

Mitigating Security Threats

Strategies

- Encrypt communications using TLS/SSL.
- Implement strong authentication and token validation.
- Monitor and log suspicious activities.
- Use consensus mechanisms in blockchain to prevent double-spending.

Practical Examples

- Cloud systems enforce MFA and short-lived API tokens.
- Blockchain nodes validate transactions using cryptographic signatures.
- Microservices implement rate limiting to prevent DoS attacks.

Practical Applications of Security in Distributed Systems

Key Use Cases

- Blockchain: secure, tamper-proof transactions using cryptography and trust models.
- Microservices: token-based authentication, encrypted communication between services.
- Cloud Security: certificate-based API authentication, TLS for data in transit, secure multi-tenant environments.

Examples

- Bitcoin and Ethereum: PKI and cryptography for transaction integrity.
- Enterprise cloud apps (AWS, Azure) using mutual TLS for service-to-service communication.

Questions?

Thank you for your attention!

References

Suggested References

1. M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., Version 4.02, February 2024.
2. G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, *Distributed Systems: Concepts and Design*, 5th ed., Addison-Wesley, 2012.
3. N. Lynch, *Distributed Algorithms*, 2nd ed., Morgan Kaufmann, 1996.
4. R. E. Bryant and D. O'Hallaron, *Computer Systems: A Programmer's Perspective*, 4th ed., Pearson, 2024.