Evan Smith

CIS 628

Section M400

HW 4

Problems from Chapter 2

**Problem 2.1**

1. Given a stream of letters (0-25), the key stream would also be made of letters. The encryption function would be: yi = esi (xi) ≡ xi +si mod 26. Decryption would be: xi = dsi (yi) ≡ yi -si mod 26
2. We convert the cipher text and key into their numerical equivalents:



Then, we can do the decryption algorithm on the cypher text values using the key values:

1-17 mod 26 ≡ 10 -> K

18-18 mod 26 ≡ 0 -> A

0-8 mod 26 ≡ 18 -> S

18-3 mod 26 ≡ 15 -> P

15-15 mod 26 ≡ 0 -> A

15-24 mod 26 ≡ 17 -> R

10-3 mod 26 ≡ 7 -> H

10-10 mod 26 ≡ 0 -> A

20-0 mod 26 ≡ 20 -> U

14-22 mod 26 ≡ 18 -> S

18-14 mod 26 ≡ 4 -> E

15-0 mod 26 ≡ 15 -> P (I think this may be an error in the key or cipher text)

1. The boy died from a stabbing.

**Problem 2.2**

If we had such a 1GB key on the CD, the first issue would be communicating that key to our recipient. That can be done with a physical transportation of a copy of the CD to maintain that the random key only exists on the disk with no possible outside readers. The generation of the key would require a true random number generator, which are non-trivial to acquire as well. The lifecycle of the key is only for a set of messages that are as long as it, so in this case 1GB of messages. During that lifecycle, the key must be held secret by both parties, and should be destroyed sufficiently after use.

**Problem 2.3**

If a 128-bit OTP-esque encryption was used, the attack vector that would crack it would rely on the attacker knowing at least 3 key lengths (384 bits) of the plaintext. As described in the book, this is common and possible in transmissions, since the header is formulaic and often easily guessable or knowable. Using the plaintext and the encryption, the key can then be reverse engineered using systems of equations and verifying attempted encryptions using the derived key.

**Problem 2.4**

We know that given a letter, we can produce a key letter that will encode to any arbitrary letter, and we know that there is not a single substitution occurring that would allow for frequency analysis. Given these two facts, the problem with brute-forcing an OTP is that it is equivalent to brute-forcing the plaintext. This is impossible, since you cannot verify that you are correct. The only way to know for sure would be to have known the message contents beforehand. While you CAN brute force the key, you would brute-force many other keys that produce viable messages, and there is no way to tell which is the real one.

**Problem 2.5**

|  |  |  |  |
| --- | --- | --- | --- |
| feedback | initial | work | output (mod 2) |
| 1 | 1 | NA | NA |
| 0 | 0 | NA | NA |
| 1 | 0 | 0 + 0 + 1 = 1 | 1 |
|  | 1 | 0 + 0 + 0 = 0 | 0 |
|  | 0 | 1 + 1+ 1 = 3 | 1 |
|  | 1 | 0 + 0 + 0 = 0 | 0 |
|  | 0 | 1 + 1+ 1 = 3 | 1 |
|  | 1 | … | … |

|  |  |  |  |
| --- | --- | --- | --- |
| feedback | initial | work | output (mod 2) |
| 1 | 0 | NA | NA |
| 0 | 1 | NA | NA |
| 1 | 1 | 1 +1 + 0 = 2 | 0 |
|  | 0 | 0 + 1 + 0 = 1 | 1 |
|  | 1 | 0 + 0 + 0 = 0 | 0 |
|  | 0 | 1 + 1 + 1 = 3 | 1 |
|  | 1 | 0 + 0 + 0 = 0 | 0 |
|  |  | … | … |

1. These sequences (1,0,0,1,0,1,0,1…) and (0,1,1,0,1,0,1…) are shifted versions that result in the same looping sequence.

**Problem 2.6**

A key thing that Oscar can know which will assist in breaking the stream cypher is some idea of the structure or the exact text of the plaintext and its corresponding. This could be possible if the communication is using headers or other structured information that is easy to infer or guess. Optimally, Oscar knows 200 bits worth of the plaintext, enough that we are sure that it would be encrypted by a single pass or less of the stream cypher. If this is the case, then the next step is to seed an LFSR with the known cyphertext and solve the set of linear equations for the key bits that produce the known corresponding plaintext. Once this is done, the key can be used to set the LFSR and decrypt the rest of the message.