proj2

February 28, 2026

1 Project 2 — Cryptography (CS-GY 6903)

Group members: - Darren Tahe (dt2607@nyu.edu) - Matthew Bentz (mb9661@nyu.edu) - Srikanth Akella (ta2728@nyu.edu) - Matthew Mobijohn (mm7655@nyu.edu)

1.0.1 Problem 1 - (Understanding One-Time Pad)

Research the theoretical basis of the one-time pad, including its requirements and operational principles. Describe one-time pad conditions clearly.

1.0.2 Solution

The One-Time Pad is a symmetric encryption scheme well known for its theroretical perfect security under strict conditions. In the One-Time Pad, a key is generated randomly using the key generation formula, KGen(1n). To encrypt a message Enc(k, m   0,1), the sender computes the ciphertext c = k   m, where   denotes XOR. Decryption follows the same operation for m = k   c. Correctness follows from the properties of XOR since for all k and m   0,1 , it is true that Dec(k, Enc(k, m)) = m, because Dec(k; Enc(k;m)) = Dec(k; k   m) = k   k   m = 0   m = m. Therefore, decryption always recovers the original message.

Perfect security means that the ciphertext does not give an attacker any information about the plaintext, even adversaries with infinite time and computing power.

For the One-Time Pad to achieve its theoretical perfect security, it must adhere to the following conditions in operation: * the key is truly random * the key is at least as long as the message * the key is used only once

Violating any of these conditions will create vulnerabilities in cryptanalysis. Theoretically, One-Time Pad is unbreakable under its assumptions but its real-world implementations face challenges in generating, distributing and securely managing large one-time keys.

1.0.3 Problem 2 - (One-Time Pad Implementation)

The encryption and decryption process between two parties, Alice and Bob.

1.0.4 Solution

1. Alice's Program

- Should prompt for a message input (plaintext), then display the ciphertext, and save both the ciphertext (in hex) and the key (in hex) in separate files.

```
[ ]: import os

     # prompt for plaintext
     plaintext = input("Enter the plaintext: ")

     # encrypt plaintext using xor
     key = os.urandom(len(plaintext.encode('utf-8')))
     ciphertext = bytes(a ^ b for a, b in zip(plaintext.encode('utf-8'), key))

     print("Ciphertext :", ciphertext.hex())

     # save ciphertext(hex) and key(hex) to separate files
     with open("ciphertext.txt", "w") as f:
         f.write(ciphertext.hex())
     with open("key.txt", "w") as f:
         f.write(key.hex())
```

```
Ciphertext : a89ecca4582088bccedc483740cf23de4a933b67
```

2. Bob's Program

- Should read the key and ciphertext from their respective files and display the decrypted plaintext.

```
[161]: # read key, ciphertext from files
       with open("ciphertext.txt", "r") as f:
           ciphertext_hex = f.read()
       with open("key.txt", "r") as f:
           key_hex = f.read()

       # decrypt
       ciphertext = bytes.fromhex(ciphertext_hex)
       key = bytes.fromhex(key_hex)
       plaintext = bytes(a ^ b for a, b in zip(ciphertext, key))

       # print plaintext
       print("Decrypted plaintext:", plaintext.decode('utf-8'))
```

```
Decrypted plaintext: this is my plaintext
```

1.0.5 Problem 3 - (Implementing Many-Time Pad)

Modify the one-time pad implementation to encrypt multiple messages with the same key, simulating a many-time pad scenario. The purpose of this problem is to see if there are any recognizable patterns by observing the outputs. You can gain insights by changing the plaintexts or the key to verify your findings. These findings would be useful in the next problem.

- The program should encrypt a list of 10 predefined plaintext messages with a single key, saving the plaintexts, key, and ciphertexts (all in hex) into a file. You can select 10 of your favorite messages. Assume the key is long enough to do encryption to all the 10 messages.

1.0.6 Solution

[163]:
```
## Plaintexts

plaintexts = []
for i in range(10):
    plaintexts.append("plaintext " + str(i))

print("plaintexts")
for plaintext in plaintexts:
    print(plaintext)
print()

## Key

key = os.urandom(len(max(plaintexts, key=len).encode('utf-8')))
print("key", key.hex())
print()

## Ciphertexts

ciphertexts = []
for plaintext in plaintexts:
    ciphertext = bytes(a ^ b for a, b in zip(plaintext.encode('utf-8'), key))
    ciphertexts.append(ciphertext)
print("ciphertexts")
for ciphertext in ciphertexts:
    print(ciphertext.hex())

# save ciphertexts(hex) and key(hex) to a file
with open("many_time_pad.txt", "w") as f:
    f.write("Plaintexts:\n\n")
    for plaintext in plaintexts:
        f.write(plaintext + "\n")
    f.write("\nCiphertexts:\n\n")
    for ciphertext in ciphertexts:
        f.write(ciphertext.hex() + "\n")
    f.write("\nKey:\n\n")
    f.write(key.hex() + "\n")
```

```
plaintexts
plaintext 0
plaintext 1
plaintext 2
plaintext 3
plaintext 4
plaintext 5
plaintext 6
plaintext 7
plaintext 8
plaintext 9

key fcbd9c2a7b5ad36e16be47

ciphertexts
8cd1fd43152eb616629e77
8cd1fd43152eb616629e76
8cd1fd43152eb616629e75
8cd1fd43152eb616629e74
8cd1fd43152eb616629e73
8cd1fd43152eb616629e72
8cd1fd43152eb616629e71
8cd1fd43152eb616629e70
8cd1fd43152eb616629e7f
8cd1fd43152eb616629e7e
```

1.0.7 Notes

It's clear that with the Many-Time Pad scenario, key reuse provides significant patterns in the resulting ciphertext. In our example, regardless of the production of a random key that fits the criteria for being at least as long as the message, the ciphertexts will always result in the same output for the same input. This violates the principle of the ciphertext giving no information about the plaintext.

1.0.8 Problem 4 - (Cryptanalysis of Many-Time Pad)

Develop a strategy to decrypt messages encrypted with a many-time pad. More specifically, assume Eva has collected the 10 ciphertexts and she knew they are generated by the same key. In addition, all the plaintexts are in English. Space, comma, period, and question mark are being used in the plaintext, but no other special characters are allowed. Eva wants to decrypt the last message (target message).

1. 71fe1ace4389087266117cd7c98c4182851b3acff3b086e3f83f94d6eb05c4ba85d8e1fa14f11d1c3b568ff6cff5c09c5d6
2. 71fe1ace559a1e7266117cd7ce8745d7be2e74c3f0f68eeef57e8884e607debf81dfa0f012f95819681ae7f29fe4839b517
3. 72fe069c51c81a20775928c7879d4fd2a93c3acff3f69fe5fe2e9493a303d9ea98c4e5b60ae40a146058e7c787fbd09a14
4. 67e543885b9a5b2267177084cf8453ccb8633ad7fdb39de5b13f8a93a304d6bf8bc4f4ef5def110b6f56a3e186e2c68c1
5. 71fe029a148c1236320d7192878a59cfbc3a6ec5e7f68befb13196d6ea1ec4ea81d9e3fe50ea0f196d02a2f7cfe2c29c55
6. 6ef914ce5989152b321a769ad79c42c7be6f6ad2fab19de1fc339d84f04ad3a589dfa0ff09ab0c196f13e7e780b4c0975
7. 71fe1ace4389087266117cd7c4865bd2b93b7fd2b5a58ce9f4308c9ff01e97ab82cbf2ef5dfc101d6a56b3fb8ab4d08b4
8. 71fe029a148c1437615978d7c58854dbec2c75cde5a39be5e37e9b97ef0697a285dfa0f01cff101d764983f29bf5
9. 71fe1ace50875b31730d6ad7cb8640c7ec3c73d4e1bf81e7b13796d6e518d8a4988ceff05dff101d2415a8fe9fe1d79a4

10. 71fe029a1483123c76597691878459cca9363ac4faf68ceffc2e8d82e61897b98fc5e5f809e20b0c7756b2e08aab83bc55

Target Message: 71fe0680149d083b7c1e3996879a42d0a92e7780f6bf9fe8f42cd898e61cd2b8ccd9f3f35dff101d241da2ea

1.0.9 Solution

Ref: https://crypto.stanford.edu/~dabo/cs255/hw_and_proj/hw1.html

1. Design an attack strategy based on the vulnerabilities of key reuse in a many-time pad scenario. You can use some of the hints given below.

Our attack strategy will be based on three properties of XOR (in ASCII): 1. `c1   c2 = m1   m2` 2. `'a'   ' ' = 'A'` 3. `k = c   m`

To begin our attack, we will XOR each ciphertext combination to reveal some information about the underlying plaintexts. This takes advantage of the first property by removing the key.

Once we have XOR'd each ciphertext, we then check for the next condition, `'a'   ' ' = 'A'` (or ' '   ' ' = 0). If each of the ciphertexts 1-11 produce an output of '[A-Z]' or 0, we can infer that the plaintext is a space. This step takes advantage of our second property.

Lastly, we take the inferred plaintext, ' ', and XOR it with the given ciphertext. This will output our partial key guess for the given column over the message length. This step takes advantage of the third property, `c   m = k`.

2. Implement the strategy in jupyter notebook. Remember, most of the time cryptoanalysis needs a human in the loop and a bit of luck.

[ ]:
```
# Define the ciphers
ciphertexts = []
ciphertexts.append(bytes.
 ↪fromhex("71fe1ace4389087266117cd7c98c4182851b3acff3b086e3f83f94d6eb05c4ba85d8e1fa14f11d1c3b
ciphertexts.append(bytes.
 ↪fromhex("71fe1ace559a1e7266117cd7ce8745d7be2e74c3f0f68eeef57e8884e607debf81dfa0f012f9581968
ciphertexts.append(bytes.
 ↪fromhex("72fe069c51c81a20775928c7879d4fd2a93c3acff3f69fe5fe2e9493a303d9ea98c4e5b60ae40a1460
ciphertexts.append(bytes.
 ↪fromhex("67e543885b9a5b2267177084cf8453ccb8633ad7fdb39de5b13f8a93a304d6bf8bc4f4ef5def110b6f
ciphertexts.append(bytes.
 ↪fromhex("71fe029a148c1236320d7192878a59cfbc3a6ec5e7f68befb13196d6ea1ec4ea81d9e3fe50ea0f196d
ciphertexts.append(bytes.
 ↪fromhex("6ef914ce5989152b321a769ad79c42c7be6f6ad2fab19de1fc339d84f04ad3a589dfa0ff09ab0c196f
ciphertexts.append(bytes.
 ↪fromhex("71fe1ace4389087266117cd7c4865bd2b93b7fd2b5a58ce9f4308c9ff01e97ab82cbf2ef5dfc101d6a
ciphertexts.append(bytes.
 ↪fromhex("71fe029a148c1437615978d7c58854dbec2c75cde5a39be5e37e9b97ef0697a285dfa0f01cff101d76
ciphertexts.append(bytes.
 ↪fromhex("71fe1ace50875b31730d6ad7cb8640c7ec3c73d4e1bf81e7b13796d6e518d8a4988ceff05dff101d24
ciphertexts.append(bytes.
 ↪fromhex("71fe029a1483123c76597691878459cca9363ac4faf68ceffc2e8d82e61897b98fc5e5f809e20b0c77
# target
```

```python
ciphertexts.append(bytes.
 ↪fromhex("71fe0680149d083b7c1e3996879a42d0a92e7780f6bf9fe8f42cd898e61cd2b8ccd9f3f35dff101d24

# helper methods
def xor(a, b):
    return bytes(x ^ y for x, y in zip(a, b))
def is_valid_ascii(byte):
    return 31 < byte < 127
def is_alpha(byte): # (A-Z, a-z)
    # (48 <= byte <= 57) -- 0-9
    return (65 <= byte <= 90) or (97 <= byte <= 122)
def to_ascii(bytes):
    return ''.join(chr(b) if is_valid_ascii(b) else '_' for b in bytes)

# xor each one together
for i in range(0, len(ciphertexts)):
    for j in range(i+1, len(ciphertexts)):
        print(f"m{i} XOR m{j}:\t", to_ascii(xor(ciphertexts[i],␣
 ↪ciphertexts[j])))
```

```
m0 XOR m1:
_______________U;5N__F___A_R______A___E_SLh_P_C__________________S__EH___ZW#__
m0 XOR m2:
___R_A_R_HT_N__P,'___F_____EH__P___L____[_h1H___I___T____R_____E_I___YF___W___
m0 XOR m3:
__YF__SP___S___N=x______I__EH_______I___T_,_I___I____^L8_E_O___E_L___SF____A__
m0 XOR m4:       ___TW__DT__EN__M9!T__F__I______P____D___VT-
__________A__YT__U____HKH,TF___W_L_
m0 XOR m5:
_______YT__M___E;tP________R_O____A__Z__TEh_OA______T_L__G__U_____T&_N_RR____K
m0 XOR m6:       _______________P<
E_F______I__S_____I___Q_<_EA_______A___C___C_E_A__EC_______L
m0 XOR m7:       ___TW__E_H_____Yi7O______A_A__S___A_____M___T_
m0 XOR m8:
______SC_______Ei'I_____I________T__I____C'_P____D___A________WE&E___S_I____L_
m0 XOR m9:       ___TW__N_H_FN__N,-___F_____T__S_________L_=_E^C ____
m0 XOR m10:      ___NW__I__EAN__R,5MO______LN____I___I____K-______D____L__C_
m1 XOR m2:       ___R_R_R_HT_I_____N______P__E__U__EF__R__B_5__S_E___O____R_____
O____R_I__[_6____S_H_H__NF_
m1 XOR m3:       __YF__EP___S_____MN__E__DA__E_____T_O_I__LD___E_E____OE&_E_N___
O____Y_I__QTb__A_S'O__E_A__
m1 XOR m4:       ___TA__DT__EI___________DO_R___U__C_B_W___E_P_A______P__ST__O__
___SYi_I__[_#E______________F___A__EC___
m1 XOR m5:       _______YT__M_____A___G___M___M_______RT______PC_____O_E__G__O__
__LL7O__\HKH#_D_U_EH____A__EP______
m1 XOR m6:       ____________________ES___N____I___R_O_H__LT__PS______P___C___R_
O_________JT'_C_____T _YD__EN__W___EC__S___GB__
m1 XOR m7:       ___TA__E_H______R____U________I_______H__Sd___
```

```
m1 XOR m8:       ______EC________R____I__DI_R_____SO_O_H_L_O___T__V___P________R
O7___U__G_WE;E___T_________EL__W_________I__TD__NU
m1 XOR m9:       ___TA__N_H_FI_____N______P____I___E___S__LU__O_'____
m1 XOR m10:      ___NA__I__EAI______C_I___RP_____M_S_O_H_L_E_P_O__V____E__C_
m2 XOR m3:       __E__RA__NXCH________E__O______U___YW_____D&________TJNO___T___
___________TT__S__5__TD__J_
m2 XOR m4:       ____ED__ETYU______T_____O__EI______HZ____ZEOH___A___NU__E___A__
A__QA;_______H___A_H_I__NT_
m2 XOR m5:       ___R_A__EC^]P____SP__G______SI_O__EI_O___K_
_O__A_____N_____A__L_VN/__EZN_H__H_EAW__R___U_
m2 XOR m6:       ___R_A_R_HT_C_____E_FS______S_NA___YW_____T<_O__U___TU_______N_
_____R_O___T__O__K_WOh_O_N_
m2 XOR m7:       ____ED____P_B___E_O__U___P__L_NH__EF______d5__
m2 XOR m8:
___R_OA__TB_L___E_I__I__O__EF__N_H_FW___DMO9____RW__LU___R____[_
_____EA__E_H_____H_A__NS_
m2 XOR m9:       ____EK____^V________________E_NS___N______U'_PS&A___
m2 XOR m10:      ____EU___G_Q______MO_I____L_E__RT__EW___DEE-H___EW__A_N____
m3 XOR m4:       __A_O_I_U___H____YT__E_____EI__U_________T__I___A_____A
H__EA__A__HAO_____TAT_R_A6O__M_A__
m3 XOR m5:       __WF__N_U_________P_____M___SN____T_TD___ED__V__A___T^_8___TA__
L_SW/__EHE__A_HSAAb___D____
m3 XOR m6:       __YF__SP___S_____XE_H___E___S_A______________V__U_____C&___E_R_
_____Y_O____E_OJ_K'PO<EC___
m3 XOR m7:       __A_O_O__N_S____TOO_____RA__L_A___T_A_____ ___
m3 XOR m8:       __YF_______S____T_I________EF____H______KC______RS____D5_E_O__Z
_#_____ES___YT_O__6O__N_A__
m3 XOR m9:       __A_O_I__N__H____U___E__M___E_A_____T________IEOA___
m3 XOR m10:      __E_O_S___I_H____MMW____E_R_E___G_______KK__I___ES___Q_;___
m4 XOR m5:       ___TM_______P____U___G__M__R_T_O__C_YA____E_OV______NAA_I______
__W_n&_ESA_H_T@_E_TH____A__R____L__
m4 XOR m6:       ___TW__DT__EC_______RS__E__I__SA_________T__EV__________A___DB_
A__REi_O___T_RG_____T!_ED__R___GL______T
m4 XOR m7:       ________ST_EB___P____U__RO_A__SH__C_L____K!_T_
m4 XOR m8:       ___TD_I_A__EL___P____I_________N_U______I___P____T______YT__N_Z
A!_\A<_EH__E_____F_________R___GT__EM___
m4 XOR m9:       ________DT________T_____M__T__SS____Y____T__EIA ____
m4 XOR m10:      ________N_H________E_I__E_NN___RM_______I________T___NA_N__
m5 XOR m6:       _______YT__M_____T__O________TD___R_TW___ET_________TAC_____DB_
L_^M+O__VP______O_EWA:_W_______B___
m5 XOR m7:       ___TM___SC_M____RC_______M___LD______T___Zd__A
m5 XOR m8:
______N_A__M____RS______M__R_R___SO_TT__K_O__U___N__LAD__G__N_JL
ZC/____T___TC_NFTH____A______B___
m5 XOR m9:       ___TM___DC__P____YP__G_______RD___E__I___EU___C+____
m5 XOR m10:      ___NM___N_O_P____A_R______E__V__E_S_TT__K_E_OM___N__A______
m6 XOR m7:       ___TW__E_H______U___P____N________R_A_____O__A
m6 XOR m8:       ______SC________U___T___E__I__O__G______NC___U___D_______C___G^
```

```
_.___U__M____RD__L__W)_TD__________EM__S_N_U_I_
m6 XOR m9:       ___TW__N_H_FC_____E_OS______________T_________S7____
m6 XOR m10:      ___NW__I__EAC______RC_____T___E_N_______NK__EM___D___NC____
m7 XOR m8:       ___TD_O__T______________RI_A__O__SO_A___R\+___
m7 XOR m9:       ___________FB___E_O__U___P________E_______1__^
m7 XOR m10:      _________GAAB___E__M_____RC___E_I_S_A___RT!_T_
m8 XOR m9:       ___TD_I__T_FL___E_I__I__M__T__O__I__T___SC___JT&_C__
m8 XOR m10:      ___ND_S___SAL___E__T____E_NN____TU__________P________ND__C_
m9 XOR m10:      _________GO_______MD_I____U___E_C___T___SK__ERO1_C__
```

We can see from the above XOR's that we have various candidates for (' ' XOR '[a-z]') which result in 'A-Z' as well as ('something' XOR 'something') which results in 0. To begin our attack, we will first be guessing based on spaces. For each c1 XOR c2, we can identify a possible space in m1 or m2 by an alpha character or a 0.

[151]:
```python
key = bytearray(max(len(ct) for ct in ciphertexts))

# We can find a space in a plaintext by XORing it's ciphertext with all other␣
↪ciphertexts.
for column in range(len(key)):
    for i in range(len(ciphertexts)):
        if column >= len(ciphertexts[i]):
            continue

        byte = ciphertexts[i][column]
        is_space = True

        # XOR with all other ciphertexts, and check for alpha or 0.
        for j in range(len(ciphertexts)):
            if i == j or column >= len(ciphertexts[j]):
                continue
            xor_result = byte ^ ciphertexts[j][column]
            if not (xor_result == 0 or is_alpha(xor_result)):
                is_space = False
                break

        # All other ciphertexts produced alpha or 0, we assume this text is a␣
↪space.
        # By XORing the ciphertext with a space, we can get the key for this␣
↪column.
        if is_space:
            key[column] = byte ^ ord(' ')
            break

print("Key guess: ", key.hex())
for i in range(0, len(ciphertexts)):
    print(i, to_ascii(xor(ciphertexts[i], key)))
```

```
Key guess:  000063ee34e87b5212790000a70000a2cc001aa095d60000915ef8f6836ab7caecac
8096008b78000000c700ef00a30034030000e800db00caf4003e21c800dd00000081bdb8ec003800
a827c400fa600028009261af54d2aa00442c00006d940000391a00007200a200a2e4320000a8d37a
cb99256bccfa7cb258
0 q_y was th|_n_A I_ off__ial hospital_ze_;VH_ _c_id_\tql_y Zouc_e>__he f_r_wal@
1 q_y are th|_i_Eur.nce __d premiums f_r _h_ _p_ _ev_^o`e_s Knor_o/T_y hi_h_
Be0au_e_they f_e __wa8_ c^_syiwg d_M
2 r_ere are (_ _0pe< of __ople in the _or_`X _h_s_ w_] en_er]tan_ 8N_ary _n_
thCse_w_o don _
3 g_ for punp_h_Sntc whe__ are naughty]di_oVd_i_e_ s_\t/ _heW ar_ ;K_ays _e_t tC
a_B_ot caj_
4 q_at did tq_ _Ymp:ter __ on its muchPaw_m_e_ _a_at_]n0a_ tFe b_a9OW It _a_ a
Kre_t_time t_rf__g 5_e B_t
5 n_w many cv_p_Beroprog__mmers does i_ t_o_ _o_c_an_W q _igFt b_l8_HNone_
_hat_s _ _ardwau_ p__bl$_
6 q_y was th|_c_[pu;er s__entist angry]wh_jVt_e_s_ud_\t0c_acEed _ 6F_e
co_p_ter_jo_e_ He dn_ n__ l(_e E_ ~n| bi_
7 q_at does x_b_Ty ,ompu__r call his f_th_vID_t_
8 q_y do catj_l_@e <itti__ in front of]th_$_o_p_t_r _^l0d_y Bong@ _B_ause_t_ey
Hon_t_want s_ l__ t)_ mC_st vut _\nu
9 q_at kind v_ _Yne6 do __mputer scien_is_wVu_e_ _ac_W
10 q_en using9_ _Bre.m ci__er never use]th_$_e_ _o_e _Za~ _ncK
```

Now we have recovered enough plaintext to begin making educated guesses on the key.

To do this, we XOR the plaintext with the ciphertext to generate a key guess. Therefore, we can guess what the plaintext is by inference, XOR it with the ciphertext, then use that key to decrypt all ciphertexts again.

```
[152]: plaintext_guess = b'Why are the insurance and premiums for _h_ _p_ _ev_^o`e_s␣
        ↪Knor_o/T_y hi_h_ Because they f_e __wa8_ c^_syiwg d_M'
       key_guess = xor(ciphertexts[1], plaintext_guess)

       print("Key guess: ", key_guess.hex())
       for i in range(0, len(ciphertexts)):
           print(i, to_ascii(xor(ciphertexts[i], key_guess)))
```

```
Key guess:  269663ee34e87b52127919f7a7e936a2cc4f1aa095d6ef80915ef8f6836ab7caecac
80967d8b78460045c7adefbba3c43403b000e800dbafcaf4003e21c84ddd00005b81bdb8ec8a38fb
a827c42cfa60da288b9261af54d2aa90442cb0846d9400c8391a00407200a200a2e432ca00
0 Why was the new IT official hospitalizeZ;_H[ NcXid_\tqlAy ZoucZe>_Ghe fQr_wall
1 Why are the insurance and premiums for _h_ _p_ _ev_^o`e_s Knor_o/T_y hi_h_
Because they f_e __wa8_ c^_syiwg d_M
2 There are 10 types of people in the worR`_ jh@s^ wR] enIer]tanV 8N]ary Yn_
those who don Y
3 As for punishment, where are naughty diMo_dLiYeH s_\t/ yheW arW ;KDays Ke_t to
a Boot caj]
4 What did the computer do on its much-aw_mGeZ YaXatS]n0aY tFe bWa9O_ It Pa_ a
great time tXrfW]g 5De B[t
```

```
5 How many computer programmers does it t_oV Jo_cSan]W q AigFt bGl8__None_
_hat's a hardwauH pL\bl$A
6 Why was the computer scientist angry wh[j_tVe_sOud_\tOc_acEed S 6F^e coUp_ter
joke? He dnI nQG l(Ge EJ ~n| biD
7 What does a baby computer call his fath[v_D_tN
8 Why do cats love sitting in front of th[$PoSpZt^r [^l0dLy Bong_ _BPause_t_ey
don't want sB l[G t)I mCKst vut _\
9 What kind of money do computer scientisJw_uMe_ xacRW
10 When using a stream cipher never use th[$XeG BoIe NZa~ BncK
```

[153]:
```python
plaintext_guess = b'There are 10 types of people in the world_ those who␣
    ↪enIer]tanV 8N]ary and those who don Y'

key_guess = xor(ciphertexts[2], plaintext_guess)

print("Key guess: ", key_guess.hex())
for i in range(0, len(ciphertexts)):
    print(i, to_ascii(xor(ciphertexts[i], key_guess)))
```

```
Key guess:  269663ee34e87b52127919f7a7e936a2cc4f1aa095d6ef80915ef8f6836ab7caecac
80967d8b78780407c7b3ef94a3ff34038a32e800dbafcaf4003e21c84ddd00005b81bdb8ecb2389b
a827c42cfa60da288b9261af54d2aa90
0 Why was the new IT official hospitalized?QHE accidentqlAy ZoucZe>_Ghe firewall
1 Why are the insurance and premiums for al_ App develo`e_s Knor_o/T_y high?
Because they f_
2 There are 10 types of people in the world_ those who enIer]tanV 8N]ary and
those who don Y
3 As for punishment, where are naughty diskQdRives sent/ yheW arW ;KDays sent to
a Boot caj]
4 What did the computer do on its much-awai_eD vacation0aY tFe bWa9O_ It had a
great time tX
5 How many computer programmers does it tak_ To change q AigFt bGl8__None;
that's a hardwauH
6 Why was the computer scientist angry whenQtHe student0c_acEed S 6F^e computer
joke? He dnI
7 What does a baby computer call his fatherNDAta
8 Why do cats love sitting in front of the _oMputer all0dLy Bong_ _BPause they
don't want sB
9 What kind of money do computer scientistsQuSe? Cache
10 When using a stream cipher never use the _eY more tha~ BncK
```

[155]:
```python
plaintext_guess = b'Why are the insurance and premiums for all app developers␣
    ↪Knor_o/T_y high? Because they f_'

key_guess = xor(ciphertexts[1], plaintext_guess)

print("Key guess: ", key_guess.hex())
for i in range(0, len(ciphertexts)):
    print(i, to_ascii(xor(ciphertexts[i], key_guess)))
```

```
Key guess:  269663ee34e87b52127919f7a7e936a2cc4f1aa095d6ef80915ef8f6836ab7caecac
80967d8b78780476c793ef94a3ff34038a32e810db82caf4003e21c84ddd00005b81bdb8ecb2389b
a827c42cfa60da288b9261af54d2aa90
0 Why was the new IT official hospitalized? He accidentally ZoucZe>_Ghe firewall
1 Why are the insurance and premiums for all app developers Knor_o/T_y high?
Because they f_
2 There are 10 types of people in the world. Those who under]tanV 8N]ary and
those who don Y
3 As for punishment, where are naughty disk drives sent? TheW arW ;KDays sent to
a Boot caj]
4 What did the computer do on its much-awaited vacation at tFe bWa9O_ It had a
great time tX
5 How many computer programmers does it take to change a ligFt bGl8__None;
that's a hardwauH
6 Why was the computer scientist angry when the student cracEed S 6F^e computer
joke? He dnI
7 What does a baby computer call his father?Data
8 Why do cats love sitting in front of the computer all day Bong_ _BPause they
don't want sB
9 What kind of money do computer scientists use? Cache
10 When using a stream cipher never use the key more than oncK
```

[159]:
```
plaintext_guess = b'How many computer programmers does it take to change a␣
 ↪light bulb? None; that\'s a hardware'

key_guess = xor(ciphertexts[5], plaintext_guess)

print("Key guess: ", key_guess.hex())
for i in range(0, len(ciphertexts)):
    print(i, to_ascii(xor(ciphertexts[i], key_guess)))
```

```
Key guess:  269663ee34e87b52127919f7a7e936a2cc4f1aa095d6ef80915ef8f6836ab7caecac
80967d8b78780476c793ef94a3ff34038a32e810db82caf42e3e21c87fdd5a276881bdb8ecb2389b
a827c42cfa60da288b9261af54d2adbd
0 Why was the new IT official hospitalized? He accidentally touched the firewall
1 Why are the insurance and premiums for all app developers enormously high?
Because they ar
2 There are 10 types of people in the world. Those who understand binary and
those who don't
3 As for punishment, where are naughty disk drives sent? They are always sent to
a Boot camp
4 What did the computer do on its much-awaited vacation at the beach? It had a
great time su
5 How many computer programmers does it take to change a light bulb? None;
that's a hardware
6 Why was the computer scientist angry when the student cracked a lame computer
```

```
joke? He did
7 What does a baby computer call his father?Data
8 Why do cats love sitting in front of the computer all day long? Because they
don't want to
9 What kind of money do computer scientists use? Cache
10 When using a stream cipher never use the key more than once
```

1.1 The decrypted message:

When using a stream cipher never use the key more than once

3. Analyze and discuss the outcomes, including any partial decryption results and insights gained from the process.

Two insights that were discovered from this process are key reuse is only vulnerable if the encrypted plaintext messages are as wide as the key, and that human inference is necessary to uncover patterns to assist with an attack. Vulnerabilities in the plaintext encryption cascade to the entire message. Each bit of the key, ciphertext, and plaintext are important in understanding and decrypting many-time pads. When implementing our attack strategy, we only needed to rely on spaces within the plaintext, and this small junction allowed us to gradually build our knowledge of the key to obtain a complete break.