Assignment 1 Write Up

E1 Rivest–Shamir–Adleman:
Cipher text is given by the formula: $m = c^d \ mod \ n$
Given that we know p, q and e. We can compute d and n. After decrypting the message, we get,
30451955574887994185760677450285729355733635378743673528612379833223427080317. Converting it into hexadecimal, we get 4353323130377B706C336173655F2433637572655F75725F705F715F5253407D. Then converting it into ascii, we get the flag, CS2107{pl3ase_$3cure_ur_p_q_RS@}.

E2 xor_secure:
Given the encrypted message and four keys, we can derive the original message by using multiple xor in the reverse order. As such, enc_msg XOR key4 XOR key3 and so on. After obtaining the final output in binary, convert it into ascii, and we get the flag, CS2107{M4St3R_0f_aNc13nT_x0R_t3CHn1qu3s!!!!}.

E3 Hash Browns:
Given the list of SHA-1 hashes, went online https://10015.io/tools/sha1-encrypt-decrypt to lookup the corresponding character for each hash in the text file. The flag is, CS2107{hash_br0wn$_cr@ck3rs}.

E4 Caesar with a capital C:
Given the cipher text and the cipher algorithm, we can scrutinize the cipher algorithm. As such, the algorithm outputs a character that is that 5 positions to the right except for the underscore character. Therefore, by shifting 5 positions to the left of the cipher text, we can thus obtain the plain text, CS2107{345y_p345y_l3m0n_5qu33zy}.

M1 AES ECB:
Since we know that AES ECB encodes are deterministic, hence we can observe the contents of the output file. More specifically, the bytes that are associated with each char in chal.py. From chal.py, we can see that the block size is 16, as such we can split the output into chunks of 16, creating a dictionary for the chunks and their corresponding characters. However, since we do not know the length of the flag, we have to do some manual visual inspection. We do know that the last 16 bytes refers to the newline token, and that the 5th and 28th characters are 'C' and 'S' respectively. We can narrow down the region in which the flag lies in the output file.
After knowing the length of the flag, we edit chal.py to add 36 filler characters in between $CS2107\{$ and $\}$, and by adding all bytes and character pairs, except these 36 characters, into the map, we can obtain the flag, CS2107{AES_ECB_1s_l1terally_0nly_subb1ng...}

M2 baby shark:

Using wireshark, we can export all items with regards to http, since the hint suggested looking at http files. We then look through each file for the flag. Parts of the flag can be found in the respective locations:

File name	Flag
Wallpaper.png	CS2107{b
Inventory_book.xlsx secret tab	@by_sha@a
Config.xml (search for part)	Rk_d00_d
Confidential.pdf words were white in colour	Oo_143192}

M3 hash_key:

Given the algorithm, we can see that the user uses a random number from 1 to 2^{25} to generate the key. Since this is an offline attack, we can brute force to get the key which in turn enable us to decrypt the cipher text using AES counter mode with the given nonce. Therefore, the flag is, CS2107{n0t_A_g00d_Id3A_t0_h4v3_sH0rt_K3y_L3ngTH}

M4 Salad:

The substitution cipher algorithm can be found in salad.py, therefore by pinpointing which parts of the ascii codes it is trying to substitute to, we can reverse the conditions as well as the number of positions shifted.

Explanation:

```
    if ac > 33 and ac <= 96:
        if ac >= 65:                        # A-H
            if ac >= 73:                    # I-Z
                if ac >= 91:                # [\]^_`
                    ac = shift(ac,-6)       # UVWXYZ
                    aabb += chr(ac)
                    continue
                ac = shift(ac,32)           # i-z
                aabb += chr(ac)
                continue
            ac = shift(ac,-25)              # ()*+,-./
            aabb += chr(ac)
            continue
```

We can see that if the ascii code is between 91 and 96, the algorithm is looking to change from [\]^_` to UVWXYZ. For ascii codes between 73 to 90, I-Z upper case to lower case i-z. As such we can slowly deduce the substitution changes for each portion of the algorithm. Full list of changes can be seen below:

```
        if ac < 65:                         #
            if ac < 58:                     # 0-9
                if ac < 48:                 # "#$%&'()*+,-./
                    ac = shift(ac,57)       # [\]^_`abcdefgh
                    aabb += chr(ac)
                    continue
                if ac >= 55:                # 789
                    ac = shift(ac,68)       # {|}
                    aabb += chr(ac)
                    continue
                ac = shift(ac,10)           # 0-6     change to  :;<=>?@
                aabb += chr(ac)
                continue
            ac = shift(ac,-10)              # :;<=>?@ change to  0-6
            aabb += chr(ac)
            continue
        continue

    if ac >= 117 and ac < 126:              # uvwxyz{|}
        if ac >= 123:                       # {|}
            ac = shift(ac,-68)              # 789
            aabb += chr(ac)
            continue
        ac = shift(ac,-83)                  # "#$%'(
        aabb += chr(ac)
        continue

    if ac > 33 and ac <= 116:               # `abcdefghijklmnopqrst
        ac = shift(ac,-32)                  # @ABCDEFGHIJKLMNOPQRST
        aabb += chr(ac)
        continue

    if ac == 126:                           # ~
        nc = 33                             # !
    aabb += chr(nc)
```

H2 Secure Password:

Given that the hint was to view the html file, we can see that the expected string is encoded within the file. Since passwords are mainly made up of alpha numeric and some special characters, which only contains up to 127 different characters. We can easily loop through each ascii code to find the matching expected output. Therefore, the flag is CS2107{1S_4Ct1y_4_Sb0x}.