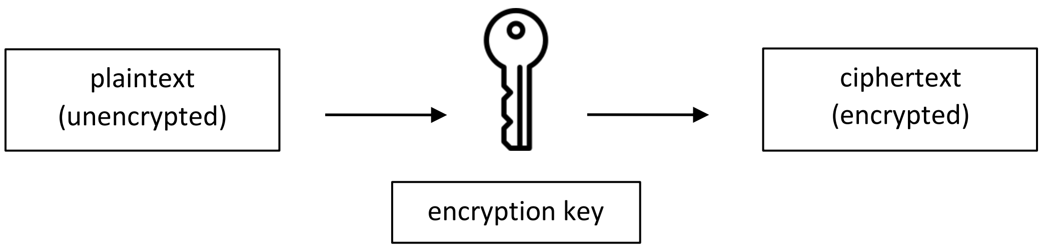
Table of contents

The current password to my computer is 37 characters long. It's a *little* excessive, I know. But when my former IT teacher told me that the longer your password is, the better, I took his advice to heart, sorting all my accounts into tiers and giving them different passwords based on that—memorizing every single one. I won't bore the reader of this exploration with the details of my system (and exactly how long it took me to memorize all those passwords), because first of all, I feel that is a bit of a security hazard in itself, and second of all, the purpose of this exploration is *not* to explore the (in)effectiveness of my old password system, but instead to create a new, better system for creating strong passwords from simple phrases.

Introduction

In order to create an algorithm that creates secure passwords, I looked for inspiration from something I already know about: ciphers. A cipher is an algorithm used for encryption and decryption. Encryption protects information from being compromised and is done by using an algorithm and an encryption key to encode information (plaintext) into an encrypted message (ciphertext), shown in Fig. 1. Decryption is the decoding of the ciphertext back into plaintext (Sander, 2021).

Fig. 1 Simplified process of encryption (candidate created)



The creation of passwords, however, does not require any type of decryption, since the whole idea behind a password is that it is "uncrackable". Even then, using the principles behind encryption can help to create unpredictable and seemingly random passwords, making them difficult for a computer to crack. The aim of this exploration is to create an algorithm that creates strong passwords, using inspiration from popular (though old) encryption methods such as the Caesar and Hill Ciphers, in addition to the use of matrix multiplication.

Caesar Cipher

The first cipher I decided to explore is the Caesar Cipher, one of the earliest and simplest ciphers, used most famously by Julius Caesar (Kumar, 2023). In it, every letter is assigned a number: A is 0, B is 1, etc.; a table relating each letter to a number can be seen in the appendix as Table 3. After being converted into a number, each one is then shifted along by a certain amount, which is called the key. If the key is 2, A (0) would be C (2), B (1) would be D (3), and so on. This type of encryption method is known as a substitution cipher; when given a key, every letter will always be substituted with the same one.

Mathematically, the Caesar Cipher can be expressed by the following equation:

$$f(x) = (x + k) \bmod 26$$

Where x is the plaintext, k is the key (shift), and $f(x)$ is the ciphertext. The modulo (mod) operation returns the remainder when divided by 26, ensuring that if the number goes above 25 (Z), it returns to the beginning as 0 (A). This is because there are only 26 possible characters (the letters of the alphabet), and since the first character starts at 0 (A), there is no corresponding character for the number 26.

Example encryption

The message "password" is given as such:

p	a	s	s	w	o	r	d
15	0	18	18	22	14	17	3

When shifted by a key of two, the equation is:

$$f(x) = (x + 2) \bmod 26$$

For the letter "P", the encryption is done as so:

$$f(S) = (S + 2) \bmod 26$$

$$(15 + 2) \bmod 26$$

$$f(S) = 17$$

Since a value of 17 relates to the letter "R", letter "P" is encrypted as "R". The same process is followed for the rest of the message.

r	c	u	u	y	q	t	f
20	22	19	19	6	15	5	6

The resulting message is therefore "rcuuyqtf". For humans, "rcuuyqtf" is much harder to guess than a common word like "password". Even to a computer, it is likely to take slightly longer to crack, as most attacks follow an algorithm that places words at a higher likelihood over a random assortment of letters.

If the key is known, decrypting the message is done as follows:

$$f(x) = (x - k) \bmod 26$$

However, even if the key is unknown, there are only 25 possible keys (solutions) to the code. This is because a key of 26 or above would loop back to 0 due to the modulus function in the encryption process, resulting in no encryption of the message. Having only 25 possible solutions makes the Caesar Cipher easy for humans—let alone computers—crack.

As the aim of this exploration is creating an algorithm that makes a secure password, the Caesar Cipher alone is not enough. So, I decided to explore different ciphers, in the hopes of finding one that could be of use in creating strong passwords.

Matrix multiplication & the Hill Cipher

The next cipher I looked at was the Hill Cipher. The Hill Cipher is an encryption method that utilizes matrix multiplication between a key and the message. To understand how the Cipher works, it is important to understand matrices and matrix multiplication.

Matrices & matrix multiplication

A matrix stores numbers in a rectangular array with rows and columns (Belcher 374). Each number in a matrix is called an element and can be represented as $a_{r,c}$, where r is the row the element is in and c is the column. Matrices can be used to solve equations in linear algebra and represent geometric transformations, among all its other uses. In this paper, matrices are looked at algebraically, specifically looking at the application of matrices in cryptography. In cryptography, the product between a key matrix and a plaintext matrix results in a ciphertext matrix.

It is important to note that multiplying matrices is not commutative. This means that multiplying matrix A with matrix B ($A \times B$) is not equal to multiplying matrix B with matrix A ($B \times A$). Furthermore, with multiple matrices, the product is calculated from the right of the equation to the left, similar to the composite form of functions.

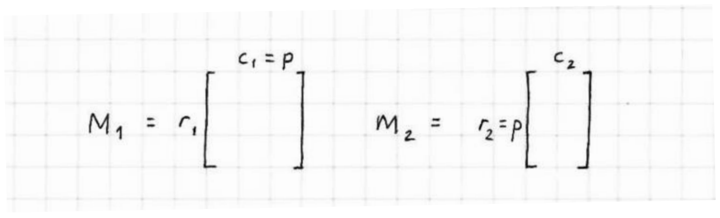
Let us take two matrices: matrix 1 with columns c_1 and rows r_1, and matrix 2 with columns c_2 and rows r_2. When multiplying matrix 1 by matrix 2, the number of columns in matrix 1, c_1, must equal the number or rows in matrix 2, r_2. Essentially,

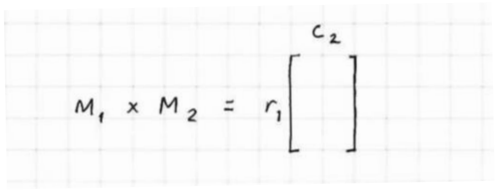
$$c_1 = r_2$$

If the above equation does not hold true, the two matrices are not compatible and cannot be multiplied. Throughout this exploration, the common value of c_1 and r_2 may in some cases be referred to as p .

When multiplying matrix 1 by matrix 2, the resulting matrix will be $r_1 \times c_2$. The following is shown in Fig. 2, with each square representing one element in a matrix.

Fig. 2 Properties of matrix multiplication (candidate created)





A few simple examples of calculating the product between two matrices are shown below:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} = x \begin{bmatrix} a \\ c \end{bmatrix} + y \begin{bmatrix} b \\ d \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix} = x \begin{bmatrix} a \\ d \\ g \end{bmatrix} + y \begin{bmatrix} b \\ e \\ h \end{bmatrix} + z \begin{bmatrix} c \\ f \\ i \end{bmatrix}$$

The same principles can be applied to multiplication between more complex matrices.

Hill Cipher

The Hill Cipher uses these principles of matrix multiplication to encrypt a message with a key (Anand).

Let us take the message to be encrypted as ‘password’ and the key as ‘mkey’ (master key). Using the same substitution method as the Caesar Cipher, ‘p’ is assigned the number 15, ‘a’ 0, and so on.

The password is therefore given as the list of numbers 15, 0, 18, 18, 22, 14, 17, 3, and the key as 12, 10, 4, 24.

We group the numbers into pairs (since we will be using 2x1 matrices) and convert them into a matrix. The password matrices can be combined into one big matrix (2×4 in this case), or separated into four, as shown below:

$$\begin{bmatrix}15\\0\end{bmatrix}, \begin{bmatrix}18\\18\end{bmatrix}, \begin{bmatrix}22\\14\end{bmatrix}, \begin{bmatrix}17\\3\end{bmatrix}$$

Separating the password into 2×1 matrices helps to simplify the process, as instead of multiplying the key by a large matrix, it can be done in steps. The key must be a square matrix (dimensions of $n \times n$), meaning that the key must have the character length of a square number. In this case, the matrix is 2×2 , meaning the key has four elements:

$$\begin{bmatrix}12 & 10\\4 & 24\end{bmatrix}$$

To encode the word, the message matrix is multiplied by the key matrix. For the first matrix, this is done as so:

$$\begin{bmatrix}12 & 10\\4 & 24\end{bmatrix} \times \begin{bmatrix}15\\0\end{bmatrix}$$

$$15\begin{bmatrix}12\\4\end{bmatrix} + 0\begin{bmatrix}10\\24\end{bmatrix}$$

$$\begin{bmatrix}(15 \times 12) + (0 \times 10)\\(15 \times 4) + (0 \times 24)\end{bmatrix}$$

$$\begin{bmatrix}180\\60\end{bmatrix}$$

If the encryption is being sent to another user or stored somewhere else, the encryption process should end here (once the same process is done to the rest of the matrices), as it allows the ciphertext to be decrypted.

To make the ciphertext readable to a user, however, we must first calculate mod 26 of the matrices:

$$\begin{bmatrix}180\\60\end{bmatrix} \bmod 26 = \begin{bmatrix}24\\8\end{bmatrix}$$

The same process is followed for the rest of the matrices, giving the result of:

$$\begin{bmatrix}24\\8\end{bmatrix}, \begin{bmatrix}6\\10\end{bmatrix}, \begin{bmatrix}14\\8\end{bmatrix}, \begin{bmatrix}0\\10\end{bmatrix}$$

When converted back into letters, the following message is received:

“yigkoiak”

When compared to the unencrypted phrase “password”, “yigkoiak” is indecipherable at first glance, and much more difficult for a computer to guess as it is not a word and is therefore more unpredictable.

The Hill Cipher can be encrypted with any other $n \times n$ matrix as well, such as 3×3 . In this case, the message would be split into 3×1 matrices, since the columns of the first matrix must be equal to the rows of the second one. The same logic is followed for larger matrices.

Decrypting a message

When given the key matrix, decrypting the message is done by finding the inverse matrix of the key matrix. The product of the inverse matrix with the ciphertext results in the plaintext, so the original message. However, if the determinant of the matrix is 0, there is no inverse matrix. This means that any key used in the Hill Cipher cannot have a determinant of 0, because otherwise the message is not decryptable.

The determinant of a 2×2 matrix can be found with the following equation:

$$\det \begin{bmatrix}a & b\\c & d\end{bmatrix} = ad - bc$$

Taking the key matrix of “mkey” shown in the example, the determinant can be found as such:

$$\det \begin{bmatrix}12 & 10\\4 & 24\end{bmatrix}$$

$$(12 \times 24) - (10 \times 4)$$

$$248$$

Since the determinant is not 0, there is an inverse to the key matrix. Finding the inverse of a 2×2 matrix is done by swapping the positions of $n_{1,1}$ (a) and $n_{2,2}$ (d), placing a negative sign in front of $n_{1,2}$ (b) and $n_{2,1}$ (c), and dividing all elements by the determinant.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

$$\begin{bmatrix} 12 & 10 \\ 4 & 24 \end{bmatrix}^{-1}$$

$$\frac{1}{248} \begin{bmatrix} 24 & -10 \\ -4 & 12 \end{bmatrix}$$

$$\begin{bmatrix} \frac{3}{31} & \frac{-5}{124} \\ \frac{-1}{62} & \frac{3}{62} \end{bmatrix}$$

For the ciphertext to be decrypted, the matrices calculated before the modulo operation must be used. Using the final text alone is not enough to decrypt the message, as the resulting matrices do not take into account the fact that the modulo operation was used. As an example, both 0 and 27 will result in the same letter. This is why decrypting the message requires the matrix before the modulo operation. However, it may be possible to try all multiples of the elements in the matrix to find the decrypted plaintext, though this is likely to take many tries and be time consuming, meaning that a computer algorithm may be used.

The inverse matrix and the encrypted matrix are multiplied to find the decrypted message.

$$\begin{bmatrix} \frac{3}{31} & \frac{-5}{124} \\ \frac{-1}{62} & \frac{3}{62} \end{bmatrix} \times \begin{bmatrix} 180 \\ 60 \end{bmatrix}$$

$$\begin{bmatrix} (\frac{3}{31} \times 180) + (\frac{-5}{124} \times 60) \\ (\frac{-1}{62} \times 180) + (\frac{3}{62} \times 60) \end{bmatrix}$$

$$\begin{bmatrix} 15 \\ 0 \end{bmatrix}$$

The matrix can be converted to the list of numbers "15, 0", which corresponds to "pa". This is the first two characters of the word "password", showing that using the inverse matrix works to find the original text. The process can be repeated to find the rest of the message.

The Hill Cipher was, for its time, secure. As the matrices became bigger and bigger, it became nearly impossible to find the determinant—and therefore decrypt the messages—by hand. However, with modern-day computers that can solve from thousands to millions of complex mathematical expressions per second, the Hill Cipher is no longer considered secure as an encryption method. Despite that, I realized that the Hill Cipher could still be of use in creating secure passwords.

Creating an algorithm for secure passwords

Passwords, in comparison to encrypted messages that are meant to be decrypted, are a little different. If a user uses the same encryption method to create a password, the user must not share their key, since the password should not be decrypted by anyone—not even the user. If the key is kept secret, a hacker will not be able to calculate the password using Hill Cipher encryption. This means that something similar to a Hill Cipher could be used to make complex passwords that are difficult for humans to guess and computers to crack. Other principles of matrix multiplication can be used in conjunction with the Hill Cipher and Caesar Cipher to try and create an algorithm that creates strong passwords, based off simple phrases that are easily memorable.

Characteristics of a secure password

Before setting out to create an algorithm that creates secure passwords, it is important to understand what makes a password strong.

Many websites require a user to input a password with a lowercase, uppercase, number, and symbol. The reason for this can be explained using the following equation, which helps to account for all possible password possibilities:

$$P = n^r$$

Where P refers to all possibilities, n refers to the number of characters, and r refers to the length of the password.

Using only lowercase letters (26 possible characters) for a 6-character long password, for example, gives a maximum number of this many possibilities:

$$P = n^r = 26^6 = 308915776$$

Using lowercase, uppercase, digits, and symbols (five common ones are chosen for this purpose: !@%*#), adds up to a total of 67 characters. For a 6-character long password, this gives this many total possibilities:

$$P = n^r = 67^6 = 90458382169$$

To hypothesize the amount of time it would take for someone to brute-force a password, the total possibilities should be halved (as an approximation for the number of possibilities the hacker needs to go through before reaching the correct one), and then divided by how many possibilities a computer can try in a second. The number of passwords a brute-force attack can check depends on the attack, but they can be upwards of 1 billion passwords per second. The time it would take for someone to brute-force a six-character password is calculated below, using the total possibilities calculated previously:

$$\frac{90458382169}{2\times1000000000} \approx 45.2 \text{ seconds}$$

At 8 characters, it would take about 56.4 hours to crack a password. The following table outlines the amount of time it would take to crack a password as the length increases.

Even utilizing a variety of characters, a password that is only six characters in length can be potentially cracked in under a minute. Using the same math, the following graph was constructed to highlight the importance of a long password.

Table 1: The relationship between length and time taken to crack a password

	Possible characters			
Length	*26*		*67*	
	hours	*simplified*	*hours*	*simplified*
6	0.0000423	0.152 seconds	0.0126	45.3 seconds
8	0.0290	1.74 minutes	56.4	56.4 hours
10	19.6	19.6 hours	253000	28.9 years
12	13300	1.51 years	1140000000	1300 centuries
14	8960000	10.2 centuries	5100000000000	5.82 million centuries
16	6057000000	6910 centuries	2.29E+16	26.1 billion centuries

As demonstrated in Table 1, both the variety and length of characters are important to creating the largest number of possible passwords. This is verified by the National Institue of Standards and Technology of the US Government, who recommends that passwords be a random mix of between 14 and 16 characters at the very least (Orenstein).

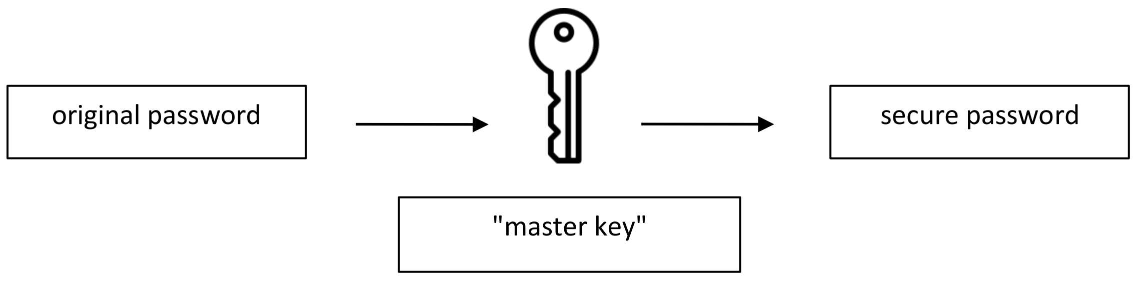
The NIST recommends a random mix of characters, since purely brute-forcing passwords is rarely done today, as hackers are much more likely to try passwords that people commonly use—ones with a pattern or ones with words. To create a strong password, then, requires not only length and a variety of characters, but unpredictability and randomness.

Principles from encryption to create strong passwords

The unpredictability and randomness resulting from encryption comes back to the aim of my algorithm: creating secure passwords. I can first substitute a letter, number, or symbol with the corresponding digit in a table, convert it into a matrix, and then multiply it by a key that is kept hidden. The result that is converted back into a string of characters should be random—or at least random enough to be secure against a human and computer.

Similar to Fig. 1 seen in the introduction, the following figure illustrates the process of my algorithm.

Fig. 3 Simplified process of my algorithm (candidate created)



To ensure the password is secure, using a variety of characters is important. In addition to uppercase (26), lowercase (26), and numbers (10), five common symbols used in passwords were chosen: "!@%*#". This adds up to a total of 67 characters. The conversion from numbers into characters can be seen in Table 3 in the Appendix. The first 30 characters of the conversion chart are seen below.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

p	q	r	s	t	u	v	w	x	y	z	A	B	C	D
15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

I started creating my algorithm by using the same process the Hill Cipher does, but with a 3×3 matrix as a key:

1. Create a key made up of 9 characters
2. Choose the website you want to create a password for
3. Split both words/phrases into matrices
4. Multiply the two matrices
5. Take mod 67 of the resulting matrix
6. Convert back into letters, numbers, and symbols

The algorithm with a key "MasterKey" and the website "managebac", for example, can be done as so:

"MasterKey" = 38, 0, 18, 19, 4, 17, 36, 4, 24

$$\begin{bmatrix} 38 & 0 & 18 \\ 19 & 4 & 17 \\ 36 & 4 & 24 \end{bmatrix}$$

"managebac" = 12, 0, 13, 0, 6, 4, 1, 0, 2

$$\begin{bmatrix} 12 & 0 & 1 \\ 0 & 6 & 0 \\ 13 & 4 & 2 \end{bmatrix}$$

The two matrices are multiplied:

$$\begin{bmatrix} 38 & 0 & 18 \\ 19 & 4 & 17 \\ 36 & 4 & 24 \end{bmatrix} \times \begin{bmatrix} 12 & 0 & 1 \\ 0 & 6 & 0 \\ 13 & 4 & 2 \end{bmatrix}$$

Only looking at the first column of the resultant matrix, the multiplication is done as so:

$$\begin{bmatrix} (38 \times 12) + (0 \times 0) + (18 \times 13) \\ (19 \times 12) + (4 \times 0) + (17 \times 13) \\ (36 \times 12) + (4 \times 0) + (24 \times 13) \end{bmatrix}$$

$$\begin{bmatrix} 690 \\ 449 \\ 744 \end{bmatrix}$$

By using the same method for the rest of the columns of the resultant matrix, the product for the matrix can be found:

$$\begin{bmatrix} 690 & 72 & 74 \\ 449 & 92 & 53 \\ 744 & 120 & 84 \end{bmatrix}$$

Mod 67 is taken from the matrix:

$$\begin{bmatrix} 690 & 72 & 74 \\ 449 & 92 & 53 \\ 744 & 120 & 84 \end{bmatrix} \bmod 67$$

$$\begin{bmatrix} 20 & 5 & 7 \\ 47 & 25 & 53 \\ 7 & 53 & 17 \end{bmatrix}$$

The matrix is then converted back into a list of numbers:

20, 5, 7, 47, 25, 53, 7, 53, 17

Referring to Table 3 in the Appendix, this can be converted back into letters, digits, and symbols, giving the final password as "ufhVz2h2r".

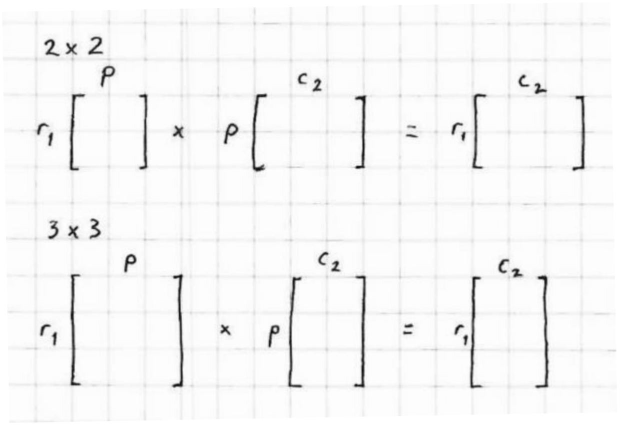
Although this password utilizes a variety of characters and is random enough (since it does not include any words or common patterns), there is one clear problem: it has a character length of only nine. According to the NIST, this is a weak password that can be cracked in hours or days. A strong password must be between at least 14 to 16 characters. This is further shown in Table 1, where the length of a password dramatically increases the amount of time it takes to crack. My next task, therefore, was to find a way to lengthen the final output without having to change the length of the input.

To do this, I began by further exploring the properties of matrices when multiplied, to try and find out whether there is any way to create larger matrices (16 or above) out of smaller ones (9 numbers).

As explained previously, when multiplying matrix 1 by matrix 2, the columns in matrix 1 must be equal to the rows in matrix 2.

With the previously used key matrix, where the number of rows has to equal the number of columns, the ciphertext matrix will always have the same number of rows as the key matrix. The number of columns is then based on the plaintext cipher. This is because the product between two matrices will have the dimensions of $r_1 \times c_2$.

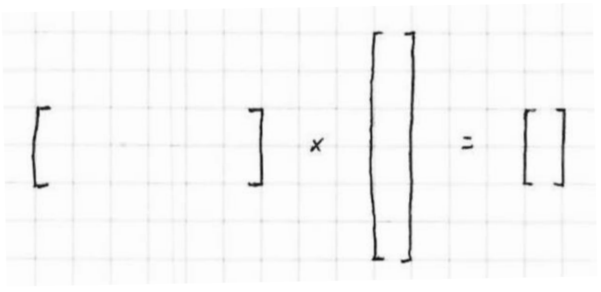
Fig. 4 Multiplication of square matrices with a ciphertext matrix



This means that, given a square key matrix, the ciphertext matrix will always have the same dimensions as the plaintext matrix. This is a problem for my algorithm, as the optimal length for a password is at least 14 characters, and if the plaintext—the name of the website—is fewer than 14 characters, it will always remain this way.

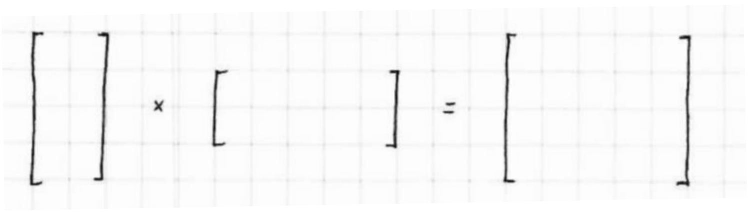
To solve the problem, then, requires the use of a key matrix that is not a square matrix. I started trying out matrices of different sizes, to try and find out when the resulting matrix is larger than two matrices being multiplied.

Fig. 5 Multiplying matrices to create a smaller matrix



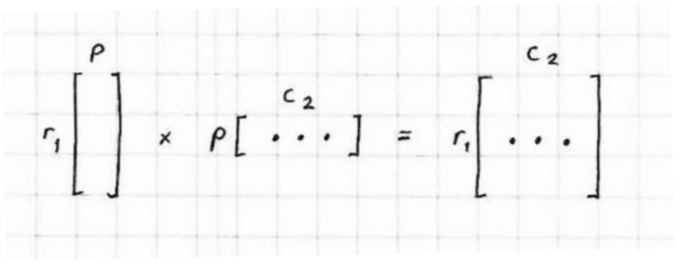
I noticed that when both r_1 and c_2 are smaller than p, the resulting matrix will be smaller than both matrices being multiplied, shown in Fig. 5. If that was the case to get a smaller matrix, the same would be true to get a larger matrix, which is illustrated by Fig. 6.

Fig. 6 Multiplying matrices to create a larger matrix



When r_1 and c_2 are larger than p, the resulting matrix will be larger than either of the matrices. To implement this into my algorithm, I can add a new step in the process where only part of the original key is used to lengthen the password. As an example, if I use a 3×3 matrix, I can use the first column of the key as a "second" key, which has the dimensions of 3×1. The password must then be converted into a $1 \times n$ matrix, where n is the length of the inputted plaintext. The ciphertext plaintext will have the dimensions of $3 \times n$, meaning that the length of the input password has increased threefold. The use of the second key is shown in Fig. 7.

Fig. 7 Use of a second key to create a larger matrix



The length of r_1 is based on the factor by which the user wishes to increase the length of the password. If the inputted plaintext is "managebac" (9 characters), a factor of two should be sufficient, as it brings the character length to 18, which is above 14. Not only does this added step to the algorithm allow for a longer, and therefore more secure, password, but it also adds another level of encryption.

Although the password itself is now hard to crack, I ran into another security problem: the master key. The master key has a length of 9 characters, meaning that it is a possible security hazard. If someone is aware of my password system—which means they know the plaintext—they can then brute force the key until they find something that works. Additionally, the key is supposed to be easily memorable by a human, meaning that it is less likely to use a large variety of symbols, making it even more prone to brute force attacks.

The simplest and most effective solution that I could come up with is changing from a 3×3 key matrix to a 4×4 one. This would make the character length 16, which is considered secure by the NIST. But since I want the key to be a phrase or something else that is easy to memorize, I need to add some other step to create the key into a “random mix” of characters. Using matrix multiplication could work, but it would require the use of another key, making the algorithm unnecessarily complicated. Something simple that could work, however, is the Caesar Cipher. The key for the Caesar Cipher is not as important, though to add an extra level of security, the shift can be the first element of the plaintext. This means that the key will be different for different websites. Having a 4×4 key that is encrypted using the Caesar Cipher allows it to be both a suitable character length and a random mix of characters.

Final algorithm

1. User inputs plaintext (website/account) name
    a. Does not need to be memorized or kept secret
    b. Character length needs to be a multiple of 4, if not, a filler character such as “x” is added until it is a multiple of 4
2. User inputs master key
    a. Needs to be memorized and be kept secret
    b. Can be used for other passwords as well
    c. Character length of 16
3. Both plaintext and master key are converted into a list of numbers, using the table seen in the Appendix (Table 3)
4. The master key is converted into a 4×4 matrix. The plaintext is converted into a $4 \times n$ matrix
    a. n is equal to the plaintext length divided by 4
5. The master key is encrypted using the Caesar Cipher by using a key that is the first element ($n_{1,1}$) of the plaintext matrix.
6. The master key is multiplied by the plaintext matrix and mod 67 is taken
7. The resulting matrix is converted into a list of numbers, and then created into a $1 \times c_2$matrix (called the ciphertext 1 matrix)
    a. c_2 is equal to the number of elements in the plaintext matrix
8. The second key is created as a $r_1 \times 1$ matrix, using some of the elements from the master key (in the same positions)
    a. $r_1 \times c_2 \geq 16$, where $r_1 \in \mathbb{Z}$
9. The second key is multiplied by the ciphertext 1 matrix and mod 67 is taken
10. The resulting matrix is converted into a list of numbers, and then converted back into characters using Table 3 in the Appendix.

Example

Step 1

Plaintext: “managebac” (9 characters)

With filler characters: “managebacxxx” (12 characters)

Step 2

Master key: “ThisMyMasterKey!” (16 characters)

Step 3

Plaintext list: 12, 0, 13, 0, 6, 4, 1, 0, 2, 23, 23, 23

Master key list: 45, 7, 8, 18, 38, 24, 38, 0, 18, 19, 4, 17, 36, 4, 24, 62

Step 4

Master key matrix:

$$\begin{bmatrix} 45 & 7 & 8 & 18 \\ 38 & 24 & 38 & 0 \\ 18 & 19 & 4 & 17 \\ 36 & 4 & 24 & 62 \end{bmatrix}$$

Plaintext matrix:

$$\begin{bmatrix} 12 & 6 & 2 \\ 0 & 4 & 23 \\ 13 & 1 & 23 \\ 0 & 0 & 23 \end{bmatrix}$$

Step 5

$$f(x) = (x + k) \bmod 67$$

$$\left(\begin{bmatrix} 45 & 7 & 8 & 18 \\ 38 & 24 & 38 & 0 \\ 18 & 19 & 4 & 17 \\ 36 & 4 & 24 & 62 \end{bmatrix} + 12 \right) \bmod 67$$

$$\begin{bmatrix} 57 & 19 & 20 & 30 \\ 50 & 36 & 50 & 12 \\ 30 & 31 & 16 & 29 \\ 48 & 16 & 36 & 7 \end{bmatrix}$$

Step 6

$$\begin{bmatrix} 57 & 19 & 20 & 30 \\ 50 & 36 & 50 & 12 \\ 30 & 31 & 16 & 29 \\ 48 & 16 & 36 & 7 \end{bmatrix} \times \begin{bmatrix} 12 & 6 & 2 \\ 0 & 4 & 23 \\ 13 & 1 & 23 \\ 0 & 0 & 23 \end{bmatrix}$$

For the first column of the resultant matrix, the process would be done as follows:

$$\begin{bmatrix} (57 \times 12) + (19 \times 0) + (20 \times 13) + (30 \times 0) \\ (50 \times 12) + (36 \times 0) + (50 \times 13) + (12 \times 0) \\ (30 \times 12) + (31 \times 0) + (16 \times 13) + (29 \times 0) \\ (48 \times 12) + (16 \times 0) + (36 \times 13) + (7 \times 0) \end{bmatrix}$$

The product of the full matrix is calculated to be:

$$\begin{bmatrix} 944 & 438 & 1701 \\ 1250 & 494 & 2354 \\ 568 & 320 & 1808 \\ 1044 & 388 & 1453 \end{bmatrix}$$

When mod 67 is applied to each element, the resultant matrix is:

$$\begin{bmatrix} 6 & 36 & 26 \\ 44 & 25 & 9 \\ 32 & 52 & 66 \\ 39 & 53 & 46 \end{bmatrix}$$

Step 7

List of numbers: 6, 44, 32, 39, 36, 25, 52, 53, 26, 9, 66, 46

$1 \times c_2$ matrix:

$$\begin{bmatrix} 6 & 44 & 32 & 39 & 36 & 25 & 52 & 53 & 26 & 9 & 66 & 46 \end{bmatrix}$$

Step 8

Second key dimensions:

$$r_1 \times c_2 \geq 16, \text{where } r_1 \in \mathbb{Z}$$

$$r_1 \times 12 \geq 16$$

$$r_1 \geq \frac{4}{3}$$

$$r_1 = 2$$

Second key matrix (using elements from the encrypted master key):

$$\begin{bmatrix} 57 \\ 50 \end{bmatrix}$$

Step 9

$$\begin{bmatrix} 57 \\ 50 \end{bmatrix} \times \begin{bmatrix} 6 & 44 & 32 & 39 & 36 & 25 & 52 & 53 & 26 & 9 & 66 & 46 \end{bmatrix}$$

$$\begin{bmatrix} 342 & 2508 & 1824 & 2223 & 2052 & 1425 & 2964 & 3021 & 1482 & 513 & 3762 & 2622 \\ 300 & 2200 & 1600 & 1950 & 1800 & 1250 & 2600 & 2650 & 1300 & 450 & 3300 & 2300 \end{bmatrix}$$

Applying mod 67:

$$\left(\begin{bmatrix} 342 & 2508 & 1824 & 2223 & 2052 & 1425 & 2964 & 3021 & 1482 & 513 & 3762 & 2622 \\ 300 & 2200 & 1600 & 1950 & 1800 & 1250 & 2600 & 2650 & 1300 & 450 & 3300 & 2300 \end{bmatrix}\right) \bmod 67$$

$$\begin{bmatrix} 7 & 29 & 15 & 12 & 42 & 18 & 16 & 6 & 8 & 44 & 10 & 9 \\ 32 & 56 & 59 & 7 & 58 & 44 & 54 & 37 & 27 & 48 & 17 & 22 \end{bmatrix}$$

Step 10

List: 7, 32, 29, 56, 15, 59, 12, 7, 42, 58, 18, 44, 16, 54, 6, 37, 8, 27, 44, 48, 10, 17, 9, 22

Converted into characters: hGD5p8mhQ7sSq3gLiBSWkrjw

From an initial password of “managebac” and a key of “ThisMyMasterKey!”, my algorithm created the password “hGD5p8mhQ7sSq3gLiBSWkrjw”, with a length of 24 characters.

A character length of 24 is well over the recommended 14-16 characters. There is a variety of characters (uppercase, lowercase, and numbers), but as can be seen, a symbol is missing from this password.

Although there is a possibility for a symbol to be a part of the password, it is of course possible that not every password will include one. Certain cybersecurity websites have algorithms that can give an estimate on the amount of time needed to crack a password, shown below in Table 2.

Table 2: Comparing passwords and their password strengths

	Password Strength		
Password	*Site 1* (security.org)	*Site 2* (passwordmonster.com)	*Site 3* (password.kaspersky.com)
password	0 seconds	0 seconds	"Frequently used words"
rcuuyqtf	5 seconds	4 months	"Password too short"
ufhVz2h2r	3 days	100,000 years	"Password too short"
hGD5p8mhQ7sSq3gLiBSWkrjw	4 hundred octillion years	2 billion trillion trillion years	10,000+ centuries

Four passwords were chosen to compare their password strengths, with the final one being the password created using the algorithm. As can be seen, there are drastic differences between the estimates from these sites. This may be because each website uses different algorithms and may consider different computing speeds. Nevertheless, it is clear that the password generated using the algorithm is the most secure, as it takes upwards of millions of years to crack according to the least amount of time estimated by the three sites. This demonstrates that the algorithm can create secure passwords from simple phrases with the use of a master key.

Conclusion

Throughout this exploration, the principles behind popular encryption methods were discussed and then applied to the creation of passwords. These passwords are created from simple phrases that the user is allowed to store somewhere, and then encrypted into secure passwords with the use of a key and by following the steps of the algorithm. This system requires memorizing only the master key, and creates unique, long, and complex passwords that are difficult to brute force.

Even then, the algorithm has some limitations. If done by hand, it is a long and complex process that one can easily make mistakes on. Having to calculate the password every time you need to use it is extremely laborsome. Realistically, it is unlikely for a user to need to input a password every time they want to use an account, as most browsers store such information in a secure way. But when the user session has expired or the user is using another computer, it would be necessary for the user to calculate the password by hand. This problem can be solved by automating the process and storing the script locally, meaning that every time a password needs to be calculated, the user simply needs to input the plaintext and the master key. The computer program would calculate the password for the user, greatly increasing the effectiveness of this system.

One possible flaw in this password system is the master key. If the master key is somehow compromised—meaning that it is no longer secure—the whole password system is rendered useless, and another key will have to be created and used to encrypt all passwords. Luckily, the nature of this password system is that the master key can be changed, and the resulting passwords will all be new. This means that if the master key is compromised, a user must reset all their passwords using a new key. Even if the master key is not compromised, the NIST recommends changing passwords frequently (Orenstein), which can be done easily using this system. A user may also forget their master key, which is problematic as the whole system is based off the master key.

Overall, and even despite its limitations, this password algorithm has the potential to be an effective system. I would much rather use this system than my old one where I had to memorize all my passwords. I feel that there are still improvements that can be made to this system, though when paired with a secure password manager, I feel that the system can work well. Not only has this exploration allowed me to create an algorithm from creating secure passwords, but it has also made me understand the principles behind encryption and decryption, password security, and brute force attacks. With our world increasingly reliant on the technological world for storing all kinds of private information, it is important to understand how safe our data is, and what we can do to increase its security.

Appendix

Table 3: Conversion chart from characters to numbers*

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

p	q	r	s	t	u	v	w	x	y	z	A	B	C	D
15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
30	31	32	33	34	35	36	37	38	39	40	41	42	43	44

T	U	V	W	X	Y	Z	1	2	3	4	5	6	7	8
45	46	47	48	49	50	51	52	53	54	55	56	57	58	59

9	0	!	@	%	*	#
60	61	62	63	64	65	66

*Note: For the Caesar Cipher, use of the table goes from 0 to 25 only

Works Cited

Anand, Daman. “Hill Cipher.” *Geeksforgeeks,* 21/07/2021, https://www.geeksforgeeks.org/hill-cipher/.

Belcher, Paul, et al. “Modelling with matrices: storing and analysing data.” Mathematics: Applications and Interpretations Higher Level Course Companion, Oxford, Oxford University Press, 2019, 374-383.

Kumar, Ashutosh. “Caesar Cipher in Cryptography.” *Geeksforgeeks*, 6/4/2023, https://www.geeksforgeeks.org/caesar-cipher-in-cryptography/.

Orenstein, Gary. “How long should my password be?” *The Bitwarden Blog*, 11/10/2022, https://bitwarden.com/blog/how-long-should-my-password-be/.

Sander, Riya. “Beginner’s guide to the basics of data encryption.” *INFOSEC*, 20/8/2021, https://resources.infosecinstitute.com/topic/beginners-guide-to-the-basics-of-data-encryption/.