A history of cryptography and cryptanalysis

Christopher Colahan
Simpson College

April 12, 2017

Contents

List of Figures **2**

1 Abstract **3**

2 Antiquated Cryptography **3**
2.1 Transposition Ciphers . . . 3
2.2 Monoalphabetic Substitution Ciphers . . . 3
2.2.1 Shift Ciphers . . . 3
2.2.2 Homophonic Substitution Ciphers . . . 3
2.3 Polyalphabetic Substitution Ciphers . . . 4
2.3.1 Vigenère Cipher . . . 4
2.3.2 One Time Pad . . . 4
2.3.3 The Enigma Machine . . . 5
2.4 Frequency Analysis . . . 6
2.4.1 Frequency Analysis on Monoalphabetic Substitution Ciphers . . . 6
2.4.2 Frequency Analysis on Polyalphabetic Substitution Ciphers . . . 6

3 Modern Cryptography **6**
3.1 One Way Hashing . . . 6
3.2 Private Key Cryptography . . . 7
3.3 Public Key Cryptography . . . 7
3.3.1 Certificate Authorities . . . 7

4 Attacks **8**
4.1 Man in the Middle Attacks . . . 8
4.2 Differential Cryptanalysis . . . 8

References **9**

A Frequency Analysis of Shift Cipher in Python 3 **10**

B Frequency Analysis of Vigenère Cipher in Python 3 **11**

List of Figures

1 Vigenère Square . . . 5
2 Frequency of Characters in English Text . . . 6

1 Abstract

Cryptography is used every day to secure private information. Since the invention of the first ciphers millennia ago, ciphers have been continuously created and broken. The history of cryptography and cryptanalysis is examined, starting with the transposition and monoalphabetic substitution ciphers of the ancient Greeks and Romans. Next, frequency analysis is examined as a technique to break monoalphabetic and polyalphabetic substitution ciphers. Finally, modern cryptography is examined.

2 Antiquated Cryptography

2.1 Transposition Ciphers

A transposition cipher rearranges the order of the plaintext to create the ciphertext. Two common transposition ciphers are the rail fence and columnar ciphers. In a columnar cipher, the plaintext is written horizontally with a fixed number of characters on each row. The cipher text is obtained by reading each column of the grid [4]. Transposition ciphers are commonly added to other ciphers to make them more secure.

A technique to determine if a ciphertext is the result of a transposition cipher is to analyze the ciphertext with frequency analysis. If the ciphertext has the same frequencies as English, then the cipher is likely to be a transposition cipher [4].

2.2 Monoalphabetic Substitution Ciphers

A monoalphabetic substitution cipher uses a mapping from one alphabet to another.

For convenience, we define a mapping M: $\{A, B, ..., Z\} \to \mathbb{Z}_{26}$ where $A \mapsto 0$, $B \mapsto 1$, ... $Z \mapsto 25$.

2.2.1 Shift Ciphers

Shift ciphers work by shifting the symbols in the plaintext by an amount. For example, if we are using the English alphabet, then there are $n = 26$ possible symbols. We could then choose some k, $0 < k < n$ for our key. In our notation, this would look like

$$E(p_i, k) = M^{-1}(M(p_i) + k \pmod{n}).$$

To get the deciphering function, we shift backwards:

$$D(c_i, k) = M^{-1}(M(c_i) - k \pmod{n}).$$

[3, pg. 98]

2.2.2 Homophonic Substitution Ciphers

A homophonic substitution cipher is a substitution cipher that maps each symbol to one of more symbols in order to prevent frequency analysis from being used.

For example, suppose we have 100 symbols $S = \{s_1, s_2, ...s_{100}\}$. The letter e would map to approximately 12 or those symbols, but the letter a would only map to about 8 of those symbols.

Before enciphering, each letter is replaced at random with one of the symbols it maps to. This means that each symbol in the ciphertext only appears with a frequency of about 1% [5, pg. 53].

Homophonic substitution ciphers can be broken by using the occurrences of various character combinations. For example, q is a rare character so is would be represented by only one symbol. Additionally, u is represented by three symbols. Since q is always followed by u, if a symbol in the ciphertext is always followed by one of three symbols, the characters probably represent q and u [5, pg. 54]. Similar techniques can be used to determine the rest of the symbols.

2.3 Polyalphabetic Substitution Ciphers

A polyalphabetic substitution cipher uses multiple monoalphabetic substitution ciphers to generate more possibilities for the ciphertext.

2.3.1 Vigenère Cipher

The vigenère cipher uses 26 alphabets to encrypt plaintext. A key is also used that consists of a string of symbols. Given a plaintext symbol p_i and a key symbol k_j, the ciphertext symbol c_i is the character in the i column and j row. Figure 1 shows the square used for encrypting and decrypting using the Vigenère cipher.

The Vigenère Cipher can be mapped to a function easily by inspecting the table. Notice that every character in the A row is the same as the letter in that column. Additionally, every letter in the B row is offset one from the letter in that column, and the C row is offset by 2, and so on.

By observation, we can see that $M(c_i) = M(p_i) + M(k_j)$. Then the encryption and decryption functions for the Vigenère Cipher can be defined as

$$E(p_i, k_j) = M^{-1}(M(p_i) + M(k_j) \pmod{26}) = c_i.$$
$$D(c_i, k_j) = M^{-1}(M(c_i) - M(k_j) \pmod{26}) = p_i.$$

If we know the length of the key is n, then the function could also defined as

$$E(p_i, k_i) = M^{-1}(M(p_i) + M(k_{i \pmod{n}}) \pmod{26})$$
$$D(c_i, k_i) = M^{-1}(M(c_i) - M(k_{i \pmod{n}}) \pmod{26})$$

2.3.2 One Time Pad

The one time pad is a special Vigenère cipher where

- the key is the same length as the plaintext,
- the key is random, and

Figure 1: Vigenère Square

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

- the same key is not used to encrypt two different plain texts.

There is no statistical analysis that can be applied to the ciphertext [2, pg. 393]. Instead of using a Vigenère cipher, modern implementations use the binary XOR operation to combine ciphertext and plaintext since enciphering and deciphering are the same operation and thus much simper.

2.3.3 The Enigma Machine

The Enigma machine was a German encryption/decryption machine used extensively during World War 2. The Enigma machine consisted of a keyboard, a display, a plugboard, multiple scrambling rotors, and a reflector. When a charcter is entered via the keyboard, an electrical signal is sent first through the plugboard. The plugboard allows pairs of letters to be swapped. Next, the signal goes through the scramblers. Each scrambler acts as a substitution cipher. The reflector then reflects the signal back through the scramblers in the reverse order. Finally, the signal is shown on the corresponding letter on the display. The Enigma machine had a possible 10,000,000,000,000,000 settings [5, pg. 136].

2.4 Frequency Analysis

2.4.1 Frequency Analysis on Monoalphabetic Substitution Ciphers

Shift ciphers are easily broken by frequency analysis. Figure 2 the letter frequency from a sample of English text. If a sufficient sample of cipher-text is acquired, The frequency of letters should be a shifted version of Figure 2.

Figure 2: Frequency of Characters in English Text

Letter	Percentage
a	8.2
b	1.5
c	2.8
d	4.3
e	12.7
f	2.2
g	2.0
h	6.1
i	7.0

Letter	Percentage
j	0.2
k	0.8
l	4.0
m	2.4
n	6.7
o	7.5
p	1.9
q	0.1
r	6.0

Letter	Percentage
s	6.3
t	9.1
u	2.8
v	1.0
w	2.4
x	0.2
y	2.0
z	0.1

[5, pg. 19]

2.4.2 Frequency Analysis on Polyalphabetic Substitution Ciphers

Breaking a polyalphabetic cipher is more difficut than a monoalphabetic cipher. For breaking a Vigenère cipher, the key length must first be found. To find the key length of a Vigenère cipher, all lengths of keys are checked. If the correct length n is chosen, every n^{th} letter taken together will form a frequency distribution similar to that of a monoalphabetic cipher [5, pg. 72,73]. This works because every if the key is length n, then every n^{th} character in the key string is the same, thus every n^{th} character in the ciphertext is encrypted using the same alphabet. Once the key length n is found, frequency analysis can be used against all an letters in the ciphertext, for $0 \leq a < n$, with each a representing the a^{th} character in the key [5, pg. 74-76]. This process requires the key to be much smaller than the plaintext to be used successfully, so the one-time pad cannot be attacked using this method.

3 Modern Cryptography

3.1 One Way Hashing

One way hashing is a technique commonly used to store passwords. The idea is to take an input set of plaintext P and map it to an output hash set C using the function $H(p) = c$. There should also be no $H^{-1}(c) = p$ (otherwise it would not be one way). To break an ideal one way hash algorithm, the fastest way should be using brute force.

One way hash functions can be used to authentication. If Alice registers a secret key with bob previously, such as a password, Bob can hash and store the key. When Alice wants to verify her identity to Bob, she sends her key to Bob. Bob then hashes the key and if it matches the stored version, then Alice is authenticated. This is safer than Bob storing the key since it is difficult for Bob (or an attacker) to recover the key from the output of the hash function [4, pg. 52]

Brute force will always work on a hashing algorithm, but it is often infeasible to complete in any meaningful time. Another attack that may be practical is the birthday attack. The *birthday attack* finds 2 input messages m and m' such that $H(m) = H(m')$ [4].

Another attack against one way hashes is the dictionary attack. Using a large list of common passwords, a table of common passwords and their hashes can be created. The hash of the unkown password can then be searched for in the list of hashes [4, pg. 52].

3.2 Private Key Cryptography

In private key cryptography, both users Alice and Bob who wish to communicate securely must have each others secret keys. The security of private key cryptography resides in the length of the key and Alice and Bob keeping the key secret. There are two main types of private key ciphers: stream and block ciphers. Stream ciphers encrypt a constant stream of bytes while clock ciphers encrypt plaintext in descrete chunks.

3.3 Public Key Cryptography

In public key cryptography each user has two keys, a private key k_{pri} and a public key k_{pub}. k_{pub} is used for encrypting messages, while k_{pri} is used for decrypting messages. Unfortunately, every user must have a list of the public keys for all users they wish to communicate with.

When a user Alice wants to send a message to another user Bob, Alice encrypts the message with Bob's public key. Since Bob is the only one with his private key, he is the only one who can decrypt the message, thus providing secure communication if the algorithm is encryption and decryption algorithm is secure.

Typically, a public key algorithm such as RSA is slow and thus not very feasible fo the real time applications in use today. However, public key cryptography algorithms are commonly used to exchange a key for use with private key algorithms such as AES.

3.3.1 Certificate Authorities

A Certificate Authority (CA) solves the problem of keeping track of keys. Instead of having a key for every other person, the CA keeps them all and each user just has the key of the CA. When the user Alice wants to communicate with Bob, Alice asks the CA for Bob's key via a secure channel. Once Alice gets Bob's key, Alice then can communicate securely with Bob without involving the CA again.

4 Attacks

There are sevaral types of cryptanalytic attacks that can be performed on a cryptographic system [1]:

- *ciphertext only attack*: The cryptanalyst knows only the cipher text
- *known plaintext attack*: The cryptanalyst possesses a substantial quantity of corresponding plaintext and ciphertext
- *chosen plaintext attack*: The cryptanalyst can submit an unlimited number of plaintext messages of their choosing and examine the resulting ciphertext

4.1 Man in the Middle Attacks

If an attacker can observe and control all communications between two parties, they might be able to read or interfere with the conversation. For example, assume that Alice and Bob are trying to communicate and Mallory is in the middle and can control all their messages. When Alice sends Bob her public key, Mallory intercepts the key and replaces it with her public key. Then when Bob sends Alice his public key, Mallory intercepts it and replaces it with her public key. Now when either Bob or Alice send a message to each other, Mallory can read or alter the message [4, pg. 48]. An authentication step is needed in the key exchange algorithm to ensure that Bob is really talking to Alice and vice versa.

References

[1] Whitfield Diffie and Matrin E. Hellman. *New Directions in Cryptography. IEEE Transactions on Information Theory*, November 1976.

[2] Michael T. Goodrich and Roberto Tamassia. *Introduction to Computer Security*. 2011.

[3] Thomas W. Judson. *Abstract Algebra*. 2016.

[4] Bruce Schneier. *Applied Cryptography*. 1996.

[5] Simon Singh. *The Code Book*. 1999.

A Frequency Analysis of Shift Cipher in Python 3

```
#find the sift of an ecoded string.
#the idea is to try all shifts (0,1,2,...25)
#and to determine which one is the least difference between
#the standard frequency table and the generated one
#from the ciphertext.
def find_shift(cipher_freq_table):
        min_shift = 0
        min_err = 1
        # try all shifts
        for i in range(0,26):
                #calculate avg freq difference from shifted tables
                err = 0
                for j in range(0,26):
                        #calculate least squares difference
                        standard_freq = standard_freq_table[j]
                        cipher_freq = cipher_freq_table[(j + i) % 26]
                        err += (standard_freq - cipher_freq) ** 2
                err /= 26
                if(err < min_err):
                        min_err = err
                        min_shift = i
        return min_shift
```

B Frequency Analysis of Vigenère Cipher in Python 3

```
#caluclate error for a shift of the tables
def find_err(cipher_freq_table, shift):
        #calculate avg freq difference from shifted tables
        err = 0
        for i in range(0,26):
                #calculate least squares difference
                standard_freq = standard_freq_table[i]
                cipher_freq = cipher_freq_table[(i + shift) % 26]
                err += (standard_freq - cipher_freq) ** 2
        return err/26

#frequency analysis for vigenere cipher
def find_key(ciphertext, standard_freq_table):
        #first, find the length of the key
        #start by assuming a length of 1 character
        key_len = 1
        min_len = 1
        min_err = 1
        cipher_len = len(ciphertext)
        min_freq_len = 100

        #test all key lengths 1 through len(ciphertext).
        #if the length of the key is the same as the length of
        #the ciphertext, then it could be a one-time-pad and we
        #wont be able to find the key set a hard limit. Otherwise
        #frequency analysis will not work on offsets close to the
        #length of the ciphertext
        while key_len < cipher_len and cipher_len // key_len > min_freq_len:
                #collect all n*i elements. they would all be
                #encoded using the same character in the key.
                first_cipher_char = ""
                j = 0
                while key_len * j < len(ciphertext):
                        first_cipher_char += ciphertext[key_len * j]
                        j += 1
                #next, perform frequency analysis to determine if
                #there this is probably the correct key length
                for k in range(26):
                        err = find_err(freq(first_cipher_char), k)
                        #heuristic used here to remove multiples of
                        #smallest key, since it would be the same
```

```
                #key repeated. But a large jump also could
                #also signify that the key is more likely
                #to be correct.
                multiple = key_len % min_len is 0
                large_jump = min_err / 2 > err
                if err < min_err and (not multiple or largejump):
                        min_err = err
                        min_len = key_len
        key_len += 1

#now that the key lenth is known, the key can be obtained by
#performing frequency analysis on each of the key characters
key = ""
for key_i in range(min_len):
        #get all charcters encoded using the same character
        cipher_chars = ""
        j = 0
        while (min_len*j)+key_i < len(ciphertext):
                cipher_chars += ciphertext[(min_len*j)+key_i]
                j += 1
        #find the least error. that is probably the shift
        least_err = 100
        least_shift = 0
        for i in range(26):
                err = find_err(freq(cipher_chars), i)
                if err < least_err:
                        least_err = err
                        least_shift = i
        #add the new key character to the key string
        key += modchar2ascii(least_shift)
return key
```