UNIVERSITY OF ALBERTA

Assignment 1

CMPUT 333

Omar Zioueche, Raymond Lieu, Cole Evans (Group 13)

10/1/2015

Group 13 assignment 1 for CMPUT 333 September 2015 Fall term.

Assignment 1
CMPUT 333 - Group 13

Part 1 and 2:

The submitted files are:
Part1&2.py = automated password and decryption of both ciphertexts
Plaintext1.txt = ciphertext1 decryption
Plaintext2.rtf = ciphertext2 decryption
Pwds.txt = passwords for ciphertext decryption
README.txt = how to use Part1&2.py

The first step in the decryption process was determining a possible key that would eventually map to a printable plaintext file. The steps were as follows:

1. Find the maximum length of repeating bytes in the cipher-text. Each iteration of the code will add 1 additional byte to the length until no repetitions are found, remembering to find the minimum length of space between the repetitions as well.

2. The maximum length of this string of bytes (repetitions plus space between them) would give us an upper bound for the key length. We could know at this point that the key would have to fit in the length. (cycle and position)

3. We then check if every repetition is the same for all cycles. If any differences are found, these cycles are no longer viable key length options because a character would map to something different.

4. Having narrowed down the possible lengths, we then check to see if every set contains only printable characters. If a non-printable character is found, this set is also removed. This is done by checking every possibility for every position in the key length being checked. We use this value to decrypt the corresponding cipher-text bytes, and see if they map to a printable character. If not, the tested character is removed as a possibility for that position. This process continues until there are no allowed characters in specific position of a key map to a printable plain-text, at which point, the possibility set is removed as a viable key, or we reach the end of the cipher and obtain a **possible** key.

5. We then use the possible values for each possible key length to try and construct a viable plaintext file. Due to pre-image resistance, it is likely only one of these keys will map to a realistic and legible text.

If the file was compressed before encryption, we would have to decrypt the ciphertext before uncompressing, and then checking the plaintext for coherent content.

The key was determined to be: '**4fathers'**

This process was initially started manually. We started finding repetitions by using python to find repeating sequences in the code. We quickly realized however that the process would

quickly become tedious if the process was not automated, as it would essentially be the same steps over and over until a viable key was found. Hence, the code “Part1&2” was born. It made the password discovery and decryption process a lot easier, and allowed us to quickly adjust some parameters and solve for part 2.

NOTE: For part 2, the ability to add in the known part of the header file was used to minimize computational time. The header used for ciphertext2 was “{\rtf1\ansi”. By adding this feature, it also eliminates many of the possible key possibilities as it narrows down the search to a very small pool of candidates. We recommend changing the maximum length of possible repetition block sizes. Change the variable ‘length’ on line 173 to 200 for Ciphertext2 so the algorithm skips unnecessary steps. We also had to adjust the code to include certain characters that were not included in part 1.

The key was determined to be: ‘**Pass_TheBoycottActionKit_GentBelgium’**

Decryption Process:

Read In the bytes and map them to the cipher provided. Then take the top and bottom 4 bits and put it through the decryption algorithm provided. This was not especially complicated once the key was found, since the code decrypts the file automatically once the key is given as a parameter. SEE README FOR INSTRUCTIONS.

Part 3:

Submitted files:
XYZ = encryption algorithm used: CBC, OFB, CFB, ECB
cipherXYZ.enc = encrypted 80 character files.
cipherXYZerror.enc = modified 19th byte files
plainXYZerror = decrypted cipher XYZerror files.
Switched = modified student id file.

All 4 encryption types were used using the same password: **cmput333**
The files are named plainXYZerror whre XYZ = the encryption algorithm OFB, CBC, CFB, ECB

Based on the error of ECB:

Since the file is encrypted block by block, any modification to the ciphertext to a given block results in an error in the decrypted file when it comes to that specific block. Since our key was 8 bytes long, the 19th byte of the ciphertext falls in the third block. This means that the contents of the third block (17th-24th bytes) will be displayed with error once the decryption algorithm is applied to the modified ciphertext.

Error of CBC:

Because of the nature of the decryption algorithm, we decrypt the ciphertext and then use XOR it with the previous ciphertext(or initialization vector for the first step).Because we only use the previous step’s ciphertext to XOR, the error is not propagated beyond the step immediately after. Since our key was only 8 bytes long, the modified byte(19th) falls in block 3 of the ciphertext. This means that during decryption, only blocks 3 and 4 would contain an

error, while all previous and subsequent blocks should contain the original information because their cipher was not modified. Because XOR modifies individual bytes, since the error was found in the 3rd byte of the third block, it will also cause an error in the third byte of the fourth block. Note that the third block(original byte modification site) has many errors caused by decryption because of this change, but the fourth block only has the error in the third byte. This is because the cypher is decrypted before being XOR'ed, meaning the fourth block is still intact up until the point where it is XOR'ed with the previous cypher, at which point only the incorrect byte is changed.

Error of CFB:

Because the current block of cipher depends on the previous block of cipher, when the previous block of cipher is passed through the block cipher encryption, it modifies the contents of the block so that it can be XOR'ed with the current cipher to obtain the plaintext. Since the third block's 3rd byte (19th overall) was modified, when it is XOR'ed with the previous encrypted cipher which did not contain any errors, we only get 1 wrong byte since the other bits of the block would not be changed outside of the modified byte, and XOR only works bit by bit. When this error-containing cipher is then used for the next block's decryption, it is passed through the encryption block, at which point, because it is modified, it comes out as incorrect bytes for decryption purposes, which in turn provide an incorrect plaintext. This explains why the third block of our plaintext only contains one wrong bite, while the fourth bite is entirely gibberish.

Error OFB:

Because of the nature of the algorithm, the decryption depends exclusively on the repeated encryption of the initialization vector (IV). The ciphertext block is only used to XOR with the output of the IV encryption, and is not used again. Therefore, only the byte corresponding to the modified byte in the ciphertext will incorrect, while the remaining bytes will be correct.

We determined that there are 7 students in the file. This was determined by looking at the number of bytes in the file (232) and then dividing it by 32 (8 for id and 24 for name), which is the size of a single student record. This gives us a maximum number of 7 students (224 bytes) without going over the file size. Allowing for 1 byte for the scholarship record, the other 7 bytes are padding. There are also 7 repetitions of a set of HEX values "BF A1 40 E9 17 0D 30 FD" which we logically assumed to be white spaces between name and the next entry to fill all of the required 24 bytes.

The student with the scholarship was student 4. To determine this, we created 8 files full of blank spaces with only the number in the expected position. After having it encrypted, we compared the hex value: 'C2 9C 38 2C 6D 6B E3 4E' which when encrypted the block to the appropriate cipher based on cyphertext3. The reason we left the file blank was because we found a repeating pattern in the 24bytes used for the names of the students, implying they were white spaces, as no student would have identical names. This allowed us to assume there could only be white spaces right before the desired byte, and padding afterwards. Based on all these factors, we were able to recreate a file with similar conditions, since the names and student ID's are not included in the encryption block we want and not relevant to our purposes.

BIBLIOGRAPHY:

https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation