Robin Farrow-Yonge
Student Number: 160719011

Distributed Systems and Security - Security Coursework

Process:

The process for deciphering messages enciphered with a many time pad is based around the fact that performing the XOR operation (the operation used to do the initial encryption) on two ciphertexts gives the XOR of the corresponding plaintexts. If I then XOR *those* with hex-encoded character strings that I think are likely to be present in one of the two plaintexts, in the position that I expect them to be in, the resulting output will - if I got the correct character string in the correct position - reveal a corresponding portion of the *other* plaintext. This technique is called crib-dragging, and using it you can potentially guess all of the contents of the file. With the amount of ciphertexts available to me, I figured once I deciphered part of one of them I would be able to use that to find part of the other file it was XOR'd with. I can then, ideally, go back and forth like this, making educated guesses and uncovering steadily more and more of each plaintext, until I have enough to find out the key. Once I have enough of the plaintext of one of the ciphertexts, I can then XOR the plaintext with the ciphertext to get the key to all the ciphertexts. Because I knew I was primarily looking for the message that was contained in Ciphertext 11, it seemed logical to start by performing the XOR operation between pairs that included 11. This way, the more of each of the non-11 ciphertexts I was able to decipher, the more of 11 I would be able to decipher which would, in turn allow me to decipher more of the other files that were XOR'd with 11. To start with, and to test my script, I performed the operation on five pairs, pairing Ciphertexts 1 to 5 with 11. I named the resulting paired files according to the convention c[number]k[number].txt, where each number represents one of the ciphertexts.

The program I wrote to help perform the XOR operations was made to be simple as possible. I was unsure as to how well the approach I intended to take would work, and was loathe to write something too specialised should I have to change tack. As it is, my current script simply takes two filenames as arguments and will XOR them together. It then writes the results in hex to an output file, while also printing them as characters to the terminal so I could more easily read it. This did require me to quickly split the ciphertexts into their own files, but this seemed a small price to pay for convenience - I have included a subfolder containing these files for testing if required. The core of the program runs as follows:

1. Create two char arrays by calling hexFileToChars, which parses the hex files passed as the two arguments at run into chars.
2. Create a third char array by calling the xor function, passing the first two arrays as arguments. This function XORs each index of the first array with the corresponding index of the second array.

3. Use writeToFile to write write out/print out the output. Using two StringBuilder objects, one that creates a string of chars and one that creates a string of hexadecimal integers. The hex string is written to a file, and the char string is printed to the terminal for review.

I have included the java file along with notes to explain the way the program runs in this hand-in folder, along with the subfolder of hex files

As my intention was to begin this process with a light bit of crib-dragging, I created a test file to XOR against my paired files. I filled this file with spaces (to account for an initial deficiency in my XOR script), then replaced the first three characters with "the" and started to XOR it against my paired files to see if any of the plaintexts contained those characters at the beginning. The second combination I tried (the test file XOR c11k2.txt) resulted in:

```
bloc;r,=?+*>sw52k+a,e*u,$17ro^Me0+1!o2n"r*#c^U+51kt%<9h.?o&ys=&"l#e'!*=o4=d
t83!!i&"a0-! 79 ;*t!
```

Though not the most immediately readable message, I guessed that "bloc;r" was probably supposed to be "block " so I replaced the "the" in my test file with "block" and re-ran the XOR operation on c11k2. This gave me "the pr" - not the clearest clue. I decided to run the same XOR between "block" and different pair files. In doing this I found that the XOR of c11k4.txt and my test file resulted in:

```
encryp7  ?&e;       ).ep3o-0>1e=fo#n!&0:n&n%  (+0!871  =&e>u";      $(7g-6
!!i<i%+di$2d'& 3e:<6
```

I guessed that "encryp7" was likely to be "encryption" and so again, replaced my test file with that character string and ran the operation again which this time returned a character string starting with "block ciphe;...", I assumed this was supposed to be "block ciphers" so I set this as my test file and continued. By this point I had surmised that file starting with "block ciphers" was the contents of ciphertext11, because it was working with, and resulting from, every XOR'd pair. I went back and forth like this for a while, XOR-ing as much of each plaintext that I could guess with the corresponding pair file (or all of the pair files in the case of 11). I have omitted the results from every single XOR I performed for the sake of brevity, but eventually I had the following:

```
block ciphers take a number of bits and encrypt them as a  i,28ej
==ta2n)u3)d0&n4t5&! %<a=!t*-& 5=i1&%
```

I took the legible first half of this, what I believed to be part of the plaintext of ciphertext11, put it into my test file, then ran the XOR script on ciphertext11 and the updated test file to try and get the key. This returned:

```
it takes a great deal of bravery to stand up to our enemie i,28ej
==ta2n)u3)d0&n4t5&! %<a=!t*-& 5=i1&%'
```

Vaguely recognising this quote, I did a Google search for the legible portion, and found the complete quote - “It takes a great deal of bravery to stand up to our enemies, but just as much to stand up to our friends”. Testing this key against ciphertext11 revealed the entire message to be “Block ciphers take a number of bits and encrypt them as a single unit and padding the plaintext so that”, and running this new key against the rest of the ciphertexts also revealed their complete plaintexts - all of which lined up with the partial results from the search process.

Results:

Key:
“It takes a great deal of bravery to stand up to our enemies, but just as much to stand up to our friends”

Ciphertext 11:
“Block ciphers take a number of bits and encrypt them as a single unit and padding the plaintext so that”

Ciphertext 10:
“RSA is made of the initial letters of the surnames of Ron Rivest Adi Shamir and Leonard Adleman”

Ciphertext 9:
“The Caesar cipher also known as a shift cipher is one of the simplest forms of encryption”

Ciphertext 8:
“Even the smallest person can change the course of the future”

Ciphertext 7:
“Strong digital signatures are an essential requirement for secure systems”

Ciphertext 6:
“The Vigenere cipher is a method of encrypting alphabetic text by using a series of different Caesar cip”

Ciphertext 5:
“There are two types of encryption algorithm symmetric encryption and asymmetric encryption”

Ciphertext 4:
“Encryption is the process of encoding a message in such a way as to hide its contents”

Ciphertext 3:

“The Data Encryption Standard is a symmetric key algorithm for the encryption of electronic data”

Ciphertext 2:
“The protocol was named after Kerberos from Greek mythology the ferocious three headed guard dog of Hade”

Ciphertext 1:
“Remember to look up at the stars and not down at your feet and never give up work”