Matrices Applied in Cryptography

Team 13

I. Abstract

Many people use the same passwords for numerous accounts, which is very dangerous. Thus, we created an encoding program to solve this problem, and used some quantitative methods to evaluate our results.

II. Introduction

Nowadays, most people have multiple accounts, yet with all passwords the same, because creating a new password is quite annoying and will be easily forgotten. However, using the same password across multiple accounts can lead to credential stuffing, which is unsafe. As a result, we came up with an idea to write a code that can convert our commonly used passwords into different passwords with higher security corresponding to the account's name.

III. Methods

The main steps are as below:

1. Enter web name and general password users usually use
2. Encryption using matrix multiplication
3. Then we can get a new password!

(Detailed steps can be found in Appendix A)

IV. Results

We collected some passwords (see Appendix B) and used the UIC Password Strength Test to test if our encryption is effective. We divided the strength into four classes: very strong, strong, good, weak, and very weak. Our most successful batch of encrypted passwords showed that for the original passwords, 81.8% were very weak and 18.2% were weak. But after encryption, only 3% were very weak and 36.4% became very strong. In addition, we calculated the increment of password complexity. For instance, if a very weak password becomes a very strong one, it increases four classes, denoted as +4. We learned that the average increment of password complexity is +2.5149. (There are more detailed graphs,

charts, and other two examples in Appendix D.) By the results we obtained from the strength test, we can prove that our encryption is really useful.

Discussion

The program that we used to convert the original passwords into more complex, thus stronger ones was successful due to the program using a randomly generated matrix to encrypt the passwords. The new generated passwords are random as well, which leads to lower possibilities of characters in predictable positions, common keyboard patterns, common words, and common phrases. Therefore, these new passwords are more secure than before. (The reason that the more complex a password is the more secure it is can be answered in Appendix F.)

Conclusion

The majority of people tend to overuse passwords that are simple and easily guessed. With the implementation of our program which uses matrices to encrypt said passwords, we are able to make the passwords more complex and harder to guess.

References

[1] Ur, B., Alfieri, F., Aung, M., Bauer, L., Christin, N., Colnago, J., ... & Melicher, W. (2017, May). Design and evaluation of a data-driven password meter. In Proceedings of the 2017 chi conference on human factors in computing systems (pp. 3775-3786).

[2] Password Meter - A visual assessment of password strengths and weaknesses. (n.d.). https://www.uic.edu/apps/strong-password/

[3] Dell'Amico, M., Michiardi, P., & Roudier, Y. (2010, March). Password strength: An empirical analysis. In 2010 Proceedings IEEE INFOCOM (pp. 1-9). IEEE.

[4] Wheeler, D. L. (2016). zxcvbn:{Low-Budget} Password Strength Estimation. In 25th USENIX Security Symposium (USENIX Security 16) (pp. 157-173).

[5] Kelley, P. G., Komanduri, S., Mazurek, M. L., Shay, R., Vidas, T., Bauer, L., ... & Lopez, J. (2012, May). Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In 2012 IEEE symposium on security and privacy (pp. 523-537). IEEE.

[6] Yan, J., Blackwell, A., Anderson, R., & Grant, A. (2004). Password memorability and security: Empirical results. IEEE Security & privacy, 2(5), 25-31.

Table of Contents for Appendices

A. Methods

Here below are the methods used to encrypt the passwords input into the program.

Step 1:

We created a special sentence by hand to build the web key. The web key is used to generate the matrix A, which is going to be used to encode the general password later.

Step 2:

The user's general web account username and password is to be entered.

Step 3:

The username will correspond to a group of numbers in the web key.

Step 4:

We use that group of numbers as a seed of the random function, and generate a random number *w*. Then, our number of rows of matrix A = (*w* % length of password) + length of password (while number of columns of matrix A = length of password). This can ensure that the matrix A is not invertible, because the number of rows doesn't equal the number of columns.

Step 5:

Generate encryption matrix and use the web key to fill it.
(according to step 4, the matrix is non-square so it is not invertible at the same time)

Step 6:

Convert the general password from type of *char* to type of *int* by using ASCII code, and transpose it in order to do the matrix multiplication in the next step.

Step 7:

Do matrix multiplication of the encrypted matrix generated in Step 5 and the number string generated in Step 6.

Step 8:

Remove characters that cannot be used as a password in the string generated in Step 7 to get the final new password.

Special Advantage:

The random number *w* will be different on each computer. But the number will be consistent on the same computer. Thus, we can ensure that we are able to get the same encoded password and this encoded password will not be known by others even if they have our general password.

B. Survey

We conducted a survey (Figure 1.) to collect some manually generated passwords.

A. How has the survey been done?

We asked each respondent to provide three passwords. One of them is for social media accounts, another is for phone unlocking, the other is for email.

B. Rules of password generation

a. Password length is preset to popular regulations. (Facebook for social media accounts, Android for phone unlocking, and Gmail for emails)

b. Arabic numerals, upper/lower case letters, and special characters (e.g., #, %, $, @) are acceptable.

After collecting passwords from the total 34 respondents (one was eliminated due to rule violation), we generated new passwords using our proposed method and evaluated the security of passwords both before and after encryption. Figure 2. shows the original passwords and the encrypted ones.

密碼習慣調查 Survey of Password Usage

國立陽明交通大學 515506 線性代數 第13組 期末報告 表單調查
This survey is part of a final project for Group 13 of Linear Algebra (Class 515506), NYCU

emilychang.cs11@nycu.edu.tw (未分享) 切換帳戶

*必填

本表單的目的以及注意事項
The Purpose of This Survey & Cautions

我們是國立陽明交通大學大一的學生，目前進行**線性代數應用於密碼學**的研究作為線性代數課程的期末報告。我們的主題為**利用矩陣產生密碼**，而我們希望能取人為設置的密碼，再利用矩陣加密，最後再比較兩者的強度。
We are freshmen at NYCU undergoing a study to **apply linear algebra in cryptography** as a final project for our linear algebra course. Our study is to **utilize matrices as a way to encrypt passwords**, and we hope to gather user-chosen passwords, encrypt them, and compare the strength of the encrypted and non-encrypted passwords.

本表單希望能調查人為設置的密碼，並且將會使用您提供的密碼作為實驗對象；因此以下填寫表單內容時，請盡量**避免使用您個人帳號的密碼**及**避免重複填寫表單**，謝謝!
The purpose of this survey is to gather examples of passwords manually chosen by users, and to use them as part of our study; therefore, when you complete the survey, please make sure **not to provide personal passwords** that you use and **refrain from filling out the survey multiple times**. We thank you for your cooperation!

Figure 1.

	時間戳記	請提供一組您在社群平台 Please provide a passwo	請提供一組您手機解鎖可（限純阿拉伯數字） Please provide a passwo	請提供一組您的電子郵件可能使用的密碼： Please provide a password that you would use for an email account.			Encrypted(social media,	Encrypted(phone, wed n	Encrypted(email, wed na	email	https://drive.google.com
2	2022/12/2 下午 7:10:01	Amy@0123	05270831	ImAmy777			$Xl@hX6Xl	YsxRJhN%%	u@ax$$#5L		https://docs.google.com
3	2022/12/8 上午 2:54:24	Leo87876987	0822	Leo87876987			#IR18G9RcRK@	7INp$	1ZrOCp#p#pu#		
4	2022/12/8 上午 3:46:55	kate<3pink	0309	kao0309			M%In2%wshQ0	C@Ipj	CVY5XaBK		
5	2022/12/8 上午 11:18:06	12358	3211	S3323675			$udI@!	%Ob9l	#$N@2gNrx		
6	2022/12/8 下午 12:28:48	xuxu900728	1990902807	xukai72890			BA2c@%ac#bl	JG9IgG#ID%j	L2qTl@#I3HA		
7	2022/12/8 下午 12:30:45	FBKaRen	0000	mailkkk1129			R@!u#@FS	6f0#o	B13iW1ZG5%$#		
8	2022/12/8 下午 12:30:45	lejduet738	03716484	Jsgenj02			@el#ID#Xe@	5Q#0%Q%SA	BQ0hk0rFS		
9	2022/12/8 下午 12:37:11	87651234	0506	a1234567			U#D#%#@#n	S@#e0	$gsFj$9/r		
10	2022/12/8 下午 1:37:27	aaa003159		1227	aaa003159		B@#!$u!@Q#	7@gpc	dqUPtfE#s9		
11	2022/12/8 下午 2:18:09	l62316206	0231	20030231			QFSF$@#FL$	xDs#a	eVk@WVnVv	L0Y@20BT8Tt0lxFTm@6T6xLT	
12	2022/12/8 下午 5:20:00	Jason911023	900409	Jason900409000			@v%#gv93qvB4	yqN%%E4	RQI@!$RQYsP@5F@		
13	2022/12/8 下午 8:15:14	022573486wendy	35754981	022573486wendy			ex#m!@#1211l0%#	qFU/C@nFN	wxhm7@0lbaxq$P		
14	2022/12/8 下午 8:26:29	password	20221208	zxovbn08			LNPe8Nhog	Rt%@RTt$	@a2mi@FI0		
15	2022/12/8 下午 8:30:04	abc9811100282	159357	sophia0802			C##wj#qvd%$aJ4	GP@#8/T	%F#rkEdg$fw		
16	2022/12/8 下午 8:36:53	jh1234jh1234	0208	Jh1234jh1234			@kfkAljfcJ!v4	qp$Z#	u3Ft!$@@%Im3N		
17	2022/12/8 下午 8:39:00	daniel0101	7676	d8m88b8			u#5#TUQIPvG	k8@!$	m#HUlv2J		
18	2022/12/8 下午 8:42:38	@nt0nych0u	2004031100	antonychou			bvC3K#ll%TQ	$wSaVwxlw4l	f!gM/dr$Pk		
19	2022/12/8 下午 8:52:51	Any900230	291311	betty880631			a@dlh@E2NS	IEXgc$e	aN#6#N@diN2p		
20	2022/12/8 下午 9:47:10	password	123456789	asdfghjkl			LNPe8Nhog	XWSL%LGLlb	O@L###@#k%		
21	2022/12/8 下午 9:49:00	Lkyin@Feiron	2481632	noriFe@1.0			q1k@@@n@0GF8$	RSFWC$#x	@%N#s@e2vIH		
22	2022/12/8 下午 9:49:37	yang5303	5303	yang5303			%rm0#KV	Spaea	#L$XAW%Wd		
23	2022/12/8 下午 10:26:08	Ruby1205	19980329	John0987			0!ISo1#1Y	I0S/f@$@$	mYF@9N%N@		
24	2022/12/8 下午 10:27:48	0972772745	0972	Csy972772745			!#oE!#PQf%	GOm9D	IUr4#%p3rU3#V		
25	2022/12/8 下午 10:34:56	Willy31415926	2580	Willy31415926			@jET@u%@H#3Yu	p#cl0	O@%HX@$RjSQ@#i		
26	2022/12/8 下午 11:26:12	JOHNjohn#0#7@0@8	09100910	HSHSB16734847#19980302			s@lbIt?Ocil%o@RW#	A1@tW1Nl%	j@6@$tB%58o@yB%@/V8jMXIt		
27	2022/12/8 下午 11:54:48	Da632046	35631252	ye529171			Dk#Ugkfk!	%%@%f%%%Y	pvdUq!hv@		
28	2022/12/9 上午 12:02:50	kenn1012	1012	kenn1012			Ks@1Us%s!	3#B#O	2M%d@XPXS		
29	2022/12/9 上午 12:15:26	Dhxishao52	1231111888	Bslqb10472			##o!@aDVS%w	N#I2I#G2$SC	$nl7IX#@ncP		
30	2022/12/9 上午 12:22:13	FB5566yaah	753217788	Qaws1456			F@hp#@M8KCh	jxfbGb#o#$	v@#C@@%@q		
31	2022/12/9 上午 12:47:04	dolly0826	9999	mmm0826			k#wP@P%#kZ	xBS7t	VIDk!S2v		
32	2022/12/9 下午 12:13:45	facebook77	2036	serena22			n!$U/k5UTI!	Epne6	3C5O%8@YW		
33	2022/12/9 下午 6:07:36	2718281828	52990969	31415926535897932384626433832795028841971693993751058209749			u%J/O%@%%Po	I%B/aE1P@	F#Qp2qs@%e%!ps%$@v#3%fAj8J0xG@w66FcQ%d?$F8#!#IdQPehrI5$		

Figure 2.

C. The UIC Password Strength Test

1. About the Developer:

   The UIC Password Strength Test is developed by the Academic Computing and Communications Center at University of Illinois at Chicago.

2. How the test works:

   The test assigns points for addition and deduction to evaluate the complexity of a hypothetical password. The individual points are then added up to a total score, which is presented by a bar on top of the screen (as shown below in Fig 3).

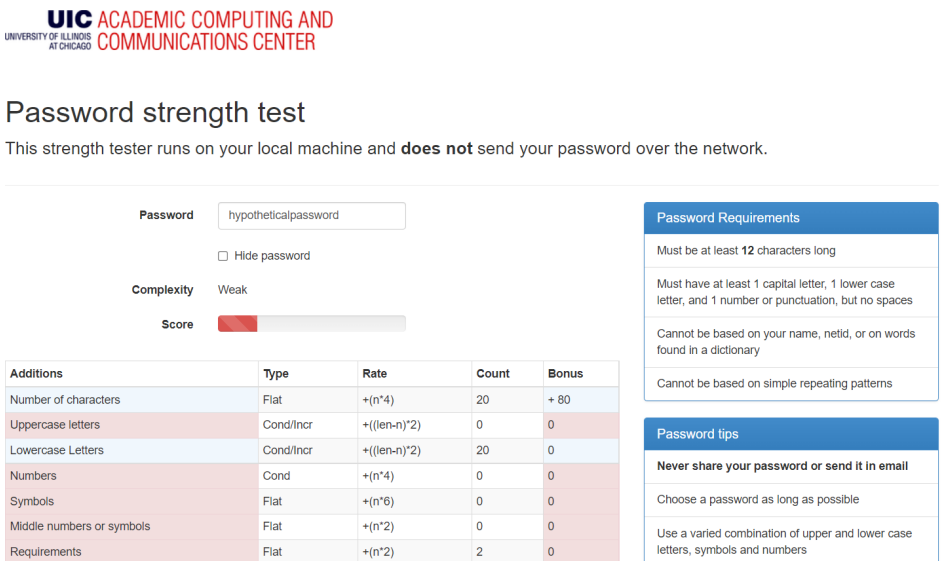


Fig 3.

The horizontal row at the top of the graph for evaluation in Fig 3 stands for how the points are added to/deducted from the overall score. *Type* in the second column corresponds to *Rate* in the third column, which indicates how the points are added/deducted and the used formula. *Count* in the fourth column stands for the occurrences for each criteria met (which are listed in the first column).

1) Points for Addition:

   a) Number of Characters

      The numbers of characters are counted, and for each character, a point is added to the overall score.

   b) Uppercase/Lowercase Letters

      For each uppercase/lowercase letter, the points increase by twice the total amount of points minus the number of occurrences of uppercase/lowercase letters.

   c) Numbers

      For each number, points are added by four times the amount of

numbers used.

d) Symbols

Symbols refers to special characters such as the $, %, &, *, etc. For each of these symbols, points are added by six times the amount of symbols used.

e) Middle Number or Symbols

For each number or symbol used in the middle of a password, the points are added by two times the occurrence of such use. (For clarification, a number or symbol used in the middle of a password means that the number or symbol is not the beginning character nor the ending character.)

f) Requirements

For each met requirement in the list in the top right corner of Fig 3, the points are added twice for each requirement met.

2) Points for Deduction:

a) Letters/Numbers only

If a password is comprised of either only letters or only numbers, the test deducts a point for each letter/number used.

b) Repeat Characters (case insensitive)

If a character is repeated, such as *abca* (*a* being the repeated character in this case), points are deducted. As this criteria is case insensitive, lowercase and uppercase letters are treated as the same letter, such as *A* and *a* in the hypothetical password *abcA*. However, how many points this particular criteria deducts is unknown to us, as it is not shown. Although it states that the method of deducting can be found in the source code, we are unable to access it.

c) Consecutive Uppercase Letters/Lowercase Letters/Numbers

For each instance of uppercase/lowercase letters or numbers used consecutively, points are deducted twice for each occurrence. An example of a password with consecutive uppercase letters would be *ZXcvbn*, where Z and X form an instance of consecutive uppercase letters.

d) Sequential Numbers/Letters/Symbols (used in a sequence

longer than three characters)

For each case of sequential characters used in a password, three times the occurrences of the mentioned case of points would be deducted. An example of this would be a hypothetical password of *password123*, where 3 points would be deducted due to the usage of *123*.

D. Detailed Results from Strength Test

1. Social Account:

1) Non-encrypted:

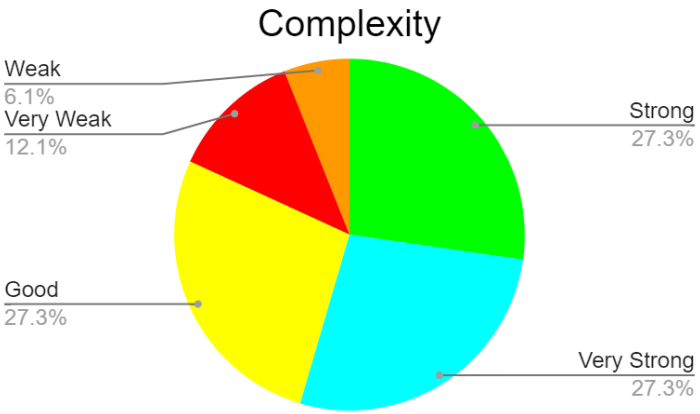


Figure 4.

2) Encrypted:

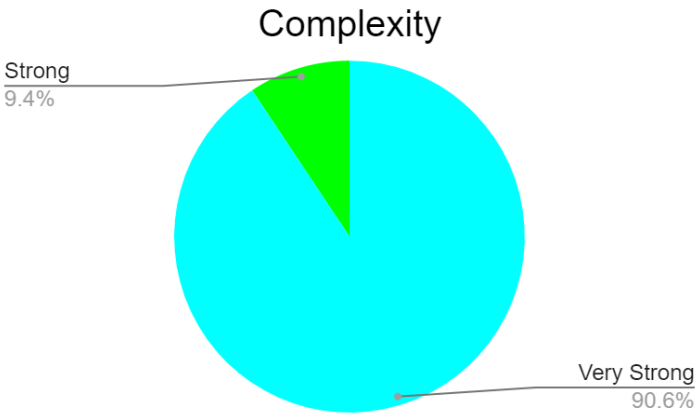


Figure5.

3) Average increment of password complexity:

Change	Proportion	Increment of Password Complexity
Very Weak → Very Strong	6.67%	+4
Very Weak → Strong	6.67%	+3
Weak → Very Strong	9.38%	+3
Good → Very Strong	18.75%	+2
Strong → Very Strong	21.88%	+1
Very Strong → Very Strong	34.38%	+0
Very Strong → Strong	3.13%	-1
Average Increment of Password Complexity		+1.3108

Table 1.

2. E-Mail:

1) Non-encrypted:

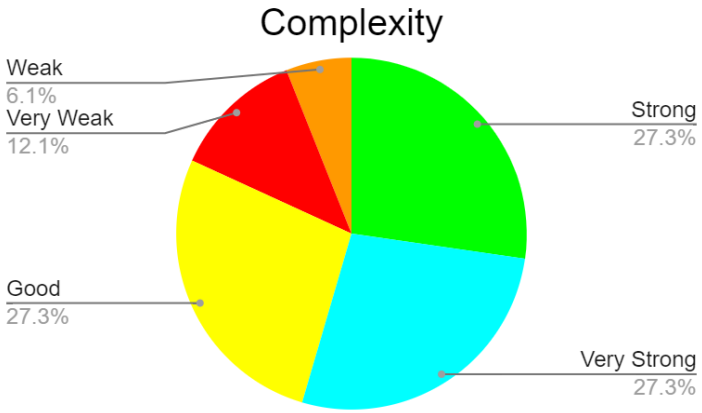


Figure 6.

2) Encrypted:

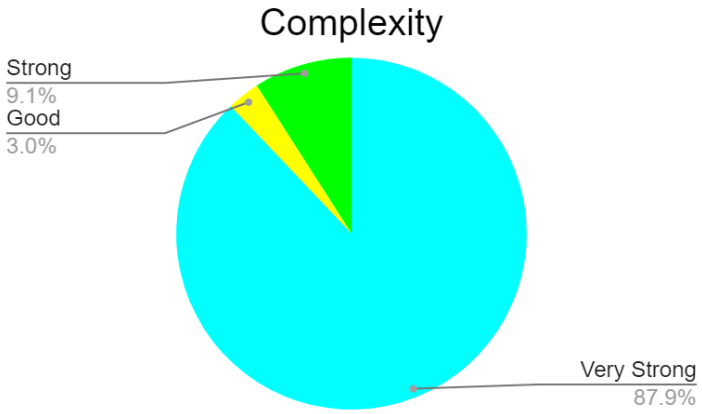


Figure 7.

3) Average increment of password complexity:

Change	Proportion	Average Increment of Password Complexity
Very Weak → Very Strong	9.09%	+4
Very Weak → Strong	3.03%	+3
Weak → Very Strong	6.06%	+3
Good → Very Strong	24.24%	+2
Strong → Very Strong	21.21%	+1
Very Strong → Very Strong	27.27%	0
Strong → Strong	6.06%	0
Good → Good	3.03%	0
Average Increment of Password Complexity		+1.3332

Table 2.

3. Phone:

1) Non-encrypted:

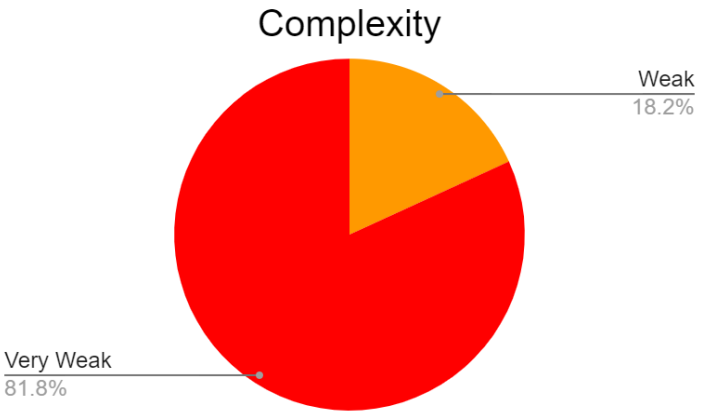


Figure 8.

2) Encrypted:

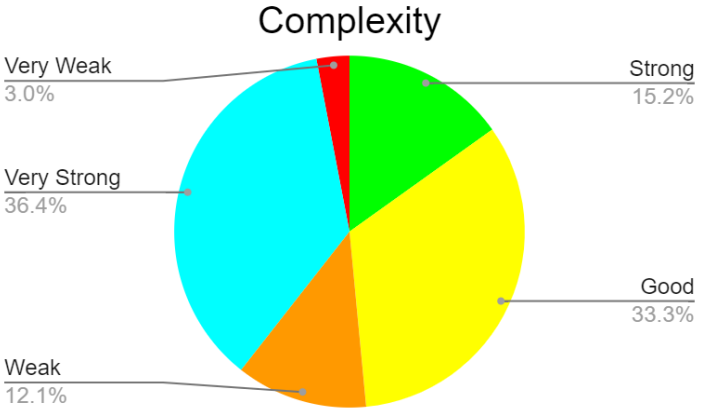


Figure 9.

3) Average increment of password complexity:

Change	Proportion	Increment of Password Complexity
Very Weak → Very Strong	24.24%	+4
Very Weak → Strong	9.09%	+3
Weak → Very Strong	12.12%	+3
Very Weak → Good	33.33%	+2
Weak → Strong	6.06%	+2
Very Weak → Weak	12.12%	+1
Very Weak → Very Weak	3.03%	0
Average Increment of Password Complexity		+2.5149

Table 3.

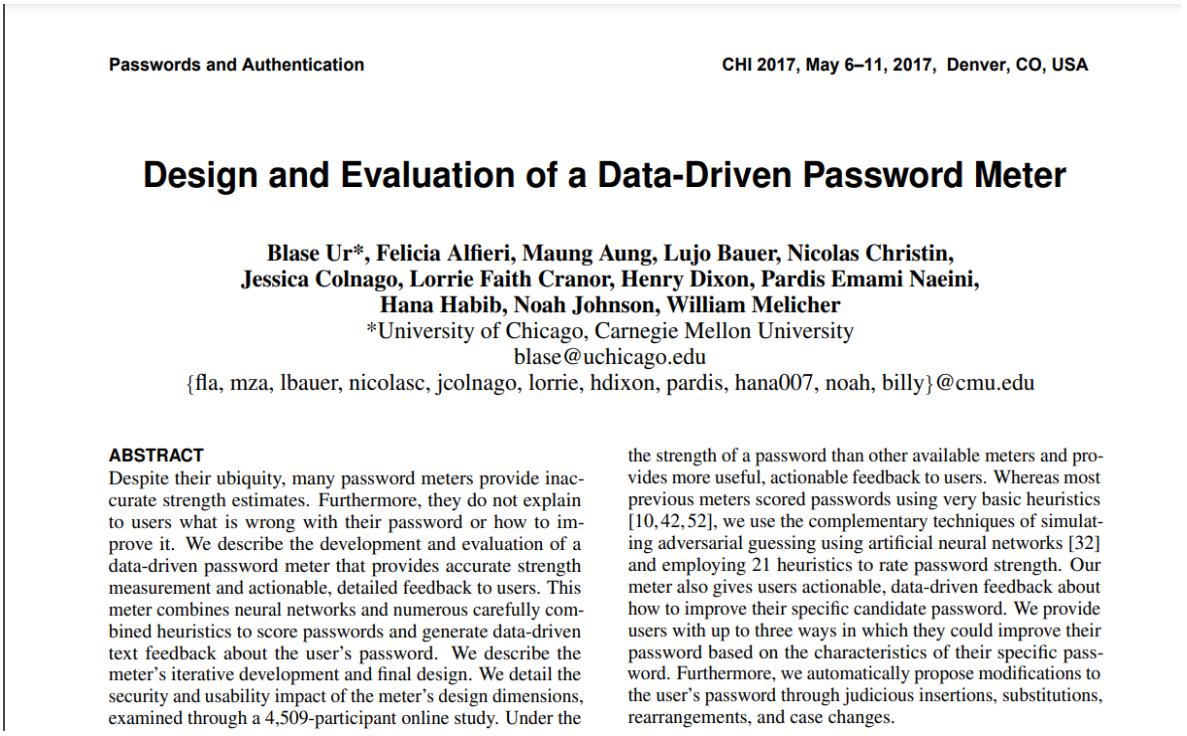
E. Validation

We used a research paper (Figure 10.) of CyLab Usable Privacy and Security Laboratory from Carnegie Mellon University to validate our proposed methods.

1. Background of the Research Paper:
    1) Title: Design and Evaluation of a Data-Driven Password Meter
    2) Authors: CyLab Usable Privacy and Security Laboratory from CMU
    3) Published: 02 May 2017
2. What we used in the research paper for validation?

    The authors mentioned the following four points to illustrate why manually generated passwords are vulnerable. Users sometimes make predictable passwords.

    1) Manually-generated passwords are easily related to words and phrases
    2) specific characters are in predictable locations
    3) Some users use keyboard patterns like “asdfghjkl” as their passwords
    4) Users frequently reuse passwords



Design and Evaluation of a Data-Driven Password Meter

Blase Ur*, Felicia Alfieri, Maung Aung, Lujo Bauer, Nicolas Christin, Jessica Colnago, Lorrie Faith Cranor, Henry Dixon, Pardis Emami Naeini, Hana Habib, Noah Johnson, William Melicher
*University of Chicago, Carnegie Mellon University
blase@uchicago.edu
{fla, mza, lbauer, nicolasc, jcolnago, lorrie, hdixon, pardis, hana007, noah, billy}@cmu.edu

ABSTRACT
Despite their ubiquity, many password meters provide inaccurate strength estimates. Furthermore, they do not explain to users what is wrong with their password or how to improve it. We describe the development and evaluation of a data-driven password meter that provides accurate strength measurement and actionable, detailed feedback to users. This meter combines neural networks and numerous carefully combined heuristics to score passwords and generate data-driven text feedback about the user's password. We describe the meter's iterative development and final design. We detail the security and usability impact of the meter's design dimensions, examined through a 4,509-participant online study. Under the the strength of a password than other available meters and provides more useful, actionable feedback to users. Whereas most previous meters scored passwords using very basic heuristics [10, 42, 52], we use the complementary techniques of simulating adversarial guessing using artificial neural networks [32] and employing 21 heuristics to rate password strength. Our meter also gives users actionable, data-driven feedback about how to improve their specific candidate password. We provide users with up to three ways in which they could improve their password based on the characteristics of their specific password. Furthermore, we automatically propose modifications to the user's password through judicious insertions, substitutions, rearrangements, and case changes.

Figure 10.

F. Other Attempted Evaluation Methods

1. Cain and Abel

This is a tool used for cracking passwords. It provides several features to crack passwords. At first, we focused on three password-cracking methods, which are the brute-force attack, dictionary attack, and the Rainbow Table Attack. We wanted to use these methods to crack the passwords both before and after encryption and measure the time taken.

1) Brute-force Attack

A brute-force attack uses trial-and-error and to work through all possible combinations in order to crack passwords. However, our passwords consist of upper and lowercase letters, Arabic numerals, and special characters, which led to too many combinations having to be run through and the time it takes is too long to measure.

2) Dictionary Attack

The dictionary attack uses a predefined list of words, which each has a

hash value[1]. If the hash of a password in the dictionary matches that of the yet-to-be-determined password, then the password is successfully identified by the attacker.

3) Rainbow Table Attack

A Rainbow Table attack is a password cracking method that uses a special table (a “rainbow table”) , which stores all the possible hashes of passwords of fixed length and components, to crack the passwords. However, there are two reasons that we gave up this method.

First, it is too strong that it cannot distinguish the cracking time between original and new passwords. Second, because our passwords consist of upper and lowercase letters, Arabic numerals, and special characters, it takes memory up to TBs to store the complete rainbow table.

Figure 11. shows we were trying to download a smaller rainbow table to use.

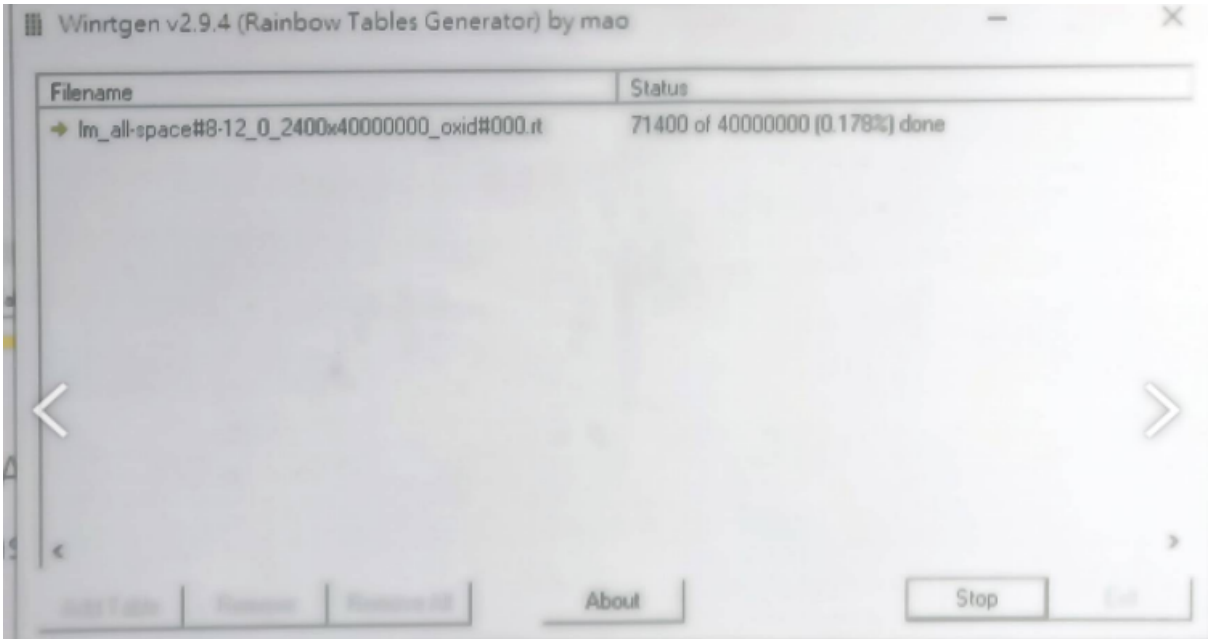


Figure 11.

G. Source Code

(For an easier experience viewing the code, please visit this website)

[1] a hash value refers to an encrypted output with a fixed length which is produced by an unique, mathematical function known as a hash function

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// find seed for random
int find_seed(int len, int seed, char *account, char *key) {
  for (int i = 0; i < len; i++)
    for (int j = 0; j < 63; j++)
      if (account[i] == key[j])
        seed += j;
  return seed;
}

int main(void) {
  char web_key[] =

"PwFeSiV2cCh7Ju8AnEsDaXd0HoTm9LrWtBb3Q5fUgRyKvZkI1xYl4MzNjOqG6p ";
  // 63 -> this is provided from a special string

  char account[100000]; // the web name you want to submit
  char general_password[100000]; // the password which you usually
used

  // scanf your web name and password
  printf("Please enter your web's name : ");
  fgets(account, sizeof(account), stdin);
  printf("Please enter your general password : ");
  fgets(general_password, sizeof(general_password), stdin);

  int acc_len = strlen(account), password_len =
strlen(general_password);

  // generate random number w -> used for deciding row numbers for
matrix A and
  // generating matrix A
  int seed = 0;
  seed = find_seed(acc_len, seed, account, web_key);
  srand(seed);
  int w = rand() % 999 + 2;

  int A_row = (w % password_len) + password_len; // row numbers
for matrix A
```

```
  // allocate memory for matrix A -> is used for encode our
passwords
  int **matrix = (int **)calloc(A_row, sizeof(int *));
  for (int i = 0; i < A_row; i++)
    matrix[i] = (int *)calloc(password_len, sizeof(int));

  // generate matrix A using web_key
  int tmp = 0, tempcount = 0;
  for (int i = 0; i < A_row; i++) {
    for (int j = 0; j < password_len; j++) {
      matrix[i][j] = ((int)web_key[tmp]) + (2 * w + 5) +
(tempcount / 63);
      tmp++;
      tempcount++;
      tmp %= 63;
    }
  }

  //
  for (int i = 0; i < A_row; i++) {
    for (int j = 0; j < password_len; j++) {
      printf("%d ", matrix[i][j]);
    }
    printf("\n");
  }
  // transpose general password
  int number_general_password[password_len][1];
  for (int i = 0; i < password_len; i++)
    number_general_password[i][0] = (int)general_password[i];

  int new_password[A_row][1];

  // encoding
  for (int i = 0; i < A_row; i++)
    new_password[i][0] = 0;
  for (int i = 0; i < A_row; i++) {
    for (int j = 0; j < password_len; j++)
      new_password[i][0] +=
          (matrix[i][j] % 10000) * (number_general_password[j][0]
% 10000);
  }
```

```
  int number[10000];
  int numberfornew[10000];

  // combine every three numbers in a group and transport it to an
ASCII code
  int j = 0;
  for (int i = 0; i < A_row; i++) {
    while (new_password[i][0] > 0) {
      number[j] = new_password[i][0] % 10;
      new_password[i][0] /= 10;
      j++;
    }
  }
  if (j % 3 == 1) {
    number[j] = 0;
    j++;
  } else if (j % 3 == 2) {
    number[j] = 0;
    number[j + 1] = 0;
    j += 2;
  }
  int numcase = 0;
  for (int i = 0; i < j; i += 3) {
    numberfornew[numcase] =
        number[i] * 100 + number[i + 1] * 10 + number[i + 2];
    numberfornew[numcase] = (numberfornew[numcase] % 89) + 33;
    numcase++;
  }

  // eliminate characters that can't be used in password
  // characters can be used in password: 33 35~37 47~57(number)
64~90(capital)
  // 97~122(lower case)
  int index[] = {
      33, 35, 36, 37,
      64}; // ASCII code of special characters that can be used in
password
  for (int i = 0; i < j; i++) {
    for (int k = 0; k < 5; k++) {
      if (numberfornew[i] == 34)
        numberfornew[i] = index[numberfornew[i] % 5];
```

```
      else if (numberfornew[i] > 37 && numberfornew[i] < 47)
        numberfornew[i] = index[numberfornew[i] % 5];
      else if (numberfornew[i] > 57 && numberfornew[i] < 64)
        numberfornew[i] = index[numberfornew[i] % 5];
      else if (numberfornew[i] > 90 && numberfornew[i] < 97)
        numberfornew[i] = index[numberfornew[i] % 5];
    }
  }

  // transport ASCII code into corresponded character
  char lastpassword[10000];
  for (int i = 0; i < numcase; i++) {
    lastpassword[i] = (char)numberfornew[i];
  }

  // printf encoded password
  printf("New password : ");
  if (strlen(lastpassword) < 10)
    printf("%10s", lastpassword);
  else
    for (int i = 0; i < password_len; i++)
      printf("%c", lastpassword[i]);
  printf("\n");
  return 0;
}
```

H. Author Contributions

Name	Student ID	Contributions
蔡承倢	111550119	Code Implementation
郭芷安	111550139	
陳妍沂	111950031	
顏名柔	111550075	Evaluation of Program
張芷瑜	111550131	
李欣穎	111950020	