Homework 05

HOMEWORK 05, DUE BY 2022-03-03

Question 1: Define the Fibonacci sequence as follows:

$$\begin{cases} F_0 = 0, \\ F_1 = 1, \\ F_{n+2} = F_{n+1} + F_n, \text{ where } n \geq 0 \end{cases}$$

Prove the following statement for any $n \in \mathbb{Z}^+$:

$$\sum_{k=1}^{n} F_{2k} = F_{2n+1} - 1.$$

Question 2: What is the original message encrypted using the RSA cryptosystem with $N = 113 \cdot 127$, and public exponent $e = 257$, if the encrypted message is `[1024, 12523, 10691]`. Each number in the list is to be decrypted separately by raising to the power of *private exponent* d modulo N.

To decrypt, first find the decryption exponent d which is the multiplicative inverse of e modulo $\varphi(N)$.

Note: Multiplicative inverses can be computed using the widget https://bit.ly/3rYFELL or a Python function `mod_inverse(...)` (see source code in Question 6 below). RSA cryptography is described in (Rosen2019), page 316.

Question 3: This problem could be tedious to compute on paper; consider using Python scripts or something similar.

(A) For each number $a \in \{1, 2, 3, \ldots, 18\}$ find its *multiplicative order* modulo 19: it is the smallest positive integer k such that $a^k \equiv 1 \pmod{19}$. Fill in all entries in this table:

a	1	2	3	4	$\cdots$	17	18
Multiplicative order of a	1				$\cdots$		

(B) For every number $a \in a \in \{1, 2, 3, \ldots, 18\}$ which is **not** a primitive root (modulo 19) find the discrete logarithm (index) of a to the base $r = 2$ modulo 19: $\text{ind}_2\, a \pmod{19}$.

(C) Alice and Bob have selected prime number $p = 19$ and a primitive root $r = 2$ for their Diffie-Hellman key exchange. Alice's secret exponent is $a = 11$, but Bob's secret is $b = 13$. Compute the message $A \equiv r^a \pmod{p}$ that Alice sends to Bob, and also the message $B \equiv r^b \pmod{p}$ that Bob sends to Alice.

Compute the secret key they have both agreed to in this key exchange. Diffie-Hellman key exchange is defined in (Rosen2019), page 319.

Question 4: Prove by mathematical induction that the number $k^{2^n} - 1$ is divisible by 2^{n+2}, if k is a positive odd integer, but n is any positive integer.

Question 5: Consider the following two propositions:

Proposition1: Let $n \in \mathbb{Z}^+$ be any positive integer. Then the following inequality is true:

$$\prod_{k=1}^{n} \frac{2k-1}{2k} \leq \frac{1}{\sqrt{3n}}.$$

Proposition2: Let $n \in \mathbb{Z}^+$ be any positive integer. Then the following inequality is true:

$$\prod_{k=1}^{n} \frac{2k-1}{2k} \leq \frac{1}{\sqrt{3n+1}}.$$

(A) Choose any of the two propositions and prove it using mathematical induction.

(B) Is any of the two propositions false, or are they both true? Justify your answer.

Question 6: Assume that the Python code given below implements RSA encryption. The public key is (N, e), where $N = 125060292466904545847084304952630886510097084 7 = p \cdot q$ is the product of two unknown primes, but public exponent is $e = 257$.

Unfortunately, an attacker has gained access to the `decrypt(message)` function as well (see Python source). The decrypt function does not contain the private exponent, but it uses a different exponent d_1 that can reverse the encryption of the public exponent $e = 257$.

(A) Factorize the modulus number N – express it as a product of two primes.

(B) Recover the proper private exponent d such that $d \cdot e \equiv 1 \pmod{\varphi(N)}$ and also $d < N$.

Note: You may need to apply some knowledge how to attack RSA algorithm. See https://stanford.io/3I6fQTG. RSA is still considered safe, if used properly; it is a popular public key method for encryption and electronic signatures. The above article summarizes known attacks discovered during 20 years of research.

For example, consider the following result (Page 3 of the article):

Fact 1: Let $(N; e)$ be an RSA public key. Given the private key d, one can efficiently factor the modulus $N = p \cdot q$. Conversely, given the factorization of N, one can efficiently recover d.

```
N = 125060292466904545847084304952630886510097084 7
e = 257

# A utility method to find multiplicative inverse a^(-1) (modulo n)
def mod_inverse(a, n):
    a1, v1 = 0, n
    a2, v2 = 1, a
    while v2 != 0:
        k = v1 // v2
        a1, a2 = a2, a1 - a2 * k
        v1, v2 = v2, v1 - v2 * k
    if a1 < 0:
        a1 += n
    return a1

# Public encryption function -- should be safe to show to everyone.
def encrypt(message):
    return pow(message, e, N)

# decrypt() is the inverse of encrypt(); exposes the RSA private key to attackers.
# (Even though the prime factors of N or the private exponent d<N are not shown.)
```

(continues on next page)

(continued from previous page)

```
def decrypt(message):
    big_num = 1157020808057230654431538840923638721379001278990144O
    d1 = mod_inverse(e, big_num)
    return pow(message, d1, N)

message = int(input("Enter a positive number to encrypt (up to 40 digits): "))
enc_message = encrypt(message)
print('Encrypted message: {}'.format(enc_message))
dec_message = decrypt(enc_message)
print('Decrypted message: {}'.format(dec_message))
```

This program lets the user enter a number (`message` variable), encrypts the number, then decrypts (should get back the original number).

```
Enter a number to encrypt (up to 40 digits): 2022
Encrypted message: 476636266497450731167878798199081427180792230
Decrypted message: 2022
```