CSE 345/545: Foundations to Computer Security
ASSIGNMENT 1 (TOTAL OF 150 POINTS)
Deadline: 25 Sep 2021

Instructions:

- Do your Assignment questions individually.
- Submit Python files individually as per guidelines.
- ***Do not* zip your submissions.**
- All the Queries, if any can be posted on google classroom.
- **Note: Keep your code efficient, make sure output matches the requirements.**
- This part uses the Python language library GMP You are required to install the library, and run all experiments.

---

1. Write a program in Python to convert plain english text into easy to remember but hard to crack password. [Ex: password -> P@s2w0rD]. Submit one python file with name <rollNo>_1.py.

   Input: a plain english string with no spaces. *Max length 100.*

   Output: a string of same length as input, but "hard to crack".

   [10 points]

2. Submit python files with names <rollNo>_2_a.py, <rollNo>_2_b.py, <rollNo>_2_c.py for the following question.

   a. Write a program in Python to generate a random number of 1023 digits, and extract an OTP of 6 digits. Use the Python standard library 'random'.

      Input: No input.

      Output: 1023 digit random number, 6 digit OTP. Each printed on separate lines.

   b. Use the GMP library in Python to generate random number of 1023 digits, and extract an OTP of 6 digits.

      Input: No input.

      Output: 1023 digit random number, 6 digit OTP. Each printed on separate lines.

   c. Time the above functions for 100 OTPs. Write a function that runs at least 80% faster than the previous implementation. [Hint: reuse the random number].

      Input: No input.

      Output: 100 OTPs. Each printed on separate lines.

      Constraint: Code should run in under 10 sec. All OTPs should be unique.

   [10 + 10 + 10 = 30 points]

Part II

Notes:

- This part uses the Python language library GMP You are required to install the library, and run all experiments.
- **Notations and algorithms in supplementary material** will be the standard for this assignment.
- Deliverables for programming part are python files in the format <RollNo>_3.py

---

1. Implement Diffie-Hellman in python using GMP library. Fill in the following functions.

   a. `bob_sends_alice(p, g)`

      Input: A large prime *p* and a large integer *g* upto 1023 digits long.

      Output: Public exchange *B*. Random integer *b*. Each printed on separate lines.

      Constraint: the random integer *b* should be randomly generated and at least 1000 digits long.

      [10 marks]

   b. `alice_sends_bob(p, g)`

      Input: A large prime *p* and a large integer *g* upto 1023 digits long.

      Output: Public exchange *A*. Random integer *a*. Each printed on separate lines.

      Constraint: the random integer *a* should be randomly generated and at least 1000 digits long.

      [10 points]

   c. `print_shared_secret_alice(B, a, p)`

      Input: Bob's public exchange key.

      Output: Print the share secret of Alice and bob.

      Constraint: Random integer *a* will be taken from the output of function: `alice_sends_bob()`

      [20 points]

   d. `print_shared_secret_bob(A, b, p)`

      Input: Alice's public exchange key.

      Output: Print the share secret of Alice and Bob.

      Constraint: Random integer *a* will be taken from the output of function: `bob_sends_alice()`

      [20 points]

Public Parameter Creation	
A trusted party chooses and publishes a (large) prime p and an integer g having large prime order in $\mathbb{F}_p^*$.	
Private Computations	
Alice	**Bob**
Choose a secret integer a. Compute $A \equiv g^a \pmod{p}$.	Choose a secret integer b. Compute $B \equiv g^b \pmod{p}$.
Public Exchange of Values	
Alice sends A to Bob $\longrightarrow$	A
B $\longleftarrow$	Bob sends B to Alice
Further Private Computations	
Alice	**Bob**
Compute the number $B^a \pmod{p}$.	Compute the number $A^b \pmod{p}$.
The shared secret value is $B^a \equiv (g^b)^a \equiv g^{ab} \equiv (g^a)^b \equiv A^b \pmod{p}$.	

Table 2.2: Diffie–Hellman key exchange

Part III

Notes:

- This part uses the Python language library GMP You are required to install the library, and run all experiments.
- **Notations and algorithms in supplementary material** will be the standard for this assignment.
- Deliverables for programming part are python files in the format <RollNo>_4.py

---

1. Implement RSA in python using GMP library.
    a. Write a program to encrypt 'm' given 'p' and 'q'.
    Input: p, q, m - in each line; p<q.
    Output: c, e, d, n - each number in a separate line
    Constraints: integers p, q and m are up to 1023 digits long. Avoid using loops.
    [30 marks]
    b. Write a program to decrypt 'm' given 'c' and 'd' and 'n'.
    Input: c, d, n - in each line.
    Output: m
    Constraints: same as 1.a.
    Note: Values of m, c, d, n will be taken from 1.a. for evaluation.
    [20 marks]

Supplementary material for Part III

---

1. The steps in an RSA Algorithm are
   a. Choose two prime numbers $p,\ q$.
   b. Let $n\ =\ p\ *\ q$
   c. Let $\phi\ =\ (p\ -\ 1)(q\ -\ 1)$
   d. Choose a large number $e\ \in\ [2,\ \phi\ -\ 1]$that is coprime toϕ.
   e. Compute $d\ \in\ [2,\ \phi\ -\ 1]$ such that $e\ \times\ d\ =\ 1\ (mod\ \phi)$, and $d$must be coprime to ϕ
   f. $(e,\ n)$ is the public key
   g. $(d,\ n)$ is the private key
   h. **Encryption**:
      i. $C\ =\ m^{e}\ (mod\ n)$
   i. **Decryption**:
      i. $m\ =\ C^{d}\ (mod\ n)$