Practical – 10

AIM: Write a program to generate digital signature using Hash code.

- **Code**

```
#include <iostream>
#include <string>
using namespace std;
int generateHash(string message) {
    int hash = 0;
    for (char ch : message) {
        hash += (int)ch;
    }
    return hash % 1009;
}
int signHash(int hash, int privateKey) {
    return (hash * privateKey) % 1009;
}
int main() {
    string message;
    int privateKey = 17;
    cout << "Enter the message to sign: ";
    getline(cin, message);
    int hash = generateHash(message);
    int digitalSignature = signHash(hash, privateKey);
    cout << "\nOriginal Message: " << message << endl;
    cout << "Hash Value: " << hash << endl;
    cout << "Digital Signature (Simulated): " << digitalSignature << endl;
    return 0;
}
```

- **Output**

```
PS D:\CSE\CSE_github\SEM 6\CNS> cd "d:\CSE\CSE_github\SE
Enter the message to sign: Hello!! RIAUHAS

Original Message: Hello!! RIAUHAS
Hash Value: 146
Digital Signature (Simulated): 464
PS D:\CSE\CSE_github\SEM 6\CNS>
```

Practical – 11

AIM: Case Study on Kerberos.

1. Introduction

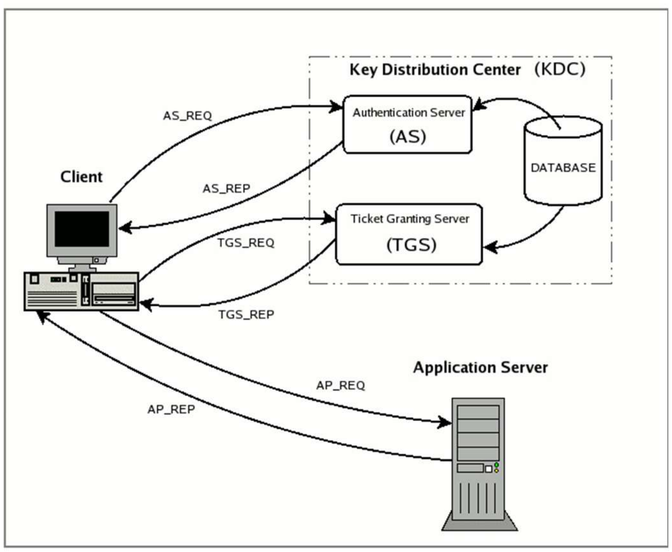
Kerberos is a **network authentication protocol** designed to provide strong authentication for client-server applications using secret-key cryptography. Developed at **MIT** as part of **Project Athena**, it allows entities communicating over a non-secure network to prove their identity securely.

Kerberos is widely used in enterprise environments, including **Microsoft Windows**, where it's the default authentication protocol in Active Directory.

2. Why Kerberos?

Traditional systems used passwords for authentication, which are vulnerable to interception and replay attacks. Kerberos addresses this with **tickets** and **time-stamped authentication**, avoiding the direct transmission of passwords.

3. Kerberos Architecture



Kerberos relies on a centralized **Key Distribution Center (KDC)**, which is split into two main parts:

- **Authentication Server (AS)**
- **Ticket Granting Server (TGS)**

Components:

- **Client/User**: The person or process requesting access.
- **KDC**: The trusted third party responsible for issuing authentication and service tickets.
- **AS**: Verifies the user's credentials and provides a TGT (Ticket Granting Ticket).

- **TGS**: Issues a service ticket using the TGT.
- **Service Server**: The final destination that the user wants to access (e.g., file server, web app).

4. How Kerberos Works (Step-by-step)

Step 1: Authentication Request

The client sends a request to the **Authentication Server**.

Step 2: Ticket Granting Ticket (TGT)

The AS verifies the credentials and sends back a **TGT**, encrypted using the user's password-derived key.

Step 3: Requesting Access

The client uses the TGT to request access to a particular service from the **TGS**.

Step 4: Service Ticket

TGS validates the TGT and sends a **service ticket**, which the client can present to the **Service Server**.

Step 5: Access Granted

The client presents the ticket to the server, and if valid, access is granted.

5. Real-World Use Case

Kerberos is the default authentication method in:

- **Microsoft Active Directory (Windows)**
- **Hadoop clusters** for securing resource manager and data nodes
- **SSH in enterprise Linux environments**

6. Advantages of Kerberos

1. **Strong security** with mutual authentication
2. **No passwords** transmitted over the network
3. **Time-based tickets** prevent replay attacks
4. **Scalable** for large networks
5. **Widely supported** across OS and services

7. Disadvantages of Kerberos

- **Single point of failure** – If KDC is down, no one can authenticate
- **Requires synchronized time** between clients and servers
- **Key management complexity** for large setups
- **Initial setup** can be complicated for beginners

8. Conclusion

Kerberos is a battle-tested and powerful authentication protocol that is especially effective in distributed systems. With its robust ticketing system and mutual authentication, it has become a critical piece of the security puzzle in many enterprise and academic systems. Understanding Kerberos helps in grasping how real-world secure communication works behind the scenes.

Practical – 12

AIM: Case Study on Kerberos.

1. Introduction

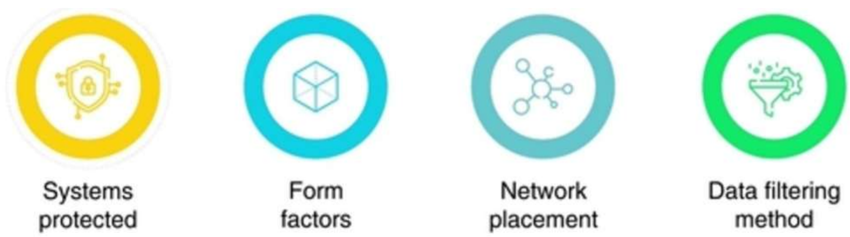
A **firewall** is a **network security system** that monitors and controls incoming and outgoing network traffic based on predetermined security rules. It acts as a barrier between a trusted internal network and untrusted external networks, like the Internet.

Firewalls are essential for both personal computers and enterprise networks to protect against unauthorized access, malware, and other cyber threats.

2. Why Use a Firewall?

- Prevent unauthorized access
- Block malicious traffic and attacks
- Filter content and data leakage
- Reduce attack surface
- Control how internal users access external resources

3. Types of Firewalls



I. Packet Filtering Firewall

- Works at **Network Layer (Layer 3)**
- Inspects source/destination IP, port, and protocol
- **Fast** but limited in context (can't detect complex attacks)

II. Stateful Inspection Firewall

- Keeps track of the **state of active connections**
- Makes decisions based on both **header information and connection state**
- More secure than simple packet filters

III. Application Layer Firewall (Proxy Firewall)

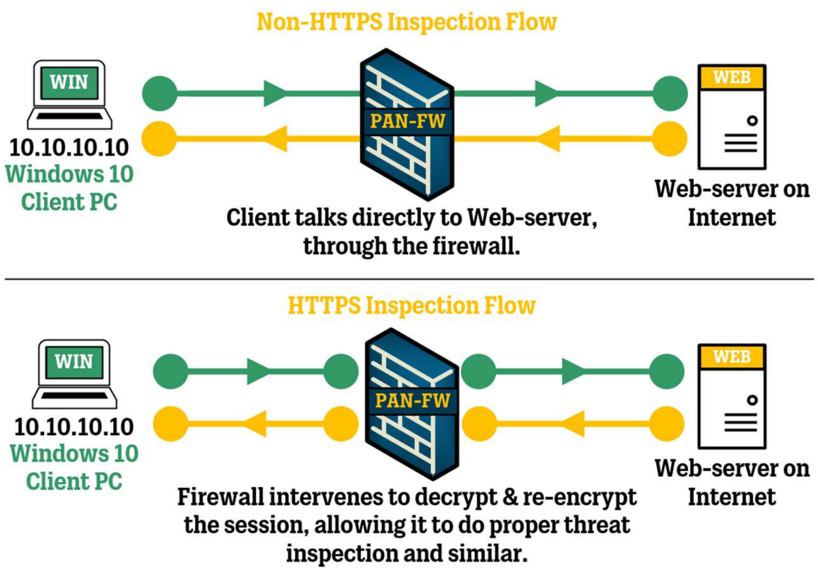
- Operates at the **Application Layer (Layer 7)**
- Can filter HTTP, FTP, DNS traffic, etc.
- Can inspect **application payloads**

IV. Next-Generation Firewall (NGFW)

Combines traditional firewall features with:

- **Deep packet inspection**
- **Intrusion prevention**
- **Antivirus**
- **Content filtering**
- **User identity tracking**

5. How Firewalls Work



- All traffic enters and exits through the firewall
- The firewall applies **predefined rules** to each packet or request
- Based on rules, traffic is either **allowed, blocked, or flagged**
- Can be **hardware**, **software**, or a **hybrid**

6. Real-World Use Case: Corporate Firewall

Scenario: A mid-sized company wants to protect its internal network from the internet.

Solution:

- Deploy a **Stateful + Application Layer Firewall**
- Allow ports 80 (HTTP) and 443 (HTTPS)
- Block social media, torrent, and gaming traffic during office hours
- Monitor outgoing traffic to detect data exfiltration

7. Advantages of Firewalls

- Provides **network perimeter security**
- Filters **unwanted traffic**
- Can **log, monitor**, and **alert** on suspicious activity
- Enforces **access control policies**
- Helps maintain **compliance** with data protection laws

8. Disadvantages of Firewalls

- Can’t protect against **internal threats**
- Not effective if misconfigured
- May cause **latency or bottlenecks**
- Some advanced attacks can bypass them (e.g., phishing, social engineering)

9. Firewall Best Practices

- Regularly **update and patch** firmware
- Implement **least privilege policies**
- **Monitor logs** and set alerts
- Conduct **regular audits** and **pen tests**
- Use **cloud-based firewalls** for remote environments

9. Conclusion

Firewalls are the **first line of defense** in network security. From personal use to enterprise-grade systems, firewalls help filter traffic, enforce policies, and prevent breaches. While they’re not a silver bullet, when combined with other security layers like IDS/IPS and antivirus, they play a critical role in building a secure network architecture.

Practical – 13

AIM: Study of MD5 hash function and implement the hash code using MD5.

- **What is MD5?**

  The **MD5 (Message Digest 5)** algorithm is a widely-used **cryptographic hash function** that produces a **128-bit (16-byte)** hash value. It was developed by **Ronald Rivest** in 1991.

  - It takes an input (or message) and returns a fixed-length hash value.
  - Commonly used for **integrity checking**, **digital signatures**, and **password hashing** (though not recommended for passwords anymore due to vulnerabilities).

- **Code**

```
#include <iostream>
#include <cstring>
#include <iomanip>
#include <sstream>
typedef unsigned int uint32;
class MD5 {
public:
    MD5() { reset(); }
    std::string digest(const std::string& str) {
        reset();
        update((const unsigned char*)str.c_str(), str.length());
        finalize();
        return toHex();
    }
private:
    uint32 a, b, c, d;
    uint32 msgLenLow, msgLenHigh;
    unsigned char buffer[64];
    uint32 block[16];
    bool finalized;
    void reset() {
        finalized = false;
        msgLenLow = msgLenHigh = 0;
        a = 0x67452301;
        b = 0xefcdab89;
        c = 0x98badcfe;
        d = 0x10325476;
    }
    static uint32 F(uint32 x, uint32 y, uint32 z) { return (x & y) | (~x & z); }
    static uint32 G(uint32 x, uint32 y, uint32 z) { return (x & z) | (y & ~z); }
    static uint32 H(uint32 x, uint32 y, uint32 z) { return x ^ y ^ z; }
    static uint32 I(uint32 x, uint32 y, uint32 z) { return y ^ (x | ~z); }
    static uint32 rotateLeft(uint32 x, int n) { return (x << n) | (x >> (32 - n)); }
    void step(uint32& w, uint32 x, uint32 y, uint32 z, uint32 data, uint32 s, uint32 ac, uint32
(*func)(uint32, uint32, uint32)) {
        w = w + func(x, y, z) + data + ac;
        w = rotateLeft(w, s) + x;
```

```
    }
    void transform(const unsigned char block[64]) {
        for (int i = 0; i < 16; ++i)
            this->block[i] = ((uint32)block[i * 4]) | ((uint32)block[i * 4 + 1] << 8) |
                             ((uint32)block[i * 4 + 2] << 16) | ((uint32)block[i * 4 + 3] << 24);
        uint32 A = a, B = b, C = c, D = d;
        step(A, B, C, D, this->block[0], 7, 0xd76aa478, F);
        step(D, A, B, C, this->block[1], 12, 0xe8c7b756, F);
        step(C, D, A, B, this->block[2], 17, 0x242070db, F);
        step(B, C, D, A, this->block[3], 22, 0xc1bdceee, F);
        step(A, B, C, D, this->block[4], 7, 0xf57c0faf, F);
        step(D, A, B, C, this->block[5], 12, 0x4787c62a, F);
        step(C, D, A, B, this->block[6], 17, 0xa8304613, F);
        step(B, C, D, A, this->block[7], 22, 0xfd469501, F);
        step(A, B, C, D, this->block[8], 7, 0x698098d8, F);
        step(D, A, B, C, this->block[9], 12, 0x8b44f7af, F);
        step(C, D, A, B, this->block[10], 17, 0xffff5bb1, F);
        step(B, C, D, A, this->block[11], 22, 0x895cd7be, F);
        step(A, B, C, D, this->block[12], 7, 0x6b901122, F);
        step(D, A, B, C, this->block[13], 12, 0xfd987193, F);
        step(C, D, A, B, this->block[14], 17, 0xa679438e, F);
        step(B, C, D, A, this->block[15], 22, 0x49b40821, F);
        a += A; b += B; c += C; d += D;
    }
    void update(const unsigned char* input, size_t length) {
        size_t index = (msgLenLow >> 3) & 0x3F;
        if ((msgLenLow += (uint32)(length << 3)) < (length << 3))
            msgLenHigh++;
        msgLenHigh += (uint32)(length >> 29);
        size_t partLen = 64 - index;
        size_t i = 0;
        if (length >= partLen) {
            memcpy(&buffer[index], input, partLen);
            transform(buffer);
            for (i = partLen; i + 63 < length; i += 64)
                transform(&input[i]);
            index = 0;
        }
        memcpy(&buffer[index], &input[i], length - i);
    }
    void finalize() {
        static unsigned char PADDING[64] = { 0x80 };
        if (finalized) return;
        unsigned char bits[8];
        for (int i = 0; i < 4; ++i) {
            bits[i] = (unsigned char)(msgLenLow >> (i * 8));
            bits[i + 4] = (unsigned char)(msgLenHigh >> (i * 8));
        }
        size_t index = (msgLenLow >> 3) & 0x3f;
        size_t padLen = (index < 56) ? (56 - index) : (120 - index);
        update(PADDING, padLen);
```

```
            update(bits, 8);
            finalized = true;
        }
        std::string toHex() const {
            std::ostringstream os;
            uint32 vals[4] = { a, b, c, d };
            for (int i = 0; i < 4; ++i)
                for (int j = 0; j < 4; ++j)
                    os << std::hex << std::setw(2) << std::setfill('0') << ((vals[i] >> (j * 8)) & 0xff);
            return os.str();
        }
    };
    int main() {
        MD5 md5;
        std::string input;
        std::cout << "Enter a message: ";
        std::getline(std::cin, input);
        std::string hash = md5.digest(input);
        std::cout << "MD5 Hash: " << hash << std::endl;
        return 0;
    }
```

- **Output**

```
PS D:\CSE\CSE_github\SEM 6\CNS> cd "d:\CSE\CSE_git
Enter a message: Hello RIAUHAS
MD5 Hash: d454a7d280959125ec94925fec8186a4
PS D:\CSE\CSE_github\SEM 6\CNS>
```

Practical – 14

AIM: Study of SHA-1 hash function and implement the hash code using SHA-1.

SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function that:

- Produces a **160-bit hash value** (40 hexadecimal characters)
- Is **deterministic**: same input gives same output
- Was **designed by the NSA**, published by NIST in 1995
- Takes any input and compresses it into a **fixed-length 160-bit hash**
- Is now considered **broken for secure cryptography** due to collision vulnerabilities, but still useful for understanding hash mechanics

Steps in SHA-1:

1. **Preprocessing**:
   - Message is padded to make its length a multiple of 512 bits.
   - Original message length is added in last 64 bits.
2. **Divide into 512-bit blocks**
3. **Initialize five 32-bit variables** (A, B, C, D, E)
4. **For each block**:
   - Expand the 16 words into 80
   - Run 80 rounds of hashing using functions & bitwise logic
   - Update A, B, C, D, E
5. **Output**: Final 160-bit hash (5 words concatenated)

- **Code**

```
#include <iostream>
#include <sstream>
#include <iomanip>
#include <cstring>
#include <vector>
typedef unsigned int uint32;
class SHA1 {
public:
    SHA1() { reset(); }
    std::string digest(const std::string &message) {
        reset();
        update((const unsigned char*)message.c_str(), message.length());
        finalize();
        return toHex();
    }
private:
    uint32 h0, h1, h2, h3, h4;
    std::vector<unsigned char> buffer;
```

```
uint64_t messageLength;
void reset() {
    h0 = 0x67452301;
    h1 = 0xEFCDAB89;
    h2 = 0x98BADCFE;
    h3 = 0x10325476;
    h4 = 0xC3D2E1F0;
    buffer.clear();
    messageLength = 0;
}
static uint32 rotateLeft(uint32 value, uint32 bits) {
    return (value << bits) | (value >> (32 - bits));
}
void processBlock(const unsigned char block[64]) {
    uint32 w[80];
    for (int i = 0; i < 16; ++i)
        w[i] = (block[i * 4] << 24) |
               (block[i * 4 + 1] << 16) |
               (block[i * 4 + 2] << 8) |
               (block[i * 4 + 3]);
    for (int i = 16; i < 80; ++i)
        w[i] = rotateLeft(w[i - 3] ^ w[i - 8] ^ w[i - 14] ^ w[i - 16], 1);
    uint32 a = h0, b = h1, c = h2, d = h3, e = h4;
    for (int i = 0; i < 80; ++i) {
        uint32 f, k;
        if (i < 20) {
            f = (b & c) | (~b & d);
            k = 0x5A827999;
        } else if (i < 40) {
            f = b ^ c ^ d;
            k = 0x6ED9EBA1;
        } else if (i < 60) {
            f = (b & c) | (b & d) | (c & d);
            k = 0x8F1BBCDC;
        } else {
            f = b ^ c ^ d;
            k = 0xCA62C1D6;
        }
        uint32 temp = rotateLeft(a, 5) + f + e + k + w[i];
        e = d;
        d = c;
        c = rotateLeft(b, 30);
        b = a;
        a = temp;
    }
    h0 += a;
    h1 += b;
    h2 += c;
    h3 += d;
    h4 += e;
}
```

```
    void update(const unsigned char *data, size_t length) {
        messageLength += length * 8;
        buffer.insert(buffer.end(), data, data + length);
        while (buffer.size() >= 64) {
            processBlock(&buffer[0]);
            buffer.erase(buffer.begin(), buffer.begin() + 64);
        }
    }
    void finalize() {
        buffer.push_back(0x80);
        while ((buffer.size() + 8) % 64 != 0)
            buffer.push_back(0x00);
        for (int i = 7; i >= 0; --i)
            buffer.push_back((messageLength >> (i * 8)) & 0xFF);

        for (size_t i = 0; i < buffer.size(); i += 64)
            processBlock(&buffer[i]);
    }
    std::string toHex() const {
        std::ostringstream result;
        uint32 words[5] = { h0, h1, h2, h3, h4 };
        for (int i = 0; i < 5; ++i)
            result << std::hex << std::setw(8) << std::setfill('0') << words[i];
        return result.str();
    }
};
int main() {
    SHA1 sha1;
    std::string input;
    std::cout << "Enter a message: ";

    std::getline(std::cin, input);
    std::string hash = sha1.digest(input);
    std::cout << "SHA-1 Hash: " << hash << std::endl;
    return 0;
}
```

- Output

```
PS D:\CSE\CSE_github\SEM 6\CNS> cd "d:\CSE\CSE_github
Enter a message: Hello RIAUHAS
SHA-1 Hash: fcb6d54b77669d33e917a3e87676f34cfa5150d0
PS D:\CSE\CSE_github\SEM 6\CNS> 
```

Practical – 15

AIM: Write a program to implement transposition Encryption Technique

- **Code**

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;
vector<int> getOrder(string key) {
    vector<pair<char, int>> keyMap;
    for (int i = 0; i < key.length(); ++i)
        keyMap.emplace_back(key[i], i);
    sort(keyMap.begin(), keyMap.end());
    vector<int> order(key.length());
    for (int i = 0; i < key.length(); ++i)
        order[keyMap[i].second] = i;
    return order;
}
string encrypt(string message, string key) {
    int cols = key.length();
    vector<int> order = getOrder(key);
    int rows = (message.length() + cols - 1) / cols;
    vector<vector<char>> grid(rows, vector<char>(cols, 'X'));
    int k = 0;
    for (int i = 0; i < rows && k < message.length(); ++i)
        for (int j = 0; j < cols && k < message.length(); ++j)
            grid[i][j] = message[k++];
    string ciphertext = "";
    for (int o = 0; o < cols; ++o) {
        for (int j = 0; j < cols; ++j) {
            if (order[j] == o) {
                for (int i = 0; i < rows; ++i)
                    ciphertext += grid[i][j];
                break;
            }
        }
    }
    return ciphertext;
}
int main() {
    string message, key;
    cout << "Enter the plaintext message (no spaces): ";
    cin >> message;
    cout << "Enter the key (e.g., word or numbers): ";
    cin >> key;
    string encrypted = encrypt(message, key);
    cout << "Encrypted message: " << encrypted << endl;
    return 0;
```

}

- **Output**

```
PS D:\CSE\CSE_github\SEM 6\CNS> cd "d:\CSE\CSE_github\
Enter the plaintext message (no spaces): RIAUHAS
Enter the key (e.g., word or numbers): RJ
Encrypted message: IUAXRAHS
PS D:\CSE\CSE_github\SEM 6\CNS>
```