

# Password generator using HTML, CSS, and JavaScript

20IT095 - Isha. Patel

[20it095@charusat.edu.in](mailto:20it095@charusat.edu.in)

Department of Information Technology

20IT088 - Disha. Patel

[20it088@charusat.edu.in](mailto:20it088@charusat.edu.in)

Department of Information Technology

**Abstract :-** Password which are used in google accounts, social media , while login in any website sometime we require password. When a person login in any website but does not remember the password or forgets the password. For changing password sometimes it takes 10 - 15 minutes for changing the password or sometimes the computers ask about strong password which is not used frequently. Using this application we can create a strong password more than one time. The users of computer technology and internet are increasing day by day. As the users are growing , the need for security is also felt very much. The data assets and other valuable facts are stored in the computer systems. One of the way to safe guard the data assets is to have a proper authorization method to access the data. This is achieved by user identification and password mechanism. The selection of password is important, since the entire authorization is dependent on the password. The password need to be strong enough to avoid brute force attack and other attacks. Here we discuss a method of generating random passwords, which are strong enough to combat the attacks.

**Keywords:-**Password, Random Password , Encryption , Decryption, Security, Symmetric Key

## **1.INTRODUCTION**

This app is going to be quite flexible and can be used to create any kind of password. The app will have a few options like:

- Setting the length of the password from 6-20 characters (you can expand the range yourself in the code)
- Choosing whether the number, uppercase letter, lowercase letter, and symbol appear in the password or not.
- And finally, a copy button to copy the password to your clipboard and use it.

We can even select the length of the password, and if we don't need any character that is also possible.

Many technologies have been devised to either improve on the level of security passwords provide, typically by: providing alternative means of user authentication, through identity management systems , or simply by making the use of strong, i.e. not readily guessable, passwords easier. In this paper, in line with the previous remarks about the continuing ubiquity of passwords, we focus on the third of the above categories.

To enhance the security of user created passwords, system administrators and organizations employ a set of rules, called password composition policy, which users should incorporate in their passwords. One of the password composition policy suggested by NIST is that a minimum of eight character length password and it must include at least one uppercase letter, one lowercase letter, one digit and one special character [1]. The purpose of such policies is to expand the search space of passwords. An important fact is that when users are allowed to select their password, they favor memorability of password without considering its security against an attacker. In environments where such password composition policies are used, users try to find an easy escape from it. For example, users create passwords like “#Diamond#”, “Alok@123”, which can be guessed easily and weak but satisfies the policy.

Text based password authentication systems involve a tradeoff between security and memorability of passwords. Some passwords are easy to remember but also easy to guess for an adversary. Random passwords are hard to remember and hard to crack because they are made up of arbitrary sequence of characters [2]. Several studies have examined how password composition policies affect users. In a study by Komanduri et al. reveals how password composition policies influence the predictability of passwords and as well how they affect the user behavior and sentiments. Their results demonstrate that successfully creating a password is significantly more difficult under stricter password composition policies. They measured how many people failed at least once to create an acceptable password and further observed how users deal with it [3].

Manoj Kumar Singh [4], proposed a method to exploit the artificial neural network to develop the more secure means of authentication, which is more efficient in providing the authentication. He discussed about architecture of neural Network, learning rule, target definition and process, which will apply for authentication. He has highlighted that neural network with intrusion detection capability can handle the challenge associated with password and authentication.

Michel D. Leonhard and V.N. Venkatakrishnan [5] analyzed three random password generation algorithms namely ALPHANUM, DICEWARE, and PRONOUNCE3. They used the metrics such as security, memorability and affinity to find out, which of the three methods produces best passwords. They used six character length random password, which contain upper-case letters, lower-case letters, and numbers. They have highlighted that random passwords have several benefits over user-chosen passwords mainly security and confidentiality. DICEWARE generator produces random lists of words. This is based on the concept of memorization and the mental connection required to remember the password. The PRONOUNCE3 produces pronounceable words in English. This aims at the speech facilities of the user's mind to assist in remembering the password.

Ayushi [6] proposed a Symmetric Key Cryptographic Algorithm for increasing the security of data. Secret key cryptography methodologies are classified as either stream ciphers or block ciphers. Stream ciphers operate on a single bit at a time, and feedback mechanism is used. Block cipher uses one block of data at a time using the same key on each block. The same plaintext block is always encrypted to the same cipher text when using the same key in a block cipher whereas the same plaintext will be encrypted to different cipher text in upstream cipher. In this paper, bit manipulation is taken for encryption and decryption. The data is converted to binary digits and reversed. The reversed digits are divided by the key and the result is converted as the encrypted data.

## 2. RELATED WORK

### 2.1. Alpha-Numeric Random Passwords

→ This is the simplest random password generation technique. In this, random passwords are generated by choosing characters randomly from the defined character set. To generate a random password of specific length, above step is repeated that many times. For example, a character set with lowercase letters (26), uppercase letters (26) and digits (10) and password length of six. The cardinality of the character set is  $26 + 26 + 10 = 62$ . Now, there are 62 choices for each six positions. Total password space of the alphanumeric scheme is given by equation (1).

$$62 \times 62 \times 62 \times 62 \times 62 \times 62 = 62^6 \approx 5.68 \times 10^{10} \approx 235.7 \text{ -----(1)}$$

The cardinality of password set is 235.7 say  $|P|$ . Any password chosen from  $P$ , has entropy of 35.7 bits. This entropy metric is used to determine the strength of passwords. The alpha-numeric passwords have highest randomness among all the three methods, which also make them harder to remember.

## 2.2. Pronounceable Random Passwords

→Pronounceable random password generation techniques make use of language specific (like English) properties and generate pronounceable random string. The objective of the scheme is to utilize the speech facilities of the user's mind to assist in remembering the password. Ganesan and Davies described a major flaw in pronounceable passwords schemes. These schemes choose syllables based on their frequency in English language, using complex rules to achieve pronounceability [4].

Leonhard and Venkatakrisnan defined a PRONOUNCE3, pronounceable random password generator that does not have the flaw described above [5]. The PRONOUNCE3 generator defined five vowel elements (a, e, i, o, u) and twenty two consonant elements (b, c, ch, d, f, g, h, j, k, l, m, n, p, ph, r, s, st, v, w, x, y, z). A password generated using PRONOUNCE3 scheme has entropy of 30.8 bits. The entropy 30.8 bits is less than alpha-numeric's 35.7 bits but PRONOUNCE3 passwords have better memorability.

## 2.3. Mnemonic Phrase-Based Random Passwords

Some systems suggest users to create mnemonic phrase-based passwords. Kuo et al, defines mnemonic phrase-based password is one, where a user chooses a memorable phrase and uses at least one character (often first character) to represent each word in the phrase [6]. Ideally, your password should contain a mixture of lowercase and uppercase letters, digits and special symbols.

The mnemonic phrase-based passwords appear hard to guess than regular passwords. The security of mnemonic phrase password is better because they do not appear in any dictionary and usually contain a good mixture of letters, digits and special symbols. Kuo et al, build a dictionary of 400,000 mnemonic passwords using mnemonic phrase found commonly on internet. They cracked 4% of mnemonic passwords, in comparison, a standard dictionary with 1.2 million entries cracked 11% of cracked passwords. They shown it is possible to create a dictionary to crack mnemonic phrase passwords [6].

## 3. ProActive Random Password Generation System

The password generation system is divided into three modules.

### 3.1. Random Password Generation

For random password generation, first a word is chosen randomly from the wordlist. The wordlist is prepared from a standard dictionary file. The memorability of these words is quite good for users. Now, random digits and special symbols are inserted in the randomly selected word at random positions. The random numbers are generated using a high quality random stream. The count of random characters can be specified according to the policy need and we recommend minimum of three. The generated password is sent for the ProActive analysis.

Procedure:

Step 1: Start the process

Step 2: Create random character list with numbers, upper & lower-case letters.

Step 3: Password must be in fixed length of 12.

*Step 4: Create Random Password Generator method to generate the password.*

*Step 5: Random Password Generator chooses any of the three character set.*

*Step 6: The index position of any one of the characters from the random character set is returned.*

*Step 7: Append the characters selected through the index, one by one.*

*Step 8: Print the password.*

*Step 9: End.*

### **3.2. ProActive Analysis**

*The ProActive analysis tests the generated password against some attacker approaches. If the generated password is found positive in ProActive test then it is discarded and the process starts all over again. This ProActive test ensures that password cannot be guessed on the fly. The ProActive analysis perform a series of tests on the generated password like it does not contain more than three consecutive letters (either lowercase or uppercase), three or more consecutive sequential letters or digits from alphabet set and all the inserted characters are not same. There is no need to perform any dictionary check as it will never happen. This ProActive analysis confirms that generated password does not contain any easily predictable pattern. If any of the tests found positive then password is dropped instantly.*

### **3.3 . New Password Encryption Method**

*Cryptography is a process of converting ordinary text or plain text to cipher text (encryption), then converting back to the original text (decryption). There are several ways to classify the various algorithms. The most common types are i) Secret Key Cryptography which is also known as Symmetric Key Cryptography and ii) Public Key Cryptography which is also known as Asymmetric Key cryptography. The algorithm that we are adopting is the symmetric key cryptographic technique. The key that is used to encrypt the password is 4 digits binary number ( $\geq 1000$ ). This algorithm maintains three separate variable lists that contain upper and lower case letters and numbers. To encrypt the password, the algorithm checks the alphabets in an inputted password, and finds out whether the alphabet is an upper-case/lower-case/number and then corresponding constant is used with the character.*

*The algorithm consists of three parts. The first part is to convert the password to decimal value. The second part is to perform binary manipulations. The third part is to converting binary value to alphanumeric value. In the first step there are three cases to change alphanumeric into decimal value: (1) If the alphabet is upper-case letter, and then the ASCII value of alphabet and constant 50 are summed. (2) If the alphabet is lower-case letter, then the constant 20 is subtracted from the ASCII value of alphabet (3) If the alphabet is number, and then the constant 10 is subtracted from the discovered ASCII value.*

*The second step converts the decimal value received from the previous procedure into binary and reverses the binary value. The reversed binary value is divided by the key. The key is selected by the user. The remainder and the quotient are formed as resultant binary value. The remainder in first 3 digits and quotient in next 5 digits are formed. The final step is to convert the resultant binary value to alphanumeric value.*

## *Procedure:*

*Step 1: Start the process.*

*Step 2: Store upper-case letters, lower-case letters and numbers as separate variable lists.*

*Step 3: Random password is used as input.*

*Step 4: Conversion from the alphabets to decimal notations. 4.1. Check the alphabet is upper-case. Find out its ASCII value and add with constant 50. 4.2 Check the alphabet is lower-case. Find out its ASCII value and subtract the constant 20 from the ASCII value. 4.3 Check the alphabet is Number. Find out its ASCII value and subtract the constant 10 from the ASCII value.*

*Step 5: Convert the resultant decimal value to binary digits.*

*Step 6: Reverse the binary digits.*

*Step 7: Get the key from the user.*

*Step 8: Divide the reversed binary digit by the key.*

*Step 9: Store the remainder in first 3 digits & quotient in next 5 digits (remainder and quotient wouldn't be more than 3 digits and 5 digits long respectively. If any of these are less than 3 and 5 digits respectively we need to add required number of 0s (zeros) in the left hand side.*

*Step 10: Convert the binary to decimal value*

*Step 11: Consider the decimal value as the ASCII value and find the related character as encrypted value.*

## *4. Implementation*

*The proposed system is implemented in 'C' language on windows 7 operating system. The wordlist is prepared from a standard dictionary file and stored in a text file, one word per line format. The output of the system present "random word", "generated password", "random character string" and "random position string" on the output screen as shown in fig. 1. Here, "random word" is word used to generate random password. Each digit in "random position string" represents the index of digits or special characters present in "random character string" respectively. Output of system can be redirected to desired application and can be stored in desired format as per need.*

### **Password Generator**

The screenshot shows a web application titled "Password Generator". It has several input fields and a button. The "Pattern" field contains "9S9UU9UL99UU". The "Min. Length" field contains "7" and the "Max. Length" field contains "12". The "Allow Duplicates" section has two radio buttons: "Yes" (unselected) and "No" (selected). The "Result" field displays "9#2TY8Zn74NV". A yellow "Generate" button is located at the bottom right of the form.

## *5. Evaluation*

*The security of generated passwords is measured on the following two metrics. The generated passwords are safe against simple dictionary attacks and common wordlists, which found on internet.*

### 5.1. Levenshtein Distance

Vulnerability of passwords to dictionary attacks with mangling rules can be measured by determining the similarity between password string and common dictionary words [7]. The Levenshtein distance metric calculates the distance between two strings by counting and then adding the minimal number of single character manipulation required, such as insertion or deletion, to make the strings equivalent. Campbell et al, in its experiment calculated the Levenshtein distance score for passwords using Fedora Core 5 English dictionary [8]. They separated passwords into two clusters, one with zero to two edit distances and other with four to six edit distances. Their results have shown that passwords with three or more Levenshtein's edit distances are safe from dictionary-style attacks. The passwords generated have minimum of three Levenshtein's distance from the dictionary words.

### 5.2. Password Entropy and Brute Force Approach

The brute force approach requires an attacker to test every possible combination as password. The character set for this technique contain lowercase (26) and uppercase (26) letters, digits (10) and special symbols (8). The cardinality of character set is  $26 + 26 + 10 + 8 = 70$  characters and minimum password length is eight. According to brute force approach, there are 70 choices available for each position in eight length password. Total search space is given by equation (3).

$$70 \times 70 \times 70 \times 70 \times 70 \times 70 \times 70 \times 70 = 70^8 \approx 5.76 \times 10^{14} \approx 249.03 \text{ (3)}$$

Here generated passwords have entropy of 49.03 bits, which is highest entropy among all the techniques discussed above in section 1.1. In general, higher the entropy for a given distribution of passwords, it is more difficult to guess the password and attackers are forced to check larger number of combinations. The table (1) presents theoretical brute force online and offline attacks scenario simulation against the password generated using proposed technique.

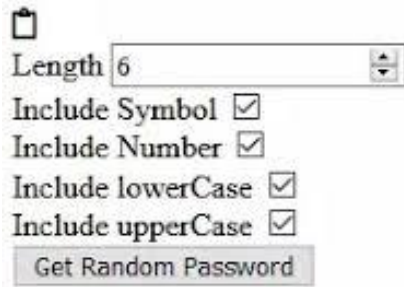
### 5.3. Memorability

A good password is often defined as one that is hard to guess and easy to remember. It implies that memorability is equally important as security. Once confirmed with security issue, memorability of generated password is considered. To help users in memorizing generated password system present users both how the password is generated and what word is used to do so. In this way password is presented with three chunks (random word, random position string and random character string) of information to help in memorizing it.

## DETAILS FOR THIS APPLICATION

The password generator in JavaScript is a simple program for generating random pass keys. The whole programming is in HTML, CSS, and JavaScript. We can make any length of password which person can remember but nearly 10 or 12 letters only. We can also recreate the password. Password is the first defence against unauthorized access to your accounts or devices, so you need a stronger password to prevent attacks from hackers. Simple and small passwords with just uppercase or lowercase letters are pretty easy to crack. Passwords with a good combination of numbers, symbols, uppercase, and lowercase letters are very hard to crack. The more complex the password is more secure and strong it is.

# Password Generator



A screenshot of a web-based password generator interface. It features a clipboard icon at the top left. Below it is a 'Length' input field with the value '6' and a dropdown arrow. Underneath are four checkboxes, all of which are checked: 'Include Symbol', 'Include Number', 'Include lowerCase', and 'Include upperCase'. At the bottom is a button labeled 'Get Random Password'.

A random password generator is software program or hardware device that takes input from a random number generator and automatically generates a password. Random passwords can be generated manually, or they can be generated using a computer. Creating or generating the password does not make the password safe and strong. Thus, there is no need at all for a password to have been produced by a perfectly random process, it just needs to be sufficiently difficult to guess.

## Conclusion

We can create a JavaScript and CSS random password generator app. It creates a strong password with variable length and the ability to select the type of characters, symbol, numeric values in the password. Complex password composition policies and policies that require password must be changed after a period of time happens to be major obstacle for users. The proposed technique can assist system administrators in creating secure and memorable passwords for users with desired complex password composition policies. The generated password along with helping information (random word, random position string and random character string) will be sent to users. This technique gives several benefits to users such as security, and confidentiality. The password generated using proposed technique is more secure because it is chosen from a large distribution of passwords and is stronger than user created passwords. The proposed technique causes more Confidentiality because in this technique, distinct passwords are given to users on different applications. If an application is compromised then rest of all are protected. Future work includes determining the memorability of the generated password. Intuitively, it can be said that the passwords generated using proposed technique are more memorable than pure random passwords.

## References

- [1] Burr WE, Dodson DF, Polk WT. Electronic authentication guideline. NIST Special Publication 800-63 version 1.0.2, 1992.
- [2] Yan J, Blackwell A, Anderson R, Grant A. Password memorability and security: Empirical Results. Security and Privacy, IEEE Journal, 2004, p. 25-7.
- [3] Komanduri S, Shay R, Kelley PG, Mazurek ML, Bauer L, Christin N, et al. Of passwords and people: Measuring the effect of password composition policies. CHI'11 Proc. of annual conf. on Human factors in computing systems, 2011, p. 2595-10.
- [4] Ganesan R, Davies C. A new attack on random pronounceable password generators. In Proceeding 17th NIST-NCSC National
- [5] Manoj Kumar Singh, " Password Based a Generalize Robust Security System Design using Neural Network", IJCSI-International Journal of Computer Science Issues, Vol. 4, No. 2, 2009.

[6] Michael D. Leonhard, V. N. Venkatakrishnan, "A Comparative Study of Three Random Password Generators", *IEEE EIT 2007 Proceedings*.

[7] Ayushi, "A Symmetric Key Cryptographic Algorithm", *International Journal of Computer Applications*, Volume 1 – No. 15, 2010.