A

Project Report on

RANDOM PASSWORD GENERATOR

**ACKNOWLEDGEMENT**

Acknowledgements

I would like to express my sincere gratitude to everyone who has supported me in the successful completion of this project on the Random Password Generator.

First and foremost, I would like to thank my project supervisor, [Supervisor's Name], for their invaluable guidance, insightful feedback, and continuous support throughout the project. Their expertise in cybersecurity has been crucial in shaping the direction and execution of this project.

I am also grateful to the faculty and staff of the [Department/Institute Name] for providing the necessary resources and a conducive environment for research and development.

Special thanks to my colleagues and friends, whose constructive criticism and encouragement have greatly contributed to the project’s progress.

Finally, I extend my heartfelt appreciation to my family for their unwavering support and understanding during the course of this project.

This project has not only been an academic endeavor but also a practical exploration of enhancing cybersecurity through robust password management. I hope that the Random Password Generator developed in this project will contribute to better security practices and awareness.

TABLE OF CONTENTS

**CHAPTER NO  TITLE OF THE CHAPTER  PAGE NO**

**1  Abstract  1**

**2  Introduction 2**

**3  Existing Method 3**

**4  Methodology  4**

**5  Implementation  6**

**6 Testing 8**

**7 Output  9**

**8 Conclusion  10**

CHAPTER 1

ABSTRACT

Security is one of the most crucial parts of our lives. In today's digital age, cybersecurity is of paramount importance as individuals and organizations increasingly rely on digital platforms for storing and transmitting sensitive information. One of the fundamental aspects of securing digital assets is the use of strong, unpredictable passwords. This project presents the development of a Random Password Generator (RPG) aimed at enhancing cybersecurity by providing users with robust, random passwords.

The RPG leverages cryptographically secure algorithms to generate passwords that are resistant to common attack vectors such as brute force and dictionary attacks. The generator allows users to customize password length and complexity, incorporating a mix of uppercase and lowercase letters, digits, and special characters to ensure high entropy.

In addition to the development of the RPG, this project also explores the importance of strong password policies and the role they play in preventing unauthorized access. The efficacy of the generated passwords is evaluated against established security criteria to ensure their strength and reliability.

By offering a user-friendly interface and integrating best practices in password management, this Random Password Generator aims to promote better cybersecurity hygiene among users. The project underscores the significance of strong passwords as a first line of defense in protecting digital identities and sensitive information.

CHAPTER 2

INTRODUCTION

In today's digital age, the importance of robust cybersecurity measures cannot be overstated. As more personal and sensitive data is stored and accessed online, the need for secure authentication mechanisms becomes paramount. One of the fundamental elements of securing access to systems and data is the use of strong, unpredictable passwords.

Passwords serve as the first line of defense against unauthorized access, and their strength can significantly impact the security of digital assets. However, creating and remembering complex passwords can be challenging for users. To address this issue, random password generators play a crucial role in enhancing cybersecurity practices.

A random password generator is a tool that automatically creates passwords that are difficult to guess or crack. These passwords typically contain a mix of letters (both uppercase and lowercase), numbers, and special characters, thereby increasing their complexity and reducing the likelihood of being compromised through brute force attacks or other common password-cracking techniques.

This project focuses on the development and implementation of a random password generator. The goal is to provide a user-friendly, efficient, and secure tool that can help individuals and organizations generate strong passwords to protect their accounts and sensitive information. The project will explore various algorithms and techniques used in random password generation, assess their effectiveness, and ensure that the generated passwords meet industry standards for security.

By understanding and implementing a robust random password generator, this project aims to contribute to the broader efforts in enhancing cybersecurity and protecting digital identities from malicious actors.

CHAPTER 3

EXISTING METHOD

Random password generation is a critical component of cybersecurity, aimed at creating secure, unpredictable passwords that enhance the security of digital assets. Various methods and algorithms are employed to generate these passwords, each with its strengths and potential vulnerabilities. Here are some of the most common existing methods used in random password generation:

1. Pseudo-Random Number Generators (PRNGs)

PRNGs are algorithms that use mathematical formulas or pre-calculated tables to produce sequences of numbers that appear random. These generators typically start with a seed value and use complex calculations to produce a seemingly random sequence. Common PRNGs include:

Mersenne Twister: Widely used for its long period and high-quality randomness.

Linear Congruential Generator (LCG): Simpler but less secure due to predictability over time.

Pros: Fast, easy to implement, suitable for general purposes.

Cons: If the seed value is known or guessed, the sequence can be reproduced, reducing security.

2. Cryptographically Secure Pseudo-Random Number Generators (CSPRNGs)

CSPRNGs are designed specifically for cryptographic applications, ensuring higher security standards. They generate random numbers that are not only unpredictable but also resistant to reverse engineering.

CryptGenRandom (Windows): Provides access to the Windows cryptographic service provider to generate secure random numbers.

Fortuna: A secure PRNG designed for cryptographic applications, incorporating multiple entropy sources.

Pros: Highly secure, suitable for cryptographic applications.

Cons: Slightly slower than non-cryptographic PRNGs, requires more system resources.

CHAPTER 4

METHODLOGY

The methodology for developing a random password generator in a cybersecurity context involves several key steps, including requirement analysis, design, implementation, testing, and evaluation. This structured approach ensures that the final product is secure, efficient, and user-friendly.

Steps in the Methodology

1. Requirement Analysis
2. Design
3. Implementation
4. Testing
5. Development and maintenance
6. REQUIREMENTS

**System requirements:**

1.Requirements at developer’s end:

* + Hardware: operating system: MS Windows XP or Ubuntu
  + Languages: python
  + RAM: 512 MB
  + Hard disk: 5GB
  + Software: python 3

2.Requirements at client ‘s end:

* + Hardware
  + Software

B. DESIGN

1. Start
2. Store all the characters as a list. Use the sting module of python to store them all.
3. Ask the user to enter the length of the password.
4. Shuffle the characters using random. shuffle method.
5. Initialize an empty list to store the password.
6. Write a loop that iterates length times
   1. Pick a random character from all the characters using random. Choice method.
   2. Append the random character to the password.
7. Shuffle the resultant character list to make it random.
8. Convert the password list to string using the join method.
9. print the password.

C. IMPLEMENTATION

During the implementation process, developers must write enough comments inside the code so that if anybody starts working on the code later, he/she can understand what has already been written. Writing good comments in very important as all others documents, no matter how good they are, will be lost eventually. Ten years after the initial work, you may find only that information which is present inside the code in the form of comments.

Development tools also play an important in this phase of the project. Good development tools save a lot of time for the developers, as well as saving money in terms of improved productivity. The most important tools for time saving are editors and debuggers. A good editor helps a developer to write code quickly. A good debugger helps make the written code operational in a short period. Before starting the coding process, you should spend some time choosing good development tools.

**CHAPTER 5**

**IMPLEMENTATION:**

import string

import random

# characters to generate password from

alphabets = list(string.ascii\_letters)

digits = list(string.digits)

special\_characters = list("!@#$%^&\*()")

characters = list(string.ascii\_letters + string.digits + "!@#$%^&\*()")

def generate\_random\_password():

# length of password from the user

length = int(input("Enter password length: "))

# number of character types

alphabets\_count = int(input("Enter alphabets count in password: "))

digits\_count = int(input("Enter digits count in password: "))

special\_characters\_count = int(input("Enter special characters count in password: "))

characters\_count = alphabets\_count + digits\_count + special\_characters\_count

# check the total length with characters sum count

# print not valid if the sum is greater than length

if characters\_count > length:

print("Characters total count is greater than the password length")

return

# initializing the password

password = []

# picking random alphabets

for \_ in range(alphabets\_count):

password.append(random.choice(alphabets))

# picking random digits

for \_ in range(digits\_count):

password.append(random.choice(digits))

# picking random special characters

for \_ in range(special\_characters\_count):

password.append(random.choice(special\_characters))

# if the total characters count is less than the password length

# add random characters to make it equal to the length

if characters\_count < length:

random.shuffle(characters)

for \_ in range(length - characters\_count):

password.append(random.choice(characters))

# shuffling the resultant password

random.shuffle(password)

# converting the list to string

# printing the list

print("".join(password))

# invoking the function

generate\_random\_password()

CHAPTER 6

**D. TESTING**

Testing is probably the most important phase for long-term support as well as for the reputation of the company. If you don’t control the quality of the software, it will not be able to compete with other products on the market. If software crashes at the customer site, your customer loses productivity as well money and you lose credibility. Sometimes these losses are huge. Unhappy customers will not buy your other products and will not refer other customers to you. You can avoid this situation by doing extensive testing.

Example test case:

Test case 1:

Sample Input:

Enter password length:

Enter alphabets countin password:

Enter digits countin password:

Enter special characters countin password:

Expected output:

A randomly generated password with the number of respective characters.

Supplied Input:

Enter password length: 10

Enter alphabets countin password: 3

Enter digits countin password: 2

Enter special characters countin password: 3

Obtained output:

V2(&#XlQq1

If you see the password generated this time, it has the minimum number of characters that the userwants. And the program has included 2 more random characters to make the password length equal touser input.

Hurray! We have a complete strong password generator.

**E. DEVELOPMENT AND MAINTENANCE**

The program should be opened in python 3 software and it is executed by the user. The input screen appears where the user inputs the required information. The output is automatically displayed on the screen i.e.; the generated password is displayed on the screen.

Since the password can be generated as per the given number of characters. It will be easy for the user to customize it as required.

CHAPTER 7

**OUTPUT**

Enter password length: 12

Enter alphabets count in password: 3

Enter digits count in password: 3

Enter special characters count in password: 6

1$\*4q$()7G$p

Enter password length: 10

Enter alphabets count in password: 3

Enter digits count in password: 2

Enter special characters count in password: 3

V2(&#XlQq1

**CHAPTER 8**

**CONCLUSION**

With the help of this project one can easily generate a random password anytime he/she wants. This project is very useful for those who are always in a perplexed state about what password to use for a tight security purpose.

As these passwords are randomly generated, no one can ever guess them and therefore it provides a tight security be it for lockers or any software etc. which requires protection from outsiders. We can also add many other features to improvise the code as per our requirement and make sure that the resultant password is strong enough.