task-06

1. Symmetric vs Asymmetric Encryption

Symmetric Encryption uses a **single secret key** for both encryption and decryption. It is **fast** and suitable for encrypting large amounts of data but requires secure key sharing.
Example: AES

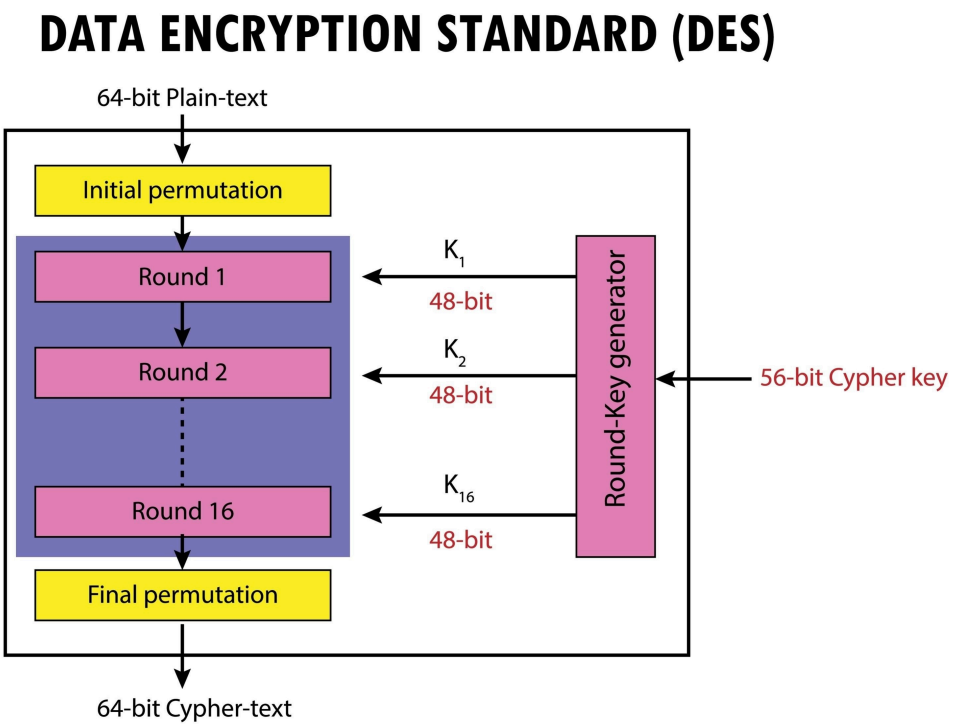
Asymmetric Encryption uses **two keys**—a **public key** for encryption and a **private key** for decryption. It is more secure for communication but **slower** than symmetric encryption.
Example: RSA

Feature	Symmetric	Asymmetric
Keys	One key	Public + Private
Speed	Fast	Slower
Security	Key sharing risk	More secure
Example	AES	RSA

2.Encrypt files using AES.

AES (Advanced Encryption Standard) is the gold standard for symmetric encryption. You can use the `openssl` command-line tool (available on Linux, macOS, and Windows via Git Bash) to encrypt a file.

Encrypt a file: CMD=openssl enc -aes-256-cbc -salt -in secret.txt -out secret.txt.enc
Decrypt the file: CMD=openssl enc -aes-256-cbc -d -in secret.txt.enc -out decrypted.txt



Best Practices for AES

When you move from learning to real-world application, keep these three rules in mind:

1. **Key Strength:** AES-128 is theoretically secure, but **AES-256** is preferred to stay ahead of future computing power (quantum resistance).
2. **Mode of Operation:** While `CBC` is common, **GCM (Galois/Counter Mode)** is often preferred in modern apps because it provides both encryption and "authentication" (it proves the file wasn't tampered with).
3. **Key Management:** In professional environments, you don't type passwords manually. Keys are often stored in a "Key Vault" or "Hardware Security Module" (HSM)

3..Generate RSA keys.

Generating **RSA (Rivest-Shamir-Adleman)** keys is the foundation of asymmetric encryption. Unlike AES, where you use one password, RSA creates a pair: a **Public Key** to lock data and a **Private Key** to unlock it.

---

1. Generate the Private Key

The private key is the most sensitive part. It is used to decrypt data and create digital signatures. We will generate a **4096-bit** key, which is currently considered the secure standard (2048-bit is the minimum).

Bash

```
openssl genrsa -out private_key.pem 4096
```

Optional: Password-Protect Your Private Key

If you want to ensure no one can use your private key even if they steal the file, add encryption to the key itself:

Bash

```
openssl genrsa -aes256 -out private_key_encrypted.pem 4096
```

---

2. Extract the Public Key

The public key is derived mathematically from the private key. You can share this file with anyone in the world.

Bash

```
openssl rsa -in private_key.pem -pubout -out public_key.pem
```

---

3. How the Key Pair Works

Now that you have your `.pem` files, here is how they interact:

Action	Key Used	Why?
Encryption	**Public Key**	Anyone can lock a message for you.
Decryption	**Private Key**	Only you (the holder of the secret) can open it.
Signing	**Private Key**	Proves the message definitely came from you.
Verification	**Public Key**	Allows others to verify your signature.

---

4. Inspecting the Key Data

If you want to see the actual "prime numbers" and math behind your key (instead of just the Base64 encoded text), run:

Bash

```
openssl rsa -in private_key.pem -text -noout
```

---

5. Security Warning

- **Keep `private_key.pem` safe:** If someone gets this file, they can impersonate you and read your encrypted messages.
- **File Permissions:** On Linux or macOS, restrict access to your private key immediately: Bash

  ```
  chmod 600 private_key.pem
  ```

4.Understand digital signatures.

Phase 1: Creating the Signature

The sender creates the signature using their **Private Key**. Note that we don't encrypt the whole document (which would be slow); we only encrypt the **hash** (the "fingerprint") of the document.

Phase 2: Verifying the Signature

The receiver uses the sender's **Public Key**. If the hash they calculate locally matches the one they decrypted from the signature, the document is authentic and unchanged.

Why this is so secure:

- **The Math:** If a single comma is changed in the document, the hash changes completely. The verification will fail.
- **The Key:** Since only the sender has the Private Key, only they could have created that specific encrypted hash.
- **Public Access:** Since everyone has the Public Key, anyone can verify the signature, but nobody can forge it.

5.Hash files and verify integrity.

1. The Simple Process

1. **Generate:** You run a file through a "hashing machine."
2. **Output:** It spits out a short string of random-looking letters and numbers (the hash).

3. **Check:** If you change even **one letter** in that file and run it again, the hash will look completely different.

---

2. How to use it (The "Download" Example)

Imagine you are downloading a new video game.

- **The Website says:** "The hash for this game is `A1B2C3`."
- **You download it:** You run a hash tool on your computer.
- **The Result:** If your computer says `A1B2C3`, the file is perfect. If it says `X9Z8Y7`, the file is broken or was hacked.

---

3. Hashing vs. Encryption

People often mix these up. Here is the easiest way to remember:

- **Encryption** is a **Lock Box**: You put a message in, lock it with a key, and later you use the key to take the message out.
- **Hashing** is a **Meat Grinder**: You put a steak in, and it comes out as hamburger meat. You can't turn the hamburger back into a steak, but you can tell if it's the *same* steak by looking at the result.

---

4. Try it right now

If you are on a computer, you can see this yourself:

1. Open your command line.
2. Type: `echo "hello" | shasum -a 256`
3. Now type: `echo "hello!" | shasum -a 256` (Note how adding one `!` changes the whole code).

6.Compare encryption algorithms. simple

.A) The Sprinters (Symmetric) and **The Bodyguards (Asymmetric).**

Feature	Symmetric (The Sprinter)	Asymmetric (The Bodyguard)
Example	**AES**	**RSA** or **ECC**
Keys	Uses **one** shared key.	Uses a **pair** (Public & Private).

Feature	Symmetric (The Sprinter)	Asymmetric (The Bodyguard)
Speed	Super fast (Like a Ferrari).	Slower (Like an armored truck).
Best Use	Encrypting large files or drives.	Securely sharing keys or signing docs.

B) The Top 3 Algorithms Compared

These are the three you will actually see in the real world:

AES (Advanced Encryption Standard)

- **The Vibe:** The gold standard for speed.
- **How it works:** It chops data into blocks and scrambles them with a single key.
- **Use it when:** You want to encrypt your hard drive, a large zip file, or your Wi-Fi traffic.

RSA (Rivest-Shamir-Adleman)

- **The Vibe:** The reliable old-timer.
- **How it works:** It uses massive prime numbers to create a public/private key pair.
- **Use it when:** You need to send a secure message to someone you've never met (like connecting to a website for the first time).
- **Downside:** It needs huge keys (3,072+ bits) to stay safe, which makes it "heavy" and slow.

ECC (Elliptic Curve Cryptography)

- **The Vibe:** The modern, high-tech replacement for RSA.
- **How it works:** It uses the math of complex curves instead of just prime numbers.
- **Use it when:** You are on a mobile phone or smart device. A tiny 256-bit ECC key is just as strong as a giant 3,072-bit RSA key.

C. The "Hybrid" Trick

In the real world (like when you see the padlock icon in your browser), we don't choose just one. We use both:

1. **Asymmetric (RSA/ECC)** is used for the "handshake" to prove who is who and safely exchange a secret key.
2. **Symmetric (AES)** is then used for the rest of the session because it's much faster for moving data.

##7.Learn real-world usage (HTTPS, VPN).

1. HTTPS (The Web Standard)

HTTPS uses a protocol called **TLS (Transport Layer Security)**. It is a "Hybrid" system because it uses both asymmetric and symmetric encryption to get the best of both worlds: security and speed.

The TLS Handshake (Simplified)

1. **The Greeting:** Your browser sends a "Hello" to the website, listing which encryption methods it supports.
2. **The Proof:** The website sends back its **Digital Certificate** (containing its **Public Key**). Your browser checks if a trusted authority (like DigiCert) signed it to prove the site isn't a fake.
3. **The Secret Handshake:** Your browser creates a random "Session Key" and encrypts it using the website's **Public Key**. Only the website can unlock this with its **Private Key**.
4. **The Switch:** Now that both have the secret Session Key, they stop using slow RSA/ECC and switch to fast **AES** for the rest of your visit.

---

2. VPN (The Secure Tunnel)

A **Virtual Private Network (VPN)** doesn't just secure one website; it secures **all** the data leaving your device. It creates an "encrypted tunnel" between you and a VPN server.

Real-World Usage:

- **Hiding from your ISP:** Your Internet Service Provider can see you are connected to a VPN, but they cannot see *which* websites you are visiting or what you are doing.
- **Public Wi-Fi Safety:** If you are at a coffee shop, a hacker on the same Wi-Fi can't see your data because it's wrapped in a layer of AES-256 before it even hits the airwaves.

How it works:

VPNs use protocols like **OpenVPN** or **WireGuard**.

- **WireGuard (Modern):** Uses **ECC** for a very fast handshake and **ChaCha20** (similar to AES) for data. It is known for being extremely fast on phones.
- **OpenVPN (Classic):** Uses **AES-256** and is highly customizable, making it the "gold standard" for privacy for the last decade.

---

3. HTTPS vs. VPN: Do you need both?

Think of it like this:

- **HTTPS** is like a **secure envelope**. It protects the letter inside, but everyone can still see who you are sending the letter to (the address on the envelope).
- **A VPN** is like a **private armored tunnel**. It hides both the letter *and* where you are going.

Feature	HTTPS	VPN
What it protects	Just the browser session	All apps/traffic on the device
Hides IP Address?	No	**Yes**
Hides Browsing History?	Partially (ISP sees domain)	**Yes** (ISP only sees VPN IP)
Setup required?	None (Automatic)	Must install an app

---

4. Documentation Lab Task

To complete your learning, try this:

1. Open any website and click the **Padlock icon** in the address bar.
2. Look for "Connection is secure" -> "Certificate is valid."
3. Find the "Details" or "Subject" to see if they use **RSA** or **ECC** for their public key.