Congratulations! You passed!

Grade received 85.71%

Latest Submission Grade 85.71%

To pass 30% or higher

Go to next item

RSA Quiz: Code

Quiz • 2h

Review Learning Objectives

Submit your assignment

Due Jan 22, 11:59 PM IST

Try again

Receive grade

To Pass 30% or higher

Your grade 85.71%

View Feedback

We keep your highest score

Like this implementation? Dislike Report an issue

1. Implement RSA encryption with the given public key $modulo, exponent$.

1 / 1 point

You have access to the function $PowMod(a, n, modulo)$ which computes $a^n \bmod modulo$ using the fast modular exponentiation algorithm from the previous module. You also have access to the function $ConvertToInt(message)$ which converts a text message to an integer.

You need to fix the implementation of the function $Encrypt(message, modulo, exponent)$ to return the integer $ciphertext$ according to RSA encryption algorithm.

```
def Encrypt(message, modulo, exponent):
  # Fix this implementation
  return PowMod(ConvertToInt(message), exponent, modulo)

#p = 1000000007
#q = 1000000009
#exponent = 23917
#modulo = p * q
#Encrypt('Hello world', modulo, exponent)
```

Run

Reset

```
No Output
```

Correct

Good job!

2. Implement RSA decryption with the given private key $p, q, exponent$.

1 / 1 point

You have access to the function $ConvertToStr(m)$ which converts from integer m to the plaintext $message$. You also have access to the function $InvertModulo(a, n)$ which takes coprime integers a and n as inputs and returns integer b such that $ab \equiv 1 \bmod n$. You also have access to the function $PowMod(a, n, modulo)$ which computes $a^n \bmod modulo$ using fast modular exponentiation.

You need to fix the implementation of the function $Decrypt(ciphertext, p, q, exponent)$ to decrypt the $message$ which was encrypted using the public key $(n = p \cdot q, e = exponent)$.

```
def Decrypt(ciphertext, p, q, exponent):
  n = p*q
  phi = (p-1)*(q-1)
  d =  InvertModulo(exponent, phi)
  return ConvertToStr(PowMod(ciphertext, d, n))

a = 3
b = 7
c = InvertModulo(a, b)
print(c)

p = 1000000007
q = 1000000009
exponent = 23917
modulo = p * q
ciphertext = Encrypt("attack", modulo, exponent)
message = Decrypt(ciphertext, p, q, exponent)
print(message)
```

Run

Reset

```
5
attack
attack
None
```

Correct

Good job!

3. Secret agent Alice has sent one of the following messages to the center:

1 / 1 point

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

Alice has ciphered her message using public key $modulo, exponent$ that is available to you, and you have intercepted her ciphertext. You want to know what was the content of her message. You have access to the function $Encrypt(message, modulo, exponent)$ which takes in a message as a string and returns a big integer as a ciphertext. It uses RSA encryption with public key $modulo, exponent$. In the starter code, you have an example usage of the function $Encrypt$.

You also have function $DecipherSimple(ciphertext, modulo, exponent, potential\_messages)$ implemented in the starter code. You need to fix this implementation to solve the problem. It should take the $ciphertext$ sent from Alice to the center, the public key $modulo, exponent$ and the set of potential messages that Alice could have sent, and return the message that Alice encrypted and sent as a string. For example, if Alice took message "wait", encrypted it with the given $modulo$ and $exponent$, and got number 139763215 as the ciphertext, you will need to return the string "wait" given the $ciphertext = 139763215$, $modulo, exponent$ and $potential\_messages = [\text{"}attack\text{"}, \text{"}don't\ attack\text{"}, \text{"}wait\text{"}]$.

```
def DecipherSimple(ciphertext, modulo, exponent, potential_messages):
  # Fix this implementation
  for potential_message in potential_messages:
    if ciphertext == Encrypt(potential_message, modulo, exponent):
      return potential_message
  return "don't know"

modulo = 101
exponent = 12
ciphertext = Encrypt("attack", modulo, exponent)
print(ciphertext)
print(DecipherSimple(ciphertext, modulo, exponent, ["attack", "don't attack", "wait"]))
```

Run

Reset

```
78
attack
attack
None
```

Correct

Good job!

4. Alice is using RSA encryption with a public key $modulo, exponent$ such that $modulo = p \cdot q$ with one of the primes p and q being less than $1\ 000\ 000$, and you know about it. You want to break the cipher and decrypt her message.

1 / 1 point

You can use the function $Decrypt(ciphertext, p, q, e)$ which decrypts the $ciphertext$ given the private key p, q and the public exponent e.

You are also given the function $DecipherSmallPrime(ciphertext, modulo, exponent)$, and you need to fix its implementation so that it can decipher the $ciphertext$ in case when one of the prime factors of the public modulo is smaller than $1\ 000\ 000$.

```

def DecipherSmallPrime(ciphertext, modulo, exponent):
  for p in range(2, 10**6):
    if p % 2 and modulo % p == 0:
      small_prime = p
      big_prime = modulo // p
      return Decrypt(ciphertext, small_prime, big_prime, exponent)
  return "don't know"

modulo = 101 * 18298970732541109011012304219376080251334480295537316123696052970419466495220522723330315111017831737980079504337868198011077274303193766040393009648852841770668239779097280026631944319501437547002412556176186750790476901358334138818 7
exponent = 239
ciphertext = Encrypt("attack", modulo, exponent)
print(ciphertext)
print(DecipherSmallPrime(ciphertext, modulo, exponent))
```

Run

Reset

```
1773488865857950936127070415107286811735036786757360876488901668828411065415319121739305112809023074111043494948978081751851444797671426477224221428915534085499185651993502038149646686545099252921036161028865989784470806731700777521041215564203613423323285066203840329997825032
attack
attack
None
```

Correct

Good job!

5. Alice is using RSA encryption with a public key $modulo, exponent$ such that $modulo = p \cdot q$ with $|p - q| < 5\ 000$, and you know about it. You want to break the cipher and decrypt her message.

0 / 1 point

You have access to the function $Decrypt(ciphertext, p, q, e)$ which decrypts the $ciphertext$ given the private key p, q and the public exponent e. You also have access to the function $IntSqrt(n)$ which takes integer n and returns the largest integer x such that $x^2 \leq n$.

You are also given the function $DecipherSmallDiff(ciphertext, modulo, exponent)$, and you need to fix its implementation so that it can decipher the $ciphertext$ in case when the difference beteween prime factors of the public modulo is smaller than $5\ 000$.