Princess Sumaya University for Technology

King Hussein School for Computing Sciences

RSA Key Pairs Generation and Encryption/Decryption Documentation

Information Systems Security (11464)

Fall - 2023/2024

Instructor:

Dr. Mustafa Al-Fayoumi

Prepared by:

Osama Alsarayreh 20200336

Fares Al-Khuffash 20200727

Omar Khalil 20200889

RSA Overview

RSA (Rivest–Shamir–Adleman) is a widely used public-key cryptosystem that is used for secure data transmission, such as in the transmission of credit card information or other sensitive data over the internet. It is also used to secure data at rest, such as when storing sensitive data in a database or on a file system. It is based on the mathematical properties of large prime numbers and their difficulty in factorization. RSA is commonly used for many online services, like: secure data transmission, digital signatures and key exchange over the internet.

Key Pair Generation (Step 1)

RSA uses a pair of keys for the encryption and decryption processes: A public key (Kpub) and a private key (Kpr). The key generation process is done in the following steps:

1. **Choose Two Prime Numbers (p and q):**

   The user inputs two large prime numbers, p and q (Two different prime numbers).

2. **Compute n and ϕ(n):**

   Compute n = p * q, where n is used as the modulus for both the public and private keys.

   Compute ϕ(n) = (p-1) * (q-1), Euler's Totient Function.

3. **Select Public Exponent (e):**

   The user inputs a public exponent e, typically a small prime number not equal to 1 or ϕ(n).

4. **Generate Keypair:**

   Use the chosen values to generate a keypair using the crypto.subtle.generateKey function (In the code as you will see or have seen).

5. **Export Keys:**

   Export the public and private keys in standard formats or infrastructure (SPKI for public key, PKCS8 for private key as you see or have seen in the code).

Encryption (Step 2)

Encryption in RSA is done using the recipient's public key. The process involves the following steps:

1. **Import Public Key:**

   The recipient's public key is imported using crypto.subtle.importKey (As you see or have seen in the code).

2. **Encrypt Data:**

   Use crypto.subtle.encrypt with the RSA-OAEP algorithm to encrypt the plaintext of 4 characters (As you see or have seen in the code).

3. **Export Ciphertext:**

   Convert the ciphertext to a hexadecimal string to display on the user’s screen that no one could encrypt without knowledge of the recipient’s private key and encryption process.

Decryption (Step 3)

Decryption in RSA is done using the recipient's private key. The process involves the following steps:

1. **Import Private Key:**

   The recipient's private key is imported using crypto.subtle.importKey (As you see or have seen in the code).

2. **Decrypt Data:**

   Use crypto.subtle.decrypt with the RSA-OAEP algorithm to decrypt the ciphertext (As you see or have seen in the code).

3. **Display Result:**

   Convert the decrypted data from an ArrayBuffer to a readable string using TextDecoder to display on the user's screen (If you copy and paste the encypted text in the text to decrypt field it would give you the original plaintext of 4 characters).

Code Explanation

HTML and CSS

The HTML is used for the simple User Interface required in the project with sections for Key Pairs Generation, Displaying Keys and Text Encryption/Decryption.

CSS is used for styling the User Interface.

JavaScript

The Web Crypto API is used in the code for cryptographic operations.

Keypair Generation:

generate_rsa_keys(p, q, e): Generates an RSA keypair based on input primes p and q and public exponent e.

Computes n and ϕ(n) and uses the Web Crypto API to generate the keypair.

The keys are exported and returned as base64-encoded strings.

Encryption:

encrypt_text(public_key, plaintext): Encrypts the input plaintext using the recipient's public key.

Converts the public key from base64, imports it, and then encrypts the plaintext using RSA-OAEP.

The ciphertext is returned as a hexadecimal string.

Decryption:

decrypt_text(private_key, ciphertext): Decrypts the input ciphertext using the recipient's private key.

Converts the private key from base64, imports it, and then decrypts the ciphertext using RSA-OAEP.

The decrypted text is returned as a string.

Utility Functions:

encodeBase64 and decodeBase64: Convert between ArrayBuffer and base64 string.

arrayBufferToHex and hexToArrayBuffer: Convert between ArrayBuffer and hexadecimal string.

Button Click Handlers:

generateKeypair(), encryptText(), and decryptText(): Handle button clicks, execute the corresponding functions, and update the interface with the results.

Finally, this is requested documentation for the project providing an overview of the RSA algorithm and all its functionalities, like: key pair generation, and encryption/decryption using the HTML and JavaScript code in the attached file to this documentation. The code shows the practical implementation and functionalities of RSA for secure communication over the internet.