A Uniform Password Generator with Required Character Sets

Rob Yoder, Pilar García, and Jeffrey Goldberg

1Password

November 19, 2018

1 The problem

The strength of a password depends on the system by which it was generated. In general the strength can be defined in terms of (twice) the expected number of guesses it would take an attacker to find that password using a guessing strategy that is aware of the distribution.

If the password generation scheme produces a uniform distribution, the strength can be taken as entropy of the distribution. For example, if a generation scheme will draw uniformly from a set of 2^{56} distinct passwords, then the average number of guesses to find the password will be half of that, and we will be happy to call the strength of such a generation scheme "56 bits".

Once the distribution becomes non-uniform, there is no principled way to determine the strength of a password generated from the system. We recommend using min-entropy (Cederlöf 2005; Goldberg 2013) as the least bad approach in that case. However, in some cases it is very difficult to compute the min-entropy of a generation scheme. The withdrawn FIPS-181 algorithm, still commonly used for pronounceable password generation, is extremely difficult to analyze. (Ganesan and Davies 1994) We therefore aim to have a password generator whose output follows a uniform distribution. We also want the generator to provide reliable information about the strength of any password it would generate.

1.1 Requirements

It is simple enough to uniformly generate a password of length n using an alphabet of, say, uppercase letters, `[A-Z]`; lowercase letters, `[a-z]`; and digits, `[0-9]`. And it is easy to calculate the strength of such a password: $n \log_2(62)$. The difficulty comes when the password to be generated must include members of certain subsets of characters. Once we require that the generated password include at least one lowercase letter, at least one uppercase letter, and at least one digit, neither producing a uniform distribution nor calculating the strength is easy.

One approach would be to first generate candidate passwords uniformly as described above and then reject those that don't meet the requirements. The obvious problem with this approach is that it is not guaranteed to terminate. Can we find an algorithm with a deterministic number of steps that produces a uniform distribution of passwords?

1.2 Shuffling off this non-uniform coil

An appealing, but doomed, approach to generating passwords with character set requirements is to generate a shorter password the simple way, collect one character randomly from each required set of characters, and then shuffle the characters that meet the requirements into the final result. This is illustrated in Algorithm 1.

Algorithm 1: Shuffle in generator

Data: A set $\mathcal{R}$ of mutually disjoint sets of required characters.
A set A of all allowed characters.
An integer n, the length of password to generate.

Result: p, a string of length n.

```
1 p ← SimpleGenerate(A, n − |R|)
2 for R ∈ R do
3     t ← RandomDraw(R)
4     p ← RandomInsert(p, t)
5 end
6 return p
```

The difficulty with Algorithm 1 is that it produces non-uniform output. If, for example, there is one way to generate some password, p_i, but two

(or more) ways to generate some other password p_j, then the distribution of generated passwords is not uniform.

Consider a case where we are generating a password of length 3, a digit is required, and the allowed characters are letters and digits. We will look at three outcomes. In each of them, the initial value of p from line 1 will be "a1" and the digit drawn in line 3 will be "1".

Line 4 could insert the random digit in one of three positions: 1, 2, or 3. Each is equally likely, and yields "1a1", "a11", and "a11", respectively.

So there are two ways to generate the string "a11" and one way to generate "1a1". Thus, we have a non-uniform distribution: some passwords are more likely to be generated than others.

We might be able to accept this non-uniform distribution if we could calculate the min-entropy of it, but since we have not figured out how to do that in the general case (with multiple required character sets), we don't know how much this is costing in strength.

1.3 A second attempt at determinism

Since we really wanted to come up with an algorithm that would terminate in a set number of steps, we tried again. Rather than shuffle a digit into an existing string that may already contain digits, what if we partition the random string by character set first, add the required character to the digit string, and then merge the component strings back together randomly, preserving the order of the characters in each component string?

This is illustrated in Algorithm 2.

Algorithm 2: Partition-then-shuffle generator

Data: A set $\mathcal{R}$ of mutually disjoint sets of required characters.
A set A of all allowed characters.
An integer n, the length of password to generate.

Result: p, a string of length n.

1 $p \leftarrow \texttt{SimpleGenerate}(A, n - |\mathcal{R}|)$
2 **for** $R \in \mathcal{R}$ **do**
3 $\quad r, p \leftarrow \texttt{Partition}(R, p)$
4 $\quad e \leftarrow \texttt{Concat}(\texttt{RandomDraw}(R), r)$
5 $\quad p \leftarrow \texttt{RandomMerge}(p, e)$
6 **end**
7 **return** p

aa	a1 or 1a	11
aa1	a11	111
a1a	1a1	
1aa	11a	

Table 1: Different probabilities for resulting passwords

Unfortunately, Algorithm 2 still manages to produce non-uniform output. Consider the even further simplified case where we are generating a password of length 3, the allowed characters are "a" and "1", and a "1" is required. Because the only illegal permutation is "aaa", it's easy to see that there are 7 possibile results: $2^3 - 1$.

Since we will draw a single "1" to add to the password, our initial random password string will be 2 characters long: "aa", "a1", "1a", or "11". Say we generated "a1" in line 1. In line 3 we separate that into the strings "a" and "1". Line 4 adds another "1" to give us "a" and "11", and line 5 merges the two together, creating either "a11", "1a1", or "11a".

Table 1 shows each of the seven possible results beneath the simple p with which we began. There is a 1/4 chance of selecting "aa" as the initial p, and when we combine the partitioned and extended strings "aa" and "1", there is a 1/3 chance of getting "aa1". This gives "aa1" (and each result in the first column) a 1/12 chance of being selected.

On the other hand, there are a 1/2 chance of generating an initial p of either "a1" or "1a", which are equivalent for our purposes because they will both end up partitioned and extended as "a" and "11". When we combine "a" and "11", there is a 1/3 chance we will get "a11", giving that result an overall probability of 1/6.

Finally, there is a 1/4 chance of generating "11" initially, but once we do, we are guaranteed to end up with "111", giving that result a probability of 1/4, the highest of all the results.

Again, we have a non-uniform distribution: some passwords are more likely to be generated than others. We must alas reject this elaborate scheme, too.

1.4 Generate and validate

Having rejected both shuffle approaches and run out of other ideas, we settled on the first approach described. Generate passwords using the simple

method until we get one that contains at least one character from each of the required sets.

Though the algorithm is not guaranteed to terminate, we accept the risk, knowing that it decreases as the password length increases. The rest of this paper focuses on calculating the strength of passwords generated this way.

1.5 Notation

- For any set A we will have "$|A|$" denote the size of A.
- For any set A, "$\mathcal{P}(A)$" will represent the power set of A, that is, the set of all subsets of A.
- "$N(n)$" represents the number of possible strings allowed by a scheme as a function on n, the length of the of the password to be created.
- For any real value x, "$\lg(x)$" denotes the base 2 logarithm of x.
- For any two sets A and B, the set $A \setminus B$ contains all elements of A that are not in B. $A \setminus B = \{x \mid x \in A \wedge x \notin B\}$

2 A concrete non-solution

To help clarify the problem, we will start by working through a concrete example.

- L is the set of letters
- D is the set of digits
- S is the set of other allowed symbols

The number of possible results for a randomly generated password of length n that allows letters and digits is given by:

$$N(n) = |L \cup D|^n \tag{1}$$

Now consider the case where the password is required to have at least one letter and one digit. The password is obtained as follows:

1. Generate a password, p, from the set $L \cup D$.
2. Check whether the password has at least one letter and one digit.

3. If it does, accept the password. If not, return to step 1.

We can compute the number of possible passwords of length n given these requirements:

$$N(n) = |L \cup D|^n - |L|^n - |D|^n \tag{2}$$

We want to include all passwords made up of letters and digits except the ones made up of only letters or only digits. Equation 2 is correct. Now, what if we want to require letters, digits, and symbols? It may seem natural to declare the following equation:

$$\begin{aligned} N(n) = &|L \cup D \cup S|^n - |L|^n - |D|^n - |S|^n \\ &- |L \cup D|^n - |L \cup S|^n - |D \cup S|^n \end{aligned} \tag{3}$$

Again we take the total number of candidate passwords, subtract all the ones that contain characters from only one of the required sets, and then this time also subtract those that are a combination of only two of the three required character sets.

But consider the following:

$$L \subseteq L \tag{4}$$

$$L \subseteq (L \cup D) \tag{5}$$

$$L \subseteq (L \cup S) \tag{6}$$

A string that contains only letters is being subtracted thrice, once for not containing a digit or symbol (L), once for not containing a symbol ($L \cup D$), and once for not containing a digit ($L \cup S$).

What we need, of course, is a way to subtract each excluded candidate password only once. Our first attempt with three required character sets (3) was not far off the mark, but it missed a key concept.

3 To recurse, divine

Among other things we now have to talk about sets of sets of characters. So let $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$ be the set of required character sets. $\mathcal{R}$ is a set of sets, not the union of those sets. Here we are calculating $N_{\mathcal{R}}(n)$, which is the number of passwords of length n given $\mathcal{R}$ as the set of required character sets.

Let's look at the simplest case, the empty set ($\varnothing$). If there are no character sets provided, then there are no passwords possible.

$$N_{\varnothing}(n) = 0 \tag{7}$$

How about a set containing just one character set:

$$N_{\{R_1\}}(n) = |R_1|^n - \Big(N_{\varnothing}(n)\Big) \tag{8}$$

The empty set may seem superfluous in equation 8, but it is there for completeness and will help us later on. Now let's look at the case with two required sets:

$$N_{\{R_1,R_2\}}(n) = |R_1 \cup R_2|^n - \Big(N_{\varnothing}(n) + N_{\{R_1\}}(n) + N_{\{R_2\}}(n)\Big) \tag{9}$$

We can define equation 9 in terms of equation 8 very naturally. If R_1 and R_2 are both required to be represented, we need to subtract the number of possibilities they represent individually. This definition is equivalent to equation 2 in section 2, which was also correct.

However, when we apply our new pattern to three sets of required characters, we find something interesting.

$$\begin{aligned} N_{\{R_1,R_2,R_3\}}(n) = &|R_1 \cup R_2 \cup R_3|^n \\ &- \Big(N_{\varnothing}(n) + N_{\{R_1\}}(n) + N_{\{R_2\}}(n) + N_{\{R_3\}}(n) \\ &+ N_{\{R_1,R_2\}}(n) + N_{\{R_1,R_3\}}(n) + N_{\{R_2,R_3\}}(n)\Big) \end{aligned} \tag{10}$$

This version is actually correct. Because equation 9 represents the number of possibilities where R_1 and R_2 are required, we can use its result in equation 10 to avoid the counting errors from before.

Look again at equations 7-10. Each of them is written in terms of the previous ones. This is a recursive function waiting to happen. Now look at the second major term in those functions. Each term inside is of the form $N_{\mathcal{X}}(n)$, with $\mathcal{X}$ ranging over all of the proper subsets of $\mathcal{R}$.

Since we are now talking about the set of all proper subsets of a set, the notion of power set becomes useful. The power set of $\mathcal{R}$, represented as $\mathcal{P}(\mathcal{R})$, is the set of all subsets of $\mathcal{R}$.

$$\begin{aligned} \mathcal{P}(\{R_1, R_2, R_3\}) = \{&\varnothing, \{R_1\}, \{R_2\}, \{R_3\}, \{R_1, R_2\}, \{R_1, R_3\}, \\ &\{R_2, R_3\}, \{R_1, R_2, R_3\}\} \end{aligned}$$

And now we have a set of sets of sets of characters.

But we want the set of all *proper* subsets of $\mathcal{R}$. That's everything in the power set but $\mathcal{R}$ itself. We can represent that as $\mathcal{P}(\mathcal{R}) \setminus \{\mathcal{R}\}$, and now we arrive at our general recursive formula:

$$N_{\mathcal{R}}(n) = \left| \bigcup_{X \in \mathcal{R}} X \right|^n - \sum_{Y \in \mathcal{P}(\mathcal{R}) \setminus \{\mathcal{R}\}} N_Y(n) \tag{11}$$

Finally, there is one more thing to add. Let A be the set of allowed characters that are not required.

$$N^A_{\mathcal{R}}(n) = \left| \bigcup_{X \in \mathcal{R} \cup \{A\}} X \right|^n - \sum_{Y \in \mathcal{P}(\mathcal{R}) \setminus \{\mathcal{R}\}} N^A_Y(n) \tag{12}$$

Note that for $N^A_{\varnothing}(n)$, we have $\varnothing \cup \{A\} = \{A\}$ and $\mathcal{P}(\varnothing) \setminus \{\varnothing\} = \varnothing$, and so the result is simply $|A|^n$. This provides us with the eventual ground to the recursive definition.

As each possible password from our generator is equally likely, it makes sense to report the entropy of any such generated password as

$$H^A_{\mathcal{R}}(n) = \lg N^A_{\mathcal{R}}(n) \tag{13}$$

References

Cederlöf, Jörgen. 2005. "Authentication in quantum key growing." PhD diss., Linköpings Universitet.

Ganesan, Ravi, and Chris Davies. 1994. "A New Attack on Random Pronounceable Password Generators." In *Proceedings of the 17th NIST-NCSC National Computer Security Conference,* 184–197.

Goldberg, Jeffrey. 2013. "Defining password strength." In *Passwords 13, Las Vegas.*