1)

Attached is the function I have created for clock function and the output after running it with what was instructed.

```
class lfsr:
    # Class implementing linear feedback shift register. Please note that it operates
    # with binary strings, strings of '0's and '1's.
    taps = []
    def __init__(self, taps):
    # Receives a list of taps. Taps are the bit positions that are XOR-ed
    #together and provided to the input of lfsr

        self.register = '1111111111111111'
        # initial state of lfsr
        self.taps = taps

    def clock(self, i='0'):
    #Receives input bit and simulates one cycle
    #This include xoring, shifting, updating input bit and returning output
    #input bit must be XOR-ed with the taps!!

    ## -- Implement me -- ##
        xor = list()
        m = 0
        for j in self.taps:
            if m == 0:
                curr = int(self.register[j])
            else:
                curr = curr ^ int(self.register[j])
            m += 1

        new = str(curr^int(i))

        leaving = self.register[len(self.register)-1]

        self.register = new + self.register[:len(self.register)-1]

        return str(int(leaving)) #returns output bit
```

```
if __name__ == "__main__":
    # l = lfsr([2,4,5,7,11,14])
    # l.seed(int_to_bin(4321))
    # print(l.gen(10,1000))
    l = lfsr([2,4,5,7,11,14])
    l.seed(int_to_bin(1234))
    print(l.gen(10,2000))
```

```
dquiroga@th121-11:~/Desktop/hw5$ python2.7 code_prototype.py
1111100000
dquiroga@th121-11:~/Desktop/hw5$
```

2)

Attached you will see a screenshot of the code I created the Hash function with along with the sentence that I decided to encrypt using the hash function.

```python
def H(inp):
    # Hash function, it must initialize a new lfsr, seed it with the inp binary string
    # skip and read the required number of lfsr output bits, returns binary string

    ## -- Implement me -- ##

    # Example: H(int_to_bin(0)) -> '10111000111111111111111000110001010010000101011101001100'
    # Example: H('0') -> '01010101010001110111000111111111111111110001100010100010000'
    # Example: H(int_to_bin(777)) -> '00010101011001100111111110011011110101011001111001101011001'
    l = lfsr([2,4,5,7,11,14])
    l.seed(inp)
    return l.gen(56, 1000)
```

```python
if __name__ == "__main__":
    # l = lfsr([2,4,5,7,11,14])
    # l.seed(int_to_bin(4321))
    # print(l.gen(10,1000))
    # l = lfsr([2,4,5,7,11,14])
    # l.seed(int_to_bin(1234))
    # print(l.gen(10,2000))


    # print(H(str_to_bin('My name is Bart Simpson and I like krusty burgers')))
    # print(H(int_to_bin(0)))
    # print(H('0'))
    # print(H(int_to_bin(777)))
    print(H(str_to_bin('My name is Daniel Quiroga and I like crispy tenders')))
```

```
dquiroga@th121-11:~/Desktop/hw5$ python2.7 code_prototype.py
00110010010000100101001101001100000011111110101010100000
dquiroga@th121-11:~/Desktop/hw5$
```

3)

Below is my implementation of the function enc_pad() and how I call the function to encrypt and decrypt my message, I have added spaces in between outputs to make readability easier.

```python
def enc_pad(m,p):
    # encrypt message m with pad p, return binary string
    o = ''

    ## -- Implement me -- ##
    for i in range(len(m)):
        o += str(int(m[i])^int(p[i]))

    return o
```

```python
if __name__ == "__main__":
    # l = lfsr([2,4,5,7,11,14])
    # l.seed(int_to_bin(4321))
    # print(l.gen(10,1000))
    # l = lfsr([2,4,5,7,11,14])
    # l.seed(int_to_bin(1234))
    # print(l.gen(10,2000))


    # print(H(str_to_bin('My name is Bart Simpson and I like krusty burgers')))
    # print(H(int_to_bin(0)))
    # print(H('0'))
    # print(H(int_to_bin(777)))
    # print(H(str_to_bin('My name is Daniel Quiroga and I like crispy tenders')))



    # s = "Hello, my name is Bobby Hill"
    # bs = str_to_bin(s)
    # print(bs)
    # print(" ")
    # l = lfsr([2,4,5,7,11,14])
    # l.seed(int_to_bin(77))
    # pad = l.gen(len(bs), 1000)
    # print(pad)
    # print(" ")
    # cipher = enc_pad(bs,pad)
    # print(cipher)
    # print(" ")
    # print(bin_to_str(cipher))
    # print(" ")
    # print(bin_to_str(enc_pad(cipher,pad)))

    s = "Hello, my name is Daniel Quiroga"
    bs = str_to_bin(s)
    print(bs)
    print(" ")
    l = lfsr([2,4,5,7,11,14])
    l.seed(int_to_bin(77))
    pad = l.gen(len(bs), 1000)
    print(pad)
    print(" ")
    cipher = enc_pad(bs,pad)
    print(cipher)
    print(" ")
    print(bin_to_str(cipher))
    print(" ")
    print(bin_to_str(enc_pad(cipher,pad)))
```

```
dquiroga@th121-11:~/Desktop/hw5$ python2.7 code_prototype.py
01001000011001010111011000110110000110111100101100001000000110110101111001001000000110111001100001011011010110010100100000011010010111001100100000100010001100
00101110111001101001011001010111011000010000001010000101110101011010010111001000110111101100111011001

00111111010011100000111011001110001101000110000111011101001110010101010111110100111010011001011011101110111011001110010010110010001011000011110000110000110100
01001101011100011000001011110101100101110010010001000011001111010011001110111111000101110101010101010110

01110110111111001011101110101111011010101000001111001011010000100101100111000111111001001001011011101101000001011010100100011000001100101001111100101110000001
01100000101111100010011101111011110001001101100100010010111001111000001011010101010111001000110111
```

```
vw]□.-□R0e>\□□;□□7

Hello, my name is Daniel Quiroga
dquiroga@th121-11:~/Desktop/hw5$
```

4)

Below is a screenshot of my implementation of the GenRSA() function and the output of each of the variables. Along with a validation using the t=100 example:

```python
def GenRSA():
    # Function to generate RSA keys. Use the Euler's totient function (phi)
    # As we discussed in lectures. Function must: 1) seed python's random number
    # generator with time.time() 2) Generate RSA primes by keeping generating
    # random integers of size 512 bit using the random.getrandbits(512) and testing
    # whether they are prime or not using the is_prime function until both primes found.
    # 3) compute phi 4) find e that is coprime to phi. Start from e=3 and keep
    # incrementing until you find a suitable one. 4) derive d 5) return tuple (n,e,d)
    # n - public modulo, e - public exponent, d - private exponent

    random.seed(time.time())

    ## -- Implement me -- ##
    p = random.getrandbits(512)
    while not is_prime(p):
        p = random.getrandbits(512)


    q = random.getrandbits(512)
    while not is_prime(q):
        q = random.getrandbits(512)

    n = p*q

    phi = (p-1)*(q-1)

    e = 3
    temp = egcd(e, phi)
    while e < phi and temp[0] != 1:
        e += 1
        temp = egcd(e, phi)

    d = modinv(e, phi)


    return (n,e,d)

if __name__ == "__main__":

    (n,e,d) = GenRSA()
    print((n,e,d))

    t = 100
    c = pow(t,e,n)
    print(pow(c,d,n) == t)
```

```
dquiroga@th121-11:~/Desktop/hw5$ python2.7 code_prototype.py
(24122266445716545875731317361681627126488020679225489459171048963520832913095115196045480884750486049973980468865287794429518797457908168464956853481449580
899404001162804306170898428662233797465930684919467122461350835472405831683585443477657368238812269740354711516779845772338244076236842877257460667258569L,
11, 19736399819222628443780168750466785830762926010275400466594494606517045110714185160400847996614034040887802201798871831805969925192833956016782880121186
0115487348252670858411359555840478126863513932525811577017703113456702799814185547088960648128397631501056346729532428472209067765488089311174440938338332351
L)
True
dquiroga@th121-11:~/Desktop/hw5$ _
```

As seen, true was outputted meaning that the encryption and decryption was successful.

5)

5.1)

Using the GenRSA() I would use those values returned along with what the message in integer is to compute an integer value and then convert that integer value into a binary representation to send the string:

```python
if __name__ == "__main__":

    (n,e,d) = GenRSA()
    s = "Daniel Quiroga dquiroga@email.wm.edu"
    bs = str_to_bin(s)
    ints = bin_to_int(bs)

    message = pow(ints,e,n)
    message = int_to_bin(message)
    print("Signature of the  message: " + message)
```

```
dquiroga@th121-11:~/Desktop/hw5$ dquiroga@th121-11:~/Desktop/hw5$ python2.7 code_prototype.py
Signature of the  message: 0000010011100000111111010010100110101010111101010100111101011001111000110010100110110111011100001010110011101000100111001000111000
0111110011011000011110101000101111001000100100011110110011010010011001001100100011000000010111100111101010010100110011101101010010010111011001001111101100001
0101111111001011100000000000001111101111100110110100100001011010100010000010001000011010011110011110011110010000001000000001101001100011110111000111101011010110
101111110011101011100011001111010111000111000100001100111111011011110101010010000000011011101110011010110001100110000000100010000101111100110101100101101101100
11001100111111111101111010000010000100101000000001100010011010011010101000101101110101000000000001111001000111110010111000011110100101101111100001010110011001
0100110000101011011011101110111100001011010001111000001100101111111101000010101001111010010011110001001101
dquiroga@th121-11:~/Desktop/hw5$ _
```

5.2)

In order to decrypt the message the other party would need the secret d key and the public modulo, n. The would need to convert the binary signature into an integer and then pow(integer, d, n) which would return the original message in integer form. By converting the int to string you would get the original message:

```python
if __name__ == "__main__":

    (n,e,d) = GenRSA()
    s = "Daniel Quiroga dquiroga@email.wm.edu"
    bs = str_to_bin(s)
    ints = bin_to_int(bs)

    message = pow(ints,e,n)
    message = int_to_bin(message)
    print("Signature of the  message: " + message)

#-----------How to verify signature----------------
    other_party = message

    other_party = bin_to_int(other_party)
    integer = pow(other_party,d,n)
    binary = int_to_bin(integer)
    original_message = bin_to_str(binary)
    print("\n\nOriginal Message: "+original_message)
```

```
dquiroga@th121-11:~/Desktop/hw5$ python2.7 code_prototype.py
Signature of the  message: 0000010011100000111111010010100110101010111101010100111101011001111000110010100110110111011100001010110011101000100111001000111000
0111110011011000011110101000101111001000100100011110110011010010011001001100100011000000010111100111101010010100110011101101010010010111011001001111101100001
0101111111001011100000000000000111110111100110110100100001011010100010000010001000011010011110011110011110010000001000000011010011000111101110001111010110101100
1011111001110101110001100111101011100011100010000110011111101101111010101001000000001101110111001101011000110011000000010001000001011111001101011001011011011100
1100110011111111110111101000000100001001010000000000110001001101001101010100010110111010100000000000111100100011111001011100001111010010110111110000101011001100 1
0100110000101011011011101110111100001011010001111000001100101111111101000010101001111010010011110001001101


Original Message: Daniel Quiroga dquiroga@email.wm.edu
dquiroga@th121-11:~/Desktop/hw5$
```