Trabalho Final de Cripto 2020.1

Nome: Ramon Oliveira de Azevedo

Observações: Professor, em alguns momentos, coloquei o símbolo ‘ ^ ’ como forma de demonstrar potenciação.

Questão 1. Temos que a fórmula apresentada é a mesma que define um número de Fermat. Ao rodarmos o programa apresentado em aula, que realiza o teste de Miller-Rabin, temos que Fn irá cair na condicional marcada:

```
if r == 1 or r == (num - 1):
    return False
i = 0
while True:
    r = pow(r, 2, num)
    print("R: ", r)
    i += 1
    if r == 1:
        return True
    if i == k:
        return True
    #Abaixo, a condicional citada na resposta
    if r == (num - 1):
        return False
```

Como vimos, o número corresponderá ao formato de r == (num - 1). Quando r entrar no loop “While” , ele terá o valor de 2. Logo, os números seguintes que serão o resultado de pow(r, 2, num), serão potências de 2. Observando a imagem abaixo, podemos notar que todos os números apresentados com o valor de r representam uma sequência, onde todos os números resultantes de pow(r, 2, num) pertencem a ordem de 2^(2^n). Como os números de Fermat podem ser definidos através da Fórmula Fn = 2^(2^n) + 1, podemos concluir q eles serão pseudoprimos de Miller Rabin, pois serão iguais a (num - 1). Além disso, vale afirmar que, como provado por Euler, o número da forma Fn = 2^(2^n) + 1 quando N = 5, é composto. E, até hoje, não se descobriu nenhum número F(n) primo com n > 4.

```
Num:  4294967297
R:  4
R:  16
R:  256
R:  65536
R:  4294967296
False
Num:  18446744073709551617
R:  4
R:  16
R:  256
R:  65536
R:  4294967296
R:  18446744073709551616
False
```

Questão 2.

Resposta: Analisando o código abaixo, temos que nos “Return” 2, é basicamente a transcrição da definição da 2° Forma do Pequeno Teorema de Fermat.. Porém, em aula, trabalhamos usando a 1° Forma, o que implica que nem todos os pseudoprimos apresentados em uma forma, serão pseudoprimos na outra forma. Para tal, o código abaixo que realiza o teste de Miller-Rabin, que por sua vez, é uma “transcrição” do algoritmo do mesmo, é capaz de solucionar esses pequenos “erros”. Para tal, ao observarmos o “Return” 1, podemos notar que caso um número b preencha os requisitos citado na condicional, ele retornará False. Esse “Return”, aliado à estrutura loop onde está inserido o 3° “Return”, é capaz de solucionar essas pequenas discrepâncias entre uma forma e outra. Isso acontece pois dentro do loop onde está o 3° Return temos a função pow, que nos serve para calcular a potência e o seu resto (mod n). Podemos perceber que ela elevará diversas vezes o número r ao quadrado. E, como podemos notar, o máximo de vezes que podemos elevar ao quadrado corresponde a um número k de vezes. Logo, para dar inconclusivo, esse i terá que ser menor que k. Assim, ficaremos com a base elevado a 2^i de um lado e (-1) do outro lado. Se elevarmos o que falta para chegar a k, então elevamos a i. Logo, elevando a k – i eu vou ter de um lado a base elevado a (n – 1) e o (-1) elevado a qualquer k vezes ao quadrado, será igual a 1. Com isso, o algoritmo de Miller-Rabin é capaz de eliminar essas discrepâncias.

Por exemplo, o número 6 na base 7 é considerado um pseudoprimo de Fermat na 1° Forma, mas o mesmo não ocorre quando analisamos a 2° Forma. Entretanto, ao utilizarmos a função abaixo com n = 6 e b = 7, ele cairá na condicional onde podemos encontrar o 1° “Return”, pois b %= n fará com que b tenha valor igual a 1.

```python
def miller_rabin(n, b):
    """
    Retorna True se o teste de Miller--Rabin para n com base b **prova**
    que n é composto, False nos outros casos
    """
    b %= n
    print(b)
    if b <= 1:
        #Return 1
        return False
    q = n - 1
    k = 0
    while q % 2 == 0:
        k += 1
        q //= 2

    r = pow(b, q, n)
    if r == 1 or r == (n - 1):
        #Return 2
        return False

    i = 0
    while True:
        r = pow(r, 2, n)
        i += 1
        if r == 1:
            return True
        if i == k:
            return True
        if r == (n - 1):
            #Return 3
            return False
```

Questão 3.

a) Fatorando N, temos que ele é igual a: 89 . 101

Logo, podemos concluir que Phi irá ter ser o resultado da multiplicação de (p - 1) * (q – 1). Portanto, Phi valerá 88 . 100 = 8800.

Por definição, temos que o E irá ter o valor do menor primo que não divide Phi. Assim, temos que E = 3. Para encontramos o expoente secreto D correspondente, basta realizarmos o Algoritmo de Euclides Estendido para acharmos o inverso de E (mod Phi). Como mostrado no print abaixo, temos que o valor será igual a 5867.

```
>>> def euclides_estendido(p, q):
    dividendo, divisor = p, q
    x_ant, y_ant = 1, 0
    x_novo, y_novo = 0, 1
    while divisor != 0:
        quociente, resto = divmod (dividendo, divisor)
        x_ant, x_novo = x_novo, (x_ant - x_novo*quociente)
        y_ant, y_novo = y_novo, (y_ant - y_novo*quociente)
        dividendo, divisor = divisor, resto
    return x_ant

>>> e = 3
>>> phi = 88 * 100
>>> d = euclides_estendido(e, phi)
>>> print(d)
-2933
>>> #Como d é negativo, temos que adicionar phi ao resultado final
>>> d += phi
>>> print (d)
5867
>>> 
```

Questão 3.

b) Usando a tabela dada como referência, temos que a mensagem 12345 será codificada como 112113114115116. Logo:

```
>>> texto_codificado = 112113114115116
>>> b1 = 1121
>>> b2 = 1311
>>> b3 = 4115
>>> b4 = 116
>>> print(pow(b1, e, n))
1404
>>> print(pow(b2, e, n))
6557
>>> print(pow(b3, e, n))
806
>>> print(pow(b4, e, n))
5799
>>> 
```

Questão 5.
Resposta: Isso ocorre porque um Expoente Público Pequeno pode comprometer a segurança do RSA, isto é, colocar a privacidade das mensagens em risco.

Temos que o mdc(E, Phi) tem que ser igual a 1. Então, quando trabalhamos com números primos grandes, temos que phi será par, e o mdc de 2 com um número par não será igual a 1. A única possibilidade do E poder ser 2 é quando o P e o Q forem dois, pois assim o Phi seria igual a:

$$(2 - 1)(2 - 1) = 1 . 1 => 1$$

E, como sabemos, o mdc (2, 1) será igual a 1. Só que dessa forma, com P e Q sendo igual a 2, eles serão primos muito pequenos e iguais, isso é algo que pode prejudicar seriamente a segurança do RSA, perdendo então, a sua eficácia. Sendo este último por sua vez, um dos “pilares” por trás do RSA.

Questão 7.
Resposta: O teorema Chinês dos Restos nos serve para acelerar o processo de descriptação. Para tanto, é de suma importância que os dados calculados mostrados no enunciado, sejam realizados. Por exemplo:

Sabendo os valores de P e Q, podemos facilmente descobrir o valor de N, pois N = P . Q
Logo, pelo Pequeno Teorema de Fermat, temos que $P \mid y^{(p-1)} - 1$. Dessa forma:

$$y^{(p-1)} = 1 \text{ (mod p) e } y^{(q-1)} = 1 \text{ (mod q)}$$

Com isso, podemos expandir essa ideia até obtermos um sistema de congruências, tal que: $x \equiv a$ (mod p) e $x \equiv b$ (mod q)
Para tal, pelo Teorema Chinês dos Restos, temos que:

$$x = (a . q . q') + (b . p . p') \text{ (mod n)}$$

Onde p' é o inverso de p em módulo q e q' é o inverso de q módulo p. Esses, por sua vez, são dois dos quatro dados armazenados pelo usuário.
Tomando como base o programa apresentado como resolução na Questão 12, temos que as duas formas reduzidas são aplicadas no processo para descobrir a congruência na qual a operação é equivalente, sendo utilizadas como expoentes na hora de calcular a aritmética modular.

A vantagem de usar o TCR no processo de descriptação é a simplificação do processo, tornando-o mais ágil.

Questão 8.
Segue a decodificação:
2823 = b
2688 = i
398 = c
4335 = h
2273 = o

Observação: Coloquei o programa que realiza o procedimento em anexo (Q8.py)

Questão 10.

Resposta: Como o enunciado nos diz, temos três chaves N. Podemos quebrar alguma delas através do cálculo do MDC utilizando um par dessas chaves. Temos que:

$$\text{mdc}(N1, N2) \mid N1 \quad \text{e} \quad \text{mdc}(N1, N2) \mid N2$$

Como sabemos, $N = P \cdot Q$, sendo P e Q primos. Assim, descobrindo um MDC, temos que esse número será capaz de dividir as duas chaves N. Como dito anteriormente, como as chaves N são o resultado da multiplicação de 2 primos, descobrindo um, podemos descobrir o outro.

Isso acontece, pois segundo o enunciado, as três chaves N foram feitas utilizando 5 números primos. Então consequentemente, um primo estará em 2 chaves ao mesmo tempo. Como achamos esse primo pelo MDC, agora podemos encontrar os outros primos que compõem as demais chaves. E assim, elas estarão fatoradas.