Implementation of secure communication

Nathan Lesourd

Abstract

In this report you will find a detailed use case illustrating the implementation of RSA encryption in the banking sector. We will analyse the strengths and limitations of this method, as well as the trade-offs involved. Finally, we will see whether RSA is still relevant today with the emergence of quantum computers.

I. Introduction

A. Understanding RSA Encryption

RSA (Rivest-Shamir-Adleman) is a widely used public-key cryptosystem that plays a crucial role in securing communications and data transmission over insecure networks. It was developed in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman and is named after their last names. RSA encryption relies on the mathematical properties of large prime numbers and modular arithmetic.

1) Public and private keys: RSA uses two keys: a public key for encryption and a private key for decryption. Key generation begins by selecting two large prime numbers, denoted as p and q. The product of p and q is used to compute a modulus. This modulus, n, is a part of both the public and private keys.

$$n = p * q \tag{1}$$

The public key consists of two components: the modulus n and an encryption exponent (usually denoted as e). The recipient shares the public key with anyone who wishes to send them encrypted messages. To encrypt a message ($plaintext$) with the recipient's public key, the sender raises the $plaintext$ to the power of e, modulo n.

$$ciphertext = plaintext^e \pmod{n} \tag{2}$$

The private key includes the modulus n and a decryption exponent (usually denoted as d). The recipient keeps the private key secret. To decrypt a ciphertext received with their public key, the recipient raises the $ciphertext$ to the power of d, modulo n. Because d is calculated from p, q, and e during key generation, only the recipient can compute d efficiently.

$$plaintext = ciphertext^d \pmod{n} \tag{3}$$

2) Security: The security of RSA relies on the difficulty of factoring the product n of two large prime numbers. Given n, it is computationally infeasible to determine p and q, making it challenging to derive the private key from the public key. The security of RSA depends on the size of the modulus n. Longer key lengths provide stronger security but also require more computational resources for encryption and decryption. Key lengths of 1024 bits or more are recommended for secure communication in contemporary applications.

3) Applications: RSA encryption is commonly used for securing data transmission over the internet, including HTTPS for secure web browsing, email encryption, and digital signatures. It is also used in key exchange protocols, such as Diffie-Hellman, to establish secure communication channels. While RSA encryption is a powerful and versatile cryptographic algorithm, its use may not be optimal or suitable in certain contexts, such as resource-constrained environments, performance-critical applications, situations requiring forward secrecy, post-quantum security concerns, and scenarios that demand efficient hashing or digital signatures. It's essential to consider the specific requirements and constraints of a given application when choosing cryptographic algorithms.

4) Modified and modern RSA-based encryption: Various enhancements and extensions have been proposed or developed to improve performance, security, and functionality in specific use cases. Here are a few notable modern or modified RSA-based encryption methods:

- RSA-PSS (Probabilistic Signature Scheme) RSA-PSS is an enhanced RSA-based signature scheme that incorporates probabilistic elements to improve security against certain attacks [1]. It uses a randomized padding scheme and is designed to provide strong security guarantees against various forms of attacks, including those related to the "padding oracle" vulnerability.
- RSA-KEM (Key Encapsulation Mechanism) RSA-KEM is a technique that combines RSA encryption with symmetric key encryption to achieve the advantages of both [2]. It generates a random symmetric key, encrypts it with RSA, and then

uses the symmetric key for efficient encryption and decryption of data. This approach can provide better performance in some scenarios.
- Ring-LWE (Ring Learning With Errors) with RSA Ring-LWE is a lattice-based encryption scheme that offers post-quantum security. Researchers have explored hybrid approaches that combine Ring-LWE with RSA to leverage the benefits of both cryptographic systems, ensuring security against quantum attacks while maintaining compatibility with existing RSA infrastructure [3].

B. Use Case scenario

Alice, a bank customer, wants to perform online banking transactions, including checking her account balance, transferring funds, and paying bills through her bank's website or mobile app. The bank wants to ensure the security and confidentiality of Alice's financial data throughout the entire transaction process.

1) Application of RSA Encryption:
- Key Pair Generation: The bank generates an RSA key pair: a public key (containing the modulus n and encryption exponent e) and a private key (containing the decryption exponent d).
- Secure Communication Setup: When Alice accesses the bank's website or app, her device establishes a secure connection (typically using protocols like HTTPS) with the bank's servers. This connection is encrypted using the bank's public key, ensuring that any data sent from Alice's device to the bank is protected from eavesdropping.
- Encryption of Sensitive Data: When Alice initiates a transaction, such as transferring funds or checking her account balance, the data (e.g., transaction details and account information) is encrypted on her device using the bank's public key. This ensures that only the bank, possessing the private key, can decrypt and access the data.
- Data Transmission: The encrypted data is transmitted securely over the internet to the bank's servers.
- Decryption at the Bank's End: Upon receiving the encrypted data, the bank's servers use their private key to decrypt the information, making it accessible for processing.
- Secure Response Transmission: Any response or confirmation from the bank, such as a transaction receipt or updated account balance, is encrypted using Alice's public key before transmission. This ensures that only Alice, with her private key, can decrypt and access the response.

2) Importance of RSA in this Use Case: RSA encryption ensures the confidentiality of sensitive financial data. Even if an attacker intercepts the data during transmission, they cannot decrypt it without the private key. It provides authentication, as only the bank's servers can decrypt data encrypted with their public key, verifying the legitimacy of the bank's communication. Furthermore, RSA helps protect against man-in-the-middle attacks, where an attacker tries to intercept and alter data exchanged between Alice and the bank.

In this scenario, RSA encryption plays a crucial role in securing online banking transactions, protecting both the customer's financial data and the integrity of the banking system. It is a fundamental technology for ensuring trust and security in the world of online finance.

II. Design and Implementation

In order to implement our use case, we need to implement the three major functions of this process, which are key generation (public and private key pairs) and the encryption and decryption functions based on these generated keys.

A. Key generation

RSA key generation is based on two very large prime numbers. The larger the size of the two prime numbers, the greater the security, but this implies a great deal of computing power. So there is a trade-off between security and process performance. The key generation function takes as input the number of digits that the two prime numbers p and q must have in order to determine the complexity of the encryption according to requirements and available computing resources. The first step is to generate two random numbers with the set number of digits. We then check whether or not these numbers are prime using the '$is\_prime\_miller\_rabin$' function. This function is a probabilistic method for determining whether a number is prime or not, with the parameter k being the number of tests to be validated to guarantee a reliable result (by default 5). The risk of having a false positive is less than 4^{-k}, which gives us a probability of less than 0.0009765625 for k equal to 5. The code implementation has been found in the following GitHub : https://gist.github.com/Ayrx/5884790.

Thus,

$$n = p * q \tag{4}$$

$$phi_n = (p-1) * (q-1) \tag{5}$$

To ensure efficiency, e is set to 65537. Despite this fixed value, the security of the encryption is guaranteed thanks to the complexity of the factorisation of the prime numbers p and q. Finally, we determine d by performing the inverse modulus of e and phi_n. The inverse modulus is calculated using the Euclidean algorithm. The function takes two arguments as input: a (an integer) and m (an integer).

- It starts by checking whether m is equal to 1. If it is, it returns 0, because the modular inverse does not exist when the modulo is equal to 1.
- Next, the function initializes three variables: m0, x0, and x1.
- The function enters a while loop that continues as long as a is greater than 1.
- After the loop, if x1 is negative, the function adjusts it by adding m0 to make it positive. This ensures that the modular inverse is positive and in the range [0, m-1].
- Finally, the function returns the value of x1, which is the modular inverse of a modulo m.

Finally we get:

$$private\_keys : [e, n] \tag{6}$$

$$public\_keys : [d, n] \tag{7}$$

B. Encryption and decryption

Encryption and decryption follow the same algorithm. Using the private and public keys, the '$fast\_modular\_exponentiation$' function is performed. This method optimises the result $a^b \pmod n$ by performing only $log(b)$ multiplications, compared with b multiplications for the naive method. The function takes three arguments as input: a (an integer), b (an integer, the exponent), and n (an integer, the modulo).

- It initializes two variables: $result$ and $base$. $result$ is initialized to 1, and $base$ is initialized to $a \pmod n$.
- The function enters a while loop that continues as long as b is greater than 0. This loop uses a clever technique to reduce the number of multiplications needed to calculate $a^b \pmod n$.
- At each iteration, the function checks whether the least significant bit (the rightmost bit) of b is equal to 1 using the operation $b \pmod 2 = 1$. If so, it multiplies result by $base \pmod n$ and updates result.
- Next, it squares base modulo n and divides b by 2 (shifts to the right). This halves the value of b.
- The loop continues until b becomes 0.
- Finally, the function returns the value of $result$, which is the result of $a^b \pmod n$ calculated quickly and efficiently.

In this way we obtain encrypted and decrypted messages:

III. Test Results

```
KEY GENERATION
This task is carried out by the bank's departments. Public key is sent to Alice to ma
nage her encryption.
Generation completed.
Number of digits for p and q:  300
In our use case, the bank id of Alice need to be protected thanks to RSA encryption.
Alice's bank id:  11820329401
Encryption of Alice's bank id thanks to the public key:  146055423812905101146476775 6
859027940881979290470595462993140845922525464523028728984308037434743842801950716050 9
133474135979712137387338128598881139341682578800046182742656226768695843855865779361 8
228350833910244166319321517358976044233804732153869423493188511452833513565282631350 9
475682732333223051566325428508994501874747570311146863909023556936332715581389386612 0
151975606488773993816486610837982557031509791833603186148524183543505305317228555944 1
046998616195950017909747769469804553943206271109568535334491417540061342729649150305 6
894322699339639498391452009956467845753092814384789575301440 17
Now that the data is encrypted, Alice's bank id can be send to the bank service.
Once received, bank's departement decrypter the cyphertext and get Alice's bank id.
11820329401
```

Fig. 1. Use case in action

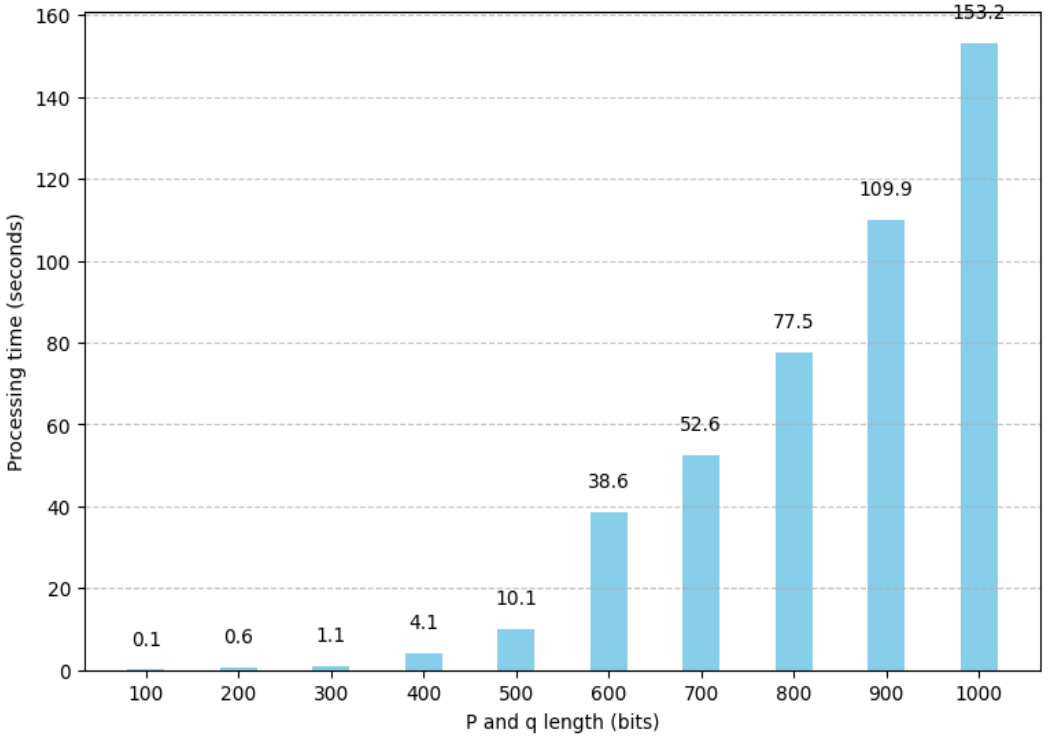

Fig. 2. Processing time for RSA encryption and decryption by p and q length

IV. Discussion

Implementing the use case with python (Fig. 1.) gave me an insight into the subtleties of encoding/decoding with RSA. The big problem was to find a trade-off between speed and security for the user who wanted to share his data. As I decided earlier to set e, the exponent of the public key, we only have one way of increasing the security of our process: the length of p and q. The larger these two numbers, the greater the security. In order to find this trade-off, I carried out a study measuring the encryption and decryption process time as a function of the length of randomly generated p and q. Since the choice of p and q is random, so is the process time. This stochastic nature was smoothed by reproducing the study 5 times and averaging the times obtained. These averages are summarised in Fig. 2. Thanks to this study, we can see that the best trade-off is found for a length between 300 and 400 decimal digits for p and q. Beyond that, the delays explode and the additional security is unnecessary. In fact, it is now considered that security is sufficient for a length of p and q of the order of 1000 binary digits, which is equivalent to 300 decimal digits.

RSA encryption has many advantages, and this is why it is so widely used. The security of this encryption is based on the complexity of factoring a number into prime numbers. RSA lets us choose the level of security we want to apply by choosing the size of the keys. In addition, the use of asymmetric keys adds an extra layer of security. Key communication is safer because man-in-the-middle attacks cannot work. With only knowledge of the public key, the attacker will not be able to extract any information from the data flow.

However, RSA also has a few drawbacks. The emergence of quantum computers calls into question the complete security of the process, since the problem of factoring into prime numbers is not a concern for it. Managing asymmetric keys is complex. Generating them and managing certificates can be a complicated task on a large scale. Because of the constant increase in the computing power of machines, it is necessary to keep sufficient key sizes up to date. This maintenance can be time-consuming and can lead to vulnerabilities. Finally, RSA is vulnerable to side-channel attacks knowing the power consumed by the computer and the time elapsed during the encryption process.

V. Conclusion

The aim of this project was to gain a deep understanding of how encryption with asymmetric keys works and to study the strengths and limitations of RSA. The implementation of the use case highlighted the trade-off between security and speed in the encryption process. With an appropriate key size and a high-performance key management system, RSA offers a simple and secure solution for exchanging data. The emergence of quantum computers calls into question the use of this method. However, even if it can no longer be used on its own to guarantee sufficient security with quantum air, it can still be used in combination with other methods. The simultaneous use of symmetric and asymmetric encryption will make it possible to obtain a process that is resistant to quantum computers.

References

[1] Jakob Jonsson. "Security Proofs for the RSA-PSS Signature Scheme and Its Variants". In: 2001. URL: https://eprint.iacr.org/2001/053.

[2] Atsushi Fujioka et al. "Practical and Post-Quantum Authenticated Key Exchange from One-Way Secure Key Encapsulation Mechanism". In: ASIA CCS '13. Hangzhou, China: Association for Computing Machinery, 2013. DOI: 10.1145/2484313.2484323.

[3] Joppe W. Bos et al. "Post-Quantum Key Exchange for the TLS Protocol from the Ring Learning with Errors Problem". In: *2015 IEEE Symposium on Security and Privacy*. 2015, pp. 553–570. DOI: 10.1109/SP.2015.40.