1. To start any two large prime numbers must be chosen, we will call them *p* and *q*.

2. Modular arithmetic will be applied. We will use mod *n* and mod *n* where *n* = *p* \* *q*

3. The sender of data must choose an *e* where *e* is co-prime to (*n*). Co-prime means that both e and (*n*) share the greatest common devisor of 1. Note that *e* can be any number it does not have to be a prime number.

4. (*n*) is found by finding (*p*) and (*q*) and multiplying them together. (*p*) = *p* -1 therefore (*q*) = *q* - 1 leading to (*p*) \* (*q*) = (*n*)

Note that it is common for *p* and *q* to have exponents to generate large numbers so know that () = - and () = - , again (*n*) = (*p*) \* (*q*)

5. A number *d* must be worked out such that *ed* 1 mod (*n*).

Note that *d* is the inverse of *e*. *d* = .

6. Euclids Algorithm must be applied to (*e,* (*n*)) to ensure the greatest common divisor (gcd) is 1. If the gcd does not equal 1 then RSA will not work.

To illustrate Euclids Algorithm (*n*) = and *e* = to make it simpler to read. The log value of *r* will increment when a new *r* is found. Lastly *s* will be the sum of all instances of *e* found within (*n*), in other words how many times you can add *e* up before surpassing the value of (*n*). if *e* surpasses *n* the final *e* value is redacted and the remainder is found. This is an iterative process until 1 is found, an infinitely repeating pattern is found or 0 is found.

7. Once 1 is successfully found back tracking must implemented, you will start with:

Eventually you will backtrack to:

Now we can swap back (*n*) and . This will give us

Because should be *d* we can swap and *e* around to better match the equation in point 5 on the previous page. The equation will now look like .

Note that mod *n* = 0. If this is not equal to 0 a mistake was made.

8. Thus 0 = 1. . Remember that *d* = (inverse).

Therefore has been turned into *ed* 1 mod (*n*).

Now a plaintext message *P* can be encrypted with a recipient’s public key (*n*, *e*) using the equation mod *n* which will transform the message from plaintext to ciphertext *C.* The recipient of the message can take the encrypted ciphertext *C* and use their secret private key (*n*, *d*) and apply the formula mod *n* to convert the ciphertext back into plaintext.

With all the key generation now defined, how to create a digital signature can now be illustrated. Technically only the public and private key of the sender is used to implement an RSA digital signature not the receiver, but the receivers public key will have to be used to cipher the plaintext unless the parties involved do not mind the information being read by strangers.

1. The sender will now cipher their plaintext message *P* into a ciphertext message *C* which will then be put through the message digest algorithm (MDA). This MDA is a cryptographic hashing function that uses the secure hashing algorithm SHA-256. The *C* gets transformed into a small, fixed length hash string called a message digest *MDS*.

Note that a cryptographic hash function is one way, it is not designed to be reversed and doing so is infeasible even for powerful computing systems.

2. This is then encrypted with the senders private key *d* which then creates the digital signature *DS*.

3. The ciphertext *C* which is then sent to the recipient along with the *DS*. Security is enhanced here thanks to the fact that *C* and *DS* do not have to be sent together down the same route.

4. Having received *DS* and *C* the recipient now puts *C* through the same message digest algorithm to produce their own message digest *MDR*.

5. The recipient will now decrypt the *DS* with the sender’s public key, revealing the original *MDS* sent to them by the sender.

6. Lastly *MDS* and *MDR* are checked to see if they are equal.

If both message digests are equal the receiver can be assured that the message and signature were not altered in transit and they are also assured that the message came from the desired sender and not someone posing as them. If the message digests are not equal, you do not trust and reject the message.

Diagram

Description automatically generated

RSA digital signature diagram

**(ii)**

The first letter in my name is R so that will equate to the numerical value of 17. Finding *n* is not necessary because it has been given to me and is equal to 33 and *e* has also been provided which is equal to 3. This makes my public key equal to (33, 3).

The formula for encryption is mod *n*, note that *P* = R so the equation will be:

Hence the encrypted form of R = 17 is 29.

To ensure that 29 is correct we can find *d* which in this case is 7, this was easy to determine because the *p* and *q* values were easy to derive from 33 which were 11 and 3 respectively.

Now we have the equation:

This answer is correct as R was determined to be equal to 17.

This equation is quite big and is difficult for scientific calculators to compute. Large numbers can be broken down into smaller components, note that any number can be expressed as a sum of exponents of 2. This is useful when dealing with large exponential values. 7 in the decryption equation can be expressed as 4 + 2 + 1.

So computing values of and so on and repeatedly squaring successive values and applying mod 33 is an alternative approach. We only need do . is already determined to by 29 as 29 mod 33 = 29.

= 841 16, therefore = 16 \* 16 = 256 25.

Now we have which is equal to

7 mod 33

So, 11,600 mod 33 = 17 = R

A number that is 5 digits long is much easier to work with than a number that is 11 digits long.