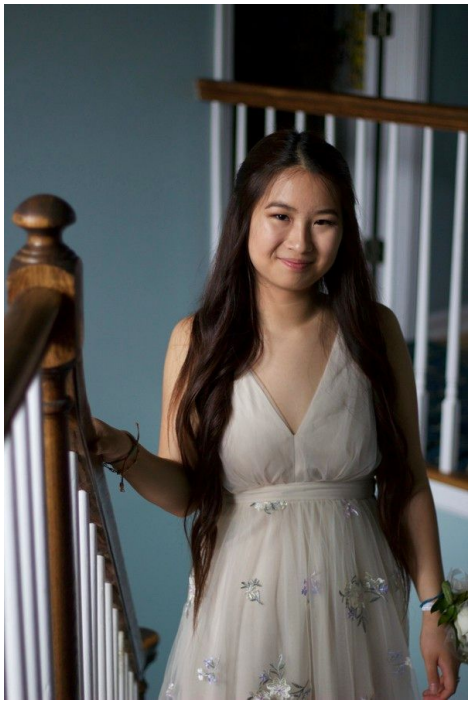
Constance Xu
10/5/2019
Homework 1
CS306

I pledge my honor that I have abided by the Stevens Honor System.

Problem 1



Problem 2

1. The two basic security properties that need to be observed are the following: confidentiality and integrity. Confidentiality is important because if they are using an insecure channel, then say an attacker Eve could potentially eavesdrop and find out their key. If they take confidentiality into consideration, they are much less likely to let an attacker like Eve find out. Furthermore, integrity is important and relevant because this key will be used for all of their communications. If this is not secure, then none of their messages are secure!
2. Confidentiality is not satisfied in this case. As you can see, Alice generated a protocol that would have some amount of randomness to it, but only at the start. This is sent to Bob after it is XOR'ed. Then, anyone who is listening or eavesdropping on the channel can do what Bob is about to do and XOR that message and have their key. The integrity of this is also compromised because if their confidentiality was compromised, then the key is not secure, meaning all of their messages are not secure.

Problem 3

1. When t=1: This is a very insecure algorithm because it is can be easily compromised through brute force. It has a very similar approach to any simple cipher where the first and second

characters could be used and the difference between them would give the attacker the information they need.

2. When *t*=2: Since there are only two passwords where ab and cd are two letters apart at most or be and dg are two letters apart at most as well, the attacker can just pick one password or the other. Since the key repeats every two letters and it is equally likely for both passwords to be used, then it is easy to figure out which one is used for a given shift in the cipher.
3. When *t*=3: There are two patterns in this case: x,y,z,x+3 or x,y,z,x+5. This is due to the only two password options. Eve would be able to figure out which pattern is being used easily.
4. When *t*=4: Both passwords are length 4. This is not new information to Eve. Hence, no new information has been shared.

2. A mono-alphabetic substitution cipher is trivial to break because a chosen-plaintext attack of 25 letters can be used to figure out the key. The attacker could easily crack the cipher and be able to decrypt all future messages or communication. This is true because there are only 26 distinct letters that will correspond to some other letter. 25 is all that is needed because the 26 letter would be the only one left through a process of elimination. The shortest chosen single-message plaintext that is a valid, English message and would successfully recover the key is “Jived fox nymph grabs quick waltz”. This mono-alphabetic substitution cipher would be perfectly secure against a ciphertext-only attacker if the probability that the mapping of the letters and the ciphertext are equally as likely to occur.

Problem 4

The texts are as follows:

“”””

Testing testing can you read this
Yep I can read you perfectly fine
Awesome one time pad is working
Yay we can make fun of Nikos now
I hope no student can read this
That would be quite embarrassing
Luckily OTP is perfectly secure
Didnt Nikos say there was a catch
Maybe but I didnt pay attention
We should really listen to Nikos
Nah we are doing fine without him

“”””

1. In order to do this, I used crib dragging. First, I computed the XOR of the first two ciphertexts. Then, I dragged lambda x: x^codecs.encode(guess_word, “hex”) function over the windows of length (len(guess_word)). This would go across the XOR of the two ciphertexts and that is to see if any probable ciphertext could be found. This gave me the single plaintext, ciphertext pair, which I then XOR’ed to get the key. Mapping lambda x:x^key over the rest gave me the rest of the text. From ASCII values, I converted to english plaintext. The key in text format is youfoundthekey!congratulations!!!

The code I used is below:

```
import codecs
import hashlib
key = b'Youfoundthekey!congratulations!!!'
def SHA_ENCODE(current_key,days_passed):
    new = current_key
    for i in range(days_passed):
        new = bytes.fromhex(hashlib.sha256(new).hexdigest()) + b'\x21'
    return codecs.encode(new, "hex")
print(SHA_ENCODE(key, 14))
```

2. The new key used in two weeks time is:
   8C2D1EAC1BF0B0910888B354D516600F1C9DEDD5804DEF51C7931360DEA6DEE000100001