### CSC165 F23 Lab 11 Cybersecurity Overview

In Lab 11, you will be “encrypting” a password. Encryption is much more involved than what you are doing, but this will give you a sense of the general process. You will be using binary and hexadecimal numbers, and using ASCII values. Dictionaries are used throughout to store data.

1. You will be using several files, so let’s start with getting things organized. From Moodle, download the lab11\_cyber folder and place in your csc165 folder. The folder has the following files:
   1. encrypt.py : a collection of functions that encrypt a password
   2. login.py : manages the username and password verification process
   3. database.py : manages the username/password file.
   4. conversions.py : defines several dictionaries used throughout the encryption process
   5. users.txt : test file that holds usernames and encrypted passwords
   6. create\_encodings.py: used to randomly generate a bitstream for each ascii character. You do not need to do anything with this file.

*NOTE: Remember to comment your code, keep it organized and use good naming conventions! Place all your test code in the body of the if \_\_name\_\_ block. Nothing should be aligned along the left edge except function definitions and the if \_\_name\_\_ statement.*

You will be using ASCII characters from 33 (‘!‘) to 126 (‘~’). There is a link to an ASCII table below. Recall that in Python, you can easily cast from character to the ascii integer value using the cast methods chr() and ord().

Translation between decimal, binary, and hexadecimal:

Deci Binary Hex

|  |  |  |
| --- | --- | --- |
| 0 | 0000 | 0 |
| 1 | 0001 | 1 |
| 2 | 0010 | 2 |
| 3 | 0011 | 3 |
| 4 | 0100 | 4 |
| 5 | 0101 | 5 |
| 6 | 0110 | 6 |
| 7 | 0111 | 7 |
| 8 | 1000 | 8 |
| 9 | 1001 | 9 |
| 10 | 1010 | A |
| 11 | 1011 | B |
| 12 | 1100 | C |
| 13 | 1101 | D |
| 14 | 1110 | E |
| 15 | 1111 | F |

Resources:

* <https://www.cisco.com/c/en/us/products/security/what-is-cybersecurity.html>
* <https://cloud.google.com/learn/what-is-encryption>
* <https://www.ssl2buy.com/wiki/difference-between-hashing-and-encryption>
* <https://www.techtarget.com/searchsecurity/definition/salt#:~:text=What%20is%20password%20salting%3F,stealing%20them%20from%20the%20database>.
* <https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/>
* <https://en.wikipedia.org/wiki/List_of_Unicode_characters>
* <https://ss64.com/ascii.html>

### CSC165 F23 Lab11 Requirements

There are 3 files to complete for the 3 levels of requirements.

1. Base Level: encrypt.py. These functions convert a user-entered password to an encrypted one.
2. Level Up: login.py. These functions get a username/password from the user and confirm that they are valid.
3. Advanced: database.py. These functions read in the username/password file, write the username/password pairs to a file, and add a user account.

It is your responsibility to make sure that your code is running correctly. There are test cases provided for both the Base Level and the Level Up.

PLEASE place test code in the body of an if \_\_name\_\_ == ‘\_\_main\_\_’ statement. You only need one of these if-statements. It should be placed at the bottom of the file.

**BASE LEVEL : encrypt.py**

Complete the functions that have been outlined for you in the file. In general, to encrypt a password, you

1. salt(): “Salt” the password by lengthening it to 16-characters and “shifting” each character.
2. create\_bitstream(): Convert the salted password to a bitstream (i.e. a series of 0s and 1s) using a random encoding that converts each character to a 10-bit string.
3. hexity(): Convert the bitstream to a string of hexadecimal characters (0 to F)
4. asciify(): Convert the hexadecimal characters to ASCII characters.
5. encrypt(): Given a password, create an encrypted version

**Passwords**

The password that the user creates must be 6 to 15 characters long and use the range of ASCII characters from ! to ~ (this is the range of decimal 33 to 126).

Each function is described below and an algorithm has been provided.

**Salting the Password**

Given a plain text password (something the user enters), create a 16-character string based on the entered password and a special 16-character string that “shifts” each character. This standardizes the length of the passwords and starts the encryption process.

Algorithm

Concatenate 3 copies of the password together to ensure it is at least 16 characters long

newpassword = Capture the first 16 characters of this concatenated password

converted = Create a new empty string for the converted password

For each character and number in newpassword and salter

c = ord(character) + number

if shifted beyond 126

Wrap value around with c%127 + 33

Add chr(c) to converted

**Create a Bitstream**

Transform a string of characters to a bitstream using a randomly generated encoding. For this assignment, each character is converted to an 10-bit stream, which in turn are concatenated. With a 16-bit character string, the resulting bitstream will be 16\*10 = 160 bits.

The encodings are randomly generated and stored in a dictionary.

Algorithm

bitstream = start with an empty string

For each character in the password

Add the 8-bit string that is associated with that character to the bitstream

**Convert a Bitstream to a Hexadecimal String**

Each hexadecimal digit (0-F) corresponds to 4 bits (0000 to 1111). Traversing from left to right over the bitstream, convert every 4 digits to its corresponding hex digit using the bin2hex dictionary.

**Convert a Hexadecimal String to an ASCII**

Each ASCII digit corresponds to 2 hexadecimal digits. Traversing from left to right over the string of hexadecimal digits, convert every 2 digits to an ASCII character in the range from 33 to 126. The digits can be converted to decimal values using the hex2deci dictionary. You can ensure it is in that range by adding 33, and then wrapping it around if it is over 126.

Algorithm

astream = start with an empty string

for every 2 characters in the hexadecimal string

d1 = Get the decimal value of the first character using the dictionary

d2 = Get the decimal value of the second using the dictionary

d = d1\*8 + d2 + 33

If the value is > 126

d = d%127 + 33

Add chr(d) to astream

Here are some test cases: These are the answers using %127:

* Password:awesome! Encrypted: }<&ww{@l-?~^dGu"!D5{
* Password:sMiLe:-) Encrypted: {;d-DyB6+E{=6o9#>l#/

**LEVEL UP : login.py**

There are 2 functions to complete in login.py

1. login(), which asks the user for their username and password. Then calls verify\_password to make sure they are valid.
2. verify\_password, which takes a username and password typed in by the user (i.e. not the encrypted password). It uses the dictionary users\_db in database.py to verify the username and password.

Here are some test cases:

* Username: user003 password: happyDay!
* Username: user005 password: ZipPidy<>++

**ADVANCED : database.py**

There are 2 functions to complete in database.py

1. write\_users() writes the contents of the users\_db dictionary to the file. This is necessary when new accounts are added.
2. add\_account() asks the user for a username and password, making sure that it meets the constraints. It then adds it to the dictionary and writes it to the file.