Something Awesome Project: 3ncrypt0r

Hayton Lam (z5358949)

What I Achieved:

From my Week 1 blog post on my Something Awesome Project, I set out different goals and stretch goals. I achieved the following:

1. Basic: ASCII substitution-based encryption.
    a. Achieved and extended: I made my Caesar Cipher encryption be flexible to custom alphabets, and had different customizable options such as ignoring whitespace, foreign characters (characters not in the given alphabet) and case-sensitivity.
    b. For the 'janky piecewise' function, I made it more cryptographically safe (by randomly alternating between functions) AND it supported Unicode character encryption.

2. (+) Better Encryption
    a. Achieved! I implemented (from scratch), DES and AES. Both functions would generate relatively cryptographically safe keys (if one isn't given) using the secrets module.
    b. Both symmetric block ciphers supported Unicode character encryption.
    c. For AES, I made it support 128 and 256 bit-length keys.
    d. Extended: Both had almost all known block cipher modes implemented (going beyond ECB, CBC and CTR to include CFB, PCBC and OFB as well).

3. (+) Ransomware Mode / Encrypting Files
    a. Achieved and extended! My program can completely encrypt almost every file type. It also can act as a ransomware, encrypting and decrypting all files (and all subdirectory files in target area).
    b. It also shields itself from encrypting itself!
    c. Allows for the following encryption methods: DES, AES, Arcfour, Chacha20

4. (+) Stream Ciphers
    a. I decided to show I went beyond course content by implementing a different type of symmetric encryption method instead of implementing RSA. I decided to explore Stream Ciphers and their PRNGs.
    b. I implemented Arcfour and Chacha20, both of which are capable of encrypting Unicode characters as well.
5. (+) Pretty CLI
    a. I might not have been able to find the time to code a proper GUI, but I managed to make a prettier terminal interface with error messages and configurations.

At the end of the day, I decided not to do some, such as RSA encryption as I felt it was more interesting to explore stream ciphers. I also didn't end up doing the wargames idea nor the password manager idea. The former because that would just be generating random words and encrypting them without giving the key or protocol. The latter because it was too ambitious… the scale was too big.

Why did I choose 3ncrypt0r (cryptography)?

At first, I really wanted to explore more cryptography. My previous experiences with cryptography have been homemade janky encryption solutions, such as Caesar ciphers and ASCII piecewise function encryptors. I saw all the online encryption tools, with things like DES, AES and ArcFour and wanted to learn how they (the proper encryption protocols) worked. Since we're doing security… why not take the chance given to explore it more fully?

Sure, I'll learn about them in the course, but that's a top-level view of the algorithm. I wanted to go in-depth, and pick apart every small implementation detail, so that I could fully understand how they worked, what their weaknesses are and to make encryption look a lot less intimidating for myself.

So, that's how I ended up rolling my own implementations of DES, AES, ArcFour and Chacha20!

What was challenging?

There were three main challenges that I had throughout the project:

1. Learning how the bit/byte manipulation worked in different encryption methods and how to apply it into Python.
    a. Had to learn how to convert between bytearray(), bit array representations and hex strings.
2. How to encrypt files, especially when padding bytes matters (I can't just zero-pad as it might break some files dependent on NULL bytes at the end). A tricky part was also how do I recursively encrypt all files (even ones inside subdirectories) inside a selected directory.
    a. Pathlib, glob and read/writing with bytes is apparently the answer!
3. How do pseudorandom number generators work? (Oh, and debugging my encryption implementations…)
    a. Pseudorandom generators involve lots of shuffling and bit manipulation. Oh, and debugging requires lots and lots of patience (as per usual) …

Proudest Achievement?

When my ransomware mode worked! It took me quite some time to figure out how to pad properly (I used zero-padding at first, but then some files couldn't be restored). It took me a lot of research to figure out how to encrypt a file and its filename, *as well as* make sure I could recover the files by decrypting them again.

What would I have done differently next time?

Honestly, I think I should've started coding earlier. I spent way too long researching all the theoretical aspects of the algorithm and forming pseudocode. A lot of my troubles happened as I was implementing them in Python. Code design, how I was going to connect it to my CLI as well as how I was going to process my data gave me troubles *when* I started coding. I didn't even think about them during my research/planning phase!

What I Learned:

This project taught me lots of things that I didn’t know!

1. Padding strategies: Some block ciphers require certain number of bits per block (such as 64) and my input might not always be a multiple of 64. That’s where adding extra bytes comes in handy. However, I can’t just pad it with NULL bytes as some files require trailing NULL bytes to work properly. That’s where I found lots of different padding strategies. The one I used was the ANSI X9.23 strategy, where I pad up to 8 bytes, and the final byte indicates how many padding bytes I added.

2. How to encrypt files. Before this project, I thought encrypting files was just reading the text content of files and encrypting them. But then how did ransomware work? You could encrypt Excel spreadsheets, but they have more than just text in them. The answer? Read the entire file as a series of bytes and manipulate the bytes directly.

3. The different types of encryption methods. Even within symmetric ciphers, stream and block ciphers have different features which make them good/vulnerable. By implementing them manually, I got a deeper understanding into what makes a good encryption protocol, and why the steps taken to shuffle the bits, XOR different sections and transpose into a matrix were necessary.

4. How random number generators work for computers. Instead of importing the random module, ArcFour and Chacha20 had their own pseudorandom number generators, which were used to make ‘random’ keys and bits. This gave me good insight into how randomness works in cryptography, but also how RNG works in other software as well… say for example, video games!