Cryptography Blockchain

Lab 2 - RSA

Lab Report

CARVAJAL Mateo
CHAHINE Maroun
DOS ANJOS GUIMARAES Bernardo

5IF | OT4
December 2025

Analysis of our RSA implementation

All our results can be found in the CSV file and in the txt file. The source codes of the RSA for each exercise are also available in the py files and named respectively.

Exercise 1

RSA key generation, encryption and decryption were implemented using two large primes and following the method described in the lecture.

Random search was used to select “e” instead of a deterministic loop in order to avoid predictable key patterns and improve security. It provides better cryptographic security and unpredictability. While a for-loop starting from a fixed value (3, 5, 7...) would quickly find a valid “e”, it makes the key generation process predictable, attackers could guess likely values of “e” based on common patterns. Random selection ensures that each key pair has a unique, unpredictable public key, rising security. Additionally, since the probability of finding a coprime to phi(n) is very high, the expected number of iterations is small (typically 1-3 attempts), making random search both secure and efficient. This approach balances cryptographic strength with computational practicality, ensuring that RSA keys remain resistant to pattern-based attacks.

Since RSA works only on integers, each character of a plaintext message was converted to its Unicode value before encryption and transformed back after decryption.

Exercise 2

Two 512-bit primes were used to generate a 1024-bit RSA key pair. Several plaintext messages were encrypted and correctly decrypted, confirming the correctness of the implementation.

Exercise 3

A single key pair must be reused throughout the lab, since randomness in generating “e” would otherwise produce different keys each time. The result will change from one run to another. For practical purposes during the lab, we created one key pair for public and private key and used them in both exercise 3 and 4. The key that we generated and used can be found in “Results_Key_gen_Ex-3-4_Needed-Data.txt”. They were generated using the script “Exercise-3-4-key-generation.py”.

Exercise 4

Digital signatures were added by first signing the message with the sender’s private key and then encrypting the signed message with the recipient’s public key.
On reception, the order is reversed: decrypt first, then verify the signature. This ensures both confidentiality and integrity.

Exercise 5

The implementation to support messages larger than the key size was already done in exercise 2.

Exercise 6

Performance was measured for messages from 2^10 to 2^16 bytes with 1024-bit keys.
Encryption time increased linearly with message size, reaching more than 200 seconds for bytes.
Ciphertext size also grew linearly, with each plaintext byte expanding to 128 bytes due to the 1024-bit modulus. This is make the ciphertext harder ot decrypt.
Overall, RSA showed poor performance for large data. It took a lot of time for encryption. The decryption process was not tested.

Exercise 7

DES performance was tested for different sizes during lab 1. It showed stable, linear scaling and significantly faster processing at all message sizes. However, in DES, ciphertext size remains equal to plaintext size, unlike RSA. In RSA, the expansion gives us a very long plaintext (128 times longer), harder to decrypt.

Overall, DES is suitable for big data encryption, while RSA becomes impractical for large messages and is best reserved for small data such as keys and passwords. We can clearly say that the RSA implementation is more secure however and more resistant to attacks.

We should note that the use cases of DES and RSA are different. RSA is mainly used to achieve confidentiality and authentification, for each message transmitted.