main

September 15, 2023

Let's walk through some basics of this SageMath-extended Jupyter notebook.

Jupyter notebooks consist of cells, which may contain python code or text. Code cell can be executed and execution results are displayed below it. Text cell can use embedded LaTeX typesetting.

Python can work on numbers of arbitrary length.

SageMath is a mathematics software system extending python with additional utilities. SageMath notebooks can be easily ran locally or using services such as CoCalc (like this instance)

[1]:
```
a = 5 # assignment
b = 2 ** 65 # two to the power of 65, a number outside of what computer CPUs␣
 ↪usually handle
a # a single final line with no assignment only will display its value
```

[1]:
```
5
```

[2]:
```
print(f"The value of a is {a}, which has {len(str(a))} decimal digits. Is it a␣
 ↪prime number? a.is_prime() == {a.is_prime()}")
print(f"The value of b is {b}, which has {b.ndigits()} decimal digits. Is it b␣
 ↪prime number? b.is_prime() == {b.is_prime()}")
```

```
The value of a is 5, which has 1 decimal digits. Is it a prime number?
a.is_prime() == True
The value of b is 36893488147419103232, which has 20 decimal digits. Is it b
prime number? b.is_prime() == False
```

1 Key generation

To generate RSA key pair, we first choose two random prime numbers p and q that are similar in size.

[3]:
```
p = 13
q = 17
```

Compute $n = p * q$

[4]:
```
n = p * q
n
```

[4]: 221

Compute least common multiple of $(q-1, p-1)$

[5]:
```
l = math.lcm(q-1, p-1)
l
```

[5]: 48

Choose a number $e \in (1, 48)$ that is coprime with 48

[6]:
```
for i in range(2, l):
    if math.gcd(i, 48) == 1:
        print(i)
```

```
5
7
11
13
17
19
23
25
29
31
35
37
41
43
47
```

Let $e = 5$

[7]:
```
e = 5
e
```

[7]: 5

Find d such that

$1 \equiv (e * d) \mod l \equiv (5 * d) \mod 48$

This can be expressed as calculation of modular multiplicative inverse of e modulo l, similarly to how we can calculate regular multiplicative inverse of $a * x = 1$ as $x = \frac{1}{a}$.

Computers can quickly calculate this value using extended Euclidean algorithm.

[8]:
```
d = inverse_mod(e, l)
d
```

[8]: 29

At this point we have concluded key generation.

As a reminder, RSA principle is $(m^e)^d \equiv m \mod n$.

In our example the public (**e**ncryption) key is a pair $(n = 221, e = 5)$ and private (**d**ecryption) key is pair $(n = 221, d = 29)$.

2 Encryption and decryption

Using RSA equation of form $(m^e)^d \equiv m \mod n$ lets encrypt a value of $m = 22$.

[9]:
```
m = 22
m
```

[9]: 22

[10]:
```
m ** e
```

[10]: 5153632

[11]:
```
encrypted_m = (m ** e) % n
encrypted_m
```

[11]: 133

We can now send public-key encrypted message 133 over insecure channel to owner of corresponding private key for decryption.

Let's see how decryption works.

[12]:
```
encrypted_m ** d
```

[12]: 3905688365736713961422773905256227860934294910937622927430165 3

[13]:
```
decrypted_m = (encrypted_m ** d) % n
decrypted_m
```

[13]: 22

3 Signing and verification

... but wait, there's more.

Asymmetric cryptography can be used for signing and verification, where holder of private key signs a message which authenticity can be later verified using corresponding public key.

Let's create a signature s of the same message $m = 22$ using our private key:

$s \equiv m^d \mod n$

[14]:
```
m ** d
```

[14]: 851643319086537701956194499721106030592

[15]:
```
s = (m ** d) % n
s
```

[15]: 3

We would usually distribute message m and its signature s publicly.

And now verify signature:

$(m^d)^e \mod n \equiv s^e \mod n$

[16]:
```
s ** e
```

[16]: 243

[17]:
```
(s ** e) % n
```

[17]: 22

Reminder to switch sharing to other tab.