Cryptography Engineering Midterm

Problem 1.

a) $f(x) = x^2 + 1, g(x) = x^3 + x^2 + 1, P(x) = x^4 + x + 1 \; over \; GF(2^4)$

$$f(x) + g(x) = x^3 + 2x^2 + 2 = x^3$$
$$f(x) - g(x) = -x^3 = x^3$$
$$f(x) \times g(x) = (x^5 + x^4 + x^2) + (x^3 + x^2 + 1) = x^5 + x^4 + x^3 + 1$$
$$= (x^2 + x) + (x + 1) + (x^3 + 1) = x^3 + x^2$$

b) $f(x) = x^2 + 1, g(x) = x + 1, P(x) = x^4 + x + 1 \; over \; GF(2^4)$

$$f(x) + g(x) = x^2 + x + 2 = x^2 + x$$
$$f(x) - g(x) = x^2 - x = x^2 + x$$
$$f(x) \times g(x) = (x^3 + x) + (x^2 + 1) = x^3 + x^2 + x + 1$$

Problem 2.

a) $f(x) = x^7 + x^5 + x^4 + x + 1, g(x) = x^3 + x + 1, m(x) = x^8 + x^4 + x^3 + x + 1 \; over \; GF(2^8)$

$$f(x) + g(x) = x^7 + x^5 + x^4 + x^3 + 2x + 2 = x^7 + x^5 + x^4 + x^3$$
$$f(x) - g(x) = x^7 + x^5 + x^4 - x^3 = x^7 + x^5 + x^4 + x^3$$
$$f(x) \times g(x) = (x^{10} + x^8 + x^7 + x^4 + x^3) + (x^8 + x^6 + x^5 + x^2 + x)$$
$$+ (x^7 + x^5 + x^4 + x + 1) = x^{10} + x^6 + x^3 + x^2 + 1$$
$$= x^2(x^4 + x^3 + x + 1) + x^6 + x^3 + x^2 + 1$$
$$= (x^6 + x^5 + x^3 + x^2) + x^6 + x^3 + x^2 + 1 = x^5 + 1$$

b) $f(x) = x^4 + 1$

$$take \; p(x) = x^2 + 1$$
$$p(x) \times p(x) = x^4 + 2x^2 + 1 = x^4 + 1 = f(x)$$

Since $p(x) \times p(x) = f(x)$, so $f(x)$ in reducible.

Problem 3.

$$P(x) = x^4 + x + 1 \; in \; GF(2^4)$$

a) $f(x) = x$

$$take \; m(x) = x^3 + 1$$
$$f(x) \times m(x) = x^4 + x = x + 1 + x = 1$$
$$so, f(x)^{-1} = m(x) = x^3 + 1$$

b) $g(x) = x^2 + x$

$$take \; m(x) = x^2 + x + 1$$
$$g(x) \times m(x) = (x^4 + x^3) + (x^3 + x^2) + (x^2 + x) = x^4 + x = x + 1 + x = 1$$
$$so, g(x)^{-1} = m(x) = x^2 + x + 1$$

Problem 4.

$$p(x) = x^8 + x^4 + 1$$

a)

$$let\, m(x) = x^5 + x^4 + x^3 + x^2$$
$$(x^3 + x^2)m(x)\, mod\, p(x) = x^8 + x^4 = 1$$
$$(x^3 + x^2 + x)(x^3 + x^2)^{-1}\, mod\, p(x) = (x^3 + x^2 + x)m(x)\, mod\, p(x)$$
$$= x^8 + x^6 + x^5 + x^3\, mod\, p(x) = x^6 + x^5 + x^4 + x^3 + 1$$

b)

$$(x^6 + x^3 + 1)(x^4 + x^3 + 1)\, mod\, p(x) = x^{10} + x^9 + x^7 + x^4 + 1\, mod\, p(x)$$
$$= x^2(x^4 + 1) + x(x^4 + 1) + x^7 + x^4 + 1$$
$$= (x^6 + x^2) + (x^5 + x) + x^7 + x^4 + 1$$
$$= x^7 + x^6 + x^5 + x^4 + x^2 + x + 1$$

Problem 5.

a) Since the matrix and the modular are predefined, and all calculation are in $\mathbb{GF}(2)$, so addition and modulation can be replaced by some XOR, also there are some multiplications with predefined multiplier, so it can be precomputed and stored in a lookup table to save computation.

b) For byte substitution, since the substitutions are predefined, so it can be replaced by a lookup table.
For shift row, since the shift amount is predefined, so each possible value for each row can be stored in a lookup table.
For mix column, as mentioned above, can be implemented by XOR and lookup table.
Since AES is based on above components and some other XOR-based operations, so the whole AES can be implemented in XORs and lookup tables.

Problem 6.

For the $InvSubBytes$ and $InvShiftRows$, since the substitution only based on the byte content, and shift rows doesn't affect the byte content by shifting predefined offset, so the order of those two operations is indifferent.
For the $AddRoundKey$ and $InvMixColumns$, since $InvMixColumns$ can be viewed as $MixColumns$ with another matrix, so it can also be represented in XORs and lookup tables.
$MixColumns$ and $InvMixColumns$ can be formulated into

$$d_j = a_i \cdot b_j \oplus a_{i+1} \cdot b_{j+1} \oplus a_{i+2} \cdot b_{j+2} \oplus a_{i+3} \cdot b_{j+3}\, for\, some\, i, coefficient\, a_{[4]}\, and\, the\, column\, b_{[4]}$$

Then $AddRoundKey$:

$$e_j = d_j \oplus key_j = (a_i \cdot b_j \oplus a_{i+1} \cdot b_{j+1} \oplus a_{i+2} \cdot b_{j+2} \oplus a_{i+3} \cdot b_{j+3}) \oplus key_j$$
$$= a_i \cdot (b_j \oplus \widetilde{key}_j) \oplus a_{i+1} \cdot (b_{j+1} \oplus \widetilde{key}_{j+1}) \oplus a_{i+2} \cdot (b_{j+2} \oplus \widetilde{key}_{j+2}) \oplus a_{i+3} \cdot (b_{j+3} \oplus \widetilde{key}_{j+3})$$
$$= (Inv)MixColumns\ after\ AddRoundKey$$

So, if the key can be adapted accordingly, then the order of $InvMixColumns$ and $AddRoundKey$ can be inverted.
Since the mentioned operations can be inverted, so decrypting can share the code or chip with encrypting with some well-designed key system.

Problem 7.

For 3DES, I think I would choose it over AES when the software is too old that doesn't have AES implementation or when the resources are too limited to perform AES.
AES is better than 3DES in many aspects:

1. AES is faster than 3DES in many circumstances, and there is even hardware designed particularly for AES for better speed.
2. AES can accept various key length, while 3DES have only one choice.
3. AES have a larger block size of 128 bits, safer than 3DES' 64 bits block.

I think 3DES is not likely susceptible to Meet-in-the-Middle attack, since MITM attack takes only $O(N)$ for 2DES, but it needs at least $O(N^2)$ for 3DES, so it is not likely susceptible to that attack in current computer or algorithm design.

Problem 8.

- The entities using the old key should be notified that the key is going to be revoked.
- Decrypt the data with the old key and re-encrypt with the new key.
- Delete all records of the old key's existence.
- Distribute the new public key if it exists.

Problem 9.

a) Her boss might discover that she is lazy, then she would be in trouble.
Also, if the exponent is small, then we can crack the encrypted message by trial.

$$Let\ Key_p = (N, e)\ where\ e\ is\ small\ prime.$$
$$then\ for\ C = Enc(M, Key_p)$$
$$we\ can\ try\ M = \sqrt[e]{C + k * N}\ for\ different\ k\ until\ it\ get\ a\ integer.$$

b) Idk.

Problem 10.

1. First assume the column getting mixed is $[a_1, a_2, a_3, a_4]$, where each a is 8

bits and represented in two hex number.

For each a, we can represent it into binary, e.g., $(be)_{16} = (1011\ 1110)_2$.

Then we treat the expanded form as coefficients of a degree 7 polynomial.

$$(be)_{16} = (1011\ 1110)_2 = x^7 + x^5 + x^4 + x^3 + x^2 + x$$

Same as columns, the values of the $MixColumns$ matrix are also processed in same way, then calculate the matrix multiplication like normal, but replace the multiplication with polynomial multiplication in $\mathbb{GF}(2^8)$ and mod polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$.

For multiplication in $\mathbb{GF}(2^8)$, it is like the original polynomial multiplication, but if the result exists any term with degree greater than seven, then it needs to take modular with the mod polynomial.

After calculating the terms needed for one of the result vectors, we can simply add them up by doing XORs to their coefficients, then convert the polynomial into hex numbers.

2. Take input column $a = [a_1, a_2, a_3, a_4]$ and the multiplicand $b = 14 = 0E = (0000\ 1110)_2$.

For each value in a, we firstly convert it into binary then translate it into a polynomial with degree seven.

$$Let\ a_1(x) = c_7x^7 + c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x^1 + c_0$$

For calculating $a_1(x) \times b$, we can use fast exponent to deal with it without transforming it into a polynomial, since if we want to do fast multiplication with polynomial, we need to times x for each iteration, so we can take advantage of that the coefficient in $\mathbb{GF}(2)$, so the calculating in hex is same as calculating in polynomials, and this is the pseudo code:

```
Let $t = 1,\ g = a,\ res = 0$.
While $t \leq b$:
    $if\ t\&b = 1$:
        $res = res + g$. (Addition can be calculated by XOR)
    $g = g * 2,\ t = t * 2$
$output\ res\ mod\ (1001\ 1011)_2$.
```

3. Since calculating the fast exponent also cost time, and looking into a table only takes (almost) constant time, so if we can build a lookup table, then it is faster.

Since the multiplier of $MixColumns$ and $InvMixColumns$ are predefined, and the result of multiplication of all number in $\mathbb{GF}(2^8)$ with those predefined multipliers can be precomputed and store in a $2^8 \times 6$ (since there are only 6 different multipliers in $MixColumns$ and $InvMixColumns$) lookup table with $table[i][j] = i \times j$ in hex.

The advantage of building a table is the implementation (code or chips) can be

simpler and shorter or smaller, the running time is also shorter.
But on the other hands, storing the table means it would need more memory space and thus leads to more cost.

Problem 11.

Since we have c_2, m_1 and m_2 so we can first get K by trying to let $Enc(m_2, K) = m_1 \oplus c_2$ for different K to get the correct one.
After we obtain K, we can calculate $c_1 = Enc(m_1, K) \oplus m_0$
And thus $m_0 = Enc(m_1, K) \oplus c_1$.

Problem 12.

If the last m_{last} of message M_1 is known, then we can try to let $Enc(m_{last} \oplus vec) = Tag_1$ to find the needed vec.
Then arbitrary construct a message M_2 and get its Tag_2.
Let $\widetilde{M} = M_2 || (m_{last} \oplus vec \oplus Tag_2)$
Then

$$\begin{aligned} MAC(\widetilde{M}) &= Enc\big(MAC(M_2) \oplus (m_{last} \oplus vec \oplus Tag_2)\big) \\ &= Enc\big(Tag_2 \oplus (m_{last} \oplus vec \oplus Tag_2)\big) = Enc(m_{last} \oplus vec) \\ &= Tag_1 \end{aligned}$$

Extra 1.

For the roles, I think there should be:

1. **System Moderator:** to moderate the whole system, including filtering inappropriate contents, dealing with users' problems, and they are be able to view, modify, and remove any repository or other contents.
2. **Repository Owner:** can be a group, some collaborators, or a single user. They should be able to have full permissions to their repositories, including delete the repo, manage pull requests, make commits, modify the repo's information, and add new collaborators.
3. **Registered Users:** they can view others' public repositories, and clone, make pull requests, create an issue, or simply star a repo. Or they can invite another user to form a group, or to join an existing group.
4. **Anonymous User:** they can only view or clone other uses' public repo.

For the System Moderators, they need to protect the users' private content, including the API keys, or any cryptographic private key. If any of then appears in any repo, the system should have a (maybe) automated way to detect and remove the content. Except the private content, they should also prevent any possible security vulnerability of the contents. If the content is using any security vulnerable library, then the repo owner should be warned not to use those dangerous libraries in any public service.

So, for the functions, I think there should be:

1. **Create repo/pull request/issue**: this should be able for any registered users, the last two should be able to any registered user for any public repos.
2. **Register New Account/ Delete Account:** as every need to register to use its service and they can delete their accounts and any relating contents if they wish.
3. **Manage Issues/Pull requests:** this should be able to the repo's owner (and maybe moderators).
4. **Manage repo content:** every repo's owner should be able to manage their repo, or the system moderator can attend in if needed (if there are any security issue).

Since we are willing to create a free and open environment for any coder, so I think the policy should be as less as possible. These are the policies that I think there should be:

1. **The moderator can modify the content if and only if the user's private content (keys, etc.) has leaked or has any risk to leak.**
2. **Any user should create a friendly community, so their content should not include harassment, racism, or any unfriendly content.**
3. **Every release should not contain any backdoor or malicious content.**

Extra 2.

a) First, I XOR the cipher texts and get 1d0017021d1b1f0a.

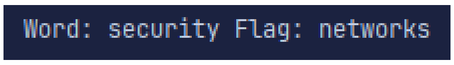
Since $C_1 = W_1 \oplus OTP$ and $C_2 = W_2 \oplus OTP$.

So, $C_1 \oplus C_2 = W_1 \oplus OTP \oplus W_2 \oplus OTP = W_1 \oplus W_2 =$
$1D\ 00\ 17\ 02\ 1D\ 1B\ 1F\ 0A$

Then I found a list of eight-letter English words on this website:
https://www.thefreedictionary.com/8-letter-words.htm

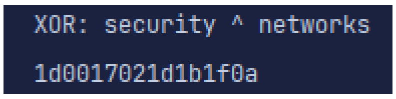
Calculate $W \oplus 1D\ 00\ 17\ 02\ 1D\ 1B\ 1F\ 0A$ for each word, and print out to find out a feasible word pair.

I find the pair from my program output:



```
Word: security Flag: networks
```

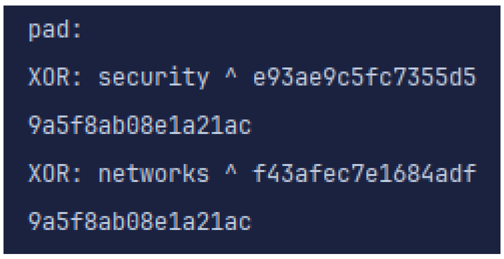
And check the XOR of the two words whether $W_1 \oplus W_2 = C_1 \oplus C_2$.



```
XOR: security ^ networks
1d0017021d1b1f0a
```

So, I think I have the right guess.

And the $OTP = W_1 \oplus C_1 = W_2 \oplus C_2$ is 9a5f8ab08e1a21ac.



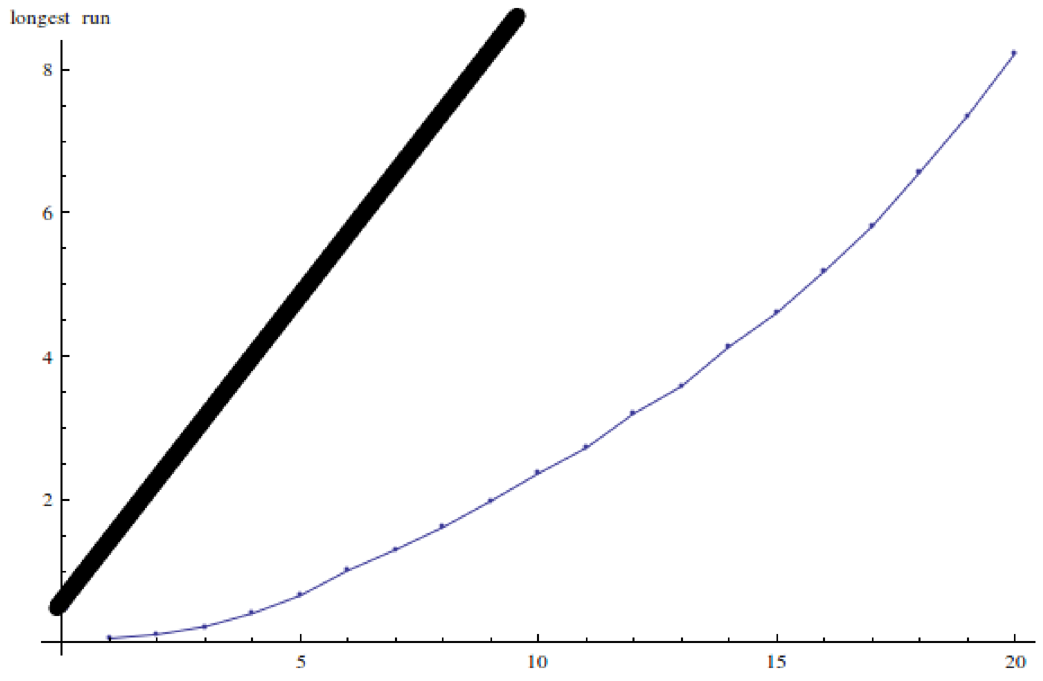
```
pad:
XOR: security ^ e93ae9c5fc7355d5
9a5f8ab08e1a21ac
XOR: networks ^ f43afec7e1684adf
9a5f8ab08e1a21ac
```

b)

Extra 3.

a) For each cipher text, I assume that the plain text is random and the OTP is also random, so the cipher text has zero with probably 0.5 for each position and one with probably 0.5.
To compare the longest common substring in c_i and c_j with $i \neq j$, we can take $c_i \oplus c_j$ and find the longest continuous zero stream.
Since c_i and c_j has uniform probability for zero and one, so do $\hat{c} = c_i \oplus c_j$.
From the formulation $\log_{\frac{1}{p}} n - \log_{\frac{1}{p}} \ln n \leq R_p(n) \leq \log_{\frac{1}{p}} n + \log_{\frac{1}{p}} \ln n$, we can plug in $p = 0.5$ to get the upper bound of longest continuous head (zero) run $R_{0.5}(n) \leq \log_2 n + \log_2 \ln n$.

b) Since $\log_2 \ln n$ is usually small, so I discard the term and change the upper bound to $R_{0.5}(n) \leq \log_2 n$.
If we draw a line with $R(n) = \lg n$, it is obvious that the bound and the read values has the large gap, thus the bound is not tight.



c) For a pad reuse, it must produce a long common substring since all pairs of plain text has long common run of identical characters.
To find the pad reuse, we firstly build a suffix tree from concatenating all the cipher texts in $O(N \log N)$.
Then, traverse the suffix tree to find the longest common suffix in the deepest common edges in $O(N)$.
From the edge found above, we can traverse up and down to construct the pair of the cipher texts that uses same padding.