6.857 Final Paper

{jimmy42, rliu42, h1212}@mit.edu

May 18, 2017

1 Introduction

Passwords are an essential part of everyday life. From email to games, from online shopping to bank accounts, it seems every website and app these days requires a new account, and with it, a new password. How do people remember all of these passwords? Some people simply use the same password for every website, but this is extremely risky because a compromised account on one website means that all of your accounts are compromised. In the 2013 Yahoo! data breach alone, over *1 billion* passwords were compromised.

In the United States, an average user has a shocking 130 online accounts [17]! That's 130 passwords to remember, if you use a different password for each site. Unable to remember all of these passwords, users typically resort to writing down their passwords, storing them in their computer via "remember password", or resetting their passwords via email. The first two options are insecure because a thief who steals your wallet (assuming you store your written passwords there) or your laptop can gain access to all of your accounts. Being able to reset passwords via email just links all of your accounts to your email address, which means that the security of each account is only as strong as the security of your email account.

In this paper, we analyze the challenge of having secure passwords for a large number of accounts. Our contribution is to first propose a framework for evaluating the usability, memorability, and security of password schemas. Then, we develop a novel scheme that allows users to securely compute different passwords for different websites. Finally, we use the frameworks we developed to analyze our proposed scheme and compare it against existing password schemas.

2 Existing References

Researchers have studied password hashes for the server side to store passwords securely. And techniques for creating and even memorizing a single strong passwords have been studied and proposed. [1] However, the challenge of creating and memorizing multiple secure passwords for tens of accounts a person normally has is not addressed enough by these researches. For example, a bio-metric authentication satisfies the requirement of being secure, with a large password

space, and being easy to remember, with minimum effort to recall. But it doesn't solve the challenge of security across multiple sites. As each site can be vulnerable of password leakage, the security analysis of a multiple accounts password scheme must address the possibility of remaining secure even after the adversary have gained access to valid account-password pairs.

To address the hardness of remembering long passwords that consist of letters and digits, [2] proposed a scheme of using a signature-based user identification system. Based on studies in cognitive science [3], humans remember pictures far better than texts and numbers. Therefore, drawing a signature would lower the burden of remembering the password. [4] also proposed a similar scheme, together with a carefully designed metric to assess the security of a drawing by evaluating the amount of information it contains. Such a metric is useful for comparing the relative of security different drawings, but doesn't provide the absolute security information against a real world adversary. Although graphical based passwords are easier to remember than the normal text based ones, users that use multiple different graphical passwords are still more likely to fail [5].

[1] also addresses the challenge of remembering a password, by converting a random assigned bit string to a set of easy to remember English words. It evaluates the resulting techniques by conducting user studies instead of using a formalized model. [6] evaluates usability by formalizing the number of rehearsals required for remembering a password, and also proposes a security game that accounts for the scenario that the adversary has gained access to some valid account-password pairs. [7] further includes the amount of computation into the model to evaluate usability of a password scheme. It also quantitatively evaluate the security by the number of valid account-password pairs the adversary needs to successfully predict a valid password for a new account. [8] formulates yet another model to evaluate the security, and only states some constraints that a usable scheme must satisfy.

3 Human Usability Analysis Framework

Of course, as we are creating a password scheme for humans, it needs to be human usable. To determine human usability, the best way is through a randomized experiment with real human subjects. Of course, human subjects are difficult to obtain and user testing is expensive and time-consuming.

3.1 Criteria for Analysis

We determine human usability through the following criteria, partially based on [7]:

- Pre-Processsing Time
- Processing Time

- Memorability
- Ease of Change
- Subjective Goodness

We will conduct a pre-test and post-test questionaire to determine subjective goodness of the scheme, use the initial test to determine pre-processing and processing time, and use the followup test to determine memorability and ease of change.

3.2 Test Subject Selection

Given that user testing is expensive and time-consuming, our user tests will probably consist of a convenience sample drawn from our peers. We may decide to ask our friends to participate in these tests, or send out an email to dorms encouraging people to participate in our tests. Of course, participation is completely voluntary, so our results may suffer from slightly voluntary response bias. We may also opt to do the testing online rather than in person, to make it easier for users to participate (described in more detail in Testing sections below).

Because the number of sites that the user needs to remember passwords for plays a critical role in memorability and processing time for any password scheme, we will divide the user groups into four classes, 5 accounts, 10 accounts, 15 accounts, and 20 accounts. This will allow us to gather better insight into how the various metrics scale with the number of accounts.

3.3 Pre-Test Questionaire

In the pre-test questionaire, we are going to ask the following questions:

1. On a scale from 1 to 5, how good do you think this password scheme is?
2. On a scale from 1 to 5, how likely are you to use this password scheme?
3. On a scale from 1 to 5, how secure do you think this password scheme is?
4. On a scale from 1 to 5, how easy to use do you think this password scheme is?
5. On a scale from 1 to 5, how memorable do you think this password scheme is?
6. On a scale from 1 to 5, how flexible do you think this password scheme is?

We will ask the list of questions above for our password scheme, as well as one or two benchmark schemes (such as using the same password across all sites, or using different passwords for each site), in order to better calibrate responses, since different users may rate on different scales (some users may be

biased towards giving high ratings, others may be biased towards giving low ratings).

The rating for each question will be computed as

$$R' = R - B$$

where R' is the adjusted rating, R is the original rating, and B is the benchmark rating.

3.4 Initial Test

We will first give the test subject a quick overview of our project and what it is that we're testing. Then, we will give the test subject the pre-test questionaire to fill out. We will then give the user our chosen password scheme, with randomly generated parameters/keys, and measure the amount of time the user takes before they can reliably type the password for each account they have. By reliably, we mean that they can type in each account's password in succession without any errors. This will yield the pre-processing time.

Once the user has memorized the scheme and can type each account's password reliably, we will ask the user type each account's password in succession, and repeat that two times (for a total of three rounds through all accounts), to get more data. This will allow us to compute the mean and variance of the processing times.

In order to make it easier to participate (and thus have more users test our scheme), we may make the testing online and fully automated. In particular, we can create a website that tells them their password generation scheme, keeps track of the amount of time they take to get all the accounts' passwords correct in one round (pre-processing time), and records the amount of time they take for each account password in each of the three later rounds (processing time).

3.5 Followup Test

The followup test will be conducted a week after the initial test. In the followup test, we will first ask the user to recite the password scheme and its keys/parameters, to see if they actually remember the scheme itself. We will measure the time taken to recite the scheme correctly, or the amount of error if they do not recite the scheme correctly. We will then ask the user to type in the password for each account in turn, measure the amount of time taken to type the password correctly, or the amount of error if they type it incorrectly. For both the scheme and for each password, we will allow the user at most three tries to get it correct.

The empiral memorability for each password will be

$$EM_p = \begin{cases} 1 \text{ if the user gets the password correct} \\ x \in [0,1] \text{ if the user gets x amount of the password correct} \end{cases}$$

The empiral memorability of the scheme is the average empirical memorability over the passwords for each site. Of course, we can get a more accurate metric if we do multiple followup tests at different time intervals, but it would be very difficult to obtain human test subjects willing to go through that many user tests.

We will also measure the flexibility in the scheme, in terms of how easy it is to change the parameters of the scheme and thus compute a different set of passwords. This may be necessary if some passwords are compromised and you want to change the passwords for those accounts, or if your workplace requires you to change passwords periodically.

To measure the flexibility and ease of change, we will change the parameters of the scheme and measure the user's new pre-processing time on the new parameters. If the preprocessing time is less than the pre-processing time for the initial test, then the scheme is conducive to change in parameters. If the preprocessing time is same as the initial test, the scheme is neutral with respect to change. If the preprocessing time is actually greater than the initial test, the scheme is unconducive to change. In this case, it means that a user who has never seen the scheme would do better than someone who has memorized the scheme with different parameters, so it is likely that the already memorized parameters conflict with the new parameters and hinder the users' ability to memorize new parameters.

Finally, we will conduct a post-test questionaire.

3.6 Post-Test Questionnaire

In the post-test questionnaire, we are going to ask the same questions as the pre-test questionnaire, and see how the users' feedback changes after they use our password scheme in the tests. We will use a weighted average of the pre-test and post-test questionnaire ratings for each question to determine the overall rating of our password scheme in a particular dimension:

$$R_o = \frac{R'_i + 2R'_f}{3}$$

where R_o is the overall rating, R'_i is the adjusted initial rating, and R'_f is the adjusted final rating.

4 Memorability Analysis Framework

Our framework for characterizing the memorability and human-computability of multi-account password generation schemas focuses on three main properties:

(i) Amount of *pre-processing time* needed to learn the scheme:

– How long does it take to commit all necessary information to long-term memory, so that it may be recalled accurately and reliably?

– How long does it take to memorize the set of instructions needed to generate a new password?

(ii) Amount of processing time or *computing effort* need to generate new passwords:

– After the private key and set of instructions for the password generation scheme is learned, how long does it take to generate a password for a new account?

(iii) *Self-rehearsability*:

– In the process of computing passwords for commonly used accounts, is each discrete instruction of the password generation scheme rehearsed often enough?

– How easy is it to recall instructions or character mappings from long-term memory that are less frequently used?

Metrics for each of these properties will be quantified in normalized units, and weighted to generate an overall memorability score for a password scheme.

4.1 Pre-processing Time (Long-term Memory Burden)

Pre-processing time is a metric for how much effort is required for a user to memorize all of the private "key" information (e.g. a character/digit mapping for each letter of alphabet), as well as the set of (published) instructions specified by the password generation scheme. All of this data must be rehearsed and committed to the user's long-term memory before the password generation scheme can be used successfully.

Let I be the number of distinct *instructions* (concretely, these could be arithmetic operations) specified in a password generation scheme. Note that looped or recursively defined instructions should not be multiply-counted, as long as they can be conceptualized in the human user's mind as a single operation that gets repeated. For example, I might be estimated by counting the number of lines needed specify the password generation scheme in a standardized pseudocode language.

Let D be the number of characters/digits required to memorize the user's private key. Concretely, this private key could be a random character mapping that the user must commit to long-term memory a priori.

Though a model might weight the difficulty of memorizing instructions (I) vs. data (D) differently, we assume that the total long-term memory information content of a password generation scheme can be equally weighted as $I + D$.

Roughly, how does the human "effort" needed to commit all this data to long-term memory scale with the information content, $I + D$? One way to produce a quantifiable metric is to estimate the expected number of trials needed before the password generation scheme is learned without error, assuming that each individual instruction or data value is misremembered or forgotten with a small probability δ during the long-term memorization (i.e. learning) phase.

If the probability of individual memorization errors is δ, and we furthermore assume that potential memorization errors occur independently from each other, then the probability that the password generation scheme is executed accurately (i.e. produces the correct password) in any single trial of the learning phase is:

$P_{success} = (1-\delta)^{(I+D)}$. Note that all instructions and private data values must be recalled correctly for the password generation scheme to succeed.

Hence, the expected number of trials – which serves as a measure of *long-term memory burden*, or "human pre-processing effort" – needed to accurately learn a new password generation scheme is modeled as:

$$\mathbf{PT} = 1/P_{success} = (1-\delta)^{-(I+D)},$$

where δ can be either an arbitrarily chosen small constant, or estimated empirically. A higher **PT** score thus indicates that a password scheme is less desirable in this human memorability framework.

4.2 Computing Time (Short-term Memory Burden)

Once the user has committed the password generation scheme's published instructions as well as his/her private key data to long-term memory so that it may be recalled accurately and reliably without error, the task then becomes to actually apply the memorized scheme to generate passwords for multiple accounts.

For our human computability model, we adopt the assumption, which is one often referenced psychology and neuroscience, that a person's *memory span* (also known as digit/character span) is $\sim$5 items long. Functionally, short-term memory span measures the number of discrete units over which a user can successively distribute his attention and still organize them into a working unit.

We thus analyze the computability of a password scheme – its short-term memory burden – as the number of "entities" (concretely, these will be digits, characters, or arithmetic operations) that must be streamed into the user's short term memory span, with the restriction that the memory span cannot store more than 5 items at any point during the computation. If executing an instruction of the password generation scheme would cause the number of items stored in the user's memory stream to exceed the memory span limit of 5, then a value already stored in the memory stream must be discarded (if that value is needed later, it must be recomputed, exhausting more stream characters).

One might imagine the analogy of computing with a fixed-size register of 5 bytes on a computer to conceptualize this framework.

The metric that models the expected *computation time* (**CT**), or short-term memory burden, of a scheme will be calculated as the total number of memory stream characters needed to compute a password, averaged a across standard collection of common online accounts (e.g. of the top 50 most frequently used Internet sites):

$$\mathbf{CT} = \frac{\sum_{a \in \mathcal{A}} MS_5(a)}{|\mathcal{A}|},$$

where $\mathcal{A}$ is a representative set of challenges (i.e. account names like $\{$GMAIL, AMAZON, FACEBOOK, $\ldots\}$), and $MS_i(a)$ is the number of of memory stream characters needed to generate

a password for account name a with memory span limit i (we will use $i = 5$ for our analyses).

A higher **CT** score thus indicates that a password scheme is less desirable in terms of computational burden.

4.3 Self-rehearsability

Another desirable property of human-usable password generation schemes that we include in our memorability analysis is self-rehearsability. This metric attempts to characterize how easy is it to recall instructions or character mappings from long-term memory that are less frequently used.

For a scheme with high self-rehearsability, we desire that during the process of computing passwords for commonly-used accounts, each discrete instruction of the password generation scheme is rehearsed often enough so as to be more easily recalled from long-term memory when required for less frequently used accounts.

For a given password generation scheme g, let its instruction set be $\mathcal{I}_g$. For each distinct instruction $i \in \mathcal{I}_g$, let the variable $X_a(i)$ denote the number of times instruction i needs to be executed to compute the password for account a.

Similarly, for each data value $d \in D$ comprising the user's private key, let $X_a(d)$, be the number of times that d needs to be accessed in the process of computing the password for account a.

We will define

$$\mathbf{SR}_g = \text{stdev}(\{X_a(i)\}_{a \in \mathcal{A}, i \in \mathcal{I}_g}) + \text{stdev}(\{X_a(d)\}_{a \in \mathcal{A}, d \in D}),$$

with a sampled from a representative set of account names $\mathcal{A}$. Note that the stdev's (standard deviations) of the variables $X_a(i)$ and $X_a(d)$ are computed across all instructions $i \in \mathcal{I}_g$ of the password generation scheme's instruction set, and all data values $d \in D$ of the user's private key, respectively.

The standard deviations of the variables $X_a(i)$ and $X_a(d)$ will be low if all instructions and data values are utilized roughly equally across different account names, hence enhancing self-rehearsability. Conversely, the standard deviations will be high if certain instructions or data values are used significantly more frequently than others, hence deterring self-rehearsability.

A higher **SR** score thus indicates that a password scheme is less desirable in terms of self-rehearsability.

4.4 Aggregating the Metrics

An aggregate metric for the memorability burden of a password generation scheme g, will be calculated as the weighted sum of (i) **PT** (pre-processing time, or long-term memory burden); (ii) **CT** (computation time, or short-term memory burden); and (iii) **SR** (self-rehearsability burden). For example, suppose we value low short-term memory burden in a password generation scheme

twice as much as the other metrics. We might choose an aggregate memorability burden metric of:

$$M_g = c_{PT} \cdot \mathbf{PT}_g + c_{CT} \cdot \mathbf{CT}_g + c_{SR} \cdot \mathbf{SR}_g,$$

where $c_{PT} = 0.25$, $c_{CT} = 0.50$, and $c_{SR} = 0.25$. M_g measures the memorability *burden* of the password generation scheme g, so higher values indicate less desirable schemes.

5 Cryptographic Security Analysis Framework

To account for the challenge of creating passwords for multiple sites, the security analysis of the schemes consider the following two attacks: offline attack, similar to the definition in [6], and random challenge attack, as defined in [8].

In the offline attack model, the adversary is given only the hash h_p of the user's password p, and the hash function h, such that $h(p) = h_p$. We will measure the security in offline attack model by the number of guesses q the adversary needs to recover the user's password p', such that $h(p') = h_p$. This model measures how strong a single password produced by the analyzed scheme is.

In the random challenge attack model, the adversary is given the password scheme, f, all the stored information (i.e. not memorized information) for computing a password, and Q randomly sampled pairs of scheme input c_i, and scheme output $p_i = f(c_i)$. We will measure the security in random challenge attack model by the number of random samples Q for the adversary to successfully recover a password p' for a new random input c'. This model measures how robust the password scheme f is against random password leaks. Note that we consider only randomly sampled pairs, instead of the adaptively chosen ones, similar to [14]. Since in the real word, adversaries generally depends on weakness of websites to gain information of user's passwords. It is more useful to model such breaches as random events not controllable by an adversary. Also in the real word, most websites have the mechanism of limiting the number of tries one can use to authenticate. For example, if the adversary gives 10 wrong passwords, the user may be notified through email and change his password accordingly. Therefore, in the random challenge model, we measure Q by the number of random pairs (c_i, p_i) an adversary needs to compute p' for a random c', within 10 tries.

6 Our Proposed Password Schemes

6.1 Parameters

Our proposed password scheme requires memorizing an English sentence plus a random letter to digit mapping. One way to generate the random English sentence is to select a random sentence from a random English book (e.g. by

using [18]). If the sentence is too short, it can be concatenated with the next sentence in the book until it reaches a minimum length. A random letter to digit mapping can be generated by picking an integer from 1 to 10 (inclusive) uniformly at random for each letter "a" through "z". We use 1 to 10 instead of 0 to 9 because our scheme requires nonzero values.

6.2 Function

To use our scheme, we first take the site name and convert it to lowercase letters. Numbers are spelled out in English, and non-alphanumeric symbols are ignored. For example, "A9" would become "anine". We then repeat the site name as many times as is necessary to reach 12 letters, and apply our letter to digit mapping to obtain 12 digits.

We first process our sentence in the same way (coverting to lowercase, spelling out numbers, removing non-alphanumeric characters like punctuation). Then, for each digit, we count forward that many characters in the sentence and output the corresponding character as a letter in the password. This is a streaming algorithm and does not require random access, so it is easier to perform these operations mentally. If we run out of characters in the sentence, we wrap around to the beginning (equivalent to having the sentence repeated as many times as necessary).

6.3 Example

An example is probably the easiest way to understand the scheme.

Let's say our letter to digit mapping maps "a" to 1, "m" to 2, "z" to 4, "o" to 3, and "n" to 6, our sentence is "mary had a little lamb", and the website we're trying to log into is Amazon. The steps in creating the password are shown below:

1. `Amazon`
2. `amazonamazon`
3. `1 2 1 4 3 6 1 2 1 4 3 6`
4. `mary had a little lamb mary had a little lamb`
5. `mryatmbardia`

6.4 Using in Real Life

One concern is that our password scheme generates passwords that are all lowercase, but most websites these days require at least one lowercase letter, one uppercase letter, one digit, and one special character. This is not a problem. We can simply append a fixed string (such as "A2@") to the end of our password to satisfy these requirements. This might not improve the security of our scheme, but it certainly won't decrease the security.

7 Results

7.1 Usability

We will call our proposed password scheme **ES12**, because it uses English Sentences and is 12 characters long. The scheme that we did user testing on is almost the same, except that the length of the password is the length of the site name, instead of always 12 characters long. We call this variant of our scheme **ES**. Of course, **ES12** would take longer to compute than **ES** and would be more secure, but otherwise the conclusions about **ES** would apply to **ES12** as well.

To test **ES**, we would ideally have a randomized sample of users. However, because students are extremely busy at this time of year, we were not able to get other students to test our scheme. Thus, we tested the scheme on each of the 3 authors of this paper.

For purposes of the questionnaire, the benchmark scheme we compared our scheme to is the **WS1** scheme from [7]. **WS1** is a scheme that asks the users to memorize 26 random words. To compute the password, the user takes the first 4 distinct characters in the site name, and for each character outputs the first 2 consonants in the random word beginning with that character.

The data that we gathered from our usability tests and the conclusions that we drew are summarized below:

7.1.1 Questionnaires

Below are the adjusted ratings for the pre-test questionnaire, post-test questionnaire, and difference between the pre- and post- test questionnaire ratings.

Category	Pre-Test R'_i	Post-Test R'_f	$\Delta = R'_f - R'_i$	$R_o = \frac{R'_i + 2R'_f}{3}$
Goodness	+0.333	+0.000	−0.333	+0.111
Likely to use	+0.333	+0.000	−0.333	+0.111
Security	+1.000	+0.667	−0.333	+0.777
Ease of use	−0.667	−1.000	−0.333	−0.888
Memorability	−0.667	+0.000	+0.667	−0.222
Flexibility	+0.667	+0.000	−0.667	+0.222

As shown by the post-test questionnaire, the users of our scheme felt it **ES** was more secure but less easy to use than **WS1**. After actually using our scheme, the ratings for each category fell, except for memorability, which rose. The overall rating, which is a weighted average of the pre-ratings and post-ratings, is positive (better than benchmark) for goodness, likelihood of use, security, and flexibility, but negative for ease of use and memorability. Again, security is the main benefit of **ES**, while ease of use is the main drawback.

7.1.2 Preprocessing Time

The Preprocessing time had a mean and standard deviation of

$$\mu = 62.000, \sigma = 19.950$$

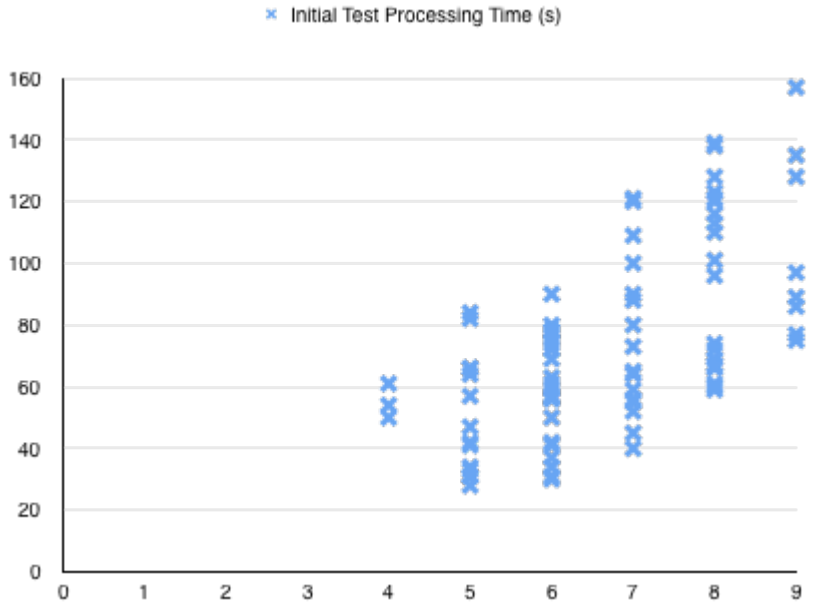
This Preprocessing time is reasonable and comparable to that of [7]. Spending an hour to remember a password scheme is very doable, since you only need to do it once. Once you remember the scheme, you automatically rehearse it through computing the passwords to your accounts.

7.1.3 Processing Time

The Processing time had a mean and standard deviation of

$$\mu = 73.356, \sigma = 29.677$$

The Processing time is the main usability concern with **ES**, our proposed password scheme. It is more than twice as long as the slowest password scheme in [7], and about 10 times as long as **WS1**, our benchmark scheme. Thus, Processing time seems to be the main drawback for **ES**, as real users are unlikely to be willing to spend over a minute to compute and type out a single password.



In the data above, it is interesting to note that although longer passwords tend to have longer processing times, there is actually a lot of variance in processing time for passwords of the same length.

7.1.4 Memorability

In the followup test that we conducted 1 week after the initial test, it took one of the users 136 seconds to recall the scheme correctly, while the other two users remembered the sentence but forgot the letter to digit mapping.

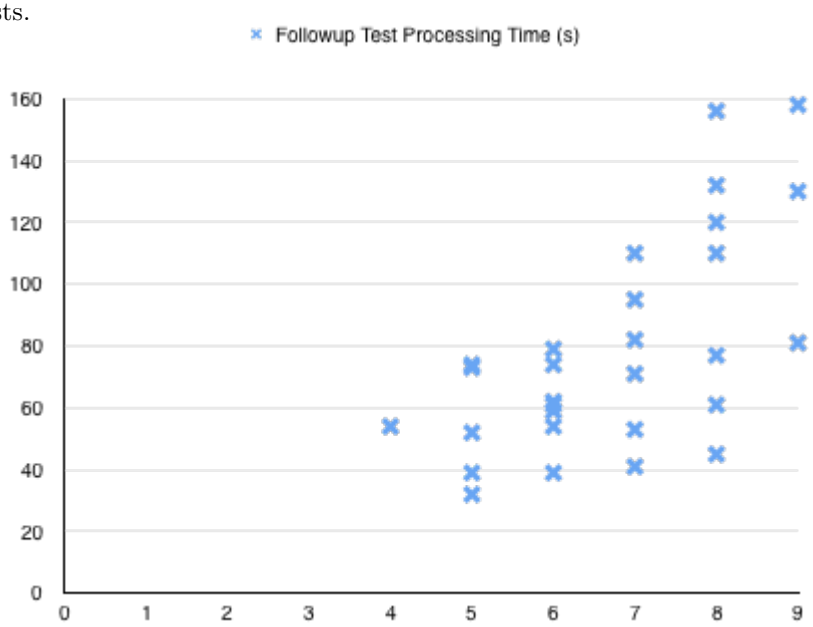
After reviewing the letter to digit mappings, all three users were able to get an empirical memorability of 1 for every account; that is, they were able to correctly compute the passwords for each site name.

What was interesting was that the users had different ways of memorizing the letter to digit mapping. One of the users simply memorized the 26-digit string composed of the digit mapped to "A", followed by the digit mapped to "B", etc. Another user memorized the same 26-digit string, but in chunks of 5 digits each. The final user memorized the inverse digit-to-letter mapping, remembering the mapping by associating letters to the digit and to each other (for example, if "G", "I", "M" map to 6, the user remembered that "G" looks like 6, and that "G" was part of "IMG"). This user was also the only one to still remember the letter to digit mapping after 1 week.

We also measured the Processing time for each account's password. The Processing time had a mean and standard deviation of

$$\mu = 77.767, \sigma = 33.646$$

This means that the Processing took a few seconds longer than a week ago, and had slightly more variance, which seems reasonable given that the users did not use the scheme at all within that week between the initial and followup tests.



As you can see in the chart, the Processing times in the followup test follow roughly the same trend and pattern as the Processing times in the initial test.

7.1.5 Flexibility

The Preprocessing time had a mean and standard deviation of

$$\mu = 50.333, \sigma = 3.682$$

This is good news for **ES**. The average Preprocessing time was more than 10 minutes shorter, and the variance decreased significantly. This shows that

ES is flexible enough to allow users to switch over to new parameters. Instead of having the old and new parameters interfere with each other, we found that having used **ES** before makes it easier to use **ES** with a new set of parameters because the user will have had practice using the scheme and perfecting their memorization and computation tricks.

7.2 Memorability

7.2.1 Long-term Memory Burden Analysis

Our proposed scheme requires the user to commit to memory a secret letter-to-digit mapping of 26 chunks, as a well as a randomly chosen sentence of roughly 20 words in length (due to the structured nature of sentences, we assume that each word in our sentence constitutes a single chunk). The total number of chunks of private data values a user must commit to long-term memory is thus $D \approx 26 + 20 = 46$.

The number of distinct instructions that need to be committed to memory can be estimated by the number of number of lines of pseudocode needed describe the password generation schema.

1. Repeat letters of the account name until it is 12-letters long.
2. For each letter, retrieve the memorized letter-to-digit mapping.
3. Advance this many indices in the memorized sentence.
4. Output the resulting letter as the next letter of the password.

Thus, we use $I = 4$ for this schema.

Applying these values the model described in Section 4.1, our framework estimates the expected number trials needed to recall the scheme without error is

$$\mathbf{PT} = (1 - \delta)^{(-I-D)},$$

where δ is the probability of independently mis-remembering any single chunk. Since we were unable to determine a suitable value for δ empirically, we can use $\delta = 0.05$ in a sample calculation and evaluate $\mathbf{PT} = (1 - 0.95)^{-46-4} = 13.0$ for our scheme.

For comparison, the **WS1** scheme presented in [7], which only requires the user to memorize a 26-chunk letter-to-word mapping ($D = 26$), and also has a public instruction set size of $I = 4$, would have a long-term memory burden of $\mathrm{PT} = (1 - 0.05)^{-26-4} = 4.66$

Thus, under a δ value of 0.05, the long-term memory burden of our scheme is roughly 2.5 times as large as the benchmark scheme, i.e. requires roughly 2.5 times as many trials to commit to long-term memory than the benchmark scheme.

7.2.2 Short-term Memory Burden Analysis

To perform a short-term memory burden analysis of our scheme (i.e. to characterize the processing time of computing new passwords), we do accounting on the number of chunks of data values that need to streamed into a user's memory span for computing each letter of the password:

- 1 chunk to store pointer into site name
- 1 chunk to store value of corresponding digit mapping for that letter
- 1 chunk to store the identity of current word in the sentence,
- 1 chunk to store moving index in the sentence
- 1 chunk to store output letter

Thus, assuming a human memory span of 5 chunks, each letter of the password is computable *without* recycling any characters out of memory stream. The total short-term memory burden our scheme is therefore $5 \cdot k$, where k is the number of letters in the output password. Since our scheme pads all account names to 12 characters in length, computing passwords with our scheme incurs a constant short-term memory burden of 60 for all accounts.

For comparison, the benchmark **WS1** scheme from [7] incurs a short-term memory burden of $4 \cdot k$, where k is the number of letters in the challenge account name. Unlike our scheme, **WS1** does not pad or truncate password challenges to a constant length. For most account names (with $k < 15$), the benchmark scheme would achieve a lower short-term memory burden than our scheme.

7.2.3 Self-rehearsability

It turns out that the self-rehearsability of both our scheme and the benchmark scheme relies solely on the distribution of letter frequencies in common account names - i.e. the frequency with which a single letter-to-digit or letter-to-word mapping is rehearsed, is proportional to the corresponding letter's occurrence frequency in the set of sampled account names. Thus, a self-rehearsability analysis would be identical for our scheme and the **WS1** scheme of [7].

7.3 Security

In the offline attack model, the adversary needs to check all possible passwords and match them to the hash. Suppose the dictionary contains L different letters, since the repeated letters don't necessarily corresponds to same password characters, the password space is simply L^{12}. That is, an adversary needs to try L^{12} times to recover the password in the offline attack model.

In the random challenge attack model, we empirically measure the security by playing the following game: for a give Q

- The challenger randomly chose a secrete s sentence from the dictionary D, and a random mapping σ from letters to digits.
- The challenger send the adversary a randomly chosen domain name c', and Q pairs of randomly chosen domain names, and their corresponding passwords $(c_i, f(s, \sigma, c_i)$.
- The adversary provides 10 guesses of the password for c', $P' = p'_1, ...p'_{10}$. If $f(s, \sigma, c') \in P'$, the adversary wins.

We measure the possibility r of an adversary winning the game for Q by running 50 such games per group, and then calculate the average of 20 groups r_{avg}. The results for several different configurations of the scheme are shown in Table 1-4.

Q	r_{avg}	std
10	0.57	0.063
12	0.64	0.072
14	0.72	0.066
16	0.74	0.069
18	0.82	0.048
20	0.83	0.063
22	0.83	0.067
24	0.88	0.046

Table 1: Measured Q using Hamlet as dictionary D, and $|s| \geq 10$

Q	r_{avg}	std
10	0.54	0.077
12	0.66	0.059
14	0.71	0.085
16	0.78	0.050
18	0.80	0.046
20	0.83	0.059
22	0.86	0.035
24	0.87	0.051

Table 2: Measured Q using Hamlet as dictionary D, and $|s| \geq 15$

Q	r_{avg}	std
10	0.544	0.063
12	0.661	0.063
14	0.745	0.048
16	0.775	0.069
18	0.825	0.062
20	0.84	0.049
22	0.857	0.050
24	0.889	0.045

Table 3: Measured Q using Hamlet as dictionary D, and $|s| \geq 20$

Q	r_{avg}	std
10	0.53	0.062
12	0.62	0.082
14	0.70	0.074
16	0.78	0.051
18	0.80	0.049
20	0.80	0.066
22	0.82	0.047
24	0.86	0.039

Table 4: Measured Q using Hamlet, and King Lear as dictionary D, and $|s| \geq 10$

In the empirical evaluation, we tried first using Hamlet as the dictionary D to choose random sentences from. In case of the sentence being too short, we concatenate consecutive sentences until it has a length of minimum 10. The result of this configuration is shown in Table 1. We next tried several variations: we used a larger dictionary, as shown in Table 4, choosing sentences from both Hamlet and King Lear; and we used different sizes of sentences as the secrete s. As shown in Table 1-3. However, none of the variations have much effect on the security of the scheme.

This is because each sentence only contains a very limited set of characters, and it's very easy for an adversary to recover the secrete sentence s by testing if a sentence contains all the provided passwords p_i's as a subsequence. In our testing cases, the adversary can easily limit the possible sentences to around 3 by the simple subsequence test. On the other hand, the scheme still has good security because the random mapping σ is hard to recover. In our testing cases, we used 1-consistency checking to eliminate the impossible mappings. For example, consider the i^{th} character in a provided website name, password pair c, p: we first rotate the recovered sentence s so that $p[i-1]$ matches the first letter of the rotated sentence s'. And then add possible mappings $\sigma(c[i]) = n$ such that $s'[n] = p[i]$, and eliminate the impossible ones. We call this 0-consistency checking since every time it only check the constrains for a single letter and ignores the rest in the website name. In the next iteration, we consider all the remaining possible mappings for $c[i]$, and checks the mappings for $c[i+1]$ and $c[i-1]$. We call this 1-consistency checking since every time it considers one pair of constrains, and therefore can eliminate more impossible mappings. The actual possible mappings after n-consistency checking can be smaller than we recovered in the empirical evaluation, so the actual rate of success may be slightly higher than our listed result. However, the difference is not be large, since the difference between 0-consistency checking and 1-consistency checking is already very small.

In comparison, the proposed scheme has better security compared to the scheme **WS1** in [7], which, in a similar evaluation described in the paper, has $7 \leq Q \leq 8$ for the adversary to get 90% success rate.

8 Comparison to Existing Schemes

In the Results section, we compared our proposed scheme, **ES12**, with the benchmark scheme, **WS1** from [7]. We found that the security was higher than **WS1**, and the ease of use was lower. In terms of memorability, the long-term and short-term memory burden of our scheme was slightly higher than the benchmark, and the self-rehearsability was identical to benchmark.

Overall, we found that the amount of information memorized is not unduly large, as shown by our reasonably short preprocessing times. The letter to digit mapping requires remembering 26 pieces of information, but so does all of the schemas in [7], which require memorizing either a letter to digit mapping, a permutation, or a collection of 26 words. We found that the English sentence was easy to remember and that the letter to digit mapping was the bulk of the memorization work.

The security is better than the schemas in [7], since the security of **ES12** is higher than **WS1**, which had the highest security in [7] (excluding the multi-map schema, which just consists of deterministically choosing one of several schemas based on the site name, then applying it).

The main drawback is the processing time, and thus ease of us, of **ES12**, which is several times slower to compute than even the slowest schema in [7].

We think that this is partly due to the large number of operations used in **ES12**, and partly due to our lack of experience in doing mental computations and memorization. If we were as experienced as the authors of [7], we would be able to compute our passwords faster. The long computation time may not be as large a handicap as it seems, however. A user's most common passwords will be typed so often the user will remember them by heart; it is only the lesser used passwords that need to be computed.

Overall, we think that our proposed scheme, **ES12**, is a reasonable scheme to use - as long as you don't mind spending a minute to compute your password.

9 Conclusion

In this paper, we first surveyed existing works that address the challenge of creating memorable passwords for multiple accounts. Combining the results from [7] and [8], we propose a comprehensive frame work to evaluate password scheme for systematically creating strong passwords for multiple accounts. The framework evaluates the usability of a scheme by conducting experiment on volunteer users. We collect data of the time it takes for each user to remember and correctly apply the scheme, and also collect their subjective feeling of the scheme's security, usability, and flexibility. The framework evaluates the memorability of a scheme using a theoretical model of human memory. It breaks down the scheme into chunks of information that needs to be stored into long term memory, and a set of algorithms that is runnable completely in a person's head. (i.e. without using a computer, or even pencils and paper.) The framework finally evaluates the security of a scheme by considering two possible attacks based on real world events. In the offline attack model, the attacker guesses a password by matching it's hash value stored on the server. In the random challenge attack model, the attacker is first given a set of valid password, and then asked to guess a password of a random account. We then propose a novel variation of the schemes from [7]. We run this sample scheme through our framework, and shows that it out performs the schemes from [7] in the security framework, and also has a slight advantage of usability, while memory burden is a little higher than the bench mar.

There are also limitations of our work. Our scheme always produces passwords of the same security, but sometimes, a user may want to sacrifice memorability for better security, for their important accounts. It would be desirable to have a scheme that support multi-level security and memorability. Also, our analysis framework is also limited to text-based password schemes, similar to those from [7]. It is desirable to extend it to also support other types of password schemes, such as graphical passwords, and biometric passwords.

References

[1] M. Ghazvininejad, and K.Knight. *How to Memorize a Random 60-bit String.* Parking. Vol. 11-70.8, 2015.

[2] A. Alam, *SUIS: An Online Graphical Signature-Based User Identification System*, DICTAP 2016, pp 85-89, 2016.

[3] D.L. Nelson, V.S. Reed, J.R. Walling, *Pictorial superiority effect*, J Exp Psychol Hum Learn, 2(5):523-8 1976.

[4] M. Sherman, C. Gradeigh, Y. Yulong, S. Shridatt, M. Arttu, L. Janne, O. Antti, and R. Teemu. *User-generated free-form gestures for authentication: Security and memorability.* In Proceedings of the 12th annual international conference on Mobile systems, applications, and services, pp. 176-189. ACM, 2014.

[5] K. M. Everitt, T. Bragin, J. Fogart, T. Kohno, *A Comprehensive Study of Frequency, Interference, and Training of Multiple Graphical Passwords* SIGCHI Conference on Human Factors in Computing Systems, pp 889-898, 2009.

[6] J. Blocki, M. Blum, A. Datta, *Naturally Rehearsing Passwords*, Advance in Cryptology ASIACRYPT 2013, pp 361-380, 2013.

[7] M. Blum, and S. Vempala. *Publishable Humanly Usable Secure Password Creation Schemas.* In Third AAAI Conference on Human Computation and Crowdsourcing. 2015.

[8] J. Blocki, B. Manuel, A. Datta, and S. Vempala. *Towards Human Computable Passwords.* arXiv preprint arXiv:1404.0024. 2014.

[9] Z. Zhao, G. Ahn, J. Seo and H. Hu. *On the Security of Picture Gesture Authentication.* Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13). 2013.

[10] R. Biddle, S. Chiasson, and P.C. Van Oorschot, *Graphical Passwords: Learning from the First Twelve Years*, ACM Comput. Surv, Vol. 44-4, 2012, doi:10.1145/2333112.2333114.

[11] X. Suo, Y. Zhu, and G. S. Owen, *Graphical Passwords: A Survey*, Proceedings of the 21st Annual Computer Security Applications Conference, 2005, doi: 10.1109/CSAC.2005.27.

[12] J Bonneau, C. Herley, P. C. van Oorschot, F. Stajano, *The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes*, IEEE Symposium on Security and Privacy, 2012, doi:10.1109/SP.2012.44

[13] G. A. Miller. *The magical number seven, plus or minus two: some limits on our capacity for processing information.* Psychological review 63.2, pp. 81. 1956.

[14] N. J. Hopper, and M. Blum. *Secure Human Identification Protocols.* In International Conference on the Theory and Application of Cryptology and Information Security (pp. 52-66). Springer Berlin Heidelberg. 2001.

[15] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, Wiley-Interscience, 2006.

[16] https://xkcd.com/936/

[17] https://blog.dashlane.com/infographic-online-overload-its-worse-than-you-thought/

[18] https://josephrocca.com/randomsentence/