Asymmetric Public Key Encryption & Hybrid Encryption

Sean Chen[a]

[a] *Bluvec Technolgies Inc., Burnaby, , British Columbia, Canada*

Abstract

The implementation of the RSA encryption system in Go requires the use of multiple for-loops due to the block length limitation imposed by the nature of the algorithm. Pure RSA encryption and decryption processes for larger files are extremely slow. However, by employing hybrid encryption with both AES and RSA, performance is enhanced by over 98%.

1. Introduction

Encryption is the process of protecting information or data by using mathematical models to scramble it in such a way that only the parties who have the key to unscramble it can access it[1]. Over the course of the almost 4000-year history of cryptography[2], different kinds of encryption were invented and used by people.

2. Symmetric Cryptography

Symmetric encryption represents the algorithms for cryptography that use the same cryptography keys for both the encryption of plain-text and the decryption of cipher-text[3]. A famous, also one of the most ancient, encryption algorithms is the Caesar cipher, or the substitution cipher. The Caesar cipher is a simple substitution cipher technique named after Julius Caesar, who is believed to have used it to communicate secret messages. It operates by shifting each letter in the plain-text a certain number of positions down the alphabet. For example, the message ‘hello’ would be encrypted as ‘ifmmp’ by shifting the letters by one position down the alphabet. However, this is extremely vulnerable to brute-force attacks due to the limited number of positions in the alphabet.

2.1. AES Encryption

The most widely used symmetric encryption today is called the AES, the Advanced Encryption Standard, or the Rijndael cipher, which was publicly announced and established by the U.S. National Institute of Standards and Technology(NIST) in 2001. It’s composed of multiple rounds of surprisingly easy permutation and substitution operations; however, the time expected for brute force attacks on AES grows exponentially as the time for each operation is linear.

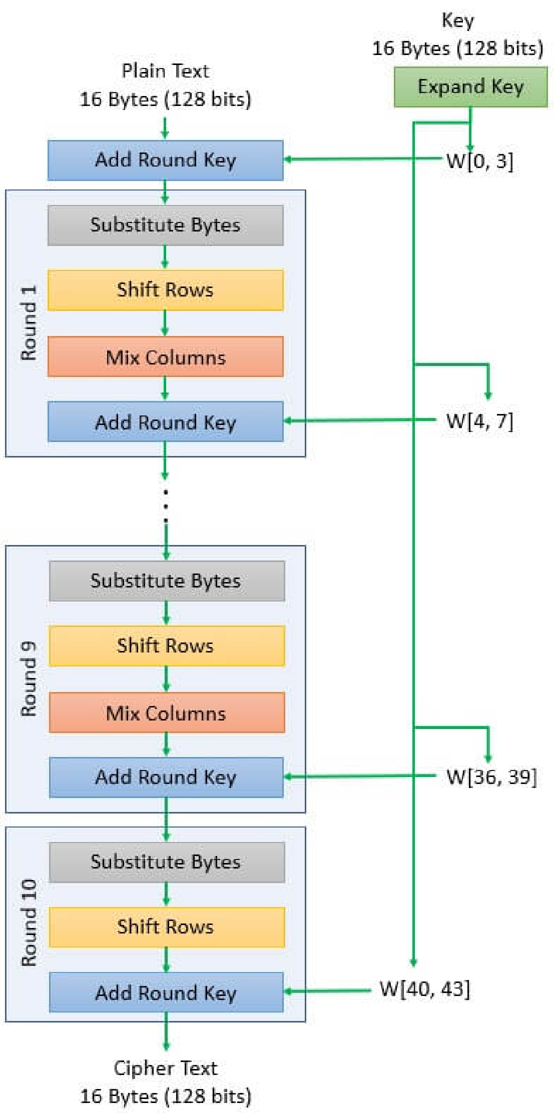

Figure 1: AES Encryption

3. Asymmetric Cryptography

Asymmetric encryption, on the other hand, uses a public key-private key pairing. Data encrypted by the public key can only be decrypted by the private key. With asymmetric encryption, we can then hide data from certain parties while still allowing them to encrypt information. For example, if Alice wants to send Bob an encrypted message while they do not have a predetermined private key for symmetric cryptography, Bob would need to set up an asymmetric encryption system and announce the public key. Alice can then communicate with Bob in encrypted messages which only Bob can decrypt. Now an obvious question arises, how does Bob reply? The answer is by having a reverse asymmetric encryption system going the other way round.

3.1. RSA Encryption

RSA (Rivest–Shamir–Adleman) is a public-key encryption system announced in 1977 by the three cryptographers who named the algorithm with their surnames[4]. This algorithm about primes is based on Fermat's little theorem.

First of all, we choose p, q as two very large primes. Then we have $n = pq$ which n is only divisible by p and q other than 1. $\phi(n)$ would be the Euler's totient function of n, which $\phi(n) = (p-1)(q-1)$, and we will discuss this in section 3.3. Then, we pick a value e which has to be coprime with $\phi(n)$ while $2 < e < \phi(n)$. A commonly picked value is $e = 65537$, which is a prime itself. Let d be the modulo multiplicative inverse of $e \mod \phi(n)$, that is,

$$d = e^{-1} \mod \phi(n)$$

3.1.1. Encryption

Note that our public key contains n and e, which are the only two integers required for the encryption process. Let p be our plain-text, the only round required to obtain the cipher-text c is

$$c = p^e \mod n$$

3.1.2. Decryption

The private key contains n and d. Notice that d was derived from $\phi(n)$, thus, p and q must be kept secret. We can then obtain our plain-text

$$p = c^d \mod n$$

3.2. RSA Example

Let's look at a simple example of the RSA encryption method. Let's say we want to encrypt and decrypt the message 'hello', and we start from the letter 'h', which is equivalent to 104 in decimal in the ASCII.

Let $p = 47$, $q = 43$, we have $n = pq = 2021$. Then $\phi(n) = 46 * 42 = 1932$. Since $2 < e < \phi(n)$, we choose $e = 257$ which is a prime.

3.2.1. Encryption

As stated above, the encryption raises the plain-text to the power of e then take the modulo of n.

$$\begin{aligned} c &= p^e \mod n \\ &= 104^{257} \mod 2021 \\ &= 105 \end{aligned}$$

3.2.2. Decryption

In order to obtain the plain-text, we will have to calculate d. As mentioned above,

$$d = e^{-1} \mod \phi(n)$$

that is,

$$ed = 1 \mod \phi(n)$$

We have $e = 257, \phi(n) = 1932$, thus

$$257d = 1 \mod 1932$$

or

$$d = 257^{-1} \mod 1932$$

We will use the Extended Euclidean Algorithm to find d. With $d = 857$, solve for

$$\begin{aligned} p &= c^d \mod n \\ &= 105^{857} \mod 2021 \\ &= 104 \end{aligned}$$

3.3. Euler's Totient Function

In number theory, Euler's Totient Function of n, usually written has $\phi(n)$, represents the number of positive integers less than n while being coprime to n. That is, $\phi(n)$ is equal to the number of $m \in \mathbb{N}$ such that $1 \leq m < n$, and $gcd(m,n) = 1$, while $gcd(m,n)$ represents the greatest common divisor between m and n.

It is trivial that $\phi(p) = p - 1$ where p is prime. Note that $n = pq$ where p and q are both prime numbers, $\phi(n)$ would then be multiples of p and multiples q which are less than n subtracted from $(n-1)$. Thus, $\phi(n) = (p-1)(q-1)$.

3.4. Proof of RSA

If

$$c = t^e \mod n$$

and

$$t = c^d \mod n$$

holds, then we want to prove that

$$(t^e)^d = t \mod n$$

Hypothesis:
1. p, q are prime numbers.
2. $n = pq$
3. $ed = 1 \mod \phi(n)$

Thesis:
$(t^e)^d = t \mod n$

By Euler's theorem, since t and n are coprime, we have

$$t^{\phi(n)} \equiv 1 \mod n \qquad (1)$$

From hypothesis 3,

$$(t^e)^d = t^{ed} = t^{1+k\phi(n)}$$

where k is a constant. Then, we have

$$t^{1+k\phi(n)} = t \cdot (t^{\phi(n)})^k$$

Note that

$$(t^{\phi(n)})^k \equiv 1^k \mod n$$

by (1), thus

$$t \cdot (t^{\phi(n)})^k \equiv t \mod n$$

by Euler's theorem. Therefore, our thesis holds.

3.5. Proof Against Attacks

The RSA system becomes more and more secure as key sizes increase. The security of the algorithm stems from the substantial computational difficulty involved in factoring an extremely large number such as a 2048-bit key. Think about numbers like 43, or 47, we are able to do some simple calculations in our mind to try and factor them. We would evidently fail because they are primes. However, think about larger numbers such as 293 or 571–most of us will not be able to tell if they are primes within several seconds, because primality tests are computationally intensive tasks either performed in our brains or computers, especially for large numbers.

The traditional algorithm for checking if positive integer n is a prime would be to check each integer individually from 1 to $n-1$. We have done some basic optimization for it to look like this:

```
if n % 2 == 0 {
    return false; // not a prime!
}
for i = 1; i <= ceil(n^0.5); i+=2 {
    if n % i == 0 {
        return false; // not a prime!
    }
}
return true; // prime!
```

The traditional algorithm would have a time complexity of $O(n^{0.5}/2)$, which would still be equivalent to $O(n^{0.5})$. By using a key size of 2048 bits, it would take over 10^{308} operations to find the key in the worst case scenario. There are only about 10^{82} atoms in the entire universe[7]. Note that as the number gets exponentially large, it simply is unfeasible to crack the algorithm with brute force.

There are primality tests such as the Fermat's test and the Fibonacci test which are only heuristic tests (not proven, but very accurate). These tests are able to tell if a very large number, n, is likely to be a prime, however, they are not able to prove if n is really a prime nor to factor n.

Since the generator of the primes p and q used in the RSA system is the only party that knows them, the system offers strong security as these private key primes are safely stored.

3.6. Drawbacks from Cipher Blocks

AES encrypts data like a chain of blocks. As illustrated in Figure 1, each block is 16 bytes or 128 bits from the plain-text. They are encrypted and then put together in order in the encryption process, and then decrypted and then put together in the decryption process. The operations from AES are quick permutations and substitutions, and the encryption and decryption on large files are relatively fast compared to other algorithms.

Unlike AES, RSA does not have built in cipher blocks. The maximum message length that RSA can encrypt depends on the key size and the padding scheme used. For example, if we have a 2048-bit key, and the OAEP padding uses SHA-256 as its hash function, we will only be able to encrypt $2048 - 256 * 2 - 16 = 1520$ bits, which is equivalent to 190 bytes. In the event where a large file would be encrypted and decrypted by pure RSA, a loop has to be implemented for dividing the file into blocks and the performance becomes noticeably slow.

One of the causes for RSA's subpar performance is that a 128-bit AES key demands a 3072-bit RSA key while 256-bit AES demands an RSA key size of 15,360 bits for equivalent security[5]. RSA then requires performing computationally intensive operations such as calculating the exponentiation of the large keys. Additionally, RSA is particularly slow for decryption, as e, the chosen coprime exponent (usually 65537), was used in the encryption process, but a much larger private exponent is used for decryption.

The Big-O notation for AES is $O(1)$ since AES operates on a block of data of fixed size (128 bits or 16 bytes) regardless of the key size. Now, another obvious question arises, if the length of my message is m, how it is possible for AES to encrypt or decrypt m in $O(1)$? The overall operation would not be $O(1)$ for a large message like m, but $O(len(m)/16)$ (the number of blocks).

The Big-O notation for RSA is significantly higher compared to AES, rendering it much slower. The encryption process is relatively fast, as the public exponent chosen is usually 65537, or 257 in our example in section 3.2. The encryption process involves raising the plain-text message to the power of the public exponent, which is considered $O(1)$, then taking the modulus of n. The time complexity of RSA encryption can then be considered as $O(n)$. When it comes to decryption of RSA, since we are computing a modular exponentiation with the private exponent, the operations become computationally expensive. Multiplication of two n bit integers takes $O(n^2)$ time, as each bit in the first number must be multiplied with each bit in the second number. Thus, decryption for RSA would have a time complexity of $O(n^3)$. Again, like the time complexity we discussed for AES, the Big-O notations for RSA are for each block of messages. Suppose we have a large message or file that we want to encrypt and decrypt with pure RSA, the time complexity for encryption would then be $O(mn/190)$, and $O(mn^3/190)$ for decryption if we use a 2048-bit key with SHA-256 padding.

4. Hybrid Encryption

Hybrid encryption combines the efficiency of symmetric encryption with the convenience of public key (asymmetric)

Cipher	AES	RSA
Encryption	$O(1)$	$O(n)$
Decryption	$O(1)$	$O(n^3)$

Table 1: Time complexity for each block

Cipher	AES	RSA
Encryption	$O(m)$	$O(mn)$
Decryption	$O(m)$	$O(mn^3)$

Table 2: Time complexity for message length m

cryptography[6]. The idea is rather ingenious, providing a resolution to both symmetric and asymmetric methods' problems. We will randomize our AES key, then encrypt it with the RSA public key, and send the key along with the message encrypted with the randomized AES key.

In an RSA scenario, what we want to achieve is the total confidentiality of our private keys. While we address the computational inefficiencies of RSA by limiting its use to encrypting the AES key, our primary goal would be undermined if this AES key were to be disclosed to the public. The essential part of this system is the randomization of the AES key. No one but the holder of the RSA private key would be able to decrypt the AES key–not even the generator of the key himself.

4.1. Benchmark

Table 3 displays the data from our benchmark of both the hybrid and pure RSA systems, each entry was obtained by computing the average value from 5-7 trials. The unit (s/MB) represents seconds per megabytes. By switching to a hybrid system with AES, we were able to decrease our encryption time by over 98.70%, and decryption time by over 99.98%.

Cipher	Hybrid	Pure RSA
Encryption Time(s/MB)	$3.96 * 10^{-3}$	$3.06 * 10^{-1}$
Decryption Time(s/MB)	$5.22 * 10^{-3}$	$4.76 * 10^{1}$

Table 3: Actual time from benchmark

4.2. Limitations

Note that although hybrid encryption with a public-key system and a symmetric cipher system provides decent security and complexity, drawbacks and limitations exist. Hybrid systems are almost always more complex to implement than systems which use only one type of encryption. Users may be vulnerable due to risk of errors. Since an additional key must be created and kept confidential, it is crucial that the randomized symmetric key must be disposed securely as well.

Acknowledgements

Thanks to Leyuan for providing me with this opportunity to implement our new file encryption and decryption system, as well as the chance for me to dig in a bit at the cryptography part. As an engineer, I am driven by a strong sense of purpose, always seeking to comprehend the what, how, and why behind our works.

I still have many topics I would like to explore, including the Euler's theorem, Fermat's little theorem, and their generalisation which serves as the foundation of the RSA algorithm. Additionally, the Chinese remainder theorem holds significant importance in the modulus calculations within the system. There is much to discuss and delve into regarding these amazing concepts.

References

1. Google. (n.d.-b). *What is encryption and how does it work? — Google Cloud.* Google. Retrieved from https://cloud.google.com/learn/what-is-encryption.

2. Binance Academy. (2020, January 19). *History of cryptography.* Binance Academy. Retrieved from https://web.archive.org/web/20200426075650/https://www.binance.vision/security/history-of-cryptography.

3. Kartit, Z. (2018). *Advances in ubiquitous networking: Proceedings of the UNET15.* SPRINGER.

4. Rivest, R. L., Shamir, A., & Adleman, L. M. (1977). *A method for obtaining digital signatures and public-key cryptosystems.* Massachusetts Institute of Technology, Laboratory for Computer Science.

5. *The next generation of cryptography.* (n.d.). Retrieved from https://www.certicom.com/content/certicom/en/code-and-cipher/the-next-generation-of-cryptography.html

6. Google. (n.d.-a). *Hybrid encryption — tink — Google for Developers.* Google. Retrieved from https://developers.google.com/tink/hybrid.

7. Baker, H. (2021, July 10). *How many atoms are in the observable universe?.* LiveScience. https://www.livescience.com/how-many-atoms-in-universe.html

Figures

1. T, N. (2021, January 18). *What is Advanced Encryption Standard (AES)? definition, encryption, decryption, advantages and disadvantages.* Binary Terms. https://binaryterms.com/advanced-encryption-standard-aes.html

Appendix A. Extended Euclidean Algorithm

We will use the Extended Euclidean Algorithm to find d.

$$\begin{aligned}
1932 &= 257(7) + 133 \\
257 &= 133(1) + 124 \\
133 &= 124(1) + 9 \\
124 &= 9(13) + 7 \\
9 &= 7(1) + 2 \\
7 &= 2(3) + 1
\end{aligned}$$

Backwards:

$$\begin{aligned}
1 &= 7 + 2(-3) \\
&= 7 + (9 + 7(-1))(-3) \\
&= 9(-3) + 7(4) \\
&= 9(-3) + (124 + 9(-13))(4) \\
&= 9(-55) + (124)(4) \\
&= (133 + 124(-1))(-55) + (124)(4) \\
&= (133)(-55) + (124)(59) \\
&= (133)(-55) + (257 + (133)(-1))(59) \\
&= (133)(-114) + 257(59) \\
&= (1932 + 257(-7))(-114) + 257(59) \\
&= (1932)(-114) + 257((-7)(-114) + 59) \\
&= 257(857) + 1932(-114) \\
&= 257(857) \mod 1932 \\
\text{thus, } d &= 857
\end{aligned}$$