Mastering System Design

Section 10: Security in System Design

Reliability- Section Agenda

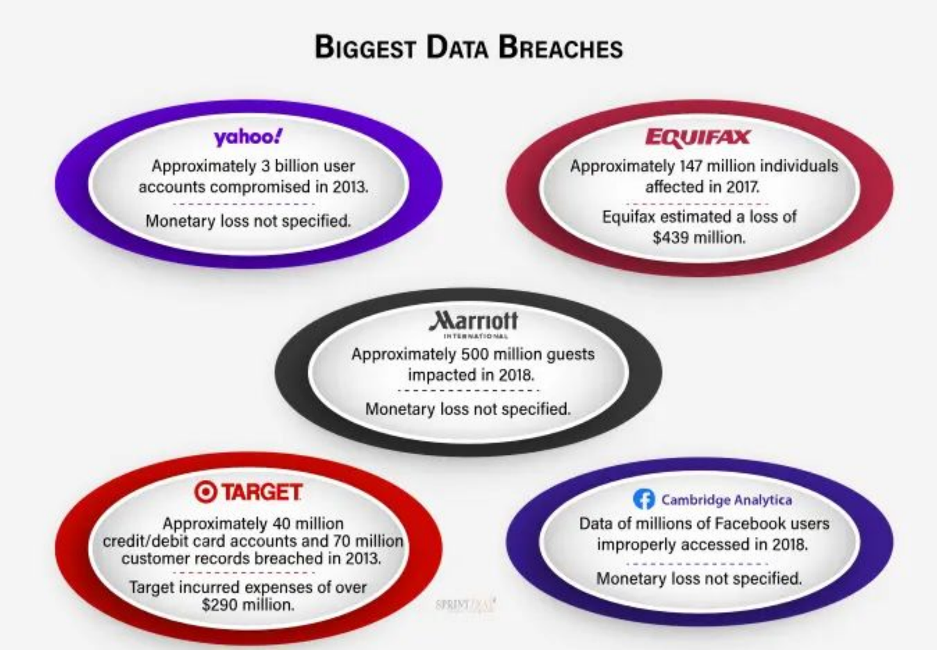
1. Introduction to Security in System Design
2. Authentication & Authorization
3. Data Protection & Secure Communication
4. Network & Infrastructure Security
5. Summary and Recap: Designing Secure Distributed Systems

Introduction to Security in System Design

Security in System Design

Why Security Matters in System Design

- Security is a non-functional requirement but critical for:
  - User trust
  - Data protection
  - System reliability
- Real-world examples: Data breaches (Equifax, Facebook, etc.)



BIGGEST DATA BREACHES

yahoo!
Approximately 3 billion user accounts compromised in 2013.
Monetary loss not specified.

EQUIFAX
Approximately 147 million individuals affected in 2017.
Equifax estimated a loss of $439 million.

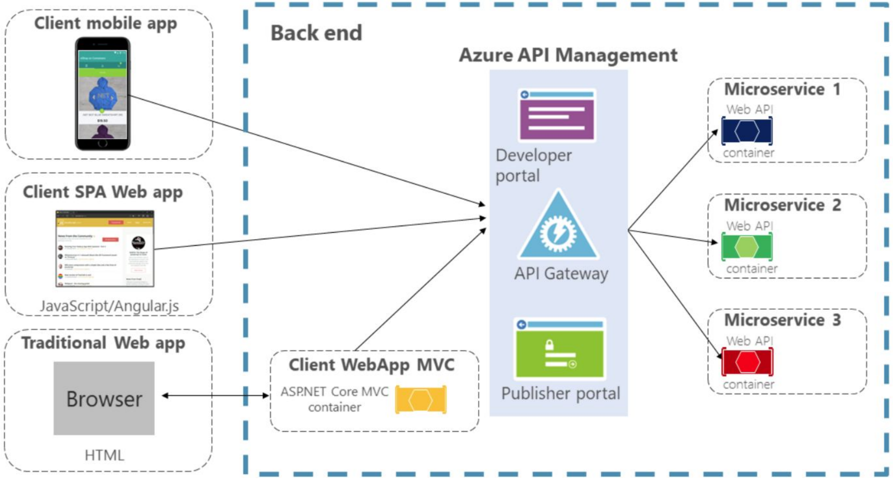
Marriott INTERNATIONAL
Approximately 500 million guests impacted in 2018.
Monetary loss not specified.

TARGET
Approximately 40 million credit/debit card accounts and 70 million customer records breached in 2013.
Target incurred expenses of over $290 million.

Cambridge Analytica
Data of millions of Facebook users improperly accessed in 2018.
Monetary loss not specified.

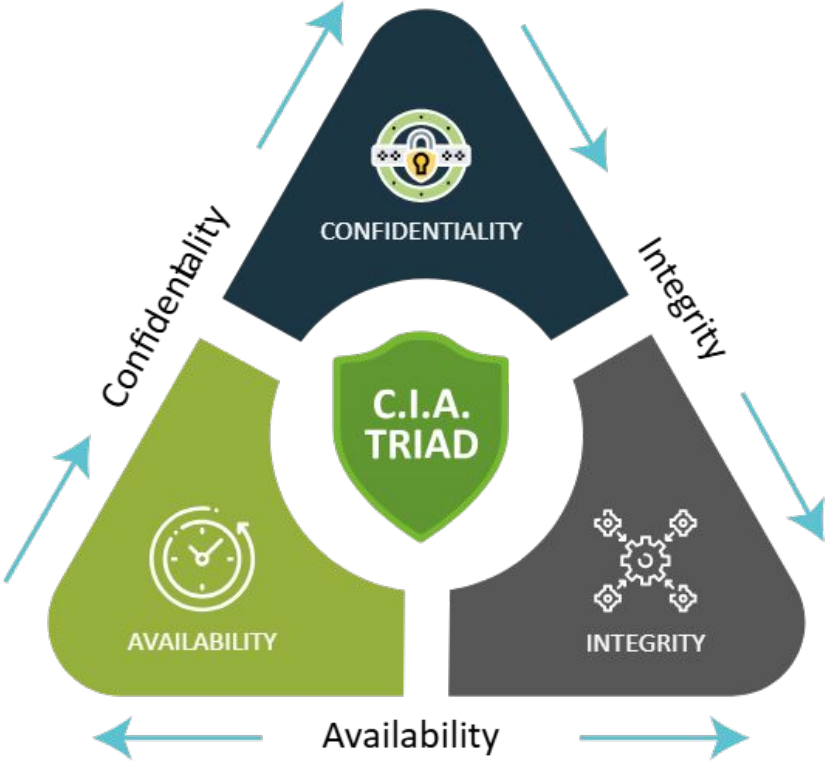
What is Security in Distributed Systems?

- Distributed systems: more entry points, more vulnerabilities
- Security considerations:
  - Data in transit & at rest
  - Authentication, Authorization
  - Secure APIs & endpoints
  - Node and network-level protection



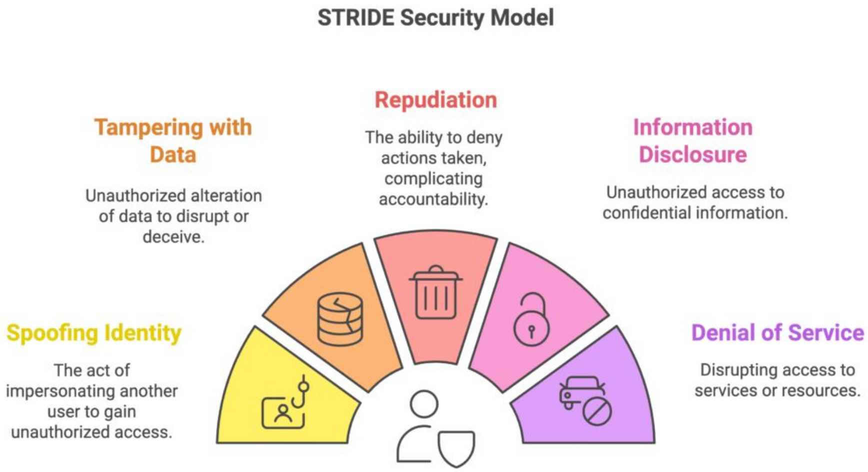
The CIA Triad: Core of System Security

- Confidentiality: Prevent unauthorized access
- Integrity: Prevent data tampering
- Availability: Ensure system uptime and access



Threat Modeling: Understanding Your Adversary

- Define potential attackers and what they want
- Consider:
  - Attack surface
  - Entry points
  - Assets to protect
- Tools: STRIDE model (Spoofing, Tampering, Repudiation, Info disclosure, Denial of service, Elevation of privilege)



STRIDE Security Model

Spoofing Identity
The act of impersonating another user to gain unauthorized access.

Tampering with Data
Unauthorized alteration of data to disrupt or deceive.

Repudiation
The ability to deny actions taken, complicating accountability.

Information Disclosure
Unauthorized access to confidential information.

Denial of Service
Disrupting access to services or resources.

Common Attack Vectors

- How attackers get in:
  - Insecure APIs
  - Misconfigured servers
  - Poor authentication
  - Open ports / services

Common Attacks

- DDoS (Distributed Denial of Service)
  - Flooding system with traffic to disrupt service
  - Target: Availability
  - Mitigation:
    - Rate limiting
    - Traffic scrubbing (e.g., Cloudflare)
    - Autoscaling and failover strategies
- Man-in-the-Middle (MITM) Attack
  - Attacker intercepts communication
  - Targets: Confidentiality & Integrity
  - Protection:
    - HTTPS (TLS)
    - Certificate pinning
    - VPNs
- Injection Attacks (e.g., SQL Injection)
  - Attacker sends malicious input to execute unwanted commands
  - Impacts: Data integrity, confidentiality
  - Mitigation:
    - Input validation
    - Parameterized queries
    - WAF (Web Application Firewall)
- Spoofing Attacks
  - Impersonation of another user/system
  - Types: IP spoofing, email spoofing, DNS spoofing
  - Defense:
    - Multi-factor authentication
    - Token-based auth
    - IP whitelisting

Security in the Software Development Lifecycle (SDLC)

- Embed security from the start (Shift Left)
- Phases:
    - Requirements: Threat modeling
    - Design: Secure architecture
    - Development: Secure coding
    - Testing: Security tests, fuzzing
    - Deployment: Secrets management
    - Maintenance: Patch management

Best Practices

- Adopt security by design
- Use encryption (TLS, at-rest)
- Harden infrastructure (firewalls, VPCs)
- Validate inputs and sanitize outputs
- Monitor and log activity

Interview Questions – Security Focused

- How would you design a secure authentication system for a distributed application?
- Explain how the CIA triad applies to system design.
- What are common security threats in a microservices architecture, and how would you mitigate them?
- How would you protect your system from a DDoS attack?
- What role does TLS/HTTPS play in system security?
- And how would you implement certificate management at scale?
- How can you ensure secure data storage in a cloud-based system?
- What is threat modeling and how would you incorporate it into your design process?

Summary and Key Takeaways

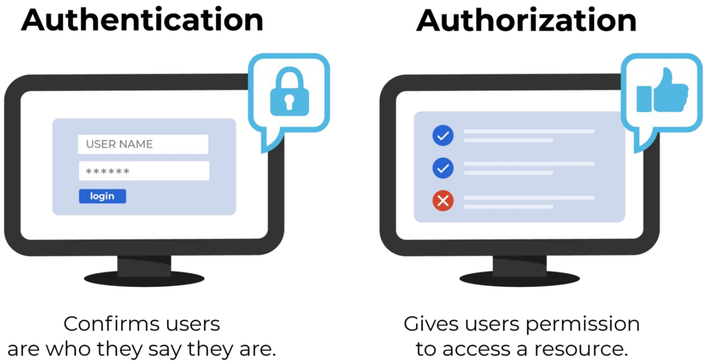
- Security = ongoing process, not a one-time task
- Understand the CIA triad
- Identify and address threat vectors
- Build secure systems from the ground up
- What’s next:
  - Authentication & Authorization

Authentication & Authorization

Security in System Design

Introduction to Authentication vs. Authorization

- Authentication: Verifying who the user is (identity verification).
- Authorization: Granting the user permission to access specific resources based on their identity.
- Key difference: Authentication is about who you are, while Authorization is about what you can do.

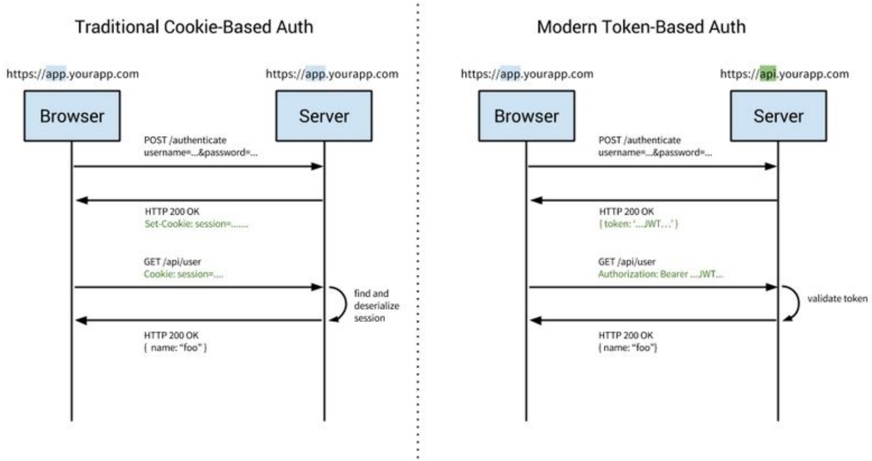


Common Authentication Methods

- Basic Authentication: Simple user and password-based authentication.
- OAuth2: Delegated access protocol, allowing third-party services to access user data without exposing credentials.
- OpenID Connect: An identity layer built on top of OAuth2 for authentication, often used for single sign-on (SSO).
- JWT (JSON Web Tokens): Token-based authentication, commonly used in stateless applications and APIs.

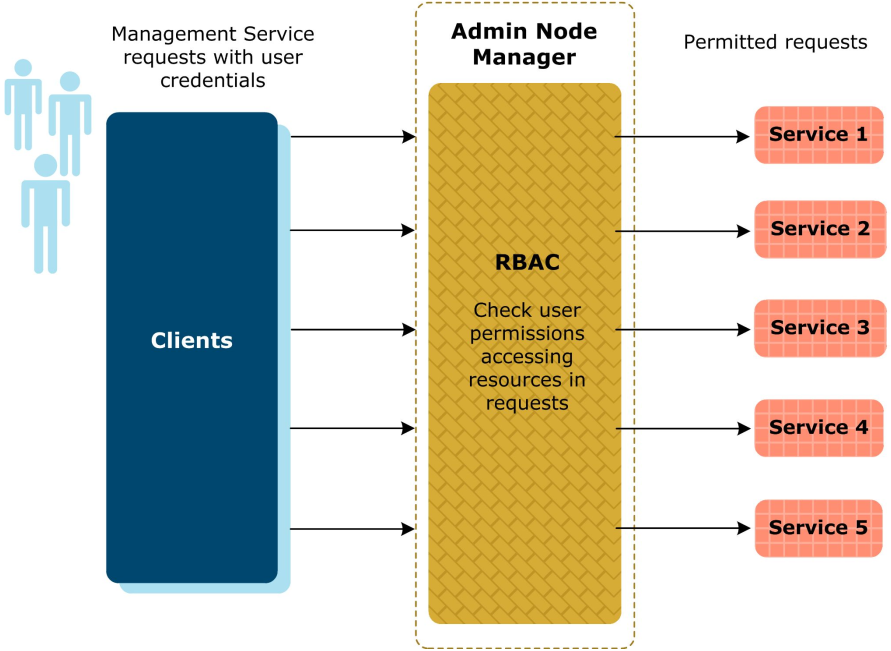
Session-based vs. Token-based Authentication

- Session-based Authentication: Server-side storage of session data, typically using cookies.
  - Pros: Easy to implement, works well with traditional web applications.
  - Cons: Scalability challenges in distributed systems.
- Token-based Authentication: Stateless, where tokens (e.g., JWT) are used to authenticate the user.
  - Pros: Scalable, decouples backend systems.
  - Cons: Requires secure token storage and handling.



Access Control Models

- RBAC (Role-Based Access Control): Assigns permissions based on user roles (e.g., Admin, User, Viewer).
  - Simpler to manage but less flexible.
- ABAC (Attribute-Based Access Control): Grants access based on attributes (e.g., department, project).
  - More fine-grained but more complex.
- DAC (Discretionary Access Control): Owner of a resource defines access controls.
- MAC (Mandatory Access Control): Access control decisions are made by a central authority based on security policies.


Management Service requests with user credentials
Admin Node Manager
Permitted requests
Clients
RBAC
Check user permissions accessing resources in requests
Service 1
Service 2
Service 3
Service 4
Service 5

SSO (Single Sign-On) and Identity Federation

- SSO: A user authentication process that allows a user to access multiple applications with a single set of credentials.
  - Benefits: User convenience, reduced password fatigue.
- Identity Federation: A system where multiple identity providers (e.g., Google, Facebook) can be used to authenticate users across different platforms.
  - Benefits: Seamless user experience across different services, reduces need for creating multiple accounts.

Interview Questions for Authentication & Authorization

- Why is JWT commonly used for stateless authentication in distributed systems?
- How does OAuth2 differ from OpenID Connect?
- Explain the difference between RBAC and ABAC. Which one is more suitable for a highly dynamic environment?
- What are the advantages of using Single Sign-On (SSO) in a system?
- What are some potential security concerns with token-based authentication?

Summary and Key Takeaways

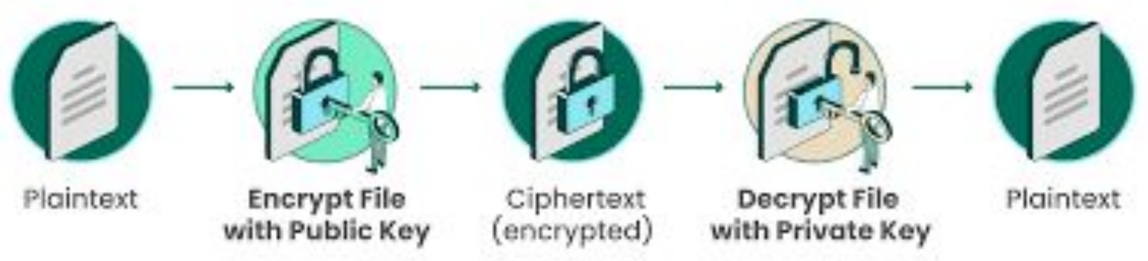
- Authentication verifies the user’s identity, while authorization defines what the user can access.
- Common authentication methods include Basic Auth, OAuth2, OpenID Connect, and JWT.
- Access control models include RBAC, ABAC, DAC, and MAC.
- SSO and identity federation simplify user authentication across multiple platforms.
- What’s next:
    - Data Protection & Secure Communication

Data Protection & Secure Communication

Security in System Design

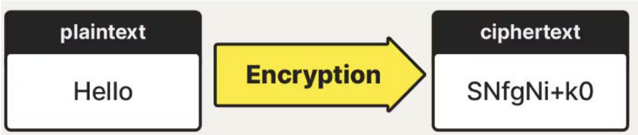
Why Data Protection Matters

- Growing threats: breaches, man-in-the-middle attacks, data leaks
- Regulations (GDPR, HIPAA, etc.)
- User trust and system reliability



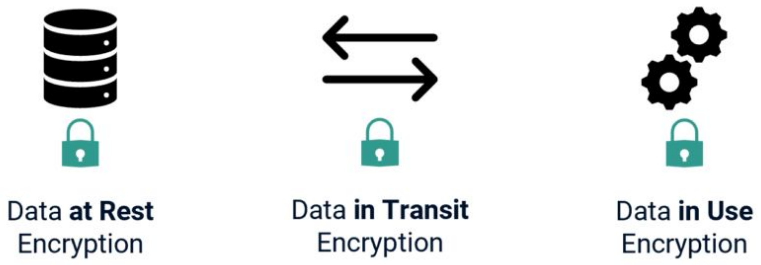
Let’s talk Encryption

- What is encryption?
- Plaintext → Ciphertext → Decryption
- Key = secret for encoding/decoding



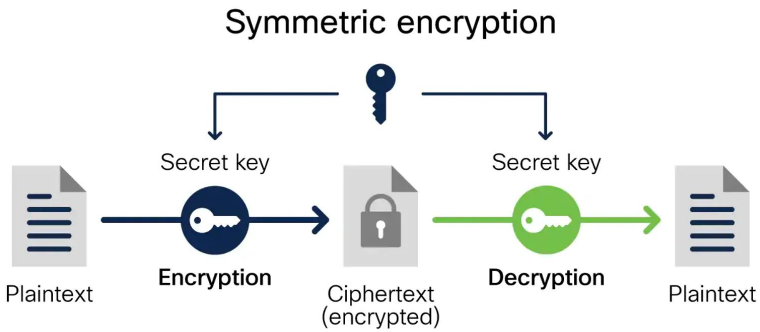
Encryption at Rest and Transit

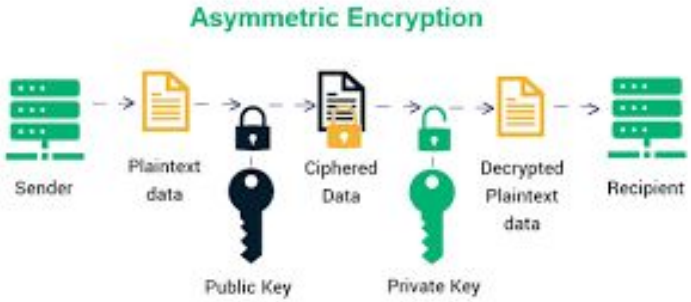
- Encryption at Rest
    - Protects stored data (disks, databases, backups)
    - Common techniques: Full-disk encryption, database-level encryption
    - Use cases: cloud storage, user files, logs
- Encryption in Transit
    - Secures data during transmission (e.g., HTTP request/response)
    - TLS/SSL protocols enable secure communication
    - Must-have for APIs, user sessions


Data **at Rest** Encryption
Data **in Transit** Encryption
Data **in Use** Encryption

Symmetric vs. Asymmetric Encryption

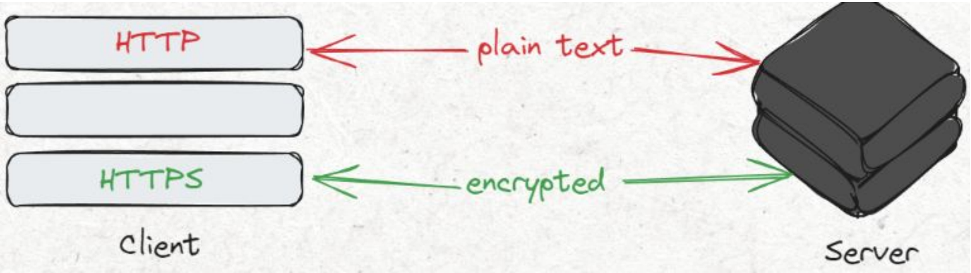
- Symmetric: one key (fast, used for large data)
- Asymmetric: public/private key pair (secure key exchange)
- Often used together (e.g., TLS handshake)





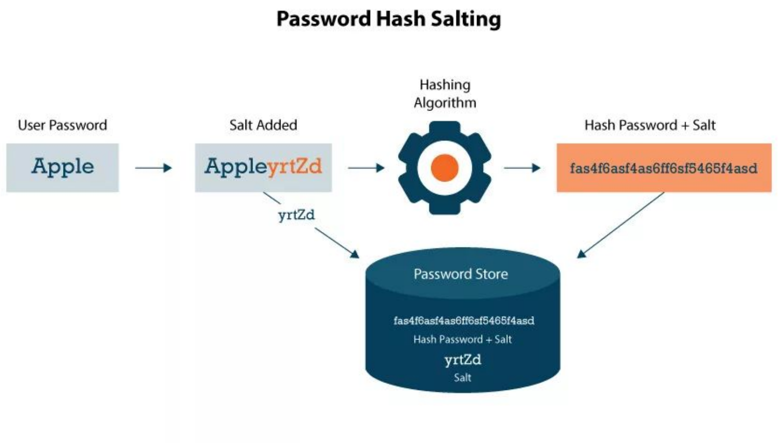
TLS/SSL and HTTPS

- HTTPS = HTTP over TLS
- Ensures confidentiality, integrity, and authenticity
- TLS handshake: key exchange + cipher negotiation



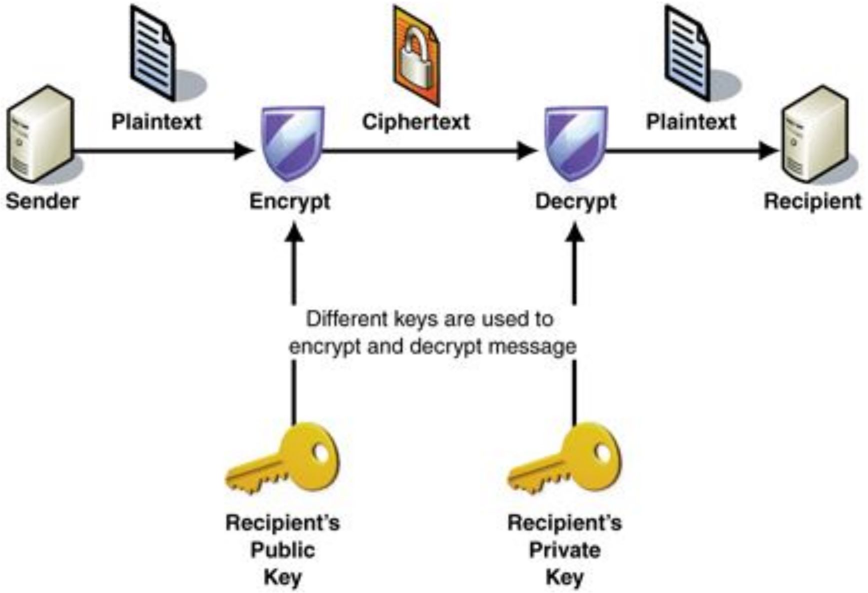
Hashing and Salting Passwords

- Hashing = one-way transformation
- Use for storing passwords (not reversible)
- Salting: add random data to prevent rainbow table attacks



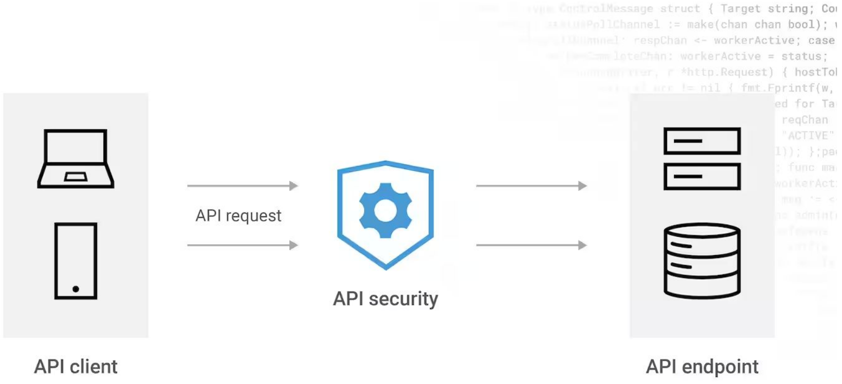
Public Key Infrastructure (PKI)

- PKI = system for managing digital certificates and keys
- Role of Certificate Authorities (CAs)
- Digital signatures & certificate chains



Secure API Communication

- Use HTTPS for all API traffic
- Auth tokens (JWT, OAuth)
- Rate limiting, IP whitelisting, mutual TLS for sensitive APIs



Interview Questions

- What is the difference between hashing and encryption?
- Why is asymmetric encryption slower than symmetric?
- How does PKI build trust online?
- What is the concept behind securing Data at rest and Data in Motion?

Summary & Best Practices

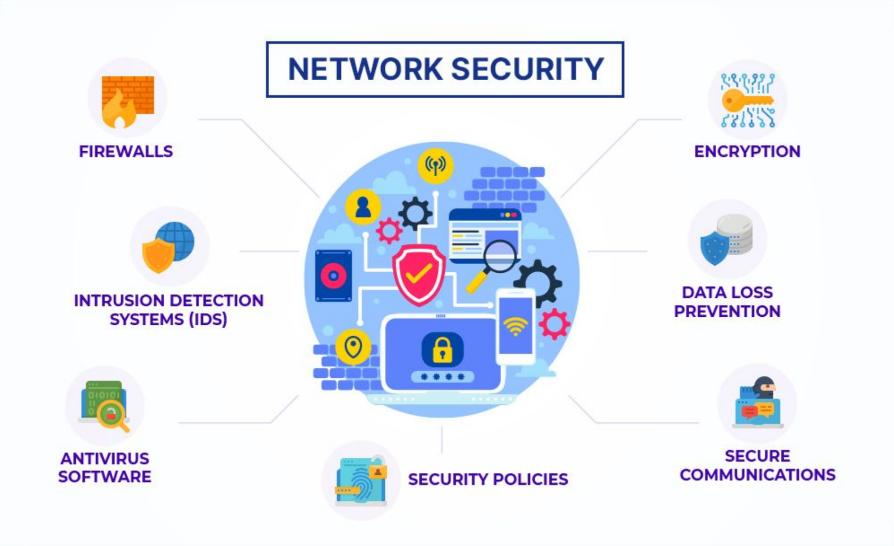
- Encrypt data both at rest and in transit
- Use hashing + salting for passwords
- Leverage TLS, PKI, HTTPS for secure communication
- Harden APIs with secure design patterns
- What’s next:
    - Network & Infrastructure Security

Network & Infrastructure Security

Security in System Design

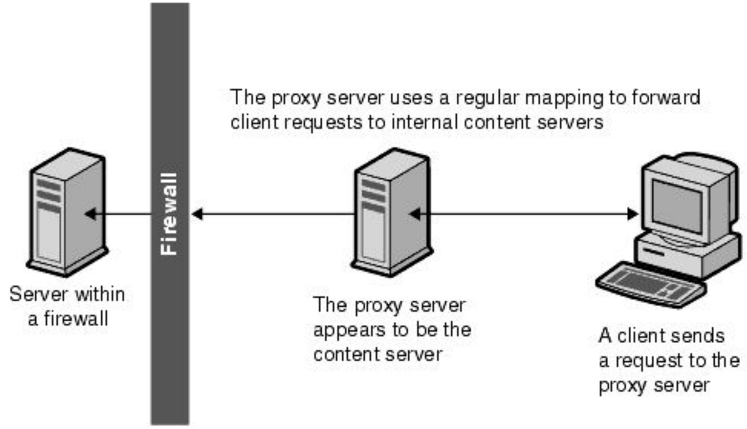
Why Network Security Matters

- External threats (DDoS, intrusion, IP spoofing)
- Internal risks (misconfiguration, lateral movement)
- Increasing cloud-native adoption = more exposure
- Reliability & user trust



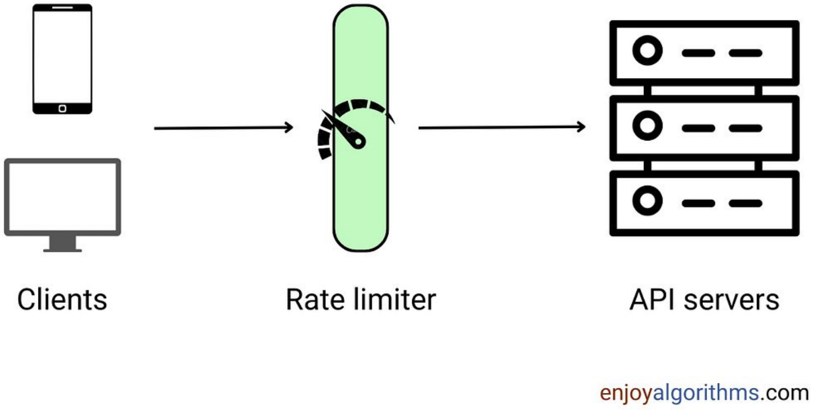
Firewalls and Reverse Proxies

- Firewalls: filter traffic based on IP, port, protocol
- Types: Network-based, Host-based, Cloud firewalls
- Reverse Proxies: route, mask backend identity, add security
- Examples: NGINX, AWS ALB


Firewall
The proxy server uses a regular mapping to forward client requests to internal content servers
Server within a firewall
The proxy server appears to be the content server
A client sends a request to the proxy server

Rate Limiting, Throttling, IP Filtering

- Rate limiting: per-user, per-IP request caps
- Throttling: graceful degradation under load
- IP Filtering: allow/block lists
- Helps protect APIs & backend systems from abuse

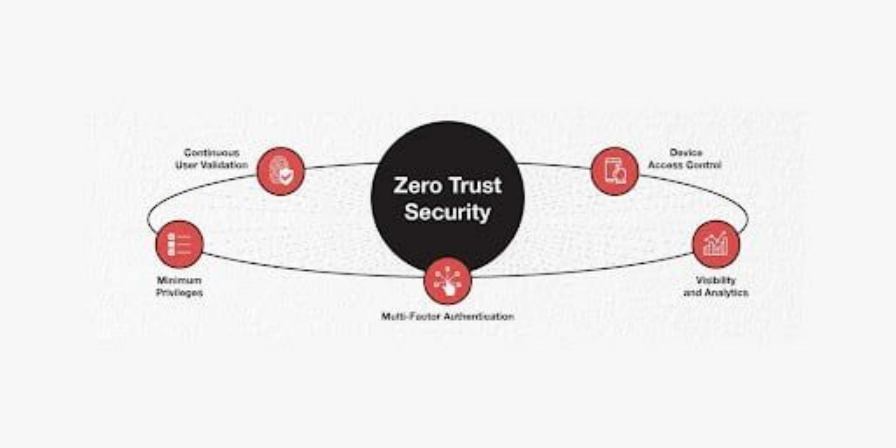


Network Segmentation & Isolation

- Split network into zones: DMZ, internal, DB, etc.
- Limit lateral movement
- Use firewalls, subnets, private VLANs
- Cloud: use VPCs, security groups, NACLs

Zero Trust Security Model

- “Never trust, always verify”
- Auth every request, even inside the network
- Microservices: mutual TLS, strict access control
- Applies to cloud, hybrid, and on-prem setups



Securing Cloud Environments

- Shared responsibility model
- Key aspects:
  - IAM
  - Encryption (EBS, S3, RDS)
  - Audit logging (CloudTrail, Stackdriver)
- CSPM (Cloud Security Posture Management) tools

Securing Serverless & Containerized Workloads

- Serverless: control IAM roles, timeouts, API gateway access
- Containers: image scanning, runtime hardening, least privilege
- Tools: AWS Lambda + API Gateway, Docker + Kubernetes security tools (OPA, Falco)

Security in Microservices

- Service-to-service auth (JWT, mTLS)
- API Gateway security: validation, auth, rate limits
- Service mesh: fine-grained control, TLS, policies (Istio, Linkerd)

Common Vulnerabilities (OWASP Top 10)

- A quick glance at top risks:
  - Injection
  - Broken Auth
  - Sensitive Data Exposure
  - Security Misconfig
  - XSS, CSRF, SSRF, etc.
- Relevance in web apps & microservices
- Focus on awareness + mitigation

Interview Questions - Network Security

- What is the difference between a firewall and a reverse proxy?
- How does rate limiting protect your backend services?
- Explain the Zero Trust security model and its importance.
- How would you secure a containerized workload in Kubernetes?
- What are some common OWASP Top 10 vulnerabilities and how do you mitigate them?
- How does a service mesh help enforce security in microservices?
- What’s the role of IAM in cloud security?

Summary & Best Practices

- Use firewalls and segment your network
- Apply rate limiting, IP filters, reverse proxies
- Embrace Zero Trust and encrypt everywhere
- Secure APIs, services, containers, and functions
- Stay alert to OWASP Top 10
- What’s next:
  - Summary and Recap: Designing Secure Distributed Systems

Section Summary: Security in System Design

- Security in Distributed Systems
  - CIA Triad: Confidentiality, Integrity, Availability
  - Common Threats: DDoS, MITM, Injection, Spoofing
- Authentication & Authorization
  - Auth Methods: OAuth2, OpenID, JWT
  - Access Control: RBAC, ABAC, SSO
- Data Protection & Secure Communication
  - Encryption: At rest & in transit
  - TLS/SSL, HTTPS, Hashing, Salting
- Network & Infrastructure Security
  - Firewalls, Reverse Proxies, Rate Limiting
  - Network Segmentation, Zero Trust
  - Cloud & Microservices Security: IAM, Encryption, API Gateway
- What’s next:
  - The System Design Blueprint