DNS Outline - Jaish Khan

1. Firewalls

> A software/hardware device that monitors and controls data entering and leaving a network. It acts as a security barrier between a trusted internal network and an untrusted external one, such as the Internet. By applying predefined rules, it decides whether to allow, block, encrypt, or log network traffic.

Limitations of a Firewall ⟶ It cannot protect against:

1. Attacks bypassing it like trusted organizations and services (SSL/SSH).
2. Internal threats (malicious employees).
3. Viruses/Trojans from reaching the machine via email, file sharing, or direct download nor their transfer.

Firewall Characteristics ⟶ These are the design goals:

1. All network traffic must pass through the firewall.
2. Only authorized traffic (defined by the local security policy) will be allowed to pass.
3. The firewall itself must be immune to penetration.

Techniques to Enforce Security by Firewall

1. **Service Control** ⟶ *Which Internet services can be accessed?*

2. **Direction Control** → *Which direction, a particular service requests can be accessed?*
3. **User Control** → *Which user is attempting to access what service?*
4. **Behavior Control** → *How a particular service is being used?*

Positives of a Firewall	Negatives of a Firewall
User authentication (Require user login, allowing control and tracking).	**Traffic bottlenecks** (Forcing all traffic through can cause congestion).
Auditing and logging (Can keep and analyze activity information).	**Single point of failure** (Incorrect configuration or unavailability can block all traffic).
Anti-Spoofing (Detecting when the source of network traffic is being "spoofed").	**Increased management responsibilities** (Adds complexity to network management and troubleshooting).

Hardware vs Software Firewall

- **Hardware Firewalls** → Protect an entire network, implemented on the router level, usually more expensive, harder to configure.
- **Software Firewalls** → Protect a single computer, usually less expensive, easier to configure.

1.1. Firewall Types

Type	Layer	Pros	Cons	How It Works
Packet Filtering (Stateless)	3, 4	Fast, simple, low cost	No context, hard for complex rules	Inspects each incoming and outgoing packet individually. Decisions are based on predefined rules involving IP address, port number, protocol, and network interface. It does not keep track of previous packets, so each one is evaluated on its own.

Type	Layer	Pros	Cons	How It Works
Stateful Inspection	3, 4	More secure, still fast	Harder to configure, no user authentication	Tracks the state of active connections (TCP handshakes). Maintains a state table of outbound requests and allows inbound traffic only if it matches an expected response in that table. This blocks unsolicited or spoofed traffic.
Application-Level Gateway (Proxy)	7	High control and visibility	Slower, separate proxy needed per service	Acts as an intermediary for specific applications like web or email. The user connects to the proxy, which then makes the request on the user's behalf. The proxy checks, filters, and may modify traffic before forwarding. It fully understands and enforces protocol rules.
Circuit-Level Gateway	4, 5	More secure than packet filters	Limited to session-level, no deep inspection	Monitors TCP or UDP session establishment. Once a session is validated, it relays traffic without inspecting packet contents. It maintains a virtual circuit for each allowed session and is commonly used with SOCKS proxies.

2. Malicious Software

> **Malware** → Software intentionally designed to harm computer systems by causing undesired actions. It spreads through email attachments, infected floppy disks, downloading/exchanging corrupted files, and embedded computer games.

Type	What It Is and What It Does
Virus	A piece of code that attaches itself to other programs and spreads by modifying them. It can damage or change data, slow down systems, and use up memory or processing power. Often includes hidden harmful functions.

Type	What It Is and What It Does
Worm	A program that spreads on its own through networks without attaching to other files. It can delete files, steal information (like passwords), and overload systems, causing crashes. Some worms can also carry viruses.
Trojan Horse	A program that looks safe but has hidden harmful actions. When run, it might install other malware, open backdoors for hackers, or delete data.
Logic Bomb	Hidden code that stays inactive until a certain condition is met (like a specific date or file change). When triggered, it can delete or damage files.
Trapdoor	A hidden way to get into a system, skipping security checks. It might be added by developers for testing, but attackers can misuse it. Hard to detect and block.
Spyware	Software that secretly watches what you do on your computer and sends the info elsewhere. It can steal personal data (like passwords), track your activity, and even install more malware.
Hoax	Fake virus alerts, often sent through chain messages or emails. They scare people into forwarding them, which clogs networks and spreads false info.

Details about Viruses

- Viruses can reside in boot sectors, memory, disk (applications/data), libraries, compilers, debuggers, and even anti-virus programs.
- Effective Viruses are hard to detect and destroy, spread widely, are easy to create and re-infect, and ideally platform-independent.
- Virus Detection involves checksums using cryptographic hash functions.
- Virus Identification relies on the fact that each virus is a unique program with unique object code that inserts itself in a deterministic manner, providing a signature based on its object code pattern and insertion location.
- **Types of Viruses**:
  1. *Boot Sector Virus* ⟶ It infects boot sector of systems and stays there. Activates when the machine boots.
  2. *File Virus* ⟶ It infects files of a program. Activates when that program is run.
  3. *Email Virus* ⟶ It is spread by infected attachments (macro viruses) triggered upon opening or even viewing (with scripting features).
- **Phases of Viruses**:
  1. *Dormant* (Waiting for trigger)
  2. *Triggering* (By an Event to execute code)
  3. *Execution* (Running code)

4. *Propagation* (Spreading to programs/disks)

Types of Trojan Horses

1. *Remote-Access Trojan* → Lets the hacker completely control your computer from somewhere else.
2. *Data-Sending Trojan* → Sends your information back to the hacker (like keyloggers that record everything you type).
3. *Destructive Trojan* → Only meant to delete and ruin your files and is often hard for virus scanners to find.
4. *Denial-of-service (DoS) attack Trojan* → Uses the power of many infected computers to attack another computer, overwhelming it so it stops working.

3. Encryption

> **Symmetric Encryption** → Uses the same key for both encryption and decryption. It is also known as conventional, secret-key, or single-key encryption. It involves an encryption algorithm, a secret key shared by the sender and receiver, and the plaintext and ciphertext. **Examples**: *AES*, *DES* and *Triple DES* algorithms.

> **Asymmetric Encryption** → Uses different keys for encryption and decryption. It is also known public-key cryptography. It involves an encryption algorithm, a secret unique key that is not shared (private key), a secret key that is shared across the network (public key), and the plaintext and ciphertext. Depending on the application, the sender uses either their private key, the recipient's public key, or both to perform cryptographic functions. **Examples:** *RSA*, *Diffie-Hellman* and *ECC* algorithms.

3.1. Ways to Distribute Keys

Symmetric and Asymmetric encryption differ in how keys are managed.

- **Symmetric encryption** requires both parties to share a common secret key, making secure key distribution a major challenge—any weakness in this process can compromise the entire system. To reduce the risk of key compromise, a strong key distribution mechanism and frequent key exchanges are essential.
- **Asymmetric encryption** uses a key pair: a public key, which is openly shared, and a private key, which is kept secret by its owner. Since the private key is generated and stored locally, it doesn’t need to be distributed like a symmetric key. However, the main challenge in asymmetric encryption is verifying the authenticity of the public key to ensure it actually belongs to the claimed owner.

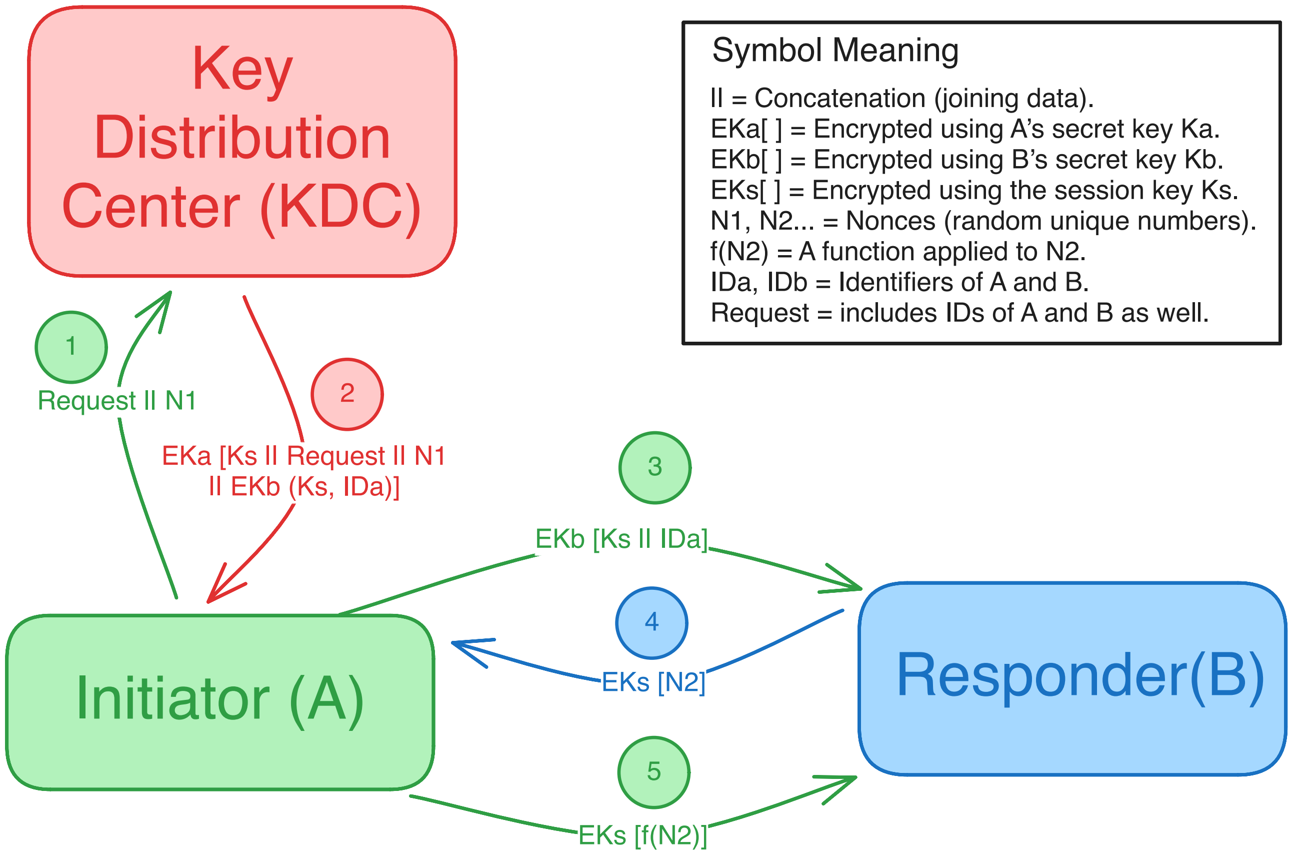
✓ **Ways to distribute Symmetric keys**

For two parties, A and B, several methods exist for distributing symmetric keys:

1. **Physical Delivery By One Party** → Party A selects a key and physically transfers it to Party B.
2. **Physical Delivery By a Trusted Third Party** → A trusted third party selects the key and physically delivers it to both Party A and Party B.
3. **Using Pre-existing Keys** → If A and B have communicated previously and share an older key, this key can be used to encrypt and exchange a new session key.
4. **Relay via a Trusted Third Party** → If both A and B have secure communication channels with a trusted third party, C (often a KDC), C can securely relay a session key between them.

Key Distribution Scenario

Purpose: To securely distribute a session key (`KS`) between two parties (A and B) using a trusted third party — the KDC — and prevent attacks like replay attacks or impersonation.



Symbol Meaning
II = Concatenation (joining data).
EKa[ ] = Encrypted using A's secret key Ka.
EKb[ ] = Encrypted using B's secret key Kb.
EKs[ ] = Encrypted using the session key Ks.
N1, N2... = Nonces (random unique numbers).
f(N2) = A function applied to N2.
IDa, IDb = Identifiers of A and B.
Request = includes IDs of A and B as well.

What's Happening

Step	What Happens
1. `A → KDC`	`A` sends a request to the Key Distribution Center (`KDC`) to communicate with `B`. This request includes a nonce N_1 (a random unique value) and their identities ID_A and ID_B. The communication between `A` and the `KDC` is encrypted using a permanent secret key K_A shared only between `A` and the `KDC`.
2. `KDC → A`	`KDC` generates a unique, one-time session key (K_S). It then sends a response to `A`, encrypted using K_A. This encrypted response contains: The session key (K_S) and the original request message (including the nonce) for verification. A separate message is also sent containing K_S and ID_A, encrypted using a permanent secret key shared only between `B` and the KDC (K_B).
3. `A → B`	`A` receives the `KDC` response, decrypts it using K_A to get K_S, and stores it. `A` then forwards the portion of the `KDC` response encrypted with K_B (which contains K_S and ID_A) to `B`.
4. `B → A`	`B` decrypts `A`'s message to get K_S, sends a nonce N_2 to `A`, encrypted K_S to verify that `A` knows the session key.
5. `A → B`	`A` receives the encrypted nonce N_2, decrypts it using K_S, performs a predefined function on it (like incrementing it by 1) and sends the result back to `B` (again encrypted with K_S). `B` decrypts the response and verifies if the function was correctly applied to its original nonce.

Step 4 and 5 is done to ensure the message received by B is not a replay of a previous communication.

✓ Ways to distribute Asymmetric keys

1. **Public Announcement** ⟶ Users share their public keys openly, such as broadcasting to others. The main risk is **forgery**—anyone can create a key pretending to be someone else and use it until detected.
2. **Public Directory** ⟶ A more secure method where users **register their public keys with a trusted directory**. It lists name–public key pairs, allows key updates, and is published regularly. Still, the directory can be **tampered with or forged** if not properly protected.
3. **Public-Key Authority** ⟶ This improves security by requiring users to **interact with a trusted authority** to get public keys. It works like a directory but adds real-time, secure access. Users must **know the authority's public key** to use it safely.
4. **Public-Key Certificates** ⟶ Certificates **bind a public key to a user's identity** and may include validity dates or usage rights. They remove the need for real-time access to an authority and allow secure key exchange offline.

How to obtain a Certificate?

A **certificate** is a digital document that contains a public key and identity information, and it's digitally signed by a trusted Certificate Authority (CA). This signature ensures the certificate's authenticity and integrity. Anyone can verify a certificate using the public key of the CA. A certificate for user A, signed by CA, is written as **CA⟨A⟩**.

To get a certificate, a user simply requests it from a CA. The CA verifies the user's identity and then issues the signed certificate. Only the CA can create or modify certificates, and because they're digitally signed, they cannot be forged.

Certificate Revocation

Certificates have a period of validity but may need to be revoked before expiry due to reasons such as:

1. The user's private key being compromised.
2. The user no longer being certified by the CA.
3. The CA's certificate being compromised.

CAs maintain a list of revoked certificates called the Certificate Revocation List (CRL), and users should check certificates against the CA's CRL.

3.2. Levels of Encryption

There are 2 levels of encryption: **Link Encryption** and **End-to-End Encryption**.

- **Best practice** is Combine link encryption (to hide headers and traffic patterns between nodes) with end-to-end encryption (to protect content and ensure authenticity).

Link Encryption	End-to-End Encryption
Encryption happens at each communication link (between every switch/router).	Encryption is done by the sender and decrypted only by the final receiver.
Each link must decrypt and re-encrypt the data, exposing it briefly at each hop.	Data remains encrypted throughout the journey, only decrypted at the endpoint.
Encrypts the entire packet (headers + payload), hiding routing info from outsiders.	Only the message content is encrypted; headers stay visible for routing.

Link Encryption	End-to-End Encryption
Requires many keys—each pair of nodes must share a unique key.	Requires only the sender and receiver to share a key.
Operates at lower OSI layers (1 or 2).	Operates at higher OSI layers (3, 4, 6, or 7).
Helps protect against **traffic monitoring between points** (e.g., routers, switches).	Helps protect **message content** and authenticate the sender.

4. Digital Signatures

Digital Signature is like an electronic fingerprint (identity card) for digital data, using the sender's private key (asymmetric encryption) and hashing.

- It allows anyone with the sender's public key to guarantee who sent it (*authentication*), that the data hasn't been changed (*integrity*) and that the sender can't deny sending it (*nonrepudiation*).
- It doesn't stop others from reading the message (*confidentiality*) but is very efficient as instead of encrypting the entire message with the private key (which would be computationally expensive), it encrypts only the small hash code of the message, ensuring integrity based on the properties of the hash function.
- It enhances message authentication by providing proof of origin and preventing tampering.
- Algorithms used for digitals signatures: **RSA**, **DSS** and **ECC**.

Mechanism

Digital signatures use asymmetric encryption and hashing. The typical process involves:

1. **Hashing:** The sender uses a secure hash function (e.g., SHA-1) to create a short message digest (hash code) of the data.
2. **Signing:** The sender then encrypts this hash code using their private key. This encrypted hash code constitutes the digital signature.
3. **Transmission:** The digital signature is appended to the message or transmitted separately.
4. **Verification:** The recipient performs the following steps:
   - Uses the sender's public key to decrypt the received digital signature, recovering the original hash code.
   - Independently computes the hash code of the received message using the same hash function used by the sender.

- Compares the computed hash code with the recovered hash code. If they match, the signature is valid.

5. Message Authentication

Message authentication is a critical procedure in secure communication, enabling communicating parties to verify the **Data Integrity** (that the message has not been changed during transmission) and **Source Authenticity** (that the message originated from the claimed sender). There are two ways:

Aspect	MAC	Hash Functions
Key Usage	Needs a shared secret key.	Doesn't use a key by default; extra steps are needed for authentication.
Purpose	Checks if the message is unchanged **and** came from the right sender.	Checks if the message is unchanged; sender identity is verified using other methods.
How It Works	1. Create MAC = F(K, M) 2. Send Message + MAC 3. Receiver checks it using the key.	Hash the message, then either: - Encrypt it (with a key) - Sign it (with private key) - Add a secret value.
What It Ensures	✓ Message integrity ✓ Sender authenticity ✓ Message order	✓ Integrity ✓ Authenticity (if encrypted or signed) ✓ Non-repudiation (if signed)
Common Types	**HMAC** (uses hash), **CMAC** (uses cipher), **CCM** (combo of CTR + CMAC)	**SHA-1**, **SHA-2** (224/256/384/512), **HMAC** when used with secret key
Security Depends On	The strength of the MAC method and keeping the key secret.	The security of the hash function and protecting the output (digest).
Used In	IPsec, SNMPv3, TLS	TLS, Wi-Fi (802.11i), WTLS, PGP, S/MIME, DKIM, IPsec/IKE

5.1. Message Authentication Codes (MACs)

> A MAC is a small block of data generated using a secret key and attached to the message. This method requires that the communicating parties share a common secret key.

Process:

1. *Generation*: `A` (with the message M and a shared secret key K_{AB}) calculates the MAC (MAC_M) using some function F.
2. *Transmission*: `A` sends the message and the calculated MAC (MAC_M) to `B`.
3. *Verification*: `B` performs the same calculation using the received message and the shared secret key to get MAC'. It then compares the received calculated MAC (MAC_M) with its' calculated MAC (MAC').

5.2. Message Authentication using Hash Functions

A hash function takes a variable-size message as input and produces a fixed-size message digest. Unlike MACs, hash functions **do not use a secret key**. Therefore, to achieve message authentication with a hash function, the integrity of the message digest itself must be ensured.

Key Properties of Hash Functions for Message Authentication:

1. **Works with any input size –** Can take messages of any length.
2. **Gives fixed-size output –** Always produces the same length hash, no matter the input size.
3. **Quick to compute –** Easy and fast to generate the hash for any message.
4. **Preimage Resistance (One-way) –** It's practically impossible to guess the original message from its hash. This is essential when using secret values.
5. **Second Preimage Resistance –** Hard to find a *different* message with the same hash as a given one. Important when using encrypted hashes.
6. **Collision Resistance –** Very hard to find *any two different messages* with the same hash. This protects against advanced attacks.

A hash function with the first 5 properties is called a **weak hash function** and one with all 6 is a **strong hash function**.

× Extra Information

HMAC (Hash-based Message Authentication Code) ⟶ Combines both methods by deriving a MAC from a hash function. Its design prioritizes using existing hash functions without modification, easy replacement of the hash function, performance preservation, simple key handling, and security analysis based on the underlying hash function's properties.

Message Authentication with either method (MAC or Hash) does not provide message confidentiality as the message itself is transmitted in plaintext.

Scenarios where message authentication (MAC or Hash) is preferred:

- *Broadcasting* → Sending the same message to multiple destinations where only one needs to verify authenticity.
- *Heavy Load* → Situations where a receiver cannot afford to decrypt all incoming messages but needs to authenticate selectively.
- *Plaintext Program Authentication* → Verifying the integrity of an executable program without requiring repeated decryption.

6. Ciphers

6.1. Stream Cipher

Stream ciphers are symmetric encryption algorithms that encrypt data one bit or byte at a time. A **key** is used to generate a **keystream** with a pseudorandom bit generator. This keystream is combined with the plaintext using XOR to produce ciphertext. Decryption works the same way—XORing the ciphertext with the same keystream to get the original data. A minimum `128-bit` key is recommended to prevent brute-force attacks.

Stream ciphers are fast and lightweight, making them ideal for real-time data like network communication. However, reusing the same key for multiple messages is dangerous, as attackers can uncover information by XORing the ciphertexts. **RC4** is a well-known example of a stream cipher.

6.2. Block Cipher

Block ciphers are symmetric key encryption algorithms that encrypt fixed-size blocks of data (instead of single bits like Stream Ciphers). Among the most well-known are **DES**, **3DES** and **AES**, each evolving to improve security and efficiency.

-	DES (Data Encryption Standard)	3DES (Triple DES)	AES (Advanced Encryption Standard)
Introduced in	1977	1985	2001
Block Size	64 bits	64 bits	128 bits
Key Length	56 bits	168 bits (3 separate keys)	128, 192, or 256 bits
Rounds	16	48 (3 sets of DES)	10 rounds (for 128-bit key)
Design	Feistel structure	DES done 3 times (Encrypt-Decrypt-Encrypt)	Substitution and permutation steps

-	DES (Data Encryption Standard)	3DES (Triple DES)	AES (Advanced Encryption Standard)
Key Handling	Makes 16 subkeys from the main key	Runs DES 3 times with 3 keys	Key is expanded for use in each round
Encryption Steps	16 steps using subkeys	3 steps: Encrypt, Decrypt, Encrypt	Starts with key mix, then 9 rounds of substitution, shifting, mixing, and key mixing
Decryption Steps	Same as encryption but reverse order	Same steps, but in reverse	Uses reverse operations with the key schedule in reverse
Security	Weak (can be cracked by brute force)	Stronger, harder to crack	Very strong with long keys; also resists cryptanalysis
Speed	Fast	Slower than DES	Very fast
Used Today	Only in old systems	Still used, but being replaced	Common and recommended today
Complexity	Simple	More complex than DES	More complex than both
Common Uses	Data verification	Banking, secure emails, VPNs	Wi-Fi (WPA2/3), VPNs, file protection, and more

× Modes of Operations for Block Ciphers

There are five modes of operation for block ciphers to accommodate various applications, with Electronic Codebook (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), and Counter (CTR) being the most significant.