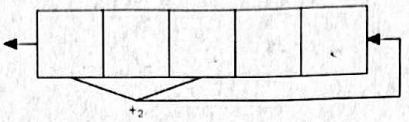
Anthony Petrosino

CSCI 325 - Fall 25 - Homework3

Due date: Sunday, Sep. 28th, 11:59 PM

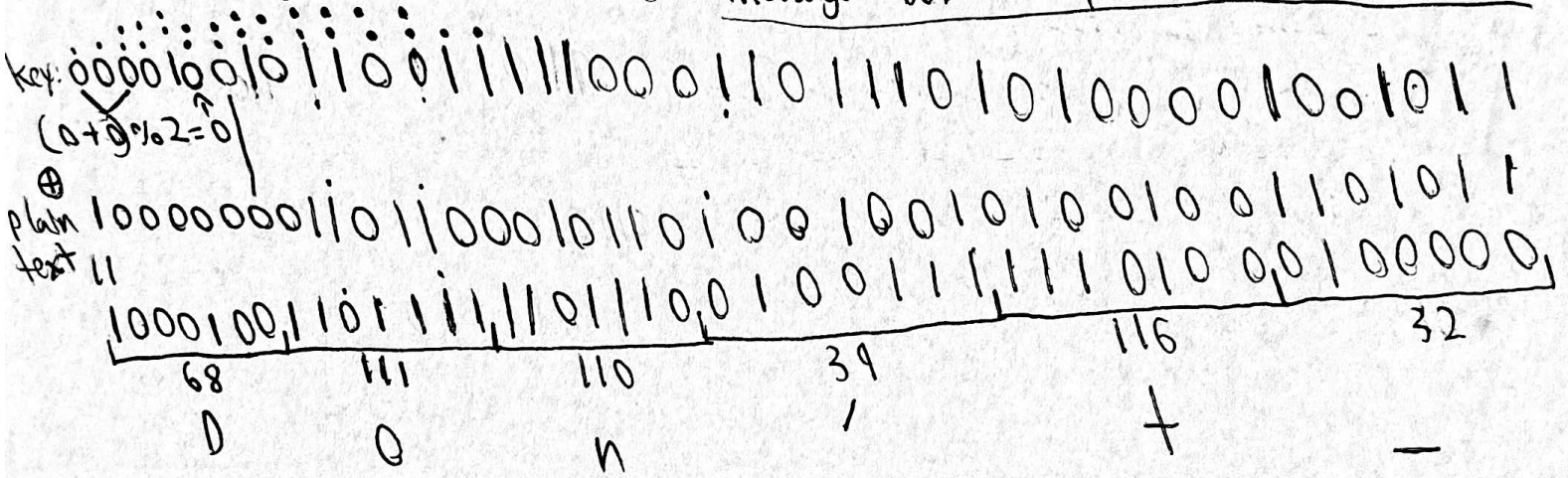
1. (5 points) You and a friend agree to use 7-bit ASCII and the recursion $b_n \equiv (b_{n-3} + b_{n-5}) \mod 2$ with keyword 00001. - Linear feedback shift register, LFSR

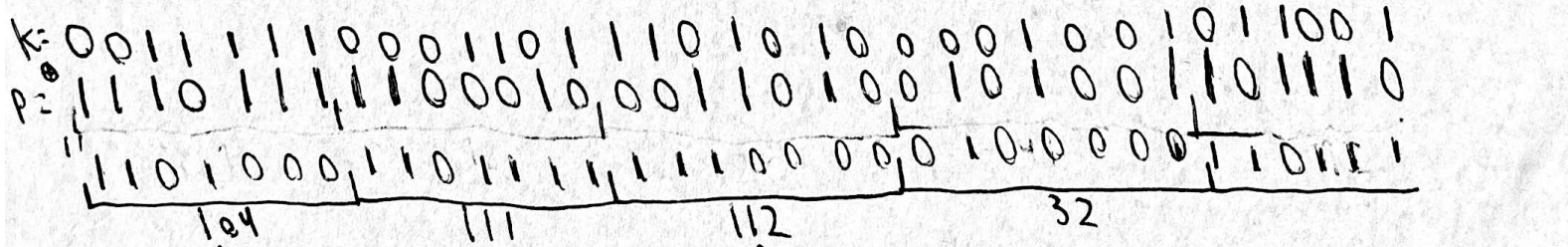


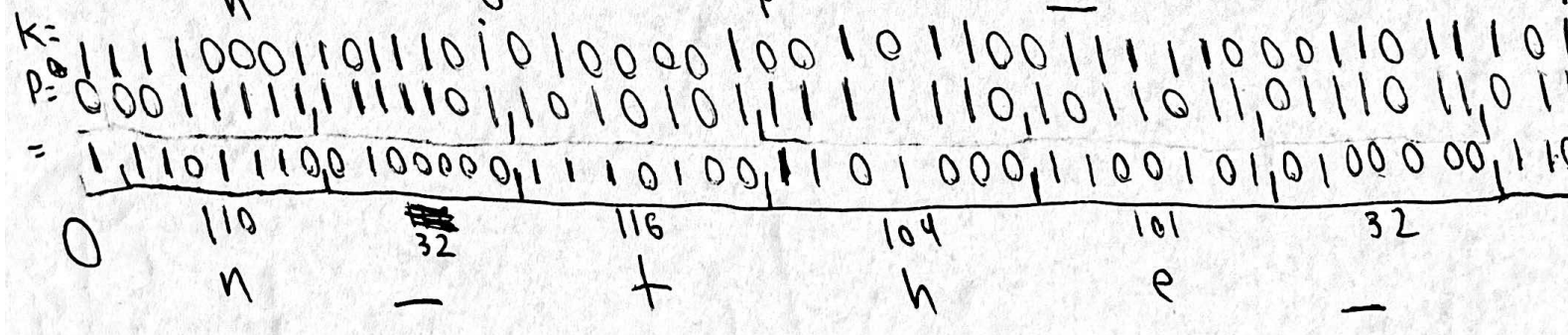
Your friend sends the following message:

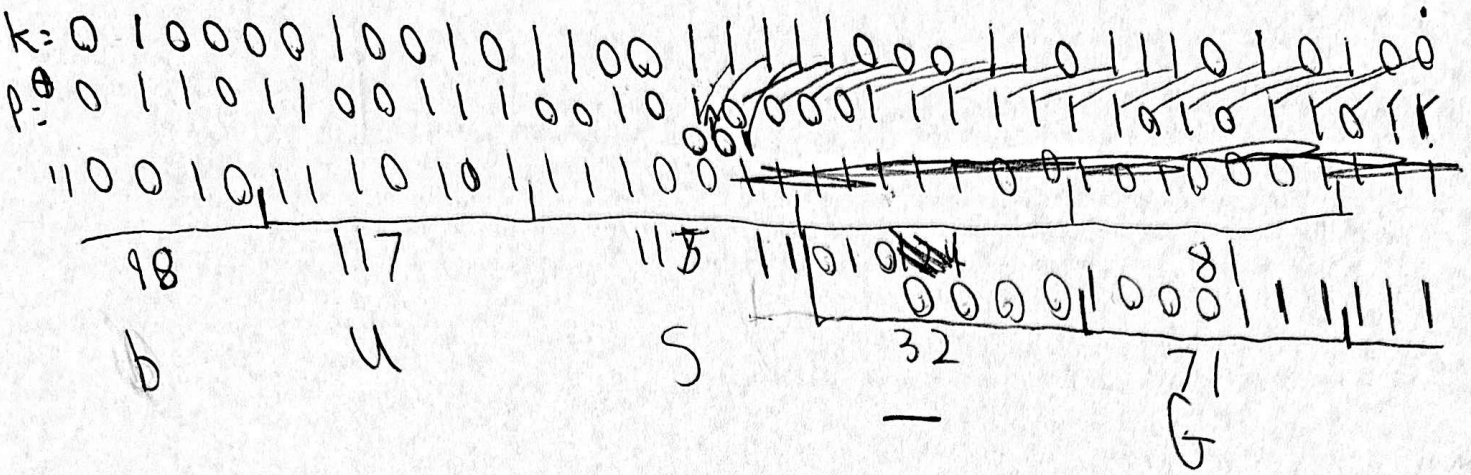
```
1000000011011000101101001001010010011010111110111110001000110100101001101110
0001111111111011010101111111010110110111011011011011001110010100100001111111
                    0101101111011111010010
```

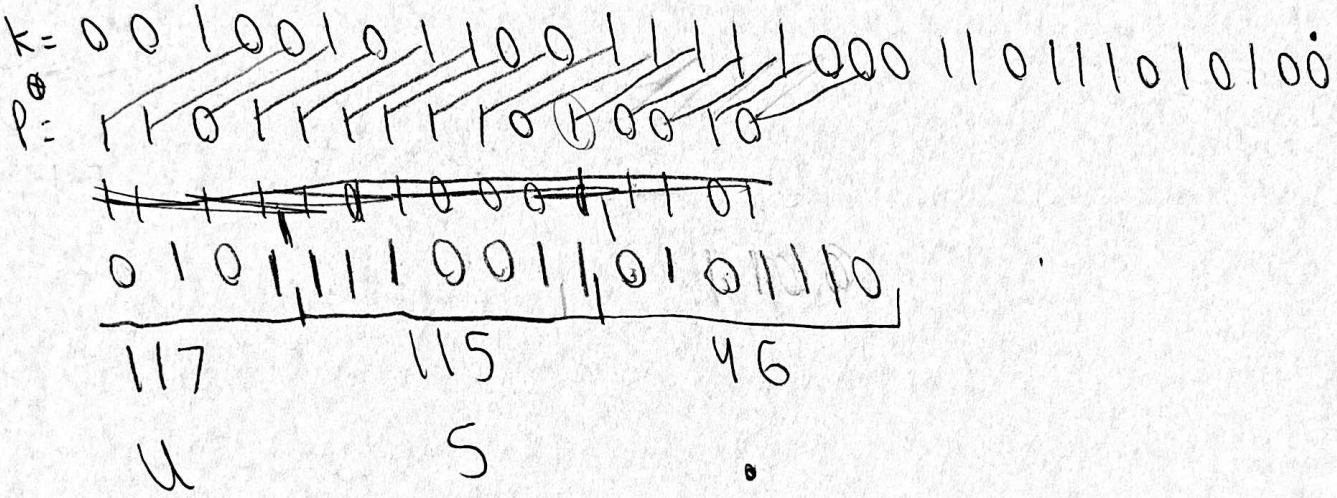
Decipher and write down the message. Message: Don't hop on the bus Gus.








k=
P=
98
117
115
32
71
b
u
s
—
G


k=
P=
117
115
46
u
s

2. (a) (2 points) What is openssl?

Open ssl is an open-source library that provides tools to implement SSL and TLS cryptographic protocols. This makes it crucial for HTTPS websites.

(b) (3 points) Please use openssl to encrypt a message "Hello world" using CBC mode with password "secret" and then decrypt the message. Please show your command below with explanation.

Encryption:

all one line: `echo -n "Hello world" | openssl enc -aes-256-cbc -pbkdf2 -pass pass:secret -base64 -out message.enc`

- `echo -n "Hello world"` — supplies plaintext
- `|` — pipe
- `openssl enc -aes-256-cbc` — use AES in CBC mode
- `-pbkdf2` — derive key using PBKDF2
- `-pass pass:secret` — set password to secret
- `-base64` — encode into Base64
- `-out message.enc` — output encrypted text to file message.enc

Decryption:

`openssl enc -d -aes-256-cbc -pbkdf2 -pass pass:secret -base64 -in message.enc`

- `openssl enc -d` — Openssl with decrypt mode
- `-aes-256-cbc` — same algorithm as encryption
- `-pbkdf2` — same key derivation
- `-pass pass:secret` — use secret as password
- `-base64` — decode from base64
- `-in message.enc` — read in message.enc file

3. Most of the encryption modes require an initialization vector (IV). A basic requirement for IV is uniqueness, which means no IV may be reused under the same key. This requirement is easy to understand when the two plaintexts are the same, because using the same IV will result in the same ciphertext. While one may argue that their plaintext will never repeat, so it is safe to use the same IV. While this may be true for some encryption modes, it is dangerous for several encryption modes. One of these mode is the Output Feedback (OFB) mode.

   (a) (2 points) Explain or draw a diagram to show how OFB mode work for encryption.

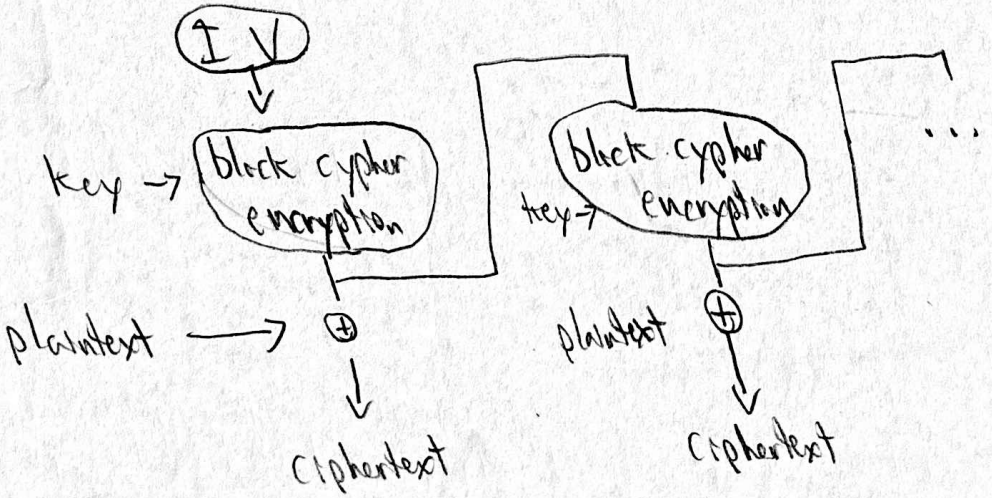
   (b) (3 points) What will happen if you reuse IVs in the OFB mode?

   (c) (3 points) Suppose two plaintexts P_1 and P_2 are encrypted to ciphertexts C_1 and C_2 respectively. If P_1 is disclosed to an attacker who also knows C_1 and C_2. The attacker also know that P_1 and P_2 are encrypted with the same IV in OFB mode. Can you help the attacker to figure out P_2?

   (d) (5 points) Please use openssl with OFB mode to simulate the attack in part (c). Demonstrate your demo a class peer with example plaintexts and ciphertexts (2 points).

explain Theoretically how it works

(a.) OFB can be used to turn a block cipher into a stream cipher by generating a keystream. First, we choose an IV with the same length as the block size, which we share with the recipient. Then, we encrypt the IV with a block cipher to produce the first keystream block, which we then XOR with the plaintext block to give us our first ciphertext block. We use that generated keystream block as an input for the next block cipher encryption and repeat.



(b.) Reusing IVs in OFB is very bad as it will produce two ciphertexts that use the same keystream, allowing for others to determine the xor of the two ciphertexts and then guess parts of the plain text.

3.

c) To figure out P_2, we can use P_1, C_1 and C_2. Since $C_1 = P_1 \oplus K$ and $C_2 = P_2 \oplus K$, the attacker can compute $K = C_1 \oplus P_1$ and use it to substitute for K, resulting in $P_2 = C_2 \oplus K$ becoming $P_2 = C_2 \oplus (C_1 \oplus P_1)$, whose values are known.

d) To simulate the attack, I first created p1.bin "Hello World!" and p2.bin "Secret message". I then created an example IV and key using command openssl rand -hex 16 and set KEY= to my generated key and IV= to my generated IV in my command line. I then encrypted both with:

- openssl enc -aes-128-cfb -k $KEY -iv $IV -in p1.bin -out c1.bin
- openssl enc -aes-128-cfb -k $KEY -iv $IV -in p2.bin -out c2.bin

Which I stored in an executable encrypt.sh.
To implement the logic to solve for p2, I created the attached python file, which executes what is described in c). The output goes into recovered_p2.bin. This only recovers up to p1.length bytes.