

# RSA

proidiot

DC405

# What is RSA?

- RSA is a very popular public key cryptography algorithm.
- RSA was described by Ron **R**ivest, Adi **S**hamir, and Leonard **A**dleman from MIT in 1978 (hence the name).
- Because RSA is so strong and was also the first public key cryptography algorithm known to be useful for both encryption and signing, RSA is widely considered to be the first practical public key cryptography system.

# Before We Get Started...

In order to understand how RSA works, there are a few other concepts you must first be familiar with:

- Modular Arithmetic
- Prime Numbers
- Coprimality
- The Totient Function

# Modular Arithmetic

- The modulo operator maps an unconstrained value in the constrained universe bound by 0 and the modulus.
- The modulo function is by definition equivalent to finding the remainder in integer division.
- For example:
  - $15 \equiv 0 \pmod{5}$
  - $7 \equiv 3 \pmod{4}$

# More About Modular Arithmetic

- In C derivatives, you would find the modulus using syntax like:
  - $d = a \% b$
- The above example may be written as an equation rather than a congruence using another constant positive integer  $k$ :
  - $a = (k * b) + d$
- It is important to know that the modulo operation is considered “destructive”; by using the modulo operator we no longer know  $k$  and so we (usually)

# Prime Numbers

- A prime number is defined to be a positive integer (other than 1) whose only factors are 1 and itself.
- For example:
  - 7 is a prime number since the only numbers that can be multiplied together to get 7 are 1 and 7.
  - 12 is not a prime number since 2, 3, 4, and 6 are all factors of 12 (in addition to 1 and 12).
- It should be noted that 1 is not prime.

# Coprimality

- Two numbers are coprime if they do not share any factors other than 1.
- For example:
  - Since 4 (whose factors are 1, 2, and 4) and 9 (whose factors are 1, 3, and 9) do not share any factors (other than 1), 4 and 9 are coprime.
  - Since 6 (whose factors are 1, 2, 3, and 6) and 9 both have 3 as a factor, 6 and 9 are not coprime.

# More About Coprimality

- Every number is coprime with 1 (since coprimality is only concerned with any common factors ***other than*** 1).
- No number other than 1 is coprime with itself (since every number has itself as a factor).
- A prime number is coprime with every number smaller than itself (makes sense if you think about it).

# The Totient Function

- The totient of a positive integer  $n$  (denoted as  $\varphi(n)$ ) is the number of positive integers less than or equal to  $n$  that are coprime with  $n$ .
- For example:
  - $\varphi(9) = 6$  because 9 is coprime with 1, 2, 4, 5, 7, and 8, but not 3 or 6 or 9.
  - $\varphi(7) = 6$  because 7 is prime, and so it is coprime with every number smaller than itself.

# More About The Totient Function

- As you've just seen, the totient of any prime number is one less than that number.
- The totient function is multiplicative, so if  $n$  and  $m$  are coprime, then:  
$$\varphi(n * m) = \varphi(n) * \varphi(m)$$
- In the event that  $n$  and  $m$  are prime numbers, the totient of the product of  $n$  and  $m$  is easy to calculate:  
$$\varphi(n * m) = (n - 1) * (m - 1)$$

# What is the RSA Algorithm?

- Use two very large prime numbers to generate the public and the private two-part keys.
- Convert the plaintext message into a series of numbers, and then encrypt that series of numbers into ciphertext using the public key.
- Decrypt the ciphertext using the private key, and then convert the resulting series of numbers back into the original plaintext.

# Key Generation

1. Choose two different prime numbers  $p$  and  $q$ .

2. Compute the product  $n$  of  $p$  and  $q$ .

$$n = p * q$$

3. Compute the totient of  $n$ .

$$\varphi(n) = \varphi(p * q) = (p - 1) * (q - 1)$$

4. Choose whatever positive integer you like between 1 and  $\varphi(n)$  (so long as that integer is also coprime to  $\varphi(n)$ ) and call that integer  $e$ .

5. Calculate the smallest  $d$  that satisfies the congruence:

# Key Generation (Cont.)

- In this case,  $n$  will be used as the modulus for encryption and decryption.
- The encryption exponent ( $e$ ) will be used in encryption.
- The decryption exponent ( $d$ ) will be used in decryption.
- So, the public key is  $(e, n)$ , and the private key is  $(d, n)$ .

# Encryption

1. Convert the plaintext into a series of integers, each of which is between 0 and  $n$ .
2. For the first number in the series ( $m$ ), find the corresponding value for the ciphertext using the public key and the following congruence:  
$$c \equiv m^e \pmod{n}$$
3. Continue to find the corresponding ciphertext values for each of the numbers in the plaintext series, and then send the resulting ciphertext series to the person who has the private key.

# Decryption

1. For the first number in the ciphertext series ( $c$ ), use the private key and the following congruence to find the corresponding value in the plaintext series:

$$m \equiv c^d \pmod{n}$$

2. Continue performing this calculation for each of the rest of the numbers in the ciphertext series in order to find the rest of the plaintext series.
3. Finally, convert the numerical series representation of the plaintext back into the original plaintext.

# Example Keygen (Finally!)

1. Let's have 61 and 53 be our prime numbers.
2. So our modulus will be  $61 * 53 = 3233$ .
3. Since we started with prime numbers, we know  $\phi(3233) = (61 - 1) * (53 - 1) = 3120$ .
4. Since 17 is prime, it's obviously coprime to 3120, so we'll use it for our encryption exponent.
5. Using a nasty-looking algorithm, we would see that 2753 is the smallest corresponding decryption exponent.

# Example Encryption

Using our public key (which is 3233, 17), if we have a message that is represented as 123 in numerical form, we calculate the ciphertext using  $123^{17} \pmod{3233}$ , which gives 855.

# Example Decryption

Using our private key (which is 3233, 2753) and the ciphertext (855), we can decrypt the message by finding  $855^{2753} \pmod{3233}$ , which, sure enough, gives us 123.

# Why Does This Work?

- Let's look at the decryption equation again:
  - $m \equiv c^d \pmod{n}$
- By using the encryption equation to substitute for  $c$ , we get:
  - $m \equiv m^{e * d} \pmod{n}$
- If you remember the equation for the modulus and how we defined  $d * e$ , we see that:
  - $d * e = (k * \phi(n)) + 1$

# Why Does This Work? (Cont.)

- Know our equation looks like:
  - $m \equiv m * m^{k * \phi(n)} \pmod{n}$
- The following (called Euler's equation) is known to be true when  $a$  and  $n$  are coprime:
  - $a^{\phi(n)} \equiv 1 \pmod{n}$
- So now we have it:
  - $m \equiv m \pmod{n}$  (obviously true since  $m < n$ )

# Where is RSA Used?

- PGP/GPG Encryption
- PGP/GPG Key Signing
- SSL Signatures

# How Do I Break RSA?

- Well, you could do it wrong if you don't pay attention...
- We don't really care about  $k$  (thank you, natural numbers!), so all we really want to know is  $\varphi(n)$ .
- Since we know  $\varphi(n) = (p - 1) * (q - 1)$ , using FOIL from Algebra 1 shows us that  $\varphi(n) = (p * q) - p - q + 1$ , and since  $(p * q) = n$ , all we need to do is factor  $n$ .
- If you know of a non-quantum algorithm that can do integer factorization faster than exponential time, please let me know. I'll pay