CSE 484 / CSE M 584 - Homework 3 - Winter 2022

This homework is focused on cryptography.

Overview

- **Due Date:** March 10, 2022, 11:45pm.
- **Group or Individual:** Do this assignment as an **individual**. But you are allowed to talk with others in advance of actually doing the assignment. Just make sure that when you write up your solutions, you are doing that on your own.
- **How to Submit:** Submit your responses via the Canvas assignment in PDF.
- **Total Points:** 35 (across 8 questions) (and one extra credit worth 3 points)

Question 1 (2 points):

What is the main concern cryptographers have with the Encrypt-and-MAC method for combining a symmetric encryption scheme with a symmetric MAC to create a symmetric authenticated encryption scheme?

Question 2 (2 points):

Consider a group of 25 people in a room who wish to be able to establish pairwise secure communications in the future. How many unique keys need to be created and then exchanged in total:

(a) Using symmetric cryptography?

(b) Using public key cryptography?

Question 3 (5 points):

The goal of this task is to give you a better understanding of Certificate Authorities (CA) and certificates.

Take a look at the CAs certificates that your computer trusts.

- Mac: Spotlight search ‘Keychain Access’
- Windows: Control Panel -> Search ‘Internet Options’ -> Content -> Certificates

Answer these questions:

(a) How many root CA certificates does your computer have?

(b) What is something that you found interesting from looking at the root CA certificates?

(c) Go to google.com using your favorite browser, and find a way to look at the certificates for google.com. List the chain of certificates your browser sees.

(d) What is a possible risk of trusting a CA?

Question 4 (5 points):

This message was encrypted with the RSA primitive, where N=77 and e=7.

(a) Decrypt it and submit the corresponding plaintext, as an ASCII string.

Tips: You are welcome to write a program to aid in the decryption, and you might want to compute the private decryption exponent d.

For this cryptogram ‘A’ is encoded as a 1 before encryption, ‘B’ as a 2, and so on.

Here is the cryptogram:

```
51 71 51 60 47 31 39 53 58 48 68 1 12 37 31 47 68 68 71 31 37 1
12 12 53 60 37 68 48 1 42 31 47 60 68 47 31 21 39 47 62 47 68 68
1 28 47
```

Question 5 (8 points):

The following question has you use RSA, but with larger values (but still not anywhere close to the size of the numbers one would use in a secure cryptographic protocol like TLS/SSL).

You may use a program that you write, Wolfram Alpha, or any other computer program to help you solve this problem.

For all of these, it is sufficient to just include your number in the answer, unless the question explicitly asks for additional detail.

Let p = 9497 and q =7187 and e = 3.

(a) Compute N = p * q. What is N?
(b) Compute Phi(N) = (p-1)(q-1). What is Phi(N)?
(c) Verify that e is relatively prime to Phi(N). What method did you use to verify this?
(d) Compute d as the inverse of e modulo Phi(N). What is d?
(e) Encrypt the value P = 22334455 with the RSA primitive and the values for N and e above. Let C be the resulting ciphertext. What is C?
(f) Verify that you can decrypt C using d as the private exponent to get back P. What method did you use to verify this?

(g) Decrypt the value C' = 55443322 using the RSA primitive and your values for N and d above. Let P' be the resulting plaintext. What is P'?

(h) Verify that you can encrypt P' using e as the public exponent to get back C'. What method did you use to verify this?

Question 6 (3 points):

Consider a Diffie-Hellman key exchange with p=29 and g=2. Suppose that Alice picks x=5 and Bob picks y=8.

(a) What will each party send to the other, and what shared key (number) will they agree on? **Show your work**.

Question 7 (5 points):

Suppose you, as an attacker, observe the following 32-byte (3-block) ciphertext C1 (in hex)

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 03
46 64 DC 06 97 BB FE 69 33 07 15 07 9B A6 C2 3D
2B 84 DE 4F 90 8D 7D 34 AA CE 96 8B 64 F3 DF 75
```

and the following 32-byte (3-block) ciphertext C2 (also in hex)

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 03
46 79 D0 18 97 B1 EB 49 37 02 0E 1B F2 96 F1 17
3E 93 C4 5A 8B 98 74 0E BA 9D BE D8 3C A2 8A 3B
```

Suppose you know these ciphertexts were generated using CTR mode, where the first block of the ciphertext is the initial counter value for the encryption. You also know that the plaintext P1 corresponding to C1 is

```
43 72 79 70 74 6F 67 72 61 70 68 79 20 43 72 79
70 74 6F 67 72 61 70 68 79 20 43 72 79 70 74 6F
```

(a) Compute the plaintext P2 corresponding to the ciphertext C2. Submit P2 as your response, using the same formatting as above (in hex, with a space between each byte).

Question 8 (5 points):

Consider an insecure version of SSH that uses ECB mode for encryption. Whenever a user types a key into the ssh client, that key is immediately encrypted and sent over the wire to the server. This immediate encrypt-after-key-press procedure is what enables the interactivity of a remote shell. Now consider the following sequence of plaintext packets (written in hex):

```
6C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // ASCII l
73 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // ASCII s
20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // ASCII
2D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // ASCII -
2D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // ASCII -
63 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // ASCII c
6F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // ASCII o
6C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // ASCII l
6F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // ASCII o
72 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // ASCII r
20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // ASCII
2D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // ASCII -
2D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // ASCII -
66 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // ASCII f
75 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // ASCII u
6C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // ASCII l
6C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // ASCII l
2D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // ASCII -
74 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // ASCII t
69 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // ASCII i
6D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // ASCII m
65 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // ASCII e
20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // ASCII
2A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // ASCII *
0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // ASCII <enter>
```

This corresponds to a user typing "`ls --color --full-time *<enter>`" into their ssh client.

Suppose an attacker knows what the user is typing via some out-of-band channel (e.g., shoulder surfing) and also eavesdrops on this communications and intercepts the corresponding ciphertexts:

```
B7 06 B1 27 FF FB 19 C5 97 59 D5 24 60 A0 72 D6
D4 AD 5F E3 3E 5E A8 51 2E D6 E1 1E 3C BA C4 72
A3 68 9F B2 FB 4E E2 03 C3 06 F2 E4 E3 EC EF 01
```

```
56 DF 80 67 90 CC D8 3B 8E C9 7C C0 86 44 94 DE
56 DF 80 67 90 CC D8 3B 8E C9 7C C0 86 44 94 DE
E7 DC 46 2B D2 46 C6 71 93 52 1B D4 29 9C 15 E8
19 6C 09 71 95 E6 A2 D5 73 A7 FD 9A 7E 90 61 3E
B7 06 B1 27 FF FB 19 C5 97 59 D5 24 60 A0 72 D6
19 6C 09 71 95 E6 A2 D5 73 A7 FD 9A 7E 90 61 3E
61 74 95 60 A8 C6 F6 4E AF 75 84 C8 C9 3D 81 F0
A3 68 9F B2 FB 4E E2 03 C3 06 F2 E4 E3 EC EF 01
56 DF 80 67 90 CC D8 3B 8E C9 7C C0 86 44 94 DE
56 DF 80 67 90 CC D8 3B 8E C9 7C C0 86 44 94 DE
2A 8A C2 D9 96 6C 02 6D B7 D4 56 10 DB 69 12 14
89 FF 5A CC 40 23 D9 87 2F CB B0 FB 5D 7F 47 62
B7 06 B1 27 FF FB 19 C5 97 59 D5 24 60 A0 72 D6
B7 06 B1 27 FF FB 19 C5 97 59 D5 24 60 A0 72 D6
56 DF 80 67 90 CC D8 3B 8E C9 7C C0 86 44 94 DE
FD A0 BD AD FD 57 F7 4D B7 4A 17 36 09 3A C4 7B
DE 4E 8C 1D 1D A5 01 E0 3B 73 0C A3 A2 55 81 1D
F1 EE 99 A0 B1 32 7C EF C2 BA 2B BD 0F CE C6 26
3D 68 D3 3C 52 CB 81 19 A5 87 3B 60 D5 19 87 23
A3 68 9F B2 FB 4E E2 03 C3 06 F2 E4 E3 EC EF 01
25 08 DA 97 27 8C CF 86 0D E8 96 5F DD 6B 60 0F
81 78 54 32 59 6B 69 02 4F FE D1 B5 52 DA F2 C4
```

The attacker can now inject messages into the communications channel from the client to the server. One thing an attacker might try to do: generate a sequence of ciphertext packets that, when decrypted, are interpreted as “rm -rf *<enter>” on the server. (In other words, a command that deletes everything!)

(a) Give the encrypted **ciphertext** (not plaintext) of “rm -rf *<enter>” in your answer below, formatted as the lines above (hex, with a space between each byte.)

Extra Credit (3 Points):

As we saw in class, 2DES (and more broadly, any double-encrypting with a block cipher) is vulnerable to a meet-in-the-middle brute-force attack. This is a chance to try it out!

To make this reasonable, we'll use DES with (effective) 14-bit keys. Thus, each 56-bit key is just the same 14-bits repeated 4 times. We're using standard ASCII/utf-8 encodings here, so 'A' is 0x41.

You've managed to get the following plaintext/ciphertext pair (maybe you just asked?)
PT: "`a short message!`"
CT: `d8bf8dc0054d525a3d7fcedb6912cc43`
(hex value, not a string)

You've captured the following ciphertext as well:
`26d22089fd6b1c9d5a33ae252bd9922a48d2a7e2d5868daf4d847d0c8eff3529e308ae7beb33015d`
(hex value, not a string)

(a) Given that you know the encryption scheme was 2"DES" using these short keys, decrypt the ciphertext and supply your answer as an ASCII string.

You will need to write a short program to solve this problem, and may use *any* language you feel comfortable with. Good options include matlab, Python, Java, etc.

Note: Our sample solution is in Python, which has a 'des' module that is fast and simple to use, but it is not installed by default. You can likely install it by running pip install des in your terminal. Our solution is less than 30 lines of code and runs in under a minute on a laptop, so you don't need to do that much compute!

Here is a snippet of code to expand a 14-bit key into a full DesKey object in Python and an example of usage.

```
from des import DesKey
def expandkey(val):
    if(val >= (2**14)):
        print("Key too large! Must fit in 14 bits")
        exit()
    k = val | (val << 14) | (val << 28) | (val << 42)
    return DesKey(bytearray.fromhex("{v:016X}".format(v=k)))
```

```
PT = b"a short message!"
CT = expandkey(1234).encrypt(PT)
print(CT.hex())
PT2 = expandkey(1234).decrypt(CT)
print(PT2)
```

Recall that 2”DES” would just be:

```
CT = expandkey(k2).encrypt(expandkey(k1).encrypt(PT))
```

You may also find this a useful way to translate a hex value into a bytes array:

```
bytes.fromhex(bighexvalue)
```