On Password Strength: A Survey and Analysis

Chapter *in* Studies in Computational Intelligence · June 2018

DOI: 10.1007/978-3-319-62048-0_12

Gongzhu Hu
Central Michigan University
121 PUBLICATIONS **989** CITATIONS

On Password Strength: A Survey and Analysis

Gongzhu Hu

Abstract Password has been a predominating approach for user authentication to gain access to restricted resources. The main issue with password is its quality or strength, i.e. how easy (or how hard) it can be "guessed" by a third person who wants to access the resource that you have access to by pretending being you. In this paper, we review various metrics of password quality, including one we proposed, and compare their strengths and weaknesses as well as the relationships between these metrics. We also conducted experiments to crack a set of passwords with different levels of quality. The experiments indicate a close positive correlation between the difficulty of guessing and the quality of the passwords. A clustering analysis was performed on the set of passwords with their quality measures as variables to show the password quality groups.

Keywords Password quality · Entropy · Levenshtein distance · Hashing · Password cracking

1 Introduction

Online security has been a major concern since the time when the Internet became a necessity for the society, from business activities to everyday life of ordinary people. A fundamental aspect of online security is to protect data from unauthorized access. The most commonly used method for doing this is to use password as part of the online access process. Password is a secret character string only the user knows and its hashed code is stored on the server that provides access to the data. When the user requests for data access, he enters the password along with other identification information, such as user name or email. A message digest cypher (hash) of the password is computed and the hashed code is transmitted to the server that matches the hash

G. Hu (✉)
Department of Computer Science, Central Michigan University, Mount Pleasant, MI 48859, USA
e-mail: hu1g@cmich.edu

Fig. 1 Password mechanism for resource access request

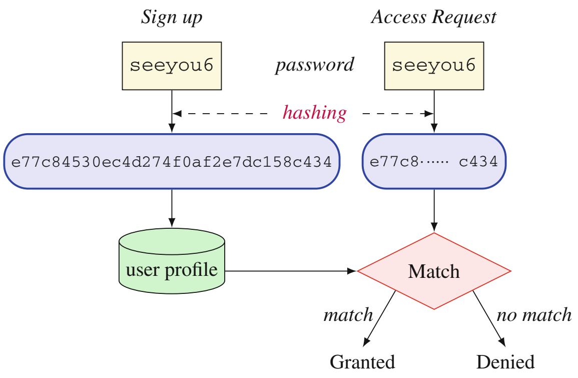
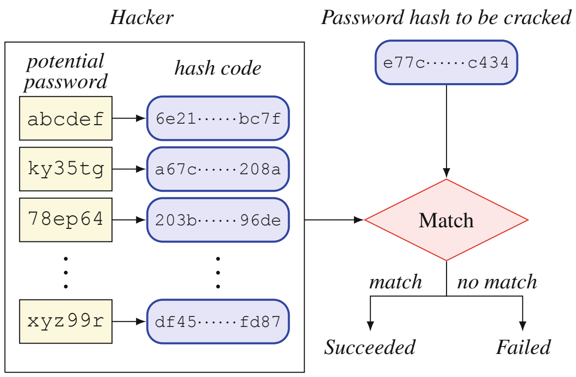


Fig. 2 Hacking the hash code of a password



code previously stored in its database. If a match is found, the user is assumed to be the one who claims who he is, and access is granted; otherwise access is denied, as illustrated in Fig. 1.

Formally, a hash function f maps a password (character string) p to a hash code (cypher) h:

$$f(p) = h \tag{1}$$

where f is irreversible, i.e. function f^{-1} does not exist.

Since the hashing function is irreversible, the only way a third person (a hacker) can guess the password is to try a possible character string to see if its hash code matches the hash code of the actual password. If no match, the hacker may try another string, and continues this process (for off-line attack) until a match is found (the password is cracked) or the hacker gives up without success. The idea of cracking a password is shown in Fig. 2, where the hacker tries a sequence of potential password $p_1, p_2, \ldots, p_k$ to generate hash codes $h_1, h_2, \ldots, h_k$, and to see if $h_k = h$, where h is the given hash code to be cracked. For online attack, a "three strikes" type of rule may prevent the hacker to try more than 3 times, though.

The problem for the hacker to figure out is how to select strings $p_1, p_2, \ldots, p_k$ as potential passwords. There are various approaches for doing so, all based on guessing. From the user's point of view, the critical point to the secrecy of the password is make it very hard for hackers to guess the password from the hash code. This is the issue of *password quality* (or strength).

In this paper, we survey various measures for password quality including a new *complexity* measure that we propose in the paper, and compare these measures by analyzing their relationships. Experiments on cracking a small set of passwords of different quality levels was conducted to confirm the correlation between the difficulty of cracking and password quality. We also applied clustering algorithm on these measures to a set of passwords to group them into four quality clusters (weak, fair, medium, strong). Since password quality is mostly an indicator as to how hard a password can be cracked, we first briefly discuss approaches to crack passwords.

2 Password Cracking Methods

Each password p is hashed using an encryption function f to generate its hash code h as defined in Eq. (1). Since only h (rather than p) is stored in the database on the resource server, the task of cracking a password is to use a method c such that $c(h) = p$. Note that c is not f^{-1} that doesn't exist. So, the hacker needs to figure out what method c to use such that he has a better chance to succeed finding p.

Commonly used methods for cracking passwords include brute-force attack, dictionary attack, and some variations of these with time-space tradeoff considerations.

2.1 *Brute-Force Attack*

Let $p = a_1 a_2 \ldots a_l, a_i \in \Sigma$ be a password of length l, where Σ is an alphabet of character set and $N = |\Sigma|$. Given the hash code $h = f(p)$, brute-force attack is to try each possible string $s = x_1 x_2 \ldots x_l, x_i \in \Sigma$ until $f(s) = h$.

The amount of time t to crack a password is proportional to the number of possible strings:

$$t \in \Theta(N^l) \tag{2}$$

where N^l is the size of string pool the possible passwords are drawn from the alphabet Σ. That is, the time complexity of brute-force attack is *polynomial* of N and *exponential* of l. In general, the alphabet Σ is one of the basic sets or combination of them. The basic sets as given in Table 1.

Most passwords are from lowercase letters ($\boldsymbol{L}$) only. In this case, for a password of length 6 (minimum length required by most service sites), there are $26^6 \approx 308.9 \times 10^6$ possible strings to try. If we can make 100,000 calls to the crypt function f per second, it will take 3089 s, or 51.5 min, to exhaust all possible strings from $\boldsymbol{L}$. For a larger alphabet, say $\boldsymbol{D} \cup \boldsymbol{L}$, a 6-char password will need over 6 h to crack, while a 7-char password needs over 9 days, and 11 months for a 8-char password.

It is practically infeasible to use brute-force approach to crack a password of length 7 or longer with a larger Σ. It is the basic reason why most service sites require passwords to be at least 8 characters long, with at least an uppercase letter, and a digit or a special character, in addition to lowercase letters. This will make $N = (62 \text{ or } 94)$ and $l \geq 8$.

2.2 Dictionary Attack

Since most people use human-memorable passwords that are likely words in dictionaries or some variations of these words, a hacker can try each word in the dictionary rather than random string used in brute-force. With this approach, each word $w_i \in D$ in a dictionary D is checked to see if $f(w_i) = h$, where h is the given hash code to be cracked. Hence, it is very easy to crack if the password is a word in the dictionary. In practice, all the hash codes $f(w_i)$ are pre-computed and stored in a database, rather than computed at run time.

There are two issues with this approach. First, the dictionary needs to contain a very large number of words to cover most (if not all) passwords that you think people may use. Second, most users are aware of dictionary attack and avoid using actual dictionary words for passwords; instead, they make small changes to the word so that it is still easy to remember. For example, rather than directly using `essay` as password, they may use `essay1`, `e55ay`, or `3ssay`. Hence, dictionary attack often apply some rules to the "spelling" of the words, such as replacing `l` by `1`, `o` by `0`, `e` by `3`, `s` by `5`, etc.

Let D be the dictionary used and m be the number of rules, the time t to crack a password using dictionary attack is

$$t \in \Theta(|D| + m) \tag{3}$$

That is, the time complexity is *linear* to the size of the dictionary and number of rules, a significant improvement from brute-force. However, the method may fail to find the password if the dictionary does not contain the password or its variations after applying the rules.

Table 1 Basic alphabet

Σ	N
$\boldsymbol{D} = \{0..9\}$	10
$\boldsymbol{L} = \{a..z\}$	26
$\boldsymbol{U} = \{A..Z\}$	26
$\boldsymbol{S}$ = {~‘!@#$%^&*()-_=+[{]}\\|;:”’,<.>/?}	32

2.3 Table Lookup and Rainbow Table Attack

To speed up the cracking time, table lookup attack stores precomputed hashes of potential passwords (dictionaries) in a database, and cracks a given password hash by searching the database. The search is a lot faster than the original dictionary attack simply because no need to compute the hash for each guess at run time. However, the method requires a huge amount of storage space to store “all” possible passwords and their hashes.

Rainbow table attack [16] is a variation of table lookup attack. Instead of pre-computing the hash codes of a large number of potential passwords and storing in a database, rainbow table approach is a time-memory trade-off to store much less number of hash codes that still represent a huge number of passwords. The basic idea is to create a password-hash chain of length k that covers k potential passwords and their hash codes.

For each word $w \in D$ in a dictionary D, we create a chain $c = (p_1, h_1, \ldots, p_k, h_k)$, where $p_1 = w, h_i = f(p_i)$ and $p_i = r(h_{i-1}), f$ is a crypt hash function and r is a *reduce* function that “reduces” a has code to a potential password. For each chain, only the pair (p_1, h_k), i.e. the *initial* password and the *ending* hash code, is stored in the database. Hence, if $k = 10,000$, the pair (p_1, h_n) represents all the 10,000 passwords and their hash codes in the chain, hence a significant saving on storage space. A rainbow table T is a set of chains: $T = \{c_i, i = 1, \ldots, n\}$, where c_i is the chain for word w_i in D.

To crack a given hash code h of password p, we check if h equals the ending hash h_k. If it does, $r(h_k)$ is the target password p. Otherwise, we keep applying r and f alternatively backward in the chain until the password is found, or we move the next chain in the table T. If all chains are exhausted, we just failed to find p.

The time complexity of rainbow table attack is

$$t \in \Theta(k|D|) \tag{4}$$

that is *linear* to the dictionary size but with a constant coefficient k that may be quite large (say, 10,000 or larger). It is a time-memory trade-off that, using the same storage space, it covers k times more words using the same dictionary but about k times slower than dictionary attack.

One of the problems with rainbow table approach is collision when two different hash codes are reduced to the same password. Another difficult is how to make the reduce function r “well behaved.” That is, r should map the hash codes into well-distributed (likely as user-selected rather than random) password set.

2.4 Attack Using Markov Model

It was argued that users prefer passwords that are easy to remember. Most users are aware of dictionary attack, so human-memorable passwords are mostly not in dictionaries, nor random (randomly generated passwords are hard to remember). One of the approaches to attack human-memorable passwords is "smart dictionary" attack using dictionaries that contain passwords that users are likely to generate. Narayanan and Shmatikov introduced a fast dictionary attack method based on the likelihood of the sequence of characters in users' passwords [15]. The method uses standard Markov model to generate smart dictionary that is much smaller than the ones used in traditional dictionary attack. The main observation given in [15] is that "the distribution of letters in easy-to-remember passwords is likely to be similar to the distribution of letters in the users' native language." Hence, the Markovian dictionary can be created based on the probability of the characters in a sequence.

Let $v(x)$ be the frequency of character x in English text, and $v(x_{i+1}|x_i)$ be the frequency of x_{i+1} given that the previously generated character is x_i. In the zero-order Markov model, the probability of a sequence $\alpha = x_1x_2 \dots x_n$ is

$$p(\alpha) = \prod_{x\in\alpha} v(x)$$

where the distribution of each character is independent of the previous character. In the first-order Markov model

$$p(x_1x_2 \dots x_n) = v(x_1)\prod_{i=1}^{n} v(x_{i+1}|x_i)$$

Markovian dictionaries are created accordingly at two levels. The zero-order dictionary is defined as

$$D_{v,\theta} = \{\alpha : \prod_{x\in\alpha} v(x) \geq \theta\}$$

where *theta* is a threshold. The first-order dictionary is

$$D_{v,\theta} = \{x_1x_2 \dots x_n : v(x_1)\prod_{i=1}^{n} v(x_{i+1}|x_i) \geq \theta\}$$

The great advantage of these models is that they drastically reduce the size of search space be eliminating the majority of words from the dictionary that are not likely to be user-selected passwords. It is shown in [15] that if $\theta = 1/7$ (i.e. only 14% of sequences are produced while 86% of sequences are ignored) the zero-order dictionary still has 90% probability covering the plausible passwords. A dictionary containing 1/11 of the keyspace has 80% coverage and 1/40 of keyspace has 50% coverage. Their experiments using the dictionaries of small fraction of the search space successfully recovered 67.6% of the passwords that is a lot higher than many previous work.

2.5 *Attack with Probabilistic Context-Free Grammar*

Dictionary attack often uses word-mangling rules, but it can be difficult to chose effective mangling rules. One approach to address this problem is to generate guesses in the order of their probability to be user passwords. This would increase the likelihood of cracking the target password in a limited number of guesses. The basic idea of this approach is to estimate the probability of the user passwords from a training set, a set of disclosed real passwords, and create a context-free grammar to be used to estimate the likelihood of the formation of a string [21, 23].

A probabilistic context-free grammar is defined as $G = (\boldsymbol{V}, \boldsymbol{\Sigma}, T, \boldsymbol{P})$, where $\boldsymbol{V}$ is a finite set of non-terminals (variables), $\boldsymbol{\Sigma}$ is a finite set of terminals, $T \in \boldsymbol{V}$ is the start variable, and $\boldsymbol{P}$ is a set of production rules of the form

$$\alpha \rightarrow \beta, (p)$$

where $\alpha \in \boldsymbol{V}$ is a variable, $\beta \in \boldsymbol{V} \times \boldsymbol{\Sigma}$ is a string of symbols (variables and terminals), and p is the probability associated with the production rule in such a way that $\sum_i p_i = 1$ for all productions i that have the same α.

In the case for passwords, the only variables (in addition to the start symbol T) are $\boldsymbol{L}$, $\boldsymbol{D}$, ans $\boldsymbol{S}$, representing letters, digits, and special characters. Notations $\boldsymbol{L_k}$, $\boldsymbol{D_k}$ and $\boldsymbol{S_k}$ represent consecutive k letters, consecutive k digits, consecutive k special symbols, respectively. The probability p_i of each production rule i is estimated using the training set. The probability of a sentential form (a string derived from T) is the product of the probabilities of the productions used in the derivation. An example of a derivation is

$$S \Rightarrow \boldsymbol{L_3 D_1 S_1} \Rightarrow \boldsymbol{L_3} 4 \boldsymbol{S_1} \Rightarrow \boldsymbol{L_3} 4\#$$

in which production rules $\boldsymbol{D_1} \rightarrow 4$ and $\boldsymbol{S_1} \rightarrow \#$ are used.

The pre-terminal structures (sentential forms) are ordered in decreasing probability, and dictionary words and hashes can be filled in for guessing.

3 Password Quality Measures

Analysis of password strength has been an active area for research and practice for a long time. The focus of these work is on the metrics of password strength and evaluation of these metrics. We shall survey several metrics for password quality including *complexity* that we propose here.

3.1 Entropy

Entropy is a measure of uncertainty and was a term used by Claude Shannon in his information theory [19]. He applied entropy to analysis of English text as “the average number of binary digits required per letter of the original language” [20]. The entropy H of variable X is defined as

$$H(X) = -\sum_{x} p(X = x) \log_2(X = x)$$

where $p(X = x)$ is the probability of X’s value being x.

National Institute of Standards and Technology (NIST) published *Electronic Authentication Guidelines* [3] that provided a metric for estimating password entropy as a measure of password strength. In the NIST Guidelines, entropy denotes the uncertainty of a password, expressed as number of bits. The high the entropy value, the higher uncertainty of the password, meaning that the password would be harder to guess. For a randomly selected password w of length m from a charset of size N, the entropy is defined in Eq. (5) below.

$$H(w) = \log_2(N^m) = m \times \log_2 N \tag{5}$$

It is obvious that longer passwords from larger charsets will have higher entropy values.

The NIST Guidelines gave an outline to estimate the entropy of user selected password based on the length of password, charsets, and possibility of dictionary attack. It mostly assigns additional bits to the entropy as the password’s length increases, and added certain “bonus” bits to the use of multiple charsets as well as dictionary check. The estimated entropy H is given below.

$$H = \begin{cases} +4 & \text{first character} \\ +2 & \text{for each of the 2nd to 8th characters} \\ +1.5 & \text{for each of the 9th to 20th characters} \\ +6 & \text{use uppercase and non-alphabetic characters} \\ +6 & \text{not in dictionaries} \end{cases}$$

For the last two bonus bits, there may involve other composition rules and properties of dictionary check.

3.2 Password Quality Indicator

Most users are aware of dictionary attack and avoid using dictionary words for passwords. However, users want passwords easy to remember, so they tend to use a word and make small changes to it. With this consideration, methods of dictionary attack also adopted various word-mangling rules to match a password with words in dictionaries. So, the strength of a password not only considers if the password is in dictionaries, but also should measure how easy (or hard) to correct the "spelling errors" in the password so it can match some words in the dictionaries. This is a commonly measure as a linguistic distance between the password and a dictionary word. The very basic linguistic distance is the *Levenshtein distance* (or edit distance) that is the minimum number of editing operations (insert, delete, and replace) needed to transfer one word to another. This idea was used in the *password quality index* (PQI) metric proposed in [13] and refined in [14].

The PQI of a password w is a pair $\lambda = (D, L)$, where D is the Levenshtein distance of w to the base dictionary words, and L is the effective length of p, which is defined as

$$L = m \times \log_{10} N \tag{6}$$

where m is the length of w and N is the size of the charset where the characters of w is drawn from.

The effective length is the length calculated in a "standardized" charset, the digit set $\boldsymbol{D}$ given in Table 1. The idea behind this is that a password (e.g. `k38P` of length 4) drawn from multiple charsets ($\boldsymbol{D} \cup \boldsymbol{L} \cup \boldsymbol{U}$) is just as hard to crack (or, has about the same number of possible candidates to crack) as another password (e.g. `378902` of length 6) from only the digit set $\boldsymbol{D}$.

It is seen that the effective length of a password given in (6) is essentially the same as the entropy value in (5), just off by a constant factor $\log_2 10$. Both consider the password's length as well as the size of the charset.

With the PQI measure, The quality criterion given in [13] states that a password is of good quality if $D \geq 3$ and $L \geq 14$.

3.3 Password Strength Meters by Service Vendors

Service vendors use various meters to measure password's strength with somewhat different algorithms. A good survey and analysis of the meters were given in [4] that listed the basic requirements at these vendors, partly shown in Table 2.

Some of these vendors also use user information (such as surname) in their classification of password quality. Most of these meters are based on the traditional LUDS requirements (**u**ppercase and **l**owercase letters, **d**igits, and **s**pecial chararacters), except Dropbox.

Table 2 Password requirements at various vendors [4]

Service	Strength scale	Length limits		Charset required
		Min	Max	
Dropbox	Very weak, Weak, So-so, Good, Great	6	72	∅
Drupal	Weak, Fair, Good, Strong	6	128	∅
FedEx	Very weak, Weak, Medium, Strong, Very strong	8	35	1+ lower, 1+ upper, 1+ digit
Microsoft	Weak, Medium, Strong, Best	1	–	∅
Twitter	Invalid/Too short, Obvious, Not secure enough Could be more secure, Okay, Perfect	6	> 1000	∅
Yahoo!	Weak, Strong, Very strong	6	32	∅
eBay	Invalid, Weak, Medium, Strong	6	20	any 2 charsets
Google	Weak, Fair, Good, Strong	8	100	∅
Skype	Por, Medium, Good	6	20	2 charsets or upper only
Apple	Weak, Moderate, Strong	8	32	1+ lower, 1+ upper, 1+ digit
PayPal	Weak, Fair, Strong	8	20	any 2 charsets[a]

[a]PayPal counts uppercase and lowercase letters as a single charset

Dropbox's password strength checker, called *zxcvbn* [24], uses a different approach to estimate the strength of a password. The basic idea is to check "how common a password is according to several sources." The sources include common passwords in leaked password sets, common names from census data, and common words in Wikipedia. The *zxcvbn* algorithm finds patterns (sub-strings) in a password that match items in the sources, and these patterns may overlap within the password. The patterns include token (`logitech`), reversed (`DrowssaP`), sequence (`jklm`), repeat (`ababab`), keyboard (`qAzxcde3`), date (`781947`), etc. It then assigns a guess attempt estimation to each match, and finally searches for non-overlapping adjacent matches that cover the password and has the minimum total guess attempt.

Table 3 Password multi-checker output for `password$1` [4]

Service	Strngth score	
Apple	Moderate	2/3
Dropbox	Very weak	1/5
Drupal	Strong	4/4
eBay	Medium	4/5
FedEx	Very weak	1/5
Google	Fair	3/5
Microsoft (v3)	Medium	2/4
PayPal	Weak	2/4
Skype	Poor	1/3
Twitter	Perfect	6/6
Yahoo!	Very strong	4/4

The algorithms used by these vendors resulted in very diverse strength scores for the same passwords. For example, as illustrated in [4], the password `password$1` scored from very weak to very strong, given in Table 3.

3.4 *Password Complexity Metric*

We propose a password *complexity* metric that considers both the common LUDS requirements and the patterns in the password. As in many other measures, the number of different charsets used in a password still plays an important role in this metric. In addition, we consider other factors that may make a password harder to guess, such as mix of same-charset-substrings, position of special symbols, and substrings in the dictionary.

As the users are aware of using different charsets to compose passwords, they are likely to put characters from the same charset together rather than mixing the them up. For example, it is more likely to have a password `horse743` instead of `ho7r4se3`. The latter is considered more "complex" than the former and harder to crack. We find substrings in a password that are from the same charset, and count the number of such substrings. Using the same example, the number of substrings in `horse743` is 2 (`horse` and `743`), while the number of substrings in `ho7r4se3` is 6 (`ho`, `7`, `r`, `4`, `se`, and `3`). However, this number may be higher for a longer password than a shorter one, so we take the ratio of this number vs the password's length as a factor to our metric.

Another factor is the position of special symbols. Many users use a common word and then add a special symbol at the end (or beginning). For example, `horse#` may be more common than `hor#se`. We apply a small penalty to this pattern if the password uses only 2 charsets (including the special symbol charset).

A password that matches a dictionary word is weak, but a password with a substring that matches a dictionary word may or may not be weak, depending on the length of the substring itself and its "weight" in the password. Many user-selected passwords contain substrings that are dictionary words but the passwords may be good. For example, `Pfan?6tk` is pretty strong by most of the measures we discussed, although it contains a dictionary word `fan`. However, a longer dictionary word (4 letters or more) in a password would make it weaker, particularly for a relatively shorter password. The password `Wfoot67` is a lot weaker that `Wdfoot6237`, with both containing the dictionary word `foot`.

With these considerations, our complexity metric of password w is calculated as given in Eq. (7).

$$C(w) = \begin{cases} 0 & \text{if } l < 4 \\ n + (k/l) + s - p - (d/l) & \text{if } l \geq 4 \end{cases} \tag{7}$$

where n is the number of charsets in w, k is the number of same-charset-substrings, l is the length of w, s is a bonus if w has special characterize, p is special character position penalty, and d/l is in-dictionary penalty. In particular, $s = 0.5$ if w contains special character, 0 otherwise; $p = 0.5$ if a special character is at the beginning or the end of w and w has no more than 2 charsets, 0 otherwise; d is the length of substring that is a dictionary word.

We may scale the metric up from the range of 0–10 to 0–100 (by a factor of 10) so it can be reasonably compared with other measures such as NIST entropy.

4 Experiments and Analysis

We conducted some preliminary experiments to compare these password strength measures. Several datasets were used in our experiments. First, we calculated some statistics of the relationships between the measures. Then, we applied clustering algorithm to passwords in the r-dimensional metric space where r is the number of metrics to see if the passwords are well grouped in strength-clusters. Finally, we tried to crack some of the passwords to confirm the relationship between the strength and the level of difficulty to crack them.

4.1 Data Sets

Three data sets were used in our experiments as explained in Table 4. The data sets D_1 and D_2 were primarily for correlation analysis, whereas D_3 was used to practice password cracking to illustrate the strength of the passwords.

4.2 Statistical Analysis of Strength Metrics

Two statistics, *Pearson correlation* and *maximum information coefficient* (*MIC*) were calculated to show the relationships between the strength measure. MIC was introduced in [18] as a new exploratory data analysis indicator that measures the strength of relationship between two variables. This measure is a statistic score between 0 and 1. It captures a wide range of relationships and is not limited to specific function types (such as linear as Pearson correlation does).

We calculated the Pearson correlation and MIC on four measures (entropy, NIST entropy, Levenshtein distance, and complexity). The pairwise results for data set D_1 and D_2 are given in Table 5(a) and (b), respectively.

It shows that the entropy, NIST entropy, and complexity have high MIC scores, indicating that the three measures are closely related and not very independent. Their

Table 4 Data sets used in experiments

Data	Size	Description of passwords in the set
D_1	10,000	Most commonly used [9]
D_2	642	Mixed user-elected and randomly generated
D_3	127	Manually selected with various composition patterns

Table 5 MIC and Pearson coefficients between measures

(a) for data set D_1

Measure 1	Measure 2	MIC	Pearson Correlation
Entropy	NIST entropy	0.79863	0.72584134
Entropy	Complexity	0.84558	0.08722651
NIST entropy	Complexity	0.89118	0.40052018
Levenshtein	Entropy	0.26666	−0.00422713
Levenshtein	NIST entropy	0.33358	0.52798074
Levenshtein	Complexity	0.48328	0.56144760

(b) for data set D_2

Measure 1	Measure 2	MIC	Pearson correlation
Entropy	NIST entropy	0.93002	0.88889660
Entropy	Complexity	0.90081	0.80279726
NIST entropy	Complexity	0.89208	0.90321730
Levenshtein	Entropy	0.70443	0.74756646
Levenshtein	NIST entropy	0.69875	0.84680110
Levenshtein	Complexity	0.66447	0.78521204

MIC scores with Levenshtein measure, though, are a lot lower indicating that it is relatively independent to the other three measures. This may confirm the claim in PQI [13] that Levenshtein distance of the password and words in the dictionaries may indeed be an alternative (and independent) to the LUDS requirements. It is interesting to note that the linear correlation between entropy and complexity is close to 0, so is the correlation between entropy and Levenshtein distance.

The p-value of the MIC statistic for the data set of size $n \geq 600$ with $MIC \geq 0.2257$ is very small ($p = 0.00001281 \pm 10^{-6}$ with $\alpha = 0.05$), indicating that the correlation of any pair of measures is significant. Here 0.2257 is the largest MIC value in the p-value table that is smaller than 0.26666, the smallest in Table 5(a).

On data set D_2, the linear correlation and the MIC scores are much higher. This may be due to composition of the patterns in the passwords in this data set.

From the correlation analysis, we see that most of these measures (entropy, NIST entropy, complexity) are highly correlated. This may lead to the conclusion that the password strength classifications using any of these measures will not differ much. The Levenshtein distance of a password from dictionary words, however, is a different measure (less closely correlated to the other measures), that may be used for estimating password's strength along with other measures, ad did in PQI [13].

4.3 Password Strength Test

To further evaluate the password strength metrics, we experimented cracking the passwords in the small data set D_3 with 127 passwords that are manually selected to include strengths as various levels.

4.3.1 Lookup Table and Rainbow Table Attack

We used two online password cracking sites to uncover the passwords (given their MD5 hashes) in the data set D_3. One site is CrackStation that "uses massive precomputed lookup tables to crack password hashes" [5]. The other is HashKiller [11] that also uses very large hash databases to crack password hashes. The sizes of the lookup table databases of the two sites are listed in Table 6 (a), and the results are given in Table 6 (b).

All the 17 MD5 hashes that HashKiller failed also failed by CrackStation. HashKiller was more successful than CrackStation simply because it has larger hash databases. These are hashes of passwords that were randomly generated (like `nXXdHtt6Q`, `2y!3e)!%)`, `fwtC9xcO`, etc.). The average strength metrics of the passwords are given in Table 6 (c). It is clearly shown that the passwords the sites failed to crack are those with higher strength than those cracked passwords on all the measures. It is also seen that HashKiller was able to crack passwords with higher strength than CrackStation.

4.3.2 John the Ripper Attack

John the Ripper (JtR) [17] is an open source software for cracking passwords. It combines dictionary attack, rainbow table attach, and brute force attack, with various rules for word composition. The free version of JtR comes with a very small dictionary (password list) containing only 3,546 words. We used JtR to crack some passwords in the data set D_3. The time spent on some of these passwords is given in Table 7. The last two passwords in the table took too long to crack and we aborted JtR before the passwords were recovered.

Figure 3 shows plots of various strength measures vs time spent by JtR to crack the passwords.

Since the default dictionary of JtR is very small and we did not use an additional dictionary, only 6 passwords (or slightly changed version of the passwords by some transformation rules in JtR) found in the dictionary very quickly. The other passwords were cracked using the brute-force approach and consumed from 10 min to 1.5 h for each password. From Fig. 3, we can see that the strength metrics entropy and complexity have positive correlations with the time spent by JtR, and the Levenshtein distance measure is even more closely correlated to the time needed to crack the passwords.

Table 6 Crack passwrods on CrackStation and HashKiller

(a) Database sizes

Hash	CrackStation	HashKiller
MD5	190 GB, 15-billion entries	829.7 GB
SHA1		312.0 GB
SHA256	19 GB, 1.5-billion entries	
NYLM		312.0 GB

(b) Cracking results

# MD5 Hashes	CrackStation		HashKiller	
	cracked	%	cracked	%
127	58	45.7%	110	86.6%

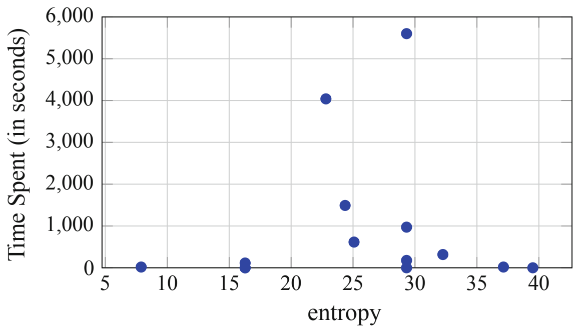
(c) Strength metrics of un-cracked and cracked passwords

Measure	CrackStation		HashKiller	
	un-cracked	cracked	un-cracked	cracked
entropy	35.75	23.73	40.02	28.76
NIST entropy	27.84	19.11	31.56	22.66
complexity	38.33	13.53	42.92	24.54
Levenshtein	4.62	1.53	5.94	2.79

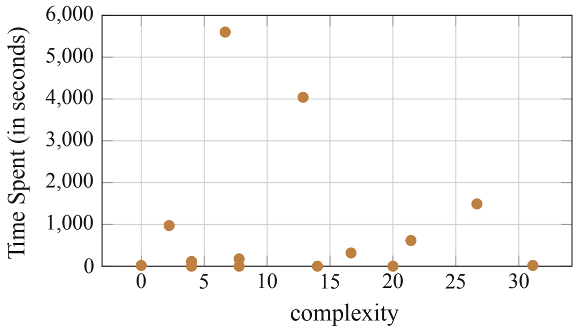
Table 7 Time spent on cracking some passwords in D_3 using JtR

Id	Password	MD5	Entropy	NIST	Complexity	Levenshtein	JtR (h:m:s)
1	kitty	`cd880b726e0a0dbd4237f10d15da46f4`	16.29	12.00	4.00	0.00	0:00:00
2	susan	`ac575e3eecf0fa410518c2d3a2e7209f`	16.29	18.00	14.00	1.00	0:00:00
3	jellyfish	`9787922f286a4349a009237d0b2ffd73`	29.32	19.50	2.22	0.00	0:16:12
4	smellycat	`5a7c5156c0f64867b607abf66f3bbe1f`	29.32	22.17	6.67	3.00	1:33:20
5	allblacks	`6186e17cdef7a72b6b44c374b7ade779`	29.32	22.83	7.78	4.00	0:02:56
6	usher	`0d5e410c96a560d494fe2b3485f5f864`	16.29	12.00	4.00	0.00	0:01:54
108	seeyou6	`e77c84530ec4d274f0af2e7dc158c434`	25.08	19.43	21.43	3.00	0:10:15
109	funlo_	`9fd0d44ade138efcb67a69c68b006f08`	24.36	17.00	26.67	3.00	0:24:15
110	Yt	`b6cc855441c4c688509ab84a34722868`	7.90	12.00	0.00	1.00	0:00:18
111	Password1	`2ac9cb7dc02b3c0083eb70898e549b63`	37.14	28.17	31.11	1.00	0:00:19
119	billabong	`c61a71b89ce304b78bce33cb56f9cffa`	29.32	22.83	7.78	3.00	0:00:00
120	Basketball	`c71a18083d9e74b4a5c5d8d9a17d68d0`	39.51	24.60	20.00	0.00	0:00:00
121	phoenix09	`6aed5d571daa97833033534a4ac2c27a`	32.25	20.83	16.67	2.00	0:05:17
126	akjuwfg	`015c09cba7ac3cdfe01e2489e2b8ad14`	22.81	22.00	12.86	4.00	1:07:20
18	oQ4cf5BDA	`3567b3ab8dbbbc1f091e284caaf7976b`	41.27	33.00	42.00	6.00	
127	D$f9	`ecad9ace9534cdb71d04a600960ca3c4`	18.17	22.00	65.00	2.00	

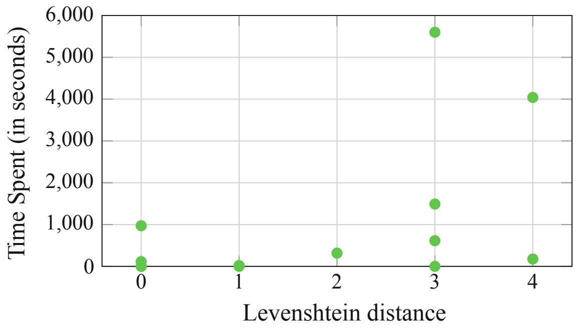
Fig. 3 Time spent by JtR



(a) Entropyvstimespent



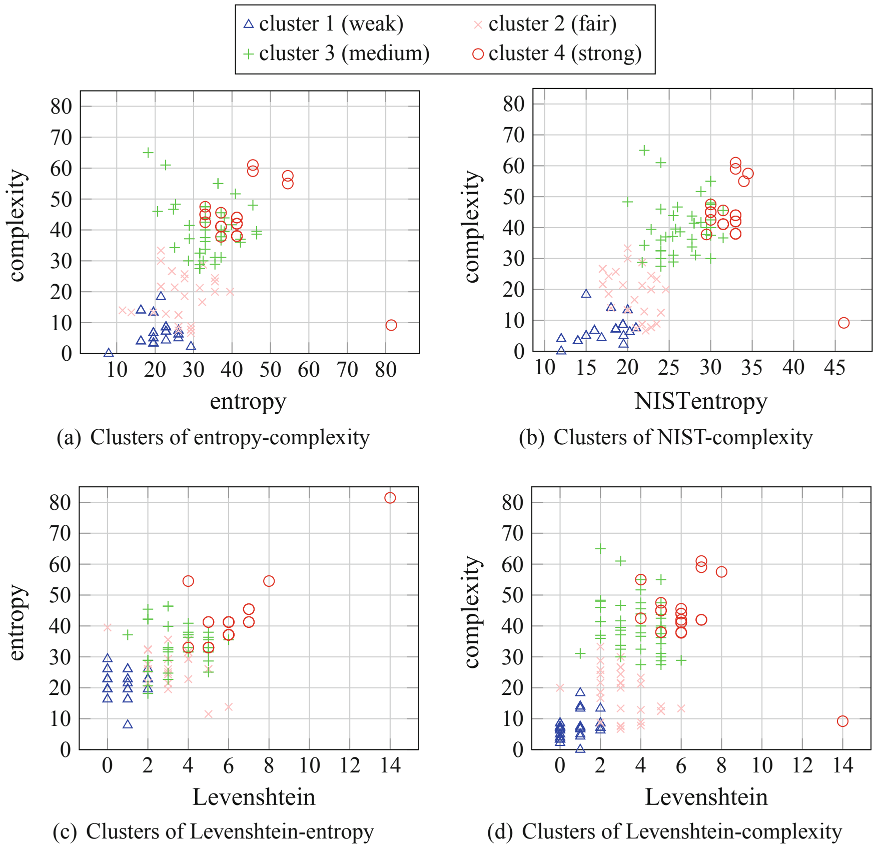
(b) Complexity vs time spent



(c) Levenshtein distance vs time spent

Table 8 Clutering result

Number of iterations:			24
Within cluster SSE (sum of squared errors):			7.23
Clustered instances	cluster	# instances	%
	1	33	26%
	2	30	24%
	3	42	33%
	4	22	17%



(a) Clusters of entropy-complexity

(b) Clusters of NIST-complexity

(c) Clusters of Levenshtein-entropy

(d) Clusters of Levenshtein-complexity

Fig. 4 Clustering plots on pair-wise strength metrics

4.4 *Clustering of Passwords*

As another evaluation of the strength metrics is to do clustering of the passwords to see if the weak passwords are grouped together and the strong ones are grouped in another cluster. In addition to the 4 variables (entropy, NIST, complexity, and Levenshtein distance), we also used two more variables when we applied the clustering algorithm. The two variables are *effective length* and a variation of the Levenshtein distance, which calculates the Lavenshtein distance of sub-words in a password against dictionary words, rather than treating the password as a single word.

We applied the K-means clustering algorithm [10] on the 127 passwords in the data set D_3 with the 6 variables into 4 clusters representing four strength levels. The results is summarized in Table 8.

The algorithm randomly selected 4 passwords (with ids 29, 35, 38, and 44) as the initial centroids, ran through 24 iterations and stabilized in 4 clusters with the within-cluster SSE (sum of squared errors) 7.23.

The projections of the 6-D space onto 2-D of some pairs of the metrics are shown in Fig. 4a–d. Note that the number of points of a cluster in the plot may appear smaller that the number in Table 8. For example, there are 22 instances in cluster 4, but only 10–14 cluster-4 points in the plots. This is because of 6-D to 2-D projection and multiple points were mapped to the same spot in the 2-D space. From theses figures, we can visually see that the passwords were reasonably clustered into strength groups, from weak to strong.

5 Related Work

A lot work has been done in assessing the quality of passwords. As mentioned before that a good survey of password meters used by service vendors was given in [4]. Florêncio and Herley studied a set of half-million user passwords (as well as other information about the user accounts) on various web sites to find the characteristics about the passwords and their usages [7]. Kelley et al. analyzed 12,000 passwords and developed methods for calculating the time needed for several password-guessing tools to guess the passwords so that the password-composition policies were better understood [12]. Dell'Amico et al. conducted an empirical analysis of password strength [6], in which the authors analyzed the success rate (number of cracked passwords versus search space) using different tools including brute force, dictionary attack, dictionary mangling, probabilistic context-free grammars [23], and Markov chains [15]. They concluded that the probability of success guessing a password at each attempt decreases roughly exponentially as the size of search space grows. An extension of the probabilistic context-free grammar approach was given in [26] that tries to find new classes of patterns (such as keyboard and multi-word) from the training set. Weir et al. tested the effectiveness of using entropy (NIST) as a measurement of password strength [22]. Their experiments on password cracking techniques on several large data sets (largest had 32 million passwords) showed that the NIST entropy does "not provide a valid metric for measuring the security provided by password creation policies."

Traditional advice to the users to select their passwords does not provide additional security, argued by Florêncio et al. [8]. They pointed out that relative weak passwords are sufficient to prevent brute-force attack on an account if "three strikes" type rule applies. Thus, making password stronger does little to address the real threats. Bonneau used statistical guessing metrics to analyze a large set of 70 million passwords [1]. The metrics is to compare the password distribution with uniform distribution "would provide equivalent security against different forms of guessing attack." The study found little variation in the password distribution that produces "similar skewed distributions with effective security varying by no more than a few bits." Bonneau and his colleagues pointed out the gaps between the academic research on password strength evaluation and the authentication methods used in the reality of the web environment today [2].

The probabilistic context-free grammar method proposed in [23] appears quite effective. A series of experiments based on training sets of relatively large number of disclosed passwords were conducted showing that the approach was able to crack 28–129% more passwords than the traditional John the Ripper method.

Another interesting work about the quality of passwords was given in [25]. The authors conducted psychological and statistic experiments with 288 students regarding their selections of passwords and memorability of these passwords. The students were divided into three groups: *control* group (traditional advice—at least 7 characters long and at least one non-letter), *random* group (random pick 8 characters with eyes closed), and *pass phrase* group (select password based on mnemonic phrases). The experiments lasted for several months (to test memorability) and various methods were used to attack the passwords. The results of their experiments showed that random passwords are hard to remember; passwords based on mnemonic phrases are harder to guess, just as strong an random passwords and as easy to remember as naively selected ones.

6 Conclusion

In this paper we reviewed the basic metrics for assessing password strength, including entropy, NIST entropy, password quality index, and Lavenshtein distance, as well as some password quality metrics developed at some popular service vendors. We proposed a password complexity metric that considers the composition patterns in a password in addition to the LUDS criteria. The correlations between these measures were analyzed using the maximum information coefficient (MIC) and Pearson coefficient. The resulting statistics show that most of the metrics are closely correlated except Levenshtein distance that may be a good measure for password quality besides the traditional parameters (length of password and size of charset).

There password strength metrics were evaluated by experiments that tried to crack the hashes of a small set of passwords to see if the difficulty of cracking a password is indeed related to the strength measures. The cracking tools used including techniques like brute force, transformation rules, dictionary attacks, and massive table look-up. Two types of results were obtained: password found or not (table look-up), and the length of time spent to discover the password (other techniques used in JtR). Results showed that the level of success cracking the passwords is highly positively related to the strength measures. It is pretty convincing from our experiments that the strength metrics are valid and can be used to assess the quality of user selected passwords. This was further validated by the clustering results.

The bottom line of our analysis comes back to the simple advice to users: select a long password with various kinds of characters (lower and uppercase letters, digits, and symbols), as the length of password and size of charset being the two most critical parameters to the strength of the password in all the metrics we studied.

References

1. Bonneau, J.: The science of guessing: analyzing an anonymized corpus of 70 million passwords. In: 2012 IEEE Symposium on Security and Privacy, pp. 538–552. IEEE (2012)
2. Bonneau, J., Herley, C., van Oorschot, P.C., Stajano, F.: Passwords and the evolution of imperfect authentication. Commun. ACM **58**(7), 78–87 (2015)
3. Burr, W.E., Dodson, D.F., Newton, E.M., Perlner, R.A., Polk, W.T., Gupta, S., Nabbus, E.A.: Draft NIST special publication 800–63-2: electronic authentication guideline. US Department of Commerce, National Institute of Standards and Technology (2013)
4. de Carné de Carnavalet, X., Mannan, M.: From very weak to very strong: analyzing password-strength meters. In: Network and Distributed System Security Symposium (NDSS 2014). Internet Society (2014)
5. Defuse Security: Crackstation: Free password hash cracker. https://crackstation.net/
6. Dell'Amico, M., Michiardi, P., Roudier, Y.: Password strength: an empirical analysis. In: 29th IEEE International Conference on Computer Communications, pp. 983–991 (2010)
7. Florencio, D., Herley, C.: A large-scale study of web password habits. In: Proceedings of the 16th International Conference on World Wide Web, pp. 657–666. ACM (2007)
8. Florêncio, D., Herley, C., Coskun, B.: Do strong web passwords accomplish anything? In: 2nd USENIX Workshop on hot Topics in Security (2007)
9. GitHub: 10k most common passwords. https://github.com/danielmiessler/SecLists/blob/master/Passwords/10k_most_common.txt
10. Han, J., Pei, J., Kamber, M.: Data mining: concepts and techniques, 3rd edn. Elsevier (2012)
11. HashC: Hash killer. https://www.hashkiller.co.uk/
12. Kelley, P.G., Komanduri, S., Mazurek, M.L., Shay, R., Vidas, T., Bauer, L., Christin, N., Cranor, L.F., Lopez, J.: Guess again (and again and again): measuring password strength by simulating password-cracking algorithms. In: 2012 IEEE Symposium on Security and Privacy (SP), pp. 523–537. IEEE (2012)
13. Ma, W., Campbell, J., Tran, D., Kleeman, D.: A conceptual framework for assessing password quality. Int. J. Comput. Sci. Netw. Secur. **7**(1), 179–185 (2007)
14. Ma, W., Campbell, J., Tran, D., Kleeman, D.: Password entropy and password quality. In: 4th International Conference on Network and System Security (NSS), pp. 583–587. IEEE (2010)
15. Narayanan, A., Shmatikov, V.: Fast dictionary attacks on passwords using time-space tradeoff. In: Proceedings of the 12th ACM Conference on Computer and Communications Security, pp. 364–372. ACM (2005)
16. Oechslin, P.: Making a faster cryptanalytic time-memory trade-off. In: Annual International Cryptology Conference, pp. 617–630. Springer (2003)
17. Openwell: John the Ripper password cracker. http://www.openwall.com/john/
18. Reshef, D.N., Reshef, Y.A., Finucane, H.K., Grossman, S.R., McVean, G., Turnbaugh, P.J., Lander, E.S., Mitzenmacher, M., Sabeti, P.C.: Detecting novel associations in large data sets. Science **334**(6062), 1518–1524 (2011)
19. Shannon, C.E.: A mathematical theory of communication. Bell Syst. Tech. J. **27**, 379–423 (1948)
20. Shannon, C.E.: Prediction and entropy of printed english. Bell Labs Tech. J. **30**(1), 50–64 (1951)
21. Weir, C.M.: Using probabilistic techniques to aid in password cracking attacks. Ph.D. thesis, Florida State University (2010)
22. Weir, M., Aggarwal, S., Collins, M., Stern, H.: Testing metrics for password creation policies by attacking large sets of revealed passwords. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, pp. 162–175. ACM (2010)
23. Weir, M., Aggarwal, S., De Medeiros, B., Glodek, B.: Password cracking using probabilistic context-free grammars. In: 30th IEEE Symposium on Security and Privacy, pp. 391–405. IEEE (2009)
24. Wheeler, D.L.: zxcvbn: low-budget password strength estimation. In: Proceedings of the 25th USENIX Security Symposium, pp. 157–173 (2016)

25. Yan, J., Blackwell, A., Anderson, R., Grant, A.: Password memorability and security: empirical results. IEEE Secur. Priv. **2**(5), 25–31 (2004)
26. Yazdi, S.H.: Probabilistic context-free grammar based password cracking: attack, defense and applications. Ph.D. thesis, Florida State University (2015)