BITNCRYPT

by: Jay Nakum

People either use insecure passwords

Or forget them entirely

Using complex passwords need a secure place to store them.

Do you trust?

There are many tools available for password management but there is always an issue of trust.

It can be so easy to guess to and leads to security breaches

There are some smart ways but not everyone uses them.

That is why we have BITNCRYPT

BitNCrypt is an easy to use, multi-threaded, fast, powerful and secure command line tool that generates passwords on the fly. It uses a custom algorithm to automate the smart method. It solves the issue of remembering the passwords and storing them securely.

But HOW?

The three terms

Key

It is a 6 digit number that user has to remember.

TXT File

It is a txt file that user enters during the setup

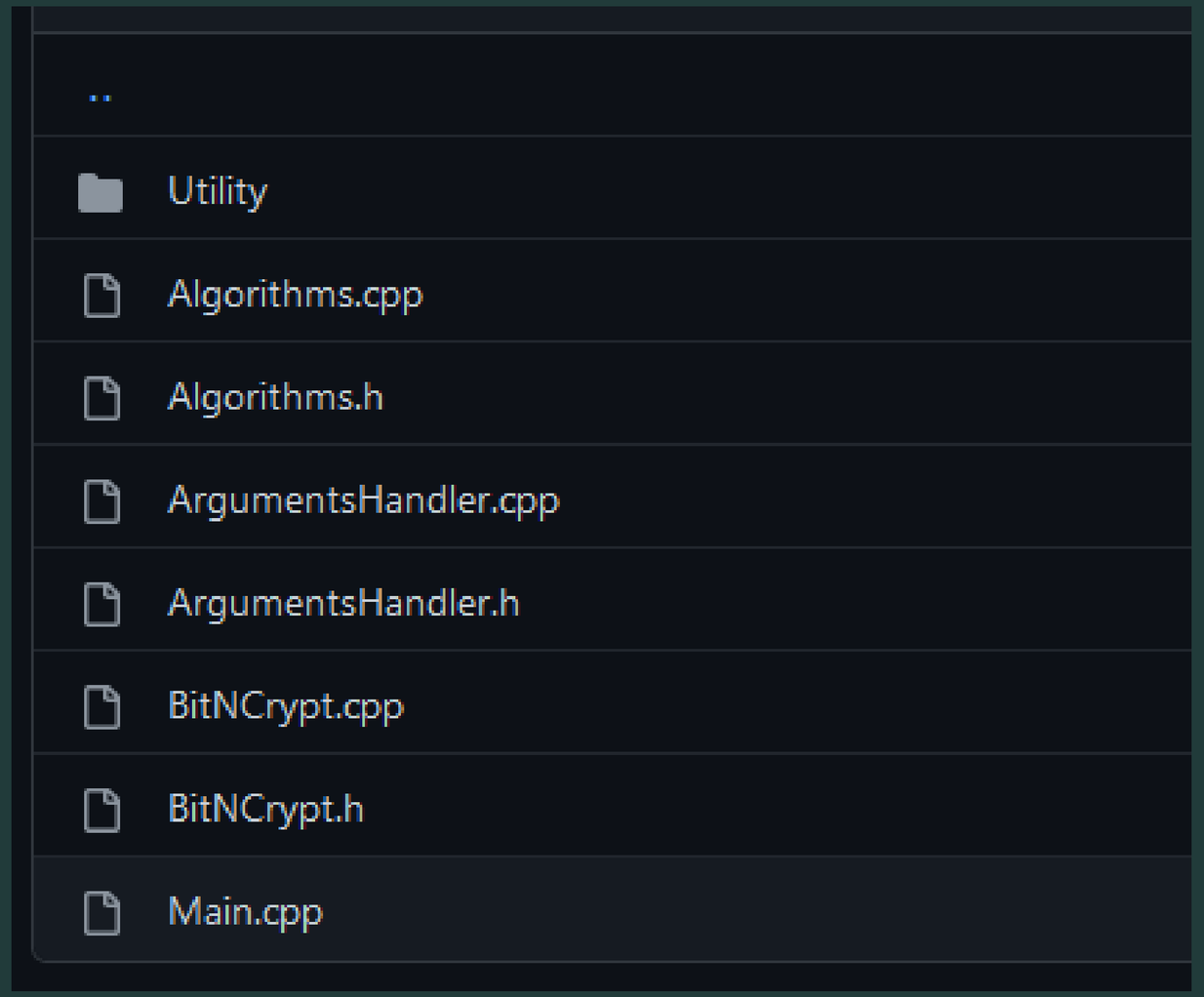
Keyword

A different keyword for each password.

HOW

Project Structure

- Arguments Handler
- BitNCrypt Class == The Brain
- Algorithms Class == The Heart
- Utility Package
  - Maths
  - IOManager
  - Logging [CppLog]

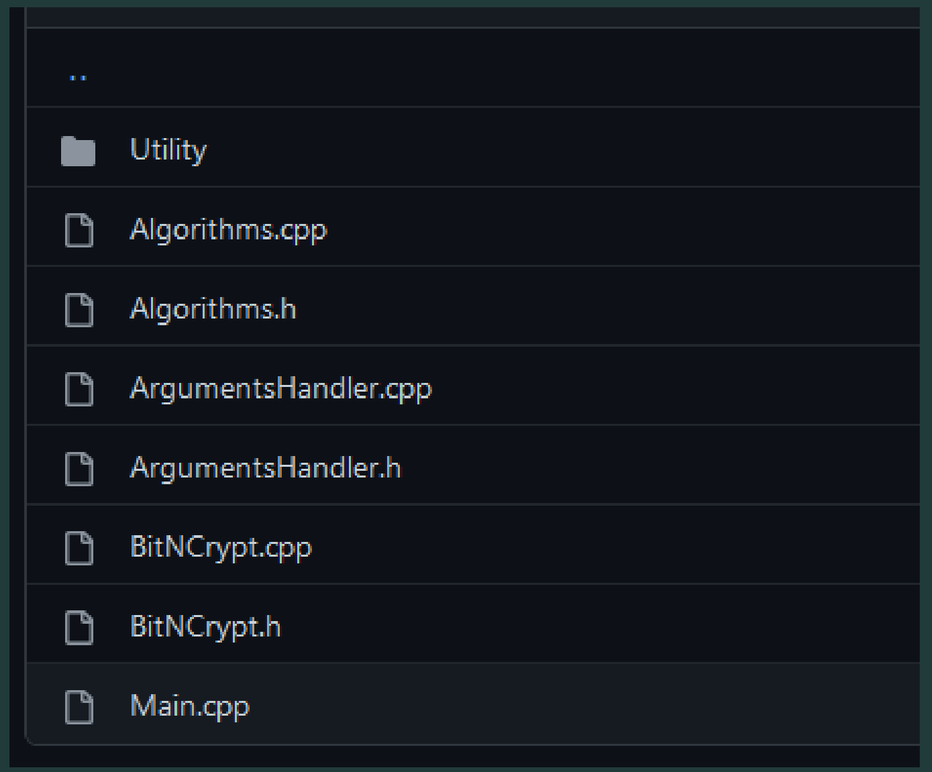


HOW

Project Structure

- Arguments Handler
- BitNCrypt Class == The Brain
- Algorithms Class == The Heart
- Utility Package
  - Maths
  - IOManager
  - Logging [CppLog]

All of this code is well documented and available on my github.




```
/// Here multiple threads are being created to
/// 1) generate and display the password key
/// 2) jumble the key file
/// 3) delete the old key.txt
/// this is done for improving execution speed
```

Algorithms

Jumble the file

- Read the file content character by character
- If 50% chance
  then
    subtract a random number to the ascii value of c
  otherwise
    add a random number to the ascii value of c
- Resave the file

Algorithms

Jumble the file

- Read the file content character by character
- If 50% chance
  then
    subtract a random number to the ascii value of c
  otherwise
    add a random number to the ascii value of c
- Resave the file

The 50% chance is there to increase the range of resulting ascii values.

Algorithms

Jumble the file

- Read the file content character by character
- If 50% chance
  then
    subtract a random number to the ascii value of c
  otherwise
    add a random number to the ascii value of c
- Resave the file

The 50% chance is there to increase the range of resulting ascii values.

Generate a key

- while key.length < 6
  - k = random(9)
  - if k in key then 1 in 12 chance to insert(k)
  - else insert(k)

Algorithms

Jumble the file

- Read the file content character by character
- If 50% chance
  then
    subtract a random number to the ascii value of c
  otherwise
    add a random number to the ascii value of c
- Resave the file

The 50% chance is there to increase the range of resulting ascii values.

Generate a key

- while key.length < 6
  - k = random(9)
  - if k in key then 1 in 12 chance to insert(k)
  - else insert(k)

6*2 = 12 chance that the duplicate number will be added. 1 in 12 because we prefer unique numbers but occasional duplicate will encourage confusion

Encryption Algorithm

Initial Section

- while password.length() < key.length()
  - password.append(keyword)
- password = password AND txt

AND operation prevents from starting with keywordkekeywordyword

Encryption Algorithm

Initial Section

- while password.length() < key.length()
  - password.append(keyword)
- password = password AND txt

Shifting

- divide the password into 4 sections
- for each section perform
  - if key[i + 2] is even
  - then
    - left circular shift each bit by key[i+2]
  - otherwise
    - right shift each bit by key[i+2]
- for each result
  - password.append(key[i%3])

AND operation prevents from starting with keywordkekeywordyword

Last four digits of key are used for each section
checking whether the number is even or odd gives a good mix of left and right shift

key[i%3] gives a number between 0 to 3 which randomizes the way results are appended. Reason why distinct numbers are encouraged.

Encryption Algorithm

XOR

- xor each bit of password with each bit of txt

AND results in more 0s
OR results in more 1s
NOT only uses password
hence XOR

Encryption Algorithm

XOR

- xor each bit of password with each bit of txt

Final Permutation

- divide the password into 6 different sections
- from each section and for each k in key
  - take char c = [key % section.length()]
  - if c is an alphabet
  - then
    - insert(c)
  - otherwise
    - insert(key[key.length() - i])
  - if k is even then insert(k)
  - if k%3==0 then insert(valid_symbol)

AND results in more 0s
OR results in more 1s
NOT only uses password
hence XOR

6 sections to utilize the whole key. And inserting one or two numbers or symbols would get the desired password length

At this point password can contain weird symbols.
This algorithm gives a good mizture of alphabets, numbers and symbols

ALGORITHM ANALYSIS

This algorithm ensures

- 8 to 10 digit password
- Good mixture of letters, numbers and symbols
- Generates same password with same key, keyword and txt.
- Txt cannot be replicated even if the content is known.
- Gives a not easy to guess, brute-force free password.

ALGORITHM ANALYSIS

This algorithm ensures

- 8 to 10 digit password
- Good mixture of letters, numbers and symbols
- Generates same password with same key, keyword and txt.
- Txt cannot be replicated even if the content is known.
- Gives a not easy to guess, brute-force free password.

WHY BITNCRYPT

- Generates good password
- Easy to use
- Fast (multi threading)
- Secure (Requires key and txt both)

Thank You!

Any Questions?

CHECK OUT BITNCRYPT

https://jaynakum.github.io/BitNCrypt/

https://github.com/JayNakum/BitNCrypt

https://jaynakum.github.io/