Even-Numbered Problem Solutions to *Understanding Cryptography* From Established Symmetric and Asymmetric Ciphers to Post-Quantum Algorithms by Christof Paar- Jan Pelzl - Tim Guneysu

Mustapha EL BOUAZAOUI.
mustaphaelbouazaoui@gmail.com

July 25, 2025

Abstract

While trying to recall a bit of cryptography to prepare for interviews. I find the masterpiece of lectures by Mr. christophe Paar in youtube. From then, I find out the book and while trying to solve some problems from it, I find out there's book only for odd-numbered problems. I find it intriguing and interesting as if the writer want someone to do the other part. And as far as I am aware, none has done the other part. So, I decided to be the one who gonna do the other one and publish them and try to add bunch of code in python as an example for those ciphers.
keywords: Mathematics, cryptography, problems, solutions.

Contents

Chapter 1 Introduction to Cryptography and Data Security

1.1 See code 1.1.py.

1.2

We know we are dealing with a shift cipher. Hence, we can perform letter frequency analysis to guess k: the number of positions by which the most frequent letter (usually "e" in English) has been shifted. After deciphering, we found:
"If we all unite, we will cause the rivers to stain the great waters with their blood" — Tecumseh in his speech to the Osages.
See 1.2.py for the code used.

1.3

There's a small mistake in the solution, as the ASIC costs \$50, not \$100.

1.4

1. For each letter, there are 128 possible characters. Since we have 8 letters, the size of the key space is 128^8.
2. Each letter uses 7 bits, so the key length is $7 \times 8 = 56$ bits.
3. Similarly, if only lowercase letters are used, the size of the key space is 26^8.
4. Representing 26 letters requires $\frac{\log 26}{\log 2} \approx 4.7$ bits, which rounds up to 5 bits per character. Hence, the key length is $5 \times 8 = 40$ bits.
5. (a) For 7-bit characters, we need $\frac{128}{7} \approx 18.3$, so we need at least 19-character passwords.
   (b) For 26 lowercase letters, we need $\frac{128}{5} = 25.6$, so we need 26-character passwords.

1.5 *Hint:* Use the identity $p^n - 1 = (p-1)\left(\sum_{i=0}^{n-1} p^i\right)$. Straightforward calculation.

1.6

Attacker	Can read?	Can alter?	Why?
Hacker between Alice and base station A	No	No	Sees only y_1 and does not have k_1.
Mobile operator on A	Yes	Yes	Controls base station A and knows k_1 and k_2.
National law enforcement agency	Yes	Yes	Same reason as (b) or (e), once access is obtained.
An intelligence agency of a foreign country	No	No	Only sees y_2.
Mobile operator on B	Yes	Yes	Same reason as (b).
Hacker between Bob and base station B	No	No	Only sees y_3 and does not know k_3.

- None can read or alter the message, since they only see c, which only Alice and Bob can decrypt using their mutual key k_{AB}.

1.7 Easy.

1.8

- $5 \times 8 = 40 \equiv 1 \mod 13$
- $5 \times 3 = 15 \equiv 1 \mod 7$
- $3 \times 2 \times 5^{-1} = 6 \times 3 = -3 \equiv 4 \mod 7$

1.9 Straightforward.

1.10

- $5 \times 9 = 45 \equiv 1 \mod 11$
- $5 \times 5 = 25 \equiv 1 \mod 12$
- $5^{-1} \equiv 8 \mod 13$

Hence, the multiplicative inverse of a number (if it exists) depends on the ring we are working in (e.g., $\mathbb{Z}_{11}$, $\mathbb{Z}_{12}$, etc.).

1.11 Straightforward calculation.

1.12

- $\phi(4) = 2$
- $\phi(5) = 4$
- $\phi(9) = 6$
- $\phi(26) = 12$

1.13 See code 1.13.py

1.14

- $y = a \cdot x + b \mod 30$
- $30 \times 30 = 900$
- Using `pow(17,-1,30)`, we get $17^{-1} \equiv 23 \mod 30$. The corresponding plaintext is **FRODO**.
- ?

1.15 Simple manipulation of the two affine equations: subtract one from the other. The inverse exists because $\gcd(x_1, m) = \gcd(x_2, m) = 1$ (since b_1 and b_2 are non-zero), so $\gcd(x_2 - x_1, m) = 1$.

1.16

1. $b_3 = a_2 \cdot b_1 + b_2$ and $a_3 = a_2 \cdot a_1$
2. $a_3 = 7$ and $b_3 = 10$
3. $K = 10 \to 9 \to 2$, and using the formula from (2), we get $K \to 2$
4. The effective key space does not change: 26×26

1.17 See odd-numbered solution.

Chapter 2 Stream cipher

2.1

1. $y_i = x_i + k_i \mod 26$
2. $x_i = y_i - k_i \mod 26$

for binary, when decoding we do $x_i = y_i + k_i \mod 2$ because $-$ and $+$ operations in $\mathbb{Z}_2$ are the same.

2.2

The lifecycle of the key-DVD:

- Creation: it should be created from a TRNG.
- Usage: a key segment must be used only once and only for one message. If reused(even partially), it compromises security.
- Destruction: Once used, the key material must be securely deleted and not just erased(recovery via forensics).

Storage of the key:

- During usage:
  - Access to the DVD must be controlled.
  - Requires physically secure environments.
- After usage: Shredded or destroyed using secure media destruction methods and no copies or backups should remain.

Key distribution:

- Two identical copies should be distributed to both parties.
- high risk of interception during delivery of the key $\Rightarrow$ tampering or duplication without detection.

Scalability:

it's not scalable, for n users wanting to communicate securely, we need $\binom{n}{2} \approx O(n^2)$ keys.

Operational implication:

- Message length: with a 1 Gigabyte key-DVD, we can only encrypt 1 Gigabyte of data maximum.
- Human error: with such system, there's high chance to human errors (reusing a key part, not destroying it properly) which is fatal to security.

2.3

We have $Y_i = X_i \oplus K_i$, and it's periodic, hence we only need the first 128 keys to decipher all the text. So, this is vulnerable to *known-plaintext attacks* or if the first 128 bits (16 bytes) are predictable (headers, protocols ...).

2.4

we have $y_i = x_i + k_i \mod 2$ for $i \in [|0, 39|]$.Hence, there's 2^{40} possible keys.
Let's suppose an attacker tries every possible key $k' \in \{0,1\}^{40}$:

$$x' = y \oplus k'$$

Then they get 2^{40} possible plaintexts, each one perfectly plausible.
The attacker has no way to know which one is the correct plaintext, because every possible key **maps** leads to a valid-looking plaintext.

2.5
hint: from hexadecimal to binary, we transform each number separately so that every couple transforms into an 8-bit. e.g $6a \rightarrow 01101010$.
Hence with straightforward calculation, the text is **LetsEncryptThisBook**.

2.6

OTP offers provable security, but for instant messaging or emails it has two major drawbacks:

- Not fast: The key should be as long as the message itself. Which means more packets to send each time you want to send a message.
- One-use only: The key should be used once, hence each time there's the need to share secretly the key (either physically or through more complicated and slow processus).

For this reason, applications such as instant messaging or emails are not practical for OTP use.

2.7

1. 0010111
2. 0111001
3. The second sequence is a rotation of the first one RT^{3L}

2.8
In the case of short period stream cipher, we can easily use known-plaintext attacks to recover the first 200 bits of the key. It's plausible, because it corresponds to 25 bytes of information (which is most likely the header used for a protocol). Since

$$k_i = y_i \oplus x_i, i \in [|0, 199|]$$

Afterward, we can easily determine the period T (using an algorithm or by brute force).
Finally, we can decipher the rest of the text using

$$x_i = y_i \oplus k_i \mod T$$

2.9 hint: $(FF)_{16} = (11111111)_2$ and $(0, 2, 3, 4, 8) \Rightarrow X^8 + X^4 + X^3 + X^2 + 1$
Unless I am mistaken, there's an error in the schema used, and the polynomial represented in the schema is $X^8 + X^4 + X^3 + X + 1$. Thus, the solution is $(1101000011111111)_2 = (D0FF)_{16}$

2.10

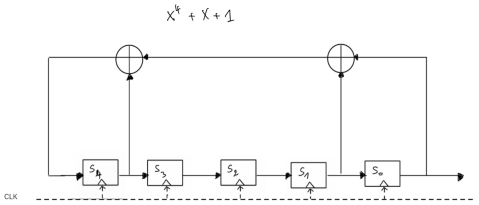

Figure 1: LFSR structure for the $X^4 + X + 1$

1. • This polynomial is primitive over $\mathbb{F}_2$.

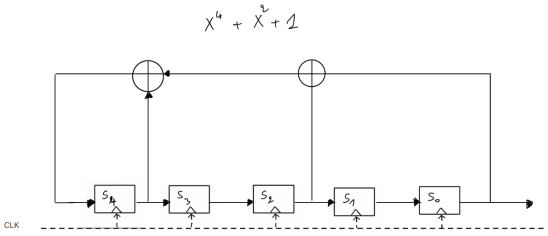

Figure 2: LFSR structure for $X^4 + X^2 + 1$

- it generates maximum-length sequence of length 15.

2.
- This polynomial is irreducible but not primitive over $\mathbb{F}_2$
- The sequence length doesn't depends on the initial value
- sequence length 5

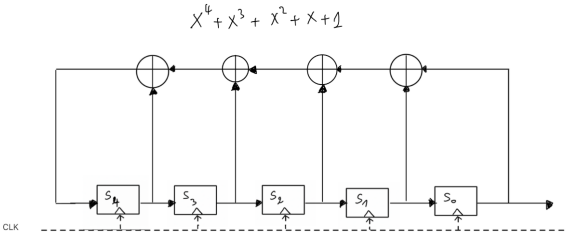

Figure 3: LFSR structure for $X^4 + X^3 + X^2 + X + 1$

3.
- This polynomial is reducible over $\mathbb{F}_2$
- factor $(X^2 + X + 1)^2$
- The sequence length depends on the initial value.

2.11
For a LSFR of degree m, you only need $2m = 512$-bit pair. The key formula is

$$S_{m+i} = \sum_{j=0}^{m-1} p_j.S_{j+i}$$

and gaussian elimination to retrieve p_j and easily build the system.

2.12

1. $K = Y \oplus X$. We find: $K = (0010111)_2 \rightarrow$ the period is 7, hence the degree of LFSR is $2^m - 1 = 7 \Rightarrow m = 3$
2. The initialization vector is $= (111)_2$ [initial content of the key].
3. We find $X^3 + X + 1 \Rightarrow (3, 1, 0)$
4. easy to verify with $S_{i+3} = S_{i+2} + S_i \mod 2$

2.13 known plaintext-attack. The key is to notice the first parts of the key are actually the initialization vector $(111111)_2$, and using the stream cipher equation

$$S_{i+5} = \sum_{i=0}^{5} p_j.S_{i+j}$$

we find an easy system to solve:

$$\begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

from here we just need to build the LFSR, find the key and XOR it with ciphertext.

2.14 Let's start with converting from ASCII to decimal (you can use some online converter):

- G $\Leftrightarrow$ 71
- I $\Leftrightarrow$ 73
- F $\Leftrightarrow$ 70

using the encoding equations we have :

$$\begin{aligned} 32 &\equiv 71 + z_1 \mod 257 \\ 166 &\equiv 73 + z_2 \mod 257 \\ 87 &\equiv 70 + z_3 \mod 257 \end{aligned}$$

hence $z_1 = 218$, $z_2 = 93$ and $z_3 = 17$. Moreover, we have :

$$\begin{aligned} 218 &\equiv a.z_0 + b \mod 257 \\ 93 &\equiv 218.a + b \mod 257 \\ 17 &\equiv 93.a + b \mod 257 \end{aligned}$$

taking the (2) - (3), we have $125.a \equiv 75 \mod 257$. we have $gcd(125, 257) = 1$, you can use the python function *pow(125,-1,257)=220* to find the reverse of 125 modulo 257.
Thus, $a \equiv 15 \mod 257$, $b = 164$ and $z_0 = 55$ (we calculate the inverse of a similarly as above).
This attack is **known-plaintext attack** and the prerequisites are minimal :

- The ciphertext values.
- the modulus m. (in case of image encrypting, it's well-known value).
- at least 3 bytes of the plaintext. (structure or fixed parts of the plaintext. (e.g. headers like GIF98a)

2.15
Inserting the values in the equations $z_(i + 2) \equiv a.z_{i+1} + b.z_i + c \mod 257$ gives us a linear system with 3 equations and 3 unknowns. Easy solving with Gaussian elimination or we solve the system directly using the NumPy library in Python.

2.16

The initial state consist of: $\begin{pmatrix} c_0 & k_0 & k_1 & k_2 \\ k_3 & c_1 & n_0 & n_1 \\ p_0 & p_1 & c_2 & k_4 \\ k_5 & k_6 & k_7 & c_3 \end{pmatrix}$

where k_i are the keys, p_i counter for the position of the current 512-bit block , n_i is the nonce used, and c_i are given by the ASCII encoded string "expand 32-byte k"[1].
Thus,

$$\begin{pmatrix} 0x61707865 & 0x00000000 & 0x00000000 & 0x00000000 \\ 0x00000000 & 0x3320646e & 0x00000000 & 0x00000000 \\ 0x00000000 & 0x00000000 & 0x79622d32 & 0x00000000 \\ 0x00000000 & 0x00000000 & 0x00000000 & 0x6b206574 \end{pmatrix}$$

2.17 straightforward calculation, other than $ROTL^7$ that needs to convert it to binary before doing it. And we have as a result

$$QR(a, b, c, d) = 0x10000001, 0x80808808, 0x01010110, 0x01000110$$

2.18
We first need to initialise A, B and C. given:
A # Key (80 zeros) + 10 zeros + 111
B # IV (80 zeros) + 1111
C # 108 zeros + 111
using the code trivium.py the answer is:
00101101110001011000000000000000000000000000000000000000000000000000000000000000

[1]'expa' → 65 78 70 61, but we use little-endian representation as '0x61707865' because salsa20 was designed with performance on mind on little-endian architecture (like x86 CPUs)