1. The nonce is determined from the email address of the recipient and the subject of the email.

   Considering that spam emails are generally sent regarding the same context/promotion/subscription, it is possible for the sender of the spam messages to keep the subject of the spam email in order to reduce the amount of competition required to generate the nonce.

   All those spam messages are sent to many different recipients, once the email address of a certain recipient has been acquired, it becomes a constant record after recipient's email address which doesn't change.

   This means there only two instances at which significant computational work will be done to generate the nonce. This is when the spam messages sent to a new recipient; new email address and when a spam message with a different subject needs to be sent. This would not deter spam message senders as the average cost of competition would be insignificant as only two instances need computational power to generate the nonce.

   A possible improvement to this idea could be to include a combination of reliable varying quantities in the determination of the nonce. This includes values such as the dates and timestamp but which the email is sent.

2.

   a.

      i. **Confidentiality**
      Yes.

      The original message may only be accessed by individuals that have the secret key. The original message is only accessible to an authorized group of people therefore the message is confidential.

      ii. **Data integrity**
      Yes.

      c is only true in the XOR case (this is done to ensure that the hash and the message are not the same). Therefore, if we hash the original message m [0, 1] with hash key h [1, 0], we get a c [1, 1] combination at the sender.

c = m XOR h	m	h
1	0	1
1	1	0

Figure 1: Alice

      Alice then sends c to Bob. We assume that Bob also has the same hash key *h*. Bob can generate this hash key using the number IV which is sent by Alice.

Bob receives the c [1, 1] combination and runs c [1, 1] XOR h [1, 0]. Bob then receives the original message [0, 1]. This confirms that this XOR check ensures data is not corrupted thus data integrity is maintained.

m = c XOR h	c	h
0	1	1
1	1	0

Figure 2: Bob

iii. Authenticity

No.

The algorithm does not provide means to identify the sender therefore Bob cannot tell if the message is indeed from Alice or someone else that used the same random IV number to generate the hash key. This means that the authenticity of the message cannot be ensured.

To combat this, a pseudo random IV number can be generated within a certain range for a certain individual. In this way, the receiver will be able to identify the sender of the message (using the range) while still maintaining the randomness of the actual IV number and as it is defined within a range and not as a specific number.

b.

```
from hashlib import sha256
import datetime


def make_hash(iv, date):
    arg = str(iv) + str(date)
    return sha256(arg.encode()).hexdigest()


def find_message(c, h):
    lst = [chr(ord(a) ^ ord(b)) for a, b in zip(c, h)]
    msg = "".join(lst)
    print("PIN:", str(ord(lst[0]))+str(ord(lst[1])))


def main():
    IV = "15304484387517434811"
    c =
"0xa75da6155e61662665dfdec2264097b460cea3eb09c84461b5f728d9b0058361"
    Date = "20211015" # Chosen secret date

    print("Date:", datetime.datetime(2021, 10, 15))

    h = make_hash(IV, Date)
    find_message(c, h)
```