RSA

Paweł Koch 145330, 16.05.2022

1.RSA

RSA (Rivest–Shamir–Adleman) is a public-key cryptosystem that is widely used for secure data transmission. It is also one of the oldest. The acronym "RSA" comes from the surnames of Ron Rivest, Adi Shamir and Leonard Adleman, who publicly described the algorithm in 1977. An equivalent system was developed secretly in 1973 at GCHQ (the British signals intelligence agency) by the English mathematician Clifford Cocks. That system was declassified in 1997.[1]

In a public-key cryptosystem, the encryption key is public and distinct from the decryption key, which is kept secret (private). An RSA user creates and publishes a public key based on two large prime numbers, along with an auxiliary value. The prime numbers are kept secret. Messages can be encrypted by anyone, via the public key, but can only be decoded by someone who knows the prime numbers.[2]

The security of RSA relies on the practical difficulty of factoring the product of two large prime numbers, the "factoring problem". Breaking RSA encryption is known as the RSA problem. Whether it is as difficult as the factoring problem is an open question.[3] There are no published methods to defeat the system if a large enough key is used.

RSA is a relatively slow algorithm. Because of this, it is not commonly used to directly encrypt user data. More often, RSA is used to transmit shared keys for symmetric-key cryptography, which are then used for bulk encryption-decryption.

2.Założenia

Liczby p i q nie powinny byc zbyt duże ze względu na wolne działanie algorytmu (jednak teoretycznie im większe tym lepsza jakośc generowanych kluczy a co za tym idzie bezpieczeństwa zaszyfrowanych informacji informacji)

3.Opis metod do wyznaczenia e i d:

Do generowania liczby e użyto najprostszej implementacji wyznaczającej wszystkie liczby względnie pierwsze a następnie wybieraj ąca loswoą z wygenerowanych liczb

```
def co_prime(n):
    coprimes = []
    for i in range(n):
        if gcd(i, n) == 1:
            coprimes.append(i)
    i = randint(0, len(coprimes)-1)
    return coprimes[i]
```

Do generowania liczby d także użyto najprostszego rozwiązania iterującego po wszystkich liczbach aż nie napotka odpowiedniej:

```
def gen_d(e, fi):
    for d in range(fi):
        if (e * d - 1) % fi == 1:
            return d
```

4. Opis realizacji zadania

Do wyznaczania liczby pierwsze o okreslonej liczby cyfr użyto prostego algorytmu wyszukiwania liczb pierwszych:

```
def gen_prime(digits: int):
    lower = 10**(digits-1)
    upper = 10**digits
    primes = []
    for num in range(lower, upper):
        if num > 1:
            for i in range(2, floor(sqrt(num))):
                if (num % i) == 0:
                    break
            else:
                primes.append(num)
    i = randint(0, len(primes)-1)
    return primes[i]
```

Nastepnie w połaczeniu z algorytmami z punktu 3 utworzono generator kluczy:

```python
def rsa():
    p, q = gen_prime(4), gen_prime(4)
    n: int = p * q
    fi = (p-1)*(q-1)
    e = co_prime(fi)
    d = gen_d(e, fi)
    print(f"P: {p} Q: {q}")
    print(f"Fi: {fi}")
    print(f"Public key e: {e} n: {n}")
    print(f"Private key d: {d} n: {n}")

    return ((e, n), (d, n))
```

Zwracane są oba klucze

Nastepnie do szyfrowania użyto nastepujących metod (UWAGA bardzo czasochłonne!!)

```python
def simple_math(m: int, power: int, modulo: int):
    return m**power % modulo


def encrypt(message, key: tuple):
    encrypted = []
    for ch in message:
        encrypted.append(simple_math(ord(ch), key[0], key[1]))
    return encrypted

(function) decrypt: (message: Any, key: tuple) -> list
def decrypt(message, key: tuple):
    decrypted = []
    for ch in message:
        decrypted.append(chr(simple_math(ch, key[0], key[1])))
    return decrypted
```

Bylo to jednak rozwizanie zbyt wolne (wymagało zaszyfrowania każdego znaku wiadomości z osobna co było piekielnie czasochłonne). zastozowano zatem rozwiazanie które wyznacza sume wartosci ASCII szyfrowanej wiadomosci (szyfrowanie takie i tak jest bardzo wolne). Znaki zostaja zmaienione na wartosci ASCCI nastepnie zsumowane i zaszyfrowane, przydeszyfrowaniu jest sprawdzane tylko cz odszyfrowana liczba jest równa sumie zaszyfrowanych znaków.

```python
def simple_encrypt(message, key: tuple):
    encrypted = 0
    for ch in message:
        encrypted += ord(ch)
    return simple_math(encrypted, key[0], key[1])


def simple_decrypt(message, key: tuple):
    decrypted = simple_math(message, key[0], key[1])
    return decrypted
```

5. Odpowiedzi na pytania

Jakie elemnty algorytmu są trudne w realizacji?
Sam algorytm nie należy do najtrudniejszych jednak korzysta z bardzo czasochłonnych obliczeń wymagajacych dużych możliwości zarówno procesora jak i pamięci.
Obliczanie liczb pierwszych jest bardzo czasochłonne, wyznaczanie liczby wzglednie pierwsze też jest czasochlonne w powyższej implementacji i dodatkowo na koniec liczby są podnoszone do ogromnych potęg co jeszcze bardziej spowalnia obliczenia.

Co stanowi o bezpieczeństwie i jakości tego algorytmu szyfrowania?
Bezpieczeństwo algorytmu opiera sia na posiadaniu 2 rodzajów kluczy które są ze sobą połaczone, w przypadku gdy ktoś zaszyfruje waidmośćdo nas naszym kluczem publiczym nikt inny nie jest w stanie tej wiadomosci odszyfrować mimo posiadania naszego klucza publicznego. tylko osoby z naszym kluczem prywatnym bed a w stanie odszyfrować wiadomość co jest głowną zaletą RSA. Sama jakośc agloyrmu jest zalezna od jak wysokich liczb pierwszych użyjemy do szyfrowania, im wieksze i z większym zakresem liczb pomiędzy nimi tym lepiej, takie liczby utworza bardzo trudny do złamania klucz który może posłużyć do szyfrowania o wysokim sopniu bezpieczeństwa.

6.Wnioski:

RSA jest bardzo bezpiecznym algorytmem, bedącym w powszechnym użyciu. Stanowi świetne zabezpieczenie, które bardzo ciężko zaatakowac. Ma jednak swoje wady, jest bardzo powolny i korzysta z własności liczb pierwszych co sprawia, że gdyby znaleziono na nie wzor matematyczny to zaszyfrowane za pomocą RSA wiadomości mogłyby być narażone na złamanie w bardzo szybkim czasie. Obecnie jednak jest szeroko wykorzystywanym algorytmem szyfrowania na którym opiera się dużo nowożytnej kryptografii.