The RSA Algorithm

Jack Griffiths, Max Johnson, James Lounds, Harvey Olive, Oscar Oliver, James Taylor

March 17, 2022

Contents

1 Introduction

In this paper, we will use $\mathbb{Z}_n$ to represent the multiplicative group of integers modulo n, and $a =_n b$ to mean b is a written in base n.

RSA is an asymmetric cryptosystem first publicly proposed by Rivest, Shamir, and Adler in 1977 [4]. It has been used since at least 1973 in secret by intelligence organisations. An asymmetric cryptosystem is a way to encrypt a message with a "key" that is public - that is anyone can know its value without compromising the secuirty of the system - and a way to decrypt the encrypted message with a *private* key - if this key is known (with the public key), it is easy for an attacker to decrypt messages. The security of the cryptosystem relies on the difficulty of factoring large prime numbers.

1.1 Original Requirements

In their 1977 paper, Rivest, Shamir and Adler proposed the following criteria an asymmetric cryptosystem should satisfy.
For an encyption procedure E, and decryption procedure D on a message M

1. $D(E(M)) = M$
2. D, E should be easy to compute
3. Revealing E does not reveal an efficient method for D
4. $E(D(M)) = M$

According to Diffie and Hellman's paper [3], satisfying (1)-(3) implies E is a "Trap-Door One-Way Function". With the added criterion of (4), E must be a "Trap-Door One-Way Permutation" - each output is a valid input to the function. A One-Way function is easy to compute one way, but hard to compute the inverse. A Trap-Door function has a hard to compute inverse, unless some private information is known, which makes the inverse easy to compute

2 The Proposed Cryptosystem

2.1 Definitions

1. Pick random primes p, q, and define $n \coloneqq p \cdot q$
2. Pick a random d such that $d^{-1} \in \mathbb{Z}_{(p-1)\cdot(q-1))}$
3. Define $e \equiv d^{-1} \ (mod \ (p-1) \cdot (q-1)) \equiv d^{-1} \ (mod \ \phi(n))$
4. Define Encryption: $E(M) \equiv M^e \ (mod n)$
5. Define Decryption: $D(M) \equiv M^d \ (mod \ n)$

We can now reveal our public key (e, n), which tells the reciever nothing about our private key (d, n).

2.2 Proof

Theorem 1. *If p, q are prime,*

$$\left.\begin{array}{l} a^b \equiv c \ (mod \ p) \\ a^b \equiv c \ (mod \ q) \end{array}\right\} \Rightarrow a^b \equiv c \ (mod \ p \cdot q)$$

Proof. If $a^b \equiv c \ (mod \ p)$, then $a^b = k_1 \cdot p + c$, similarly $a^b = k_2 \cdot q + c$.

$$\begin{aligned} a^b &= k_1 \cdot p + c = k_2 \cdot q + c, && (k_1, k_2 \in \mathbb{Z}) \\ &\Rightarrow k_1 \cdot p = k_2 \cdot q \Leftrightarrow k_1 = \alpha_1 \cdot q, k_2 = \alpha_2 \cdot p && \text{(since p, q are prime)} \\ a^b &= k_1 \cdot p + c = \alpha_1 \cdot p \cdot q + c = \alpha_2 \cdot p \cdot q + c \\ a^b &= \alpha_1 \cdot (p \cdot 1) + c \Leftrightarrow a^b \equiv c \ (mod \ p \cdot q) \end{aligned}$$

□

Theorem 2. *If p, q are prime and $n \coloneqq p \cdot q$, then for any $d \equiv e^{-1} \ (mod \ \phi(n))$.*

$$E(M) \equiv M^e \ (mod \, n), \quad D(M) \equiv M^d \ (mod \, n)$$

$$\Downarrow$$

$$D(E(M)) \equiv M \ (mod \, n) \equiv E(D(M)) \ (mod \, n)$$

Proof. If a prime p does not divide M,

$$\begin{aligned} M^{\phi(p)} &\equiv M^{p-1} \ (mod \, p) \equiv 1 \ (mod \, p) && (\textit{by Fermat's Little Theorem}) \\ &\equiv M^{k\phi(n)} \ (mod \, p) \equiv 1 \ (mod \, p) && (\textit{Since } \phi(p) \textit{ divides } \phi(n)) \\ M^p &\equiv M^{k\phi(n)+1} \ (mod \, p) \equiv M \ (mod \, p) && (\textit{By multiplying by } M) \end{aligned}$$

The same can be argued for q

By Theroem 1:

$$\left.\begin{aligned} M^{k\phi(n)+1} &\equiv M \,(mod\, p) \\ M^{k\phi(n)+1} &\equiv M \,(mod\, q) \end{aligned}\right\} \Rightarrow M^{k\phi(n)+1} \equiv M \,(mod\, n)$$

By 3. $e \equiv d^{-1} \,(mod\, \phi(n))$, so

$$\begin{aligned} e \cdot d &\equiv 1 \,(mod\, \phi(n)) \\ e \cdot d &= k\phi(n) + 1 \qquad\qquad \text{(for some } k \in \mathbb{N}) \\ \Rightarrow D(E(M)) &\equiv M^{e \cdot d} \,(mod\, n) \\ &\equiv M^{k \cdot \phi(n)+1} \,(mod\, n) \\ &\equiv M \,(mod\, n) \end{aligned}$$

□

2.3 Efficient Computation

2.3.1 Modular Exponentiation

Consider the binary representation of some exponent e:

$$e = 2^0 \cdot e_0 + 2^1 \cdot e_1 + \ldots + 2^n \cdot e_n$$

By the rules of indicies,

$$\begin{aligned} M^e &= M^{2^0 \cdot e_0 + 2^1 \cdot e_1 + \ldots + 2^n \cdot e_n} \\ &= M^{2^0 \cdot e_0} \cdot M^{2^1 \cdot e_1} \cdots M^{2^n \cdot e_n} \end{aligned}$$

So, in $\mathbb{Z}_n$

$$\begin{aligned} M^e &\equiv M^{2^0 \cdot e_0} \cdot M^{2^1 \cdot e_1} \cdots M^{2^n \cdot e_n} \ (mod\ n) \\ &\equiv (M^{2^0 \cdot e_0} \ (mod\ n)) \cdot (M^{2^1 \cdot e_1} \ (mod\ n)) \cdots (M^{2^n \cdot e_n} \ (mod\ n)) \ (mod\ n) \end{aligned}$$

We can easily generate the elements $M^{2^i e_i}$ by repeatedly squaring M. We only include this element if e_i is 1, that is the bit is true. So we only need to square M $log_2(e)$ times, then perform at most $log_2(e)$ multiplications and remainder calculations. But space is where the main savings are: the largest number we need to store will necessarily be less than n. We only need to store the running total, and the current square (e is given to us). So we only need to store 2 n-sized blocks.

2.3.2 Prime Generation

There are a variety of prime generation algorithms, but we will only consider the probabilistic method in RSA's paper. If a is prime less than a, then:

$$gcd(a, b) = 1, \text{ and } J(a, b) \equiv a^{(b-1)/2} (mod b)$$

Where J is the Jacobi symbol. However, this is also true with probability $\frac{1}{2}$ when b is not prime. So if we test r values of a, and the equations are valid for each a we choose, there is only a $(\frac{1}{2})^{100} \approx \frac{1}{1.26 \cdot 10^{30}}$ chance b is not prime. Other algorithms exist, and in our implementation, we used python's `Crypto.Util.number.getPrime`, which generates a random number and checks it with a probabilistic sieve.

3 Utility

Assymetric Encryption is fundamental to how modern networks communicate securely, albeit using a different algorithm Assymetry makes the ability to communicate securely with no private channel, and the ability to sign messages easy. There are symmetric algorithms for both, but assymetry removes some

trust - since the other party cannot give access to your private key, but could give access to a shared key.

The ability to send information about how to communicate securely over an insecure channel was vital in the development of secure email, and http protocols. Before modern encryption standards, parties would swap some physical data, be it a hard drive, or a post it note. From this shared data, which was known to be private, both parties could establish shared secrets generated by the original physical secret, using a publicly available algorithm. This reliance on physical secrecy was not scalable for individuals to secure their communications. It would be akin to sending Google a letter with your password on. Diffie and Hellman [3] propose a system for creating a shared secret over an insecure channel using modular exponentiation.

The ability to sign a message is more unique to assymetric cryptography. If we had a symmetric system, in order for Bob to prove to Charlie that Alice sent him a message, Bob must reveal Alice and Bob's shared secret to Charlie. Since Charlie now knows the secret, she can decrypt any message Alice and Bob sent to one another with this key. However, in a public key system, anyone can know Alice's public key, so for Bob to prove to Charlie that Alice sent him a message, he only needs to tell Charlie Alice's public key, maintaining the privacy of Alice's key pair. In their paper [4], Rivest, Shamir and Adler propose a signing system, in which the Alice decrypts her plaintext message, and sends it to Bob (with the message). Since $E(D(M)) = M$, Bob can encrypt the signature with Alice's public key, and check that the messages match. This can be optimised by hashing (a one way map) the message before decrypting it. That is $S = D(\mathcal{H}(M))$. Since hashes are a fixed length we can require the hash to be smaller than the size of a message block, and only need to transmit one more block instead of twice as many. This signature algorithm is very rarely used, in favour of DSA, or ECDSA [citation needed].

Importantly, RSA is only as secure as its randomness. Other encryption algorithms that rely on cyclic groups (that are not $\mathbb{Z}/n\mathbb{Z}$). If the group is such that there is some knowledge that makes it trivial to reverse some group operation, security firms, or governments who created the standardised group may be able to decrypt any message with the standard. An example of this is the US NIST's deterministic psudeorandom number generation algorithm Dual_EC_DRBG. The RSA Security company released this standard as part of their RSafe program, after a $10 million payment from the NSA [2]. This would have allowed the NSA to know what your values for p, q were *when you generated them.* This was part of Operation Bullrun, an NSA to crack encryption of online communications. This operation was undertaken with strong support from the UK's GCHQ, in Operation Edgehill [1].

3.1 Blockchain

At its core, a blockchain is a long list of messages that lots of people have access to (a distributed ledger). But how can one ensure that Alice did in fact send Bob something. Alice can sign her transaction with her public key. Her public

key is typically her address, or identifier within the network, so we know the person who has the private key to the "account" something was initially sent to has signed the transaction.

However, more utility can be gained using mutli-sig. All n members of a group who want to exchange some thing broadcast this to the blockchain. To do this, any one member of the group can submit n public keys to the network, not necessarily including their own. On creation, the creator tells the network how many group members need to agree to move funds.

3.2 PGP

PGP - Pretty Good Privacy - is an open source set of standards to encrypt and decrpyt messages, and create a "web of trust". To encrypt, first the encrypter generates a random (symmetric) encryption key. The message is encrypted with this key, and the key is encrypted with each of the recievers' private keys. Each of these reciever-specific encrypted keys is prefixed with the reciever's fingerprint - a shorter representation of their private key.

To decrypt, the reciever locates their fingerprint, and decrypts the corresponding message. They then use the message in this key to decrypt the main message.

PGP also offers signatures, which can be done with either RSA or DSA. Since any message can be signed, other people's public keys can be signed. PGP creates a "web of trust" by allowing users to publish a signed public key. If a large number of people you trust (read people whose public key you have) sign a public key, you can be reasonably convinced the public key is legitimate.

3.3 TLS

TLS - Transport Layer Security - is responsible for securing HTTP communications. TLS relies on no single encryption or signature standard, instead agreeing a "cipher suite" as its first step. To convince your browser you're speaking to the right server, the server sends a signed message (certificate) including information about who they are, and a message "vouching" for the certificate's legitimiacy. These "vouches" are signatures by Certificate Authorities: entities that are trusted to sign certificates.

Now the client knows the server is who they say they are, they can either choose a random number, encrypt it with the server's public key and send it. This is now the symmetric key. Or, they can perform Diffie-Hellman Key Exchange [3]. The second methodi is preferred since should the server's private key be compromised, no communications can be decrypred. This protects against Man In The Middle attakcs, where an attacker intercepts traffic between you and the genuine server.

4 Examples

4.1 Simple Example

$$\begin{aligned} &\text{Let } p = 5, q = 11, \\ \Rightarrow\ & n \coloneqq (5) \cdot (11) \\ &= 55 \\ \Rightarrow\ & \phi(n) = ((5) - 1) \cdot ((11) - 1) \\ &= 40 \end{aligned}$$

We must choose d such that d^{-1} exists in $\mathbf{Z}_{40}$. Let's choose $e = 3 \Rightarrow d = 27$.
Let's encrypt the first few digits of π, $M = \{31, 14\}$.

Encryption:

$$\begin{aligned} 31^e &\equiv 31^3 \ (mod\ 55) \\ &\equiv 31 \cdot 26 \ (mod\ 55) \\ &\equiv 36 \ (mod\ 55) \end{aligned}$$

$$\begin{aligned} 41^e &\equiv 41^3 \ (mod\ 55) \\ &\equiv 41 \cdot 31 \ (mod\ 55) \\ &\equiv 6 \ (mod\ 55) \end{aligned}$$

Decryption:

$$\begin{aligned} 36^d &\equiv 36^{27} \ (mod\ 55) \\ &\equiv 36 \cdot 31 \cdot 16 \cdot 36 \ (mod\ 55) \\ &\equiv 31 \ (mod\ 55) \end{aligned}$$

$$\begin{aligned} 6^d &\equiv 6^{27} \ (mod\ 55) \\ &\equiv 6 \cdot 36 \cdot 26 \cdot 16 \ (mod\ 55) \\ &\equiv 41 \ (mod\ 55) \end{aligned}$$

4.2 Large Example

$$\begin{aligned} &\text{Let } p = 12412304997166831007, q = 16909567760735815829, \\ \Rightarrow\ & n \coloneqq (12412304997166831007) \cdot (16909567760735815829) \\ &= 209886712416512307428626688728718609803 \\ \Rightarrow\ & \phi(n) = ((12412304997166831007) - 1) \cdot ((16909567760735815829) - 1) \\ &= 209886712416512307399304815970815962968 \end{aligned}$$

For large examples, it is easier to use a reasonable size known prime than to check manually for divisors.
Let's choose $e = 65537 \Rightarrow d = 14805472809276231702808889089111222619 3$.
We have 128 bits in n, so we can encrypt up to 16 bytes (read characters). Using (8-bit) ascii:

$$\begin{aligned} \text{"Hello World"} &=_2 \text{01001000 01100101 01101100 01101100 01101111 00100000} \\ &\quad \text{01010111 01101111 01110010 01101100 01100100} \\ &=_{10} 875216180888825337921158 12 \end{aligned}$$

Now we have a numeric representation of our number, we can encrypt.

$$\begin{aligned} M^e &\equiv 875216180888825337921158 12^{65537} \ (mod\ n) \\ &\equiv 209682609758627220578365935367187247507 \ (mod\ n) \end{aligned}$$

$$\begin{aligned} (M^e)^d &\equiv 209682609758627220578365935367187247507^{1480547280927623170280888908911122261 93} \ (mod\ m) \\ &\equiv 875216180888825337921158 12 \ (mod\ n) \\ &\equiv M \ (mod\ n) \end{aligned}$$

5 Code

The code can be found at `https://github.com/jameslounds/142_RSA`

References

[1] J. Ball, J. Borger, and G. Greenwald. Revealed: how us and uk spy agencies defeat internet privacy and security. *The Guardian*, 2013.

[2] Daniel R. L. Brown. Conjectured security of the ansi-nist elliptic curve rng. *Reuters*, 2006.

[3] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[4] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.