CT255 Cybersecurity

Assignment 1

Maxwell Maia 21236277

Problem 1

The code starts at the main method.
A plaintext input is placed in the main method's argument. This is a String type.

The main method…
It checks that an argument is present.
//args != null && args.length > 0

If there are no arguments, show an error message.

If there are arguments, put the plaintext input into the hashF1 function.
// hashF1(args[0]);

Inside the hashF1 function…
- Create an array of integers with all indexes equal to 1.
// int[] hashA = new int[]{1, 1, 1, 1};

- Create a filler string
"ABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGH".

- Check that the input String is of 1-64 charcters in length (inclusive). If not, return -1 to the main method.
// if ((s.length() > 64) || (s.length() < 1))

- Add the filler after the input and limit the string to the first 64 characters.
// sIn = s + filler;
// sIn = sIn.substring(0, 64);

- Loop through the 64 characters. For each character, do the following:
Calculate the character's ascii code MULTIPLIED by 17 [17 is a number chosen to encode this hash function].
Add this calculated number to the integer array at its corresponding index.
// hashA[0] += (byPos * 17);
Do this for 17, 31, 101, and 79.
// hashA[1] += (byPos * 31);
// hashA[2] += (byPos * 101);
// hashA[3] += (byPos * 79);

- Then modulus each index of the integer array by 255.
// hashA[0] %= 255;
// etc…

- Add the (indexes of the integer array)*256^index. The result of this is the hash.
// ret = hashA[0] + (hashA[1] * 256) + (hashA[2] * 256 * 256) + (hashA[3] * 256 * 256 * 256);

- If the hash is negative, make it positive.

- Return the integer hash to the main method.

Returning to the main method…
- If the hash function was successful, print out the plaintext input and the hash function.

- Start searching for collisions… Which is done in problem 2.