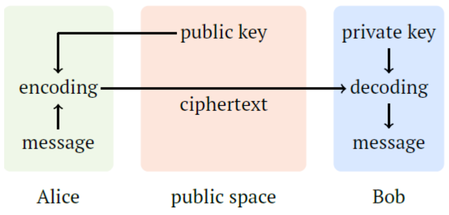
RSA Cryptosystem

RSA is a public key cryptosystem that allows two parties to exchange information securely even if they don't share a secret key yet. Programs based on RSA are among the most frequently run: online shopping, banking, digital signatures, authentication, etc. The cryptosystem is named after its inventors Ron Rivest, Adi Shamir, and Leonard Adleman who published it in 1977. In 2002, they received the Turing Award, one of the most prestigious computer science awards, "for their ingenious contribution for making public-key cryptography useful in practice".

Ron Rivest (born 1947), Adi Shamir (born 1952), and Leonard Adleman (born 1945). Source: Wikipedia.

RSA is an algorithm for *asymmetric* or *public key* encryption. This distinguishes it from one-time pad where both parties share a private key and use it for both encryption and decryption (hence, one-time pad is symmetric and private key). In RSA, a public key is used for encryption and a private key is used for decryption. Namely, Bob generates two random keys: a public key and a private key. Then, Bob publishes the public key for anyone to access. This way, *anyone* can encrypt a message for Bob. However, it is Bob only who can decrypt an encrypted message efficiently: with the private key, it is easy to decrypt, whereas it is not currently known how to decrypt efficiently without using the private key.



Keys

To generate a pair of keys, Bob generates two big random primes p and q (consisting of several thousand bits) and computes $n = pq$. Then, Bob takes an integer e coprime with $\varphi(n) = (p-1)(q-1)$ (recall that φ is Euler's totient function: $\varphi(n)$ is the number of integers from 1 to $n-1$ coprime with n). Bob publishes (say, on his webpage) a pair of numbers (n, e) as a public key. Using this key, anyone can encode a message for Bob. (We explain below how this should be done.) Then, Bob computes the modulo inverse d of e modulo $\varphi(n)$:

$$de \equiv 1 \mod \varphi(n)\,.$$

Recall that it can be computed efficiently using the generalized Euclid algorithm. The private key is the pair (n, d) and Bob keeps it secret.

Encoding and Decoding

In RSA cryptosystem, a message m is an integer from 0 to $n-1$. Thus, if Alice wants to send a message to Bob, she first needs to convert it to an integer. A common way to achieve this is to map every symbol of the message to an integer and then to concatenate the resulting integers.

To encode m, Alice takes Bob's public key (n, e) and raises m to the power of e modulo n:

$$c \equiv m^e \mod n\,.$$

As we discussed in the previous module, such modular exponentiation can be performed efficiently.

It turns out, that to decode Bob can also compute a modular exponentiation (using his private key d):

$$m \equiv c^d \mod n\,.$$

We will prove that decoding works correctly below. Before this, let us give an example.

Example

For demonstration purposes, in this example, Bob uses relatively small prime numbers (in practice, it is recommended that p and q are about four thousand bits long).

```
from math import gcd
p, q = 5915587277, 2860486313

n = p * q
print(f'n={n}')
```

Run

Reset

```
n=16921456439215439701
```

To prepare the keys, Bob computes $\varphi(n) = (p-1)(q-1)$ and takes e such that $\gcd(e, \varphi(n)) = 1$. Then, (n, e) is his public key.

```
phi = (p - 1) * (q - 1)
print(f'phi={phi}')
e = 3
print(f'e={e}')
assert gcd(phi, e) == 1
```

```
phi=16921456430439366112
e=3
```

To prepare the private key, Bob computes $d = e^{-1} \mod \varphi(n)$. In python (starting from version 3.8), this can be done using the **pow** method. Bob keeps d secret.

```
assert d * e % phi == 1
```

```
d=11280970953626244075
```

To encode her message $0 \le m < n$, Alice computes $c = m^d \mod n$. To decode, Bob computes $c^e \mod n$. As the following code snippet shows, at least in this case, the message is indeed restored.

```
def encode(m):
    assert 0 <= m < n
    return pow(m, e, n)


message = 92616855427
print(f'message: {message}')
cipher_text = encode(message)


def decode(c):
    return pow(c, d, n)


print(f'decoded: {decode(cipher_text)}')
```

```
message: 92616855427
decoded: 92616855427
```

As usual, we encourage you not only to read this python code snippet, but also to use it as an interactive example. You may want to change the primes to, say, $p = 5$ and $q = 11$ so that it is easier to verify all the computations.

Efficiency

Encoding and decoding in RSA boils down to computing modular exponentiation and hence is efficient. To verify this, you may Google for prime numbers with several hundred (or even thousand) digits and check that all the computations in the code above are performed quickly.

Generating large random primes (Bob's initialization phase) is a non-trivial task, but it still can be efficiently implemented: one generates a large random number and then checks whether it is a prime. If it is composite, one iterates. The prime number theorem guarantees that there are many primes and this, in turn, ensures that this process converges quickly. Currently, the most efficient algorithms for checking whether a given integer k is prime are randomized: roughly, one selects random $0 < a < k$ and checks whether a and k satisfy $a^{k-1} \equiv 1 \mod k$; if they don't, we know for sure that k is composite (otherwise it would violate Fermat's Little Theorem). A curious property of this probabilistic check: it can convince us that k is composite without specifying any non-trivial divisor of k.

Correctness

We are ready to prove that the RSA cryptosystem is correct.

Theorem

For any $0 \le m < n$,

$$(m^e)^d \equiv m \mod n\,.$$

Proof

Since d is the inverse of e modulo $\varphi(n)$, there exists an integer k such that $de = k\varphi(n) + 1$. Assuming that $\gcd(m, n) = 1$, we have that $m^{\varphi(n)} \equiv 1 \mod n$, by Euler's Totient Theorem. Thus,

$$(m^e)^d \equiv m^{ed} \equiv m^{k\varphi(n)+1} \equiv (1)^k m \equiv m \mod n\,.$$

In case $\gcd(m, n) > 1$, we know that m is divisible by either p or q. It cannot be divisible by both, since $m < n$. Assume, without loss of generality, that p divides m. Then, $\gcd(m, q) = 1$. In this case, $m^{ed} \equiv 0 \mod p$. By the argument above, $m^{ed} \equiv m \mod q$. Since, $m < n = pq$, we conclude that $m^{ed} \equiv m \mod n$, by the Chinese Reminder Theorem.