RSA Algorithm Implementation in Python

Erikas Eigėlis
Vinius University Kaunas Faculty
Information Systems and Cyber Security (ISKS6)

Exponentiation by squaring

Description

- This method is based on the observation, that for a positive integer n we have:

$$x^n = \begin{cases} x\,(x^2)^{\frac{n-1}{2}}, & \text{if } n \text{ is odd} \\ (x^2)^{\frac{n}{2}}, & \text{if } n \text{ is even.} \end{cases}$$

- In python this is implemented by using recursive function, until the power is equal to 1

Python implementation

```
def pow_es(self, base, power):
    # Raise to power by using exponentiation by squaring
    # If power is equal to 1...
    if power == 1:
        # return base value (stop recursion cycle)
        return base
    # If power is even...
    if power % 2 == 0:
        # call same function with squared base half power
        return self.pow_es(base * base, power / 2)
    # If power is odd...
    else:
        # multiply base by value of same function with
        # squared base and half of (power - 1) as arguments
        return base * self.pow_es(base * base, (power - 1) / 2)
```

Finding inverse number

Description

- Finding inverse number (in this case the one that q mod p = 1) is achieved by checking if xq mod p is equal to 1
- This is achieved by by incrementing x from 0 until such number is generated

Python implementation

```python
def q_inv(q, p):
    x = 0
    # Do while xq%p is not equal to 1:
    while x * q % p != 1:
        # Increment x by one:
        x += 1
    return x
```

Chinese remainder algorithm

Description

- p and q are prime numbers used for key generation
- $d_p = d \bmod (p-1)$
- $d_q = d \bmod (q-1)$
- $q_{inv} = q^{-1} \bmod p$
- These values allow the recipient to compute exponentiation $m = c^d \bmod (pq)$ efficiently as follows:
- $m_1 = c^{dp} \bmod p$
- $m_2 = c^{dq} \bmod q$
- $h = q_{inv}(m_1 – m_2) \bmod p$
- Final result (m) is: $m = m_2 + hq$

Python implementation

```
def chinese_remainder(self, c, p, q, d):

    dp = d % (p - 1)
    dq = d % (q - 1)

    qinv = self.q_inv(q, p)

    m1 = self.pow_es(c, dp) % p
    m2 = self.pow_es(c, dq) % q

    h = qinv * (m1 - m2) % p

    return m2 + h * q
```

Live showcase:
https://isks6.pythonanywhere.com/rsa/

View source code:
https://github.com/EErikas/PythonCryptoExample