Explanation of Cryptographic Scripts (Part A and B)

🔐 Part A: Secure Chat Using ECC (Elliptic Curve Cryptography)

What This Code Does:

This script simulates a secure conversation between Alice and Bob using modern cryptographic techniques. They:

1. Create their own private/public keys.

2. Use ECDH (Elliptic Curve Diffie-Hellman) to agree on a shared secret key.

3. Use AES encryption to encrypt a message.

4. Use digital signatures to sign and verify the message, ensuring authenticity.

Main Concepts:

- ECDH: Lets two people generate a shared secret over an insecure channel.

- AES-GCM: A fast and secure encryption method using a shared key.

- ECDSA: Lets a sender "sign" a message so the recipient knows it really came from them.

Key Functions Explained:

- generate_ecdh_key() – Creates a new ECC private key.

- derive_shared_key(private_key, peer_public_key) – Uses ECDH to compute a shared key from two parties' private and public keys.

- encrypt_message(key, plaintext) – Encrypts a message using AES with the shared key.

- decrypt_message(key, iv, ciphertext, tag) – Decrypts the AES-encrypted message.

- sign_message(private_key, message) – Signs a message with the sender's private key.

- verify_signature(public_key, message, signature) – Verifies the signature using the sender's public key.

What Happens in ecdh_chat_simulation():

1. Alice and Bob generate their own ECC key pairs.

2. They exchange public keys and derive the same shared key.

3. Alice encrypts a message and signs it.

4. Bob decrypts the message and verifies the signature.

Output Example:

Encrypted Message: 4a6c0f... (in hex)

Decrypted Message: Hello Bob, this is Alice.

Signature Valid: True

🔓 Part B: Understanding Digital Signatures in RSA

What This Code Does:

This script shows how digital signatures work in RSA, and why they're secure — unless someone tries to cheat the system by flipping the process.

Main Concepts:

- RSA Key Pair: Two large numbers (a public key and a private key).

- Sign (with private key): Takes a message and creates a signature.

- Verify (with public key): Takes a signature and checks if it's valid for a message.

Key Functions:

- generate_keys(bits=16) – Generates small RSA keys (for demonstration purposes only).

- sign(message, priv_key) – Creates a signature using private key: signature = message^d mod n.

- verify(signature, pub_key) – Checks a signature: message = signature^e mod n.

Eve's Two Attacks:

Part (a): Why Eve Can’t Forge a Signature for a Real Message

Eve can’t generate a signature for a specific message (like 123456789) unless she knows the private key. That’s the RSA security assumption.

Part (b): Eve Cheats by Choosing Signature First

Here, Eve cheats by choosing a random "signature" s, and then computes the message m = s^e mod n. The result is a valid (but meaningless) message-signature pair.

This trick shows why signing without checking the message is dangerous.

Output Example:

--- Part (a): Eve cannot forge signature for fixed m ---

Eve cannot easily find s such that s^e ≡ 123456789 mod n

Because it would require solving the RSA problem (modular root).

--- Part (b): Eve forges signature s = 112090305 ---

Eve forged a valid signature!

Signature s = 112090305

This corresponds to message m = 79337809