Number Theory

Assignment 3 — MTH3251

RSA Cryptosystem

RSA is one of the most common cryptography algorithms. In this assignment, you will implement a basic RSA cryptosystem in Python, and implement some simple RSA attacks.

RSA algorithm deal with integers. So, in the following questions, you will need a function to convert a string to an integer `ConvertToInt`, and another function to convert an integer to string `ConvertToStr`. You can use the following functions:

```
def ConvertToInt(message_str):
    res = 0
    for i in range(len(message_str)):
        res = res * 256 + ord(message_str[i])
    return res
def ConvertToStr(n):
    res = ""
    while n > 0:
        res += chr(n % 256)
        n //= 256
    return res[::-1]
```

1. **Encryption and Decryption Modules**: First, you will need to implement the encryption and decryption modules as you learned in class. Please fill in the following functions, where

   - `m` is a string representation of the message to be encrypted.
   - `c` is an integer representation of the encrypted text.
   - `n` is the modular arithmetic modulo and `e` is the modular exponent. The pair $\{n, e\}$ is the RSA public key.
   - `p` and `q` are the prime factors of n. The pair $\{p, q\}$ is the RSA private key.

```
def Encrypt(m, n, e):
    ...
    return c
def Decrypt(c, p, q, e):
    ...
    return m
```

2. **Simple Attack**: Assume the sender needs to send only one the following messages:

    1. attack
    2. don't attack
    3. wait

    You knew this information by watching the sender's activity over time and connecting every statement to a potential message that agrees with the receiver's action to the ciphered message. Given a set of potential messages {`attack`, `don't attack`, `wait`}, fill in the following function to decipher the message using only the public key $\{n, e\}$.

```python
def DecipherSimple(c, n, e, potential_messages):
    ...
    return decipheredtext
```

3. **Small Prime Attack**: The sender made a mistake and used a small prime less than $1,000,000$ for one of the factors of the modulo n. Knowing this information, you can easily do an exhaustive search for this factor. Fill in the following function to decipher a message with a small prime factor using only the public key $\{n, e\}$.

```python
def DecipherSmallPrime(c, n, e):
    ...
    return decipheredtext
```

4. **Small Difference Attack**: The sender made a mistake by picking up two prime factors p, q with small difference $r = |p - q| < 5,000$. Knowing this information, fill in the following function to break the cipher using only the public key $\{n, e\}$

```python
def DecipherSmallDiff(c, n, e):
    ...
    return decipheredtext
```

5. **Common Divisor Attack**: The sender used the same random seed for generating two public keys. This mistake made the two public keys n_1 and n_2 have a common factor p while the second factor q is different. Knowing this information, fill in the following function to break the cipher using only the public key $\{n, e\}$.

```python
def DecipherCommonDivisor(c1, n1, e1, c2, n2, e2):
  ...
  return first_decipheredtext, second_decipheredtext
```

6. **Hastad's Attack**: The sender has to broadcast the same message to 2 different receivers using 2 different public keys: $\{n_1, e = 2\}$ and $\{n_2, e = 2\}$ with the same exponent $e = 2$. Knowing this information, fill in the following function to Implement Hastad's broadcast attack using the intercepted ciphertexts and the public keys.

```python
def DecipherHastad(c1, n1, c2, n2, e):
    ...
    return broadcast_message
```

- For every function you implement, you will need to test the its correctness using the testing script attached to this assignment. For this assignment, we will use Python only since C++ will need special libraries to do arithmetic in arbitrary precision to support huge RSA integers that cannot fit in regular C types.

- For solving these questions, you will need to use the following functions

```python
def GCD(a, b):
    if b == 0:
        return a
    return GCD(b, a % b)
def ExtendedEuclid(a, b):
    if b == 0:
        return (1, 0)
    (x, y) = ExtendedEuclid(b, a % b)
    k = a // b
    return (y, x - k * y)
def InvertModulo(a, n):
    (b, x) = ExtendedEuclid(a, n)
    if b < 0:
        b = (b % n + n) % n # we don't want -ve integers
    return b
def PowMod(a, n, mod):
    if n == 0:
        return 1 % mod
    elif n == 1:
        return a % mod
    else:
        b = PowMod(a, n // 2, mod)
        b = b * b % mod
        if n % 2 == 0:
          return b
        else:
          return b * a % mod
```

- Note: To get the square root or cubic root of an integer, you can do it first in floating-point precision, then round to integer number, then cast to integer type, i.e.,

```
sqrtN = int(round(np.sqrt(float(N))))
```

Due date for this assignment is 11:59pm, January, 10th, 2022.

This assignment was adapted from the RSA quiz in Number Theory and Cryptography course on Coursera website.

Omar Elgendy: oelgendy@ieee.org