ENCS5121
Information Security and
Computer Networks Laboratory

EXPERIMENT #5
RSA Public-Key Encryption and Signature

Slides by: Mohamad Balawi
Updated By: Tariq Odeh

Overview

- **Objective:**
  - Gain hands-on experience with the RSA algorithm to reinforce theoretical understanding of key generation, encryption, decryption, and digital signatures.
- **Scenario:**
  - Implement RSA using C to practice each step in the encryption and signature processes with actual numbers.
- **Task:**
  - Apply RSA fundamentals by generating public/private keys, performing encryption/decryption, and implementing digital signature generation and verification.

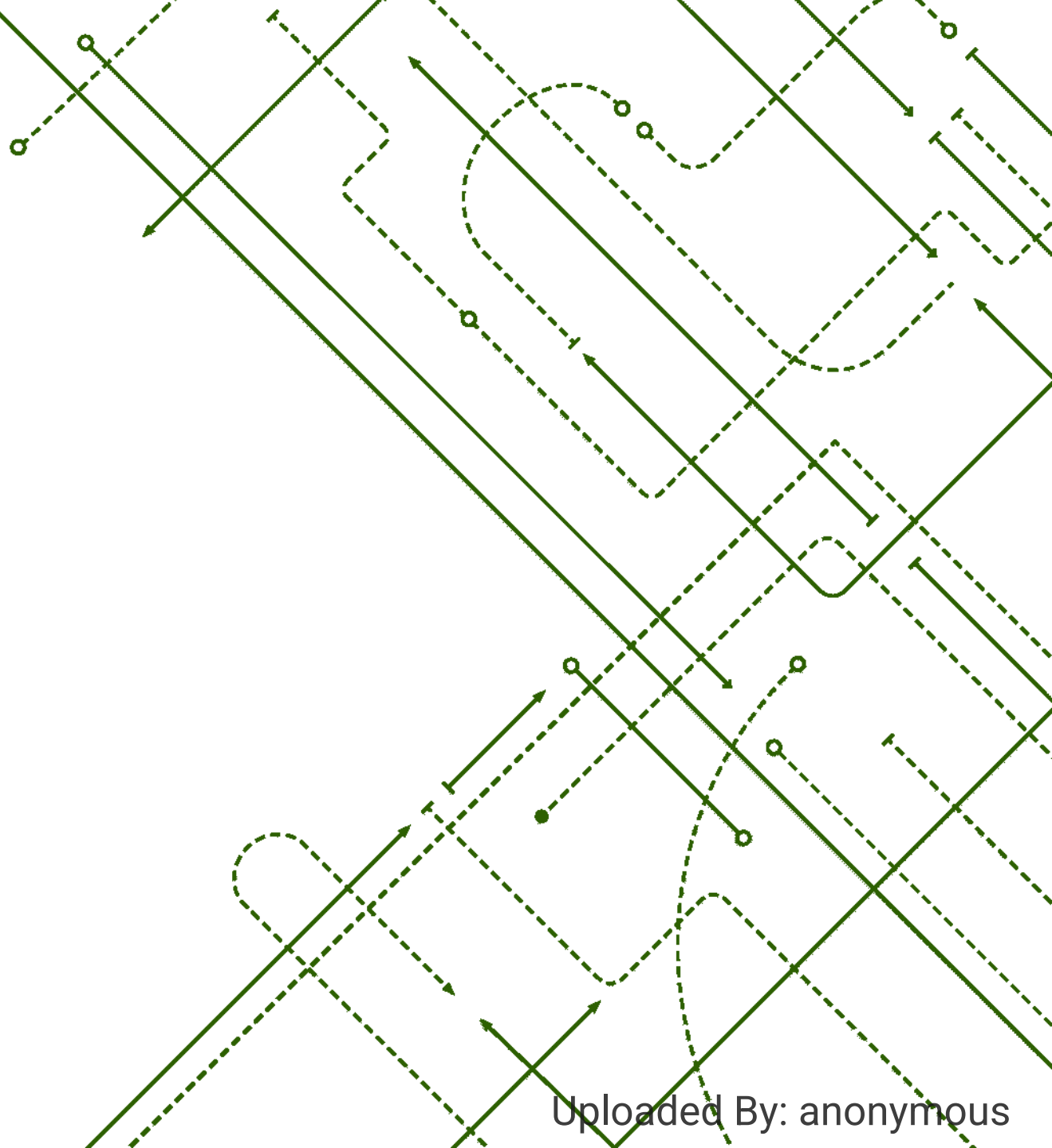
Overview (Cont.)

- **Topics Covered:**
  - Public-key cryptography
  - RSA algorithm and key generation
  - Big number calculations
  - RSA encryption and decryption
  - Digital signatures
  - X.509 certificates

Outline

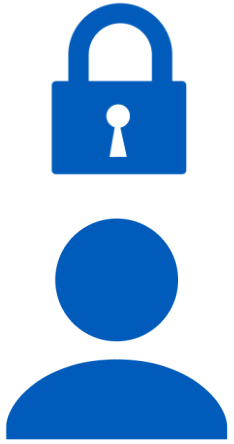
- **Introduction**
  - Web
  - MAC/MITM
  - Hashing/HMAC
- **Lab Environment**
- **Task 1: Send Request to List Files**
- **Task 2: Create Padding**
- **Task 3: The Length Extension Attack**
- **Task 4: Attack Mitigation using HMAC**

Introduction



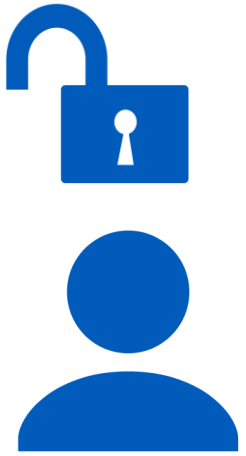
Secret Key Encryption

Secret Key Encryption uses a single shared key for both encryption and decryption, ensuring that only parties with the key can securely exchange messages.



Alice

But there is a fundamental issue in secret key encryption that prevents it from being used as an encryption standard.

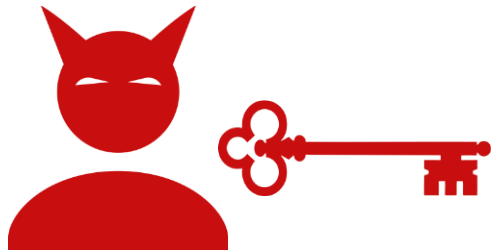


Bob

Secret Key Encryption

The challenge is securely distributing and managing the shared secret key among communicating parties.

Consider a situation where Alice seeks to establish a secure connection with a local bank she hasn't interacted with previously. Since they lack a shared secret key, Alice's options are limited to either transmitting the key openly or visiting the bank and receiving assistance from a dedicated employee for this purpose.



Public Key Encryption

Public key encryption solves the issue by using a pair of keys: a public key and a private key . The public key is freely distributed, allowing anyone to encrypt messages , while the private key remains secret and is used for decryption. This eliminates the need for sharing a secret key beforehand, enabling secure communication without prior contact or key exchange.

Bank

Alice

Examples for SKE & PKE

PUBLIC KEY ENCRYPTION

Asymmetric scheme, has two keys, public one for encryption, and secret one for decryption.

- RSA (Rivest-Shamir-Adleman)
- Diffie-Hellman Key Exchange
- ElGamal
- Elliptic Curve Cryptography (ECC)

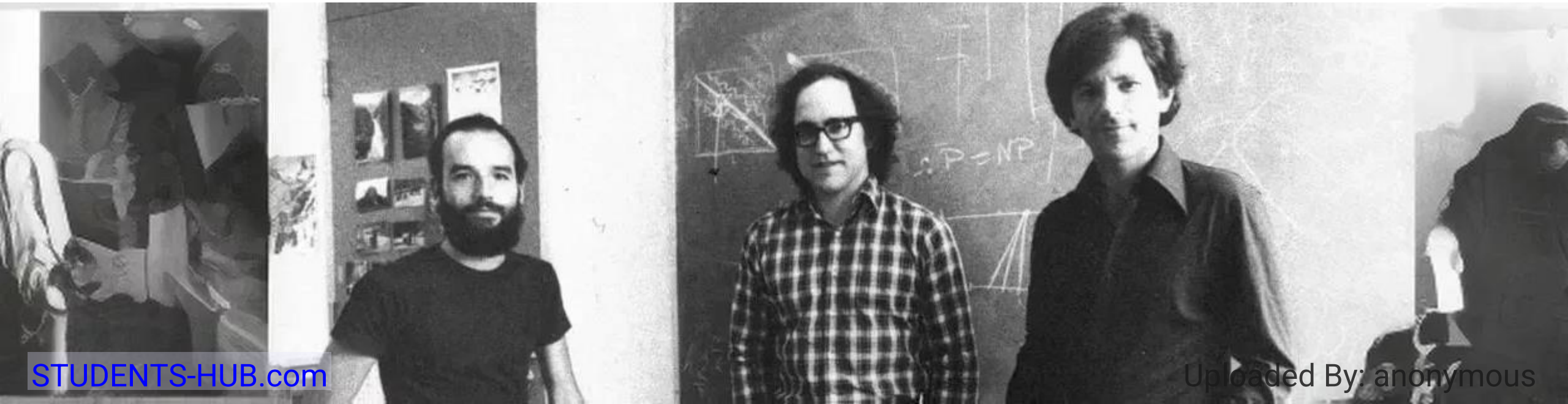
SECRET KEY ENCRYPTION

Symmetric scheme, has one key used for both encryption and decryption.

- AES (Advanced Encryption Standard)
- DES (Data Encryption Standard)
- 3DES (Triple Data Encryption Standard)
- Blowfish

RSA (Rivest-Shamir-Adleman)

RSA is an asymmetric encryption algorithm that relies on the difficulty of prime factorization. It involves generating a public key and private key pair, where the public key is used for encryption and the private key is used for decryption.



RSA (How does it work?)

1. Choose two random large prime numbers. p, q
2. Multiply them to get the modulus n. $n = pq$
3. Calculate the totient (euler) of n. $\varphi(n) = (p-1)(q-1)$
4. Choose e such that $e = \begin{cases} 1 < e < \varphi(n) \\ \gcd(e, \varphi(n)) = 1 \end{cases}$
5. Determine d such that $ed \equiv 1 \pmod{\varphi(n)}$

   $d \equiv e^{-1} \pmod{\varphi(n)}$

RSA (How does it work?)

6. Announce the public key. (n, e)
7. Store the secret key. (n, d)
8. You can encrypt using: $c \equiv m^e \pmod{n}$
9. You can decrypt using: $m \equiv c^d \pmod{n}$

Digital Signature

Digital signatures, used in asymmetric encryption like RSA, ensure data integrity and non-repudiation. In contrast, MACs are employed in symmetric encryption for message authentication.

Unlike the encryption process, we sign the message using the secret key, and that is how we ensure a unique signature, and since everyone knows the public key, they can use it to verify the signature.

- Generate a signature using secret key. $s = m^d \pmod n$
- Verify the signature using the public key. $m = s^e \pmod n$

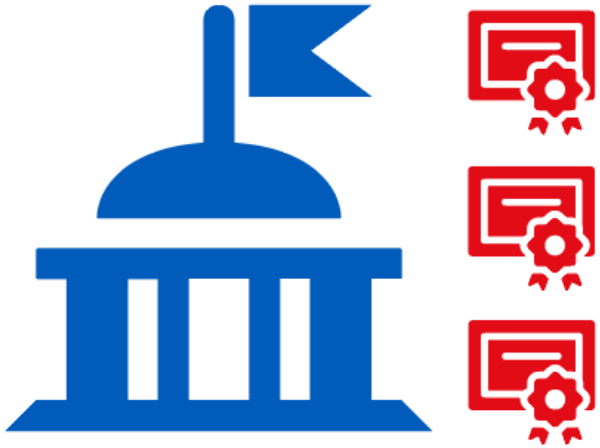
X.509 Certificate

X.509 is an International Telecommunication Union (ITU) standard defining the format of public key certificates, it is used in many Internet protocols, including TLS/SSL, which is the basis for HTTPS. It includes important info such as:

- **Subject**: Identifies the entity the certificate is issued to.
- **Issuer**: Identifies the entity that issued the certificate.
- **Validity Period**: consisting of a start date and an expiration date.
- **Public Key**: Includes the public key of the subject.
- **Signature Value**: Contains the digital signature generated by the issuer's private key to verify the integrity and authenticity of the certificate.

Certificate Authority

A Certificate Authority (CA) is a trusted entity responsible for issuing digital certificates used to verify the authenticity and identity of individuals, organizations, or websites on the internet. CAs validate the information provided by the certificate requester and digitally sign the certificate to attest to its legitimacy.

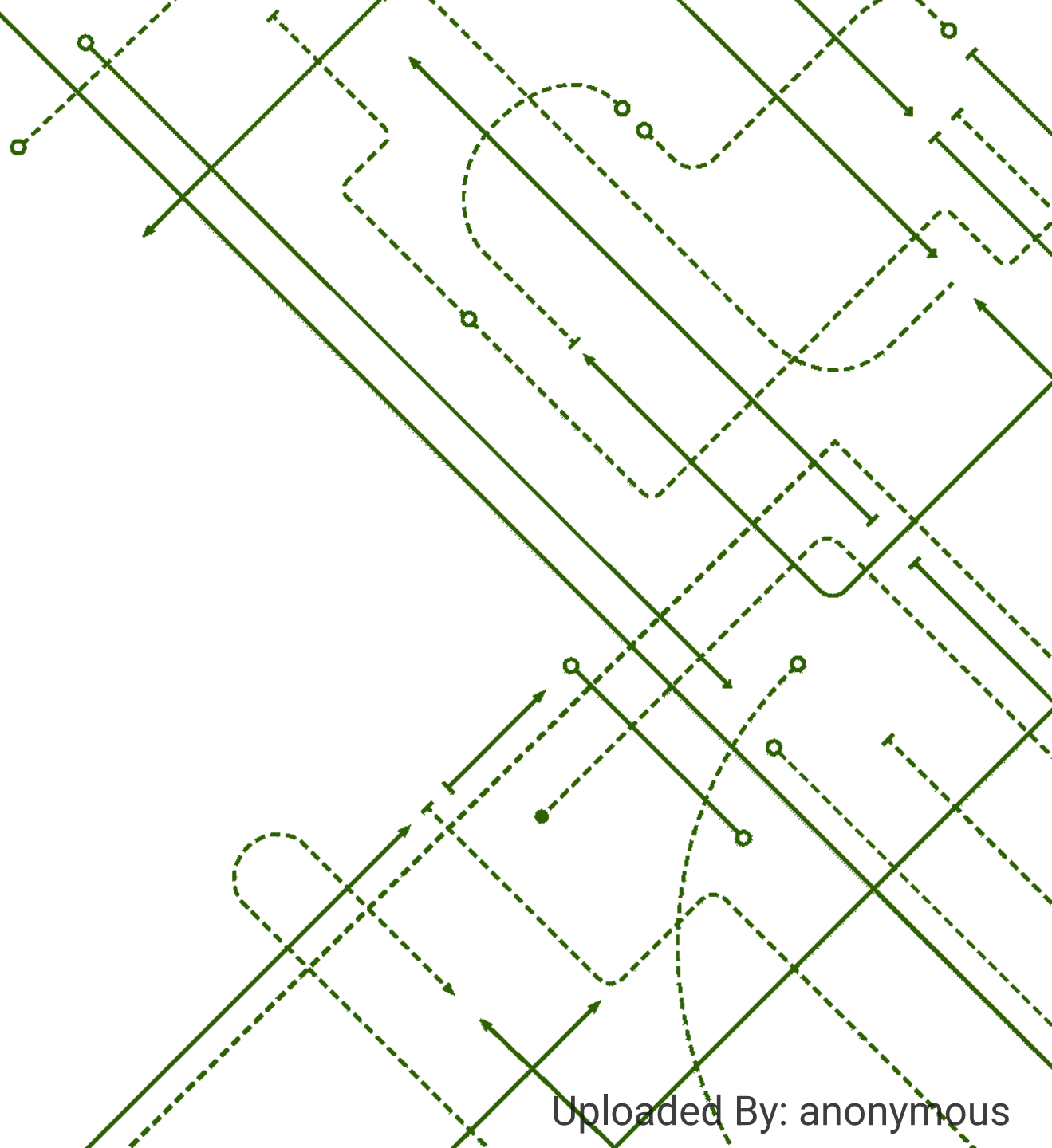


BIGNUM APIs

BIGNUM APIs in C are necessary for handling large integers that exceed the range of built-in data types like int or long. These APIs provide functions and structures for performing arithmetic operations, such as addition, subtraction, multiplication, and division, on arbitrarily large numbers. This capability is crucial for cryptographic algorithms, mathematical computations, and other applications where precision and size are significant factors.

Largest datatype in C is **long long int** (64-bit).

Tasks



Task 1: Deriving the Private Key

- **Objective:** Calculate the private key d given the public key components p,q, and e..
- **Given Values:**
  - p = F7E75FDC469067FFDC4E847C51F452DFp=F7E75FDC469067FFDC4E847C51F452DF
  - q = E85CED54AF57E53E092113E62F436F4Fq=E85CED54AF57E53E092113E62F436F4F
  - e = 0D88C3e=0D88C3

- **Note:** For simplicity, these values are smaller (128 bits) than typical secure values (at least 512 bits).

Task 2: Encrypting a Message

- **Objective:** Encrypt a message using RSA public key encryption.
- **Components:**
  - Public Key: Comprises two values, (e, n)
  - Message: The text to be encrypted.
- **Process:**
  1. Convert Message: Change the plaintext message into a format suitable for encryption (e.g., hex).
  2. Encrypt: Use the public key (e, n) to perform encryption on the formatted message.

Task 3: Decrypting a Message

- **Objective:** Decrypt a ciphertext using RSA.
- **Components:**
  - Private Key: Used for decryption.
  - Ciphertext: The encrypted message to be decrypted.
- **Process:**
  1. Decrypt: Apply the private key to the ciphertext to retrieve the original data.
  2. Convert: Transform the decrypted data into a readable format (e.g., plain ASCII).
- **Outcome:** The original plaintext message.

Task 4: Signing a Message

- **Objective:** Generate a digital signature for a message.
- **Components:**
  - Private Key: Used for signing.
  - Message: The original message to be signed (e.g., "I owe you $2000").
- **Process:**
  1. Sign the Message: Create a signature directly from the message.
  2. Modify the Message: Make a slight change (e.g., change $2000 to $3000) and sign the modified message.
  3. Comparison: Analyze and compare both signatures.
- **Observation:** Describe how the signatures differ based on the changes made to the message.

Task 5: Verifying a Signature

- **Objective:** Verify the authenticity of a signature against a message.
- **Components:**
  - Public Key: Used for verification (e, n)
  - Message: The received message (e.g., "Launch a missile.").
  - Signature: The digital signature provided by the sender.
- **Process:**
  1. Verify the Signature: Use the public key to check if the signature corresponds to the message.
  2. Corruption Simulation: Introduce a small change (e.g., modify the last byte of the signature) and repeat the verification.
- **Outcome:**
  - Describe the result of the verification process with the modified signature and discuss what happens.

Task 6: Manually Verifying an X.509 Certificate

1. **Download Certificate:** Use the command openssl s_client -connect [server_name]:443 -showcerts to retrieve the server's certificate and save it as c0.pem. Download the corresponding CA certificate and save it as c1.pem.
2. **Extract Public Key:** From c1.pem, extract the modulus (n) using openssl x509 -in c1.pem -noout -modulus and find the public exponent (e) with openssl x509 -in c1.pem -text -noout.
3. **Extract Signature:** Use openssl x509 -in c0.pem -text -noout to retrieve the signature from the server certificate, then format it by removing spaces and colons.
4. **Extract Certificate Body:** Parse c0.pem with openssl asn1parse -i -in c0.pem, identify the body's offset, and extract it using openssl asn1parse -i -in c0.pem -strparse [offset] -out c0_body.bin -noout. Compute its hash with sha256sum c0_body.bin.
5. **Verify Signature:** Validate the server certificate's signature against the computed hash using the CA's public key in your own verification program.