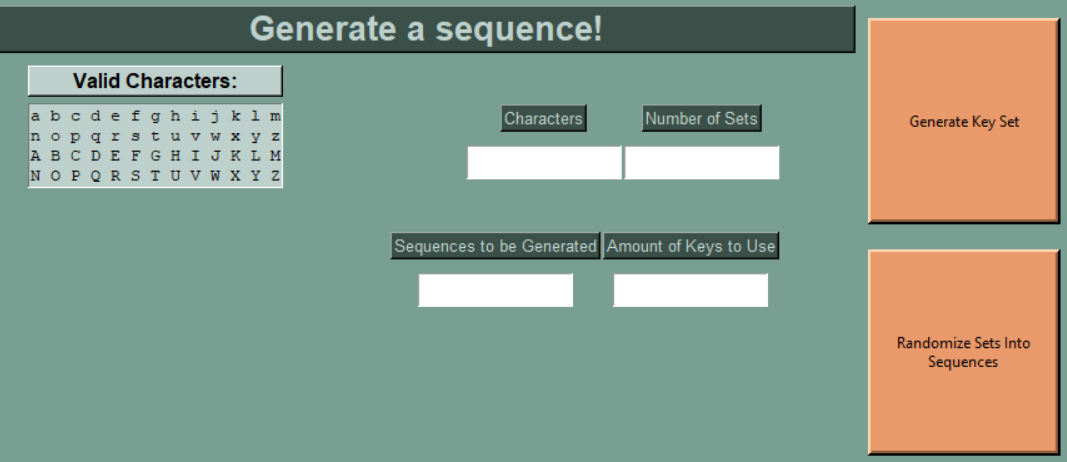
This is a TKinter application intended to provide users with a way to safely store passwords locally. There are three primary functionalities related to this app:

* Generating key sets
* Encryptor
* Decryptor

## Generate Key Sets and Sequences

This part of the application randomizes a list of 72 integers (one for each valid character) and into “key sets” which are columns that a character’s positional value can be mapped to. These key sets can then be turned into sequences either manually or through the built-in functionality.



**Generating Key Sets:**

* To generate these key sets, enter in any of the alphabetical letters displayed in the “Valid Characters” box and the number of sets for each character you would like created
* If you only want 2 sets created for the characters “ABC”, the CSV output would have 6 columns in total with the headers being A0, A1, B0, B1, C0 and C1 with each column having auniquely randomized order of integer placements (character reassignments).

**Randomizing Existing sets**

* Enter the number of sequences (an order of key sets as a single string) to be created and the number of keys to be randomly selected from the input. Both input values for this part of the page are **integer values only**.
* This functionality will request for you to upload a csv in the same format/layout as the one that is generated.
* Example Input:
  + Sequences to be generated: 3
  + Number of Keys to Use: 2
* Example Output:
  + A1B1
  + C1A0
  + A0B0

## Encrypting

The Encryption technique used in this application is a variation of the polygraphic cipher and the permutation cipher in that an inputted string of characters is passed through one or more sets of integer keys that reassign that character’s positional values to the location of that character in the new set(s).



**Password Requirements:**

* Strings that need to be less than 30 characters in length
* Cannot contain spaces
* Need to contain characters from the allowed characters table below:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a | b | c | d | e | f | g | h | i | j | k | l |
| m | n | o | p | q | r | s | t | u | v | w | x |
| y | z | A | B | C | D | E | F | G | H | I | J |
| K | L | M | N | O | P | Q | R | S | T | U | V |
| W | X | Y | Z | ! | @ | # | $ | % | ^ | & | \* |
| ? | ~ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

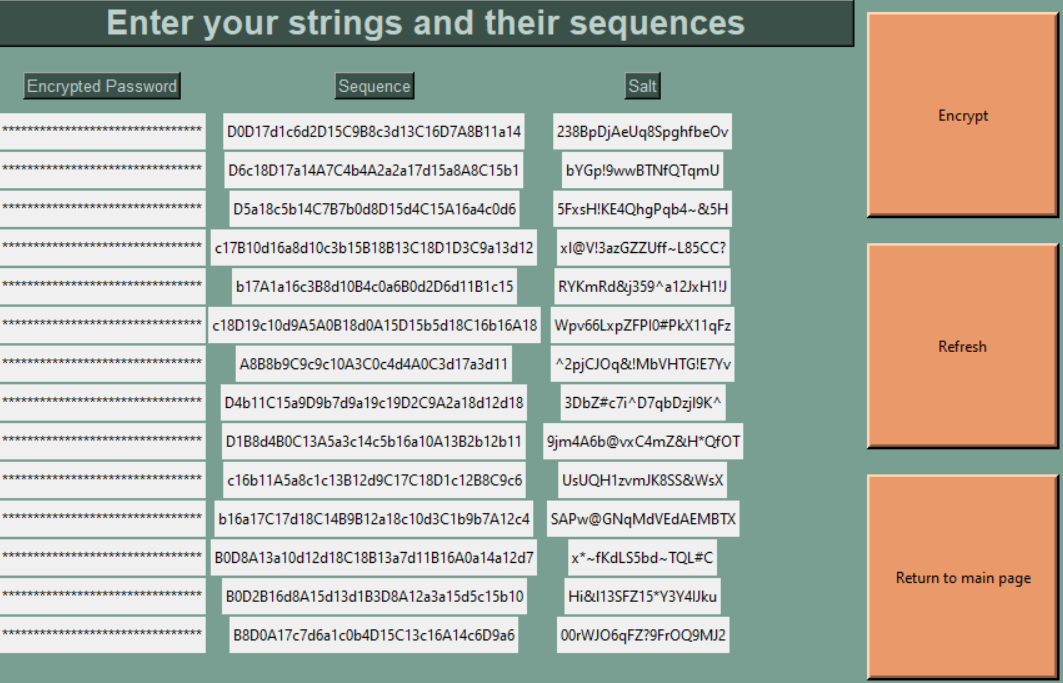
**Sequences**:

* A sequence is an ordered string of key sets with each set being denoted by alphabetical characters
* The order the sequence is recorded/entered in is the order the key sets in it will be read (from left to right) and applied to the password string

**Salt:**

* The salt in this process is a random selection of characters from the allowable characters table that **are not** present in the password string
* These salt characters are sprinkled randomly throughout the password string. Once salted, the whole string will be passed through its respective sequence.

**Output:**

* The outputted fields should be stored for stress-free decryption later on down the line as both sequence and salt are required for decryption.
* If the option is ticked before executing, the user will be prompted to select a folder destination and a name for their output file.

## Decrypting

The decryption process works similarly to the encryption process with the key difference (no pun intended) being that it requires the salt in order to undo the encryption.



**Sequence**:

* Since the encryptor reads the sequence order from left-to-right, the decryptor reverses the process by passing the password string through key sets from right-to-left.

**Salt:**

* The salt is removed from the string by searching for and removing only the salt characters from the final string to reveal the original password

**Output:**

* The output view consists of the inputted sequences, salt strings and their respective decrypted password strings

