Midterm

(Deadline May 16, 2024)

Problem 1

In $\mathbb{GF}(2^4)$ Compute $f(x)+g(x)$, $f(x)-g(x)$, $f(x)\times g(x)\ mod\ P(x)$, where $P(x)=x^4+x+1$.

a) $f(x)=x^2+1$, $g(x)=x^3+x^2+1$.

b) $f(x)=x^2+1$, $g(x)=x+1$.

Problem 2

In $\mathbb{GF}(2^8)$, $f(x)=x^7+x^5+x^4+x+1$ and $g(x)=x^3+x+1$.

a) Calculate $f(x)+g(x)$, $f(x)-g(x)$, $f(x)\times g(x)\ mod\ m(x)$, where $m(x)=x^8+x^4+x^3+x+1$.

b) Show that $f(x)=x^4+1$ is reducible over $\mathbb{GF}(2^8)$.
Hint: it can be written as the product of two polynomials.

Problem 3

Consider the field $\mathbb{GF}(2^4)$ with the irreducible polynomial $P(x)=x^4+x+1$. Find:

a) the inverse of $f(x)=x$ and

b) the inverse of $g(x)=x^2+x$ by trial and error.

Problem 4

In $\mathbb{GF}(2^8)$, find:

a) $(x^3+x^2+x)(x^3+x^2)^{-1}\ mod\ (x^8+x^4+1)=$

b) $(x^6+x^3+1)(x^4+x^3+1)\ mod\ (x^8+x^4+1)=$

Problem 5

Regarding the mix column operation of the AES round function, it is performed with a pre-defined matrix, i.e.,

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

a) Explain why this **mix column operation** can be implemented with a simple look up table and XOR.

b) Apply the same idea used above, explain why the **byte substitution**, **shift row** and **mix column** can be combined and implemented as a simple look-up table operation. And then the whole AES is implemented by look up table and few XORs.

Problem 6

In order to make AES encryption and decryption more similar in structure, the MixColumns operation is missing in the last round.

Explain how to take **advantage** of this property to share some of the code (for software implementation) or chip area (for hardware implementation) for AES encryption and decryption.

Hint: Denote InvSubBytes, InvShiftRows, and InvMixColumns as the inverses of SubBytes, ShiftRows, and MixColumns operations, respectively. Try to show:
1. The order of InvSubBytes and InvShiftRows is indifferent;
2. The order of AddRoundKey and InvMixColumns can be inverted if the round key is adapted accordingly.

Problem 7

Under **what circumstances** could you choose 3DES over AES? Also, what are the **advantages** of choosing AES over 3DES? Is 3DES **susceptible** to Meet-in-the-Middle Attacks like 2DES? Please explain.

Problem 8

If a company has encrypted its most sensitive data with a key held by the chief technology officer and that person was fired, the company would want to change its encryption key. Describe **what would be necessary** to revoke the old key and deploy a new one.
Hint: NIST Special Publication 800-57 Part 1

Problem 9

Bitdiddle Inc. requires every employee to have an RSA public key. It also requires the employee to change his or her RSA key at the end of each month.

a) Alice just started working at Bitdiddle, and her first public key is (n, e) where n is the product of two safe primes, and $e = 3$.
Whenever a new month starts, Alice (being lazy) changes her public key as little as possible to get by the auditors. What she does, in fact, is just to advance her public exponent e to the next prime number. So, month by month, her public keys look like:

$$(n, 3), (n, 5), (n, 7), (n, 11), ...$$

Explain how Alice's laziness might get her in trouble.

b) The next year, Alice tries a different scheme.
In January, she generates a fresh public key (n, e) where n is the product of two primes, p and q. In February, she advances p to the next prime p_l after p, and q to the next prime q_l after q, and sets her public key to (n_l, e_l) for a suitable e_l. Similarly, in March, she advances p_l to p_2 and q_l to q_2, and so on.

Explain how Alice's scheme could be broken.

Problem 10

The Advanced Encryption Standard (AES) requires not only the MixColumns step during encryption but also the Inverse MixColumns during decryption.

a) Given the matrix for the Inverse MixColumns operation:

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}$$

Describe how the matrix multiplication in the Inverse MixColumns step is performed within AES decryption. Your answer should include a **brief explanation of the arithmetic** used in the finite field $\mathbb{GF}(2^8)$ and **how it differs from standard matrix multiplication**.

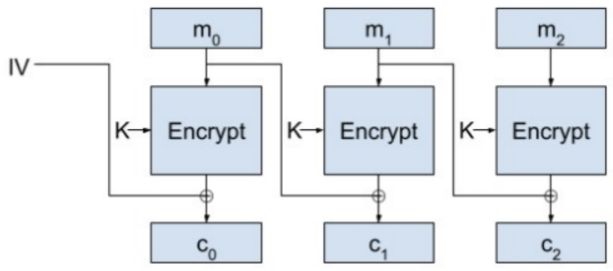
b) Multiplications in $\mathbb{GF}(2^8)$ can be complex. However, they can be simplified using repeated application of simpler operations. Explain how multiplication by 9, 11, 13, and 14 in $\mathbb{GF}(2^8)$ can be implemented using the simpler multiplication by 2 and addition (XOR). Provide the **mathematical expressions** that represent these multiplications.

c) Discuss how the use of lookup tables (LUTs) can further simplify the implementation of Inverse MixColumns in AES decryption. What are the **advantages** and potential **drawbacks** of using LUTs for this purpose?

Hint 1: https://www.youtube.com/watch?v=dRYHSf5A4lw
Hint 2: https://dl.acm.org/doi/abs/10.1145/3405669.3405819
Hint 3: https://xilinx.github.io/Vitis_Libraries/security/2019.2/guide_L1/internals/aes.html

Problem 11

In class we saw several modes such as ECB, CBC and CTR mode. Let's look at another possible mode of operation "plaintext block chaining" (PBC) which is similar to cipher block chaining (CBC) but allows for encryption in parallel:

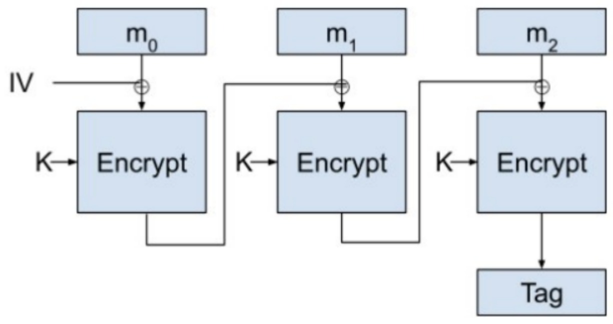


Unfortunately, PBC mode is not secure.

To see this, show how an attacker who knows IV, C_0, C_1, C_2 (which are public) and also knows that $m_1 = m_2 = x$ (for a known x) can easily compute m_0.

Problem 12

Alice is trying to design a MAC using a block cipher. She decides to use the following construction, which is essentially just CBC encryption, throwing away all but the final block.



Unfortunately, this construction is not secure.

Describe how to produce an existential forgery against this MAC scheme.

Hint 1: Start with two messages M_1 and M_2 (not to be confused with the individual blocks of a message in the diagram above) for which you know the outputs (IV_1, T_1) and (IV_2, T_2). Produce another message M_3 for which (IV_1, T_2) will be the MAC. M_3 will be close to the concatenation $M_1||M_2$, but with one block altered.

Hint 2: There is also a way to produce a forgery with only one known block if you look closely.

Caution: The blocks $m_0, m_1, \ldots$ in the diagram are distinct from the complete messages $M_1, M_2, \ldots$

Extra Credit 1

To help you understand how to develop security policies for systems used by many people and might be particularly relevant if you use GitHub or plan to design similar systems.

Please write a security policy for a website like GitHub that hosts distributed version control systems. Define the various **roles**, **functions**, and **policy** that should be included. This policy should be sufficiently detailed for someone who wants to create a site similar to GitHub. For this task, assume that GitHub is the official maintainer of the Git technology. If you cannot find relevant material on GitHub as currently implemented, invent new material as appropriate. Try to be as complete as possible, but emphasize the GitHub-specific aspects: particularly, **what security goals are the most relevant for GitHub? What roles does (or, rather, should) GitHub have, and what should each principal be allowed to do?**

Extra Credit 2

It is well known that re-using a "one-time pad" can be insecure. This problem explores this issue, with some variations.
In this problem all characters are represented as 8-bit bytes with the usual US-ASCII encoding (e.g.,"A" is encoded as 0x41). The bitwise exclusive-or of two bytes x and y is denoted $x \oplus y$.

Let $M = (m_1, m_2, \ldots, m_n)$ be a message, consisting of a sequence of n message bytes, to be encrypted. Let $P = (p_1, p_2, \ldots, p_n)$ denote a pad, consisting of a corresponding sequence of (randomly chosen) "pad bytes" (key bytes).
In the usual one-time pad, the sequence $C = (c_1, c_2, \ldots, c_n)$ of ciphertext bytes is obtained by xor-ing each message byte with the corresponding pad byte:

$$c_i = m_i \oplus p_i, \text{ for } i = 1 \ldots n \ .$$

When we talk about more than one message, we will denote the messages as $M_1, M_2, \ldots, M_k$ and the bytes of message $M_j = (m_{j1}, m_{j2}, \ldots, m_{jn})$; we'll also use similar notation for the corresponding ciphertexts.

a) Here are two 8-character English words encrypted with the same "one-time pad".

e9 3a e9 c5 fc 73 55 d5

f4 3a fe c7 e1 68 4a df

What are the words? Describe how you figured out the words.

b) Ben Bitdiddle decided to fix this problem by making sure that you can't just "cancel" pad bytes by xor-ing the ciphertext bytes.

In his scheme the key is still as long as the ciphertext. If we define $c_0 = 0$ for notational convenience, then the ciphertext bytes $c_1, c_2, \ldots, c_n$ are obtained as follows:

$$c_i = m_i \oplus ((p_i + c_{i-1}) \mod 256).$$

That is, each ciphertext byte is added to the next key byte and the addition result (modulo 256) is used to encrypt to the next plaintext byte.

Ben is now confident he can reuse his pad, since $(k_i + c_{i-1}) \mod 256$ will be different for different messages, so nobody would be able to cancel the k_i's out. You are provided with `otp-feedback.py`, which contains an implementation of Ben's algorithm.
You are also given the file `tenciphs.txt`, containing ten ciphertexts $C_1, C_2, \ldots, C_{10}$ produced by Ben, using the same pad P. You know that these messages contain valid English text.
Submit the messages and the pad, along with a careful explanation of how you found them, and any code you used to help find the messages. The most important part is the explanation.

Extra Credit 3

In the previous problem, we saw how to attack a scheme in which a one-time pad, or a scheme like it, is reused. This problem will walk you through how the rare reuse of the pad for the standard one-time pad (not the scheme of the previous part) can be efficiently detected. We will look at the extreme case in which a pad is reused just once.

The following fact may be useful in this problem: Let $R_p(n)$ be the longest run of heads in n coin flips, each of which is heads with probability p. With high probability as n grows, $R_p(n)$ is between $\log_{1/p} n - \log_{1/p} \ln \ln n$ and $\log_{1/p} n + \log_{1/p} \ln n$.

a) Suppose you are given poly(n) n-bit ciphertexts $c_1, \ldots, c_q$, each of which is properly encrypted with an independent uniformly randomly chosen one-time pad. Show that with high probability, the length of the longest repeated bitstring (i.e. a substring of both c_i and c_j for some $i \neq j$) is less than $\log_2 n + \log_2 \ln n$.

b) We have collected data on the average length of the longest run of identical characters in identical positions in passages of English text by randomly sampling the collected works of the American writer Winston Churchill. You can find a graph of run length (in characters) versus log of the message length in characters at https://courses.csail.mit.edu/6.857/2014/files/run_lengths.png. How do these run-lengths compare to the upper bound of the previous part on the longest repeated bit string if the English plaintext is US-ASCII encoded?

c) Assume that each pair of plaintexts shares a long common run of identical characters, and no pair of ciphertexts with independently chosen pads does. Show that given poly(n) n-bit ciphertexts with total length N and one instance of pad reuse, you can find the two ciphertexts that share a key in time which is $O(N)$ in the total ciphertext length. This means that your solution should run in $O(N \log N)$ time for some constant c.

Hint: You may wish to read about and use suffix trees in your solution. They are a type of tree that stores all suffixes of a string of n characters, can be constructed in $O(n \log n)$ time and can be used, among other things, to find the longest repeated substring of a string in linearithmic time by finding the deepest non-leaf node. Note that the longest repeated substring is not exactly what you need to solve this problem. If you are not familiar with suffix trees, you may or may not find suffix arrays simpler and easier to understand.

What to turn in:

a) The file you need to upload is structured as follows:

- <student_id>.pdf

b) The <student_id>.pdf file should contain the solution to the provided problems.

c) Plagiarism of answers from the Internet or ChatGPT is prohibited.

Grading Policy

a) The total score of 120% is achievable by answering all 12 regular questions, each worth 10%.

b) You can earn 10% for each extra credit question.

c) Successfully answering all extra credit questions can increase the maximum score by 30%, resulting in a total possible score of 150%.