A History of Cryptography and Cryptanalysis

Christopher Colahan
Simpson College

April 19, 2017

Contents

List of Figures **2**

1 Abstract **4**

2 Antiquated Cryptography **4**
2.1 Transposition Ciphers 4
2.2 Monoalphabetic Substitution Ciphers 4
2.2.1 Shift Ciphers 4
2.2.2 Homophonic Substitution Ciphers 5
2.3 Polyalphabetic Substitution Ciphers 5
2.3.1 Vigenère Cipher 5
2.3.2 One Time Pad 6
2.3.3 The Enigma Machine 6
2.4 Frequency Analysis 7
2.4.1 Frequency Analysis on Monoalphabetic Substitution Ciphers 7
2.4.2 Frequency Analysis on Polyalphabetic Substitution Ciphers 7
2.4.3 Implementing Frequency Analysis in Python 8

3 Modern Cryptography **11**
3.1 One Way Hashing 11
3.2 Private Key Cryptography 11
3.3 Public Key Cryptography 11
3.3.1 Certificate Authorities 12

4 Attacks **12**
4.1 Man in the Middle Attacks 12
4.2 Side-Channel Attacks 12
4.2.1 Timing Attacks 13
4.2.2 Power Consumption Attacks 13
4.2.3 Fault Attacks 13
4.3 Differential Cryptanalysis 13

5 Conclusion **14**

References **15**

A Frequency Analysis of Shift Cipher in Python 3 **16**

B Frequency Analysis of Vigenère Cipher in Python 3 **17**

List of Figures

1 Vigenère Square 6
2 Frequency of Characters in English Text 7

3 Attack on Shift Cipher 8
4 Attack on Vigenère Cipher With Key Length 1 9
5 Attack on Vigenère Cipher With Key Length 2 9
6 Attack on Vigenère Cipher With Key Length 5 10

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 and an example implementation is provided. Finally, modern cryptography and various cryptanalysis techniques are 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 [6]. 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 frequency analysis is performed on the ciphertext and the result is close to the frequency of normal English text, then the cipher is likely to be a transposition cipher [6].

2.2 Monoalphabetic Substitution Ciphers

A monoalphabetic substitution cipher uses a mapping from one alphabet to another. For convenience converting between the English alphabet and $\mathbb{Z}_{26}$, a mapping M: $\{A, B, ..., Z\} \to \mathbb{Z}_{26}$ is defined where $A \mapsto 0$, $B \mapsto 1$, ... $Z \mapsto 25$. Then M^{-1} is defined to be $0 \mapsto A$, $1 \mapsto B$, ... $25 \mapsto Z$.

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:

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

[5, 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% [7, 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 [7, 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

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

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

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

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.

There is no statistical analysis that can be applied to the ciphertext [4, pg. 393]. This is because if each key is equally likely to occur, then each ciphertext is equally likely to occur. Take, for example, the plaintext $p_1 = ATTACK$. If p_1 is then encrypted using the key $k_1 = ZGRBNL$ then $E(p_1, k_1) = ZZKBPV$. By looking at the Vigenère square, a new key k_2 can be generated for any p_2 of the same length as p_1 such that $E(p_2, k_2) = ZZKBPV$. So if the desired plaintext is $p_2 = DEFEND$, the corresponding key $k_2 = WVFXCS$.

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 has a possible 10,000,000,000,000,000 settings [7, 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

[7, pg. 19]

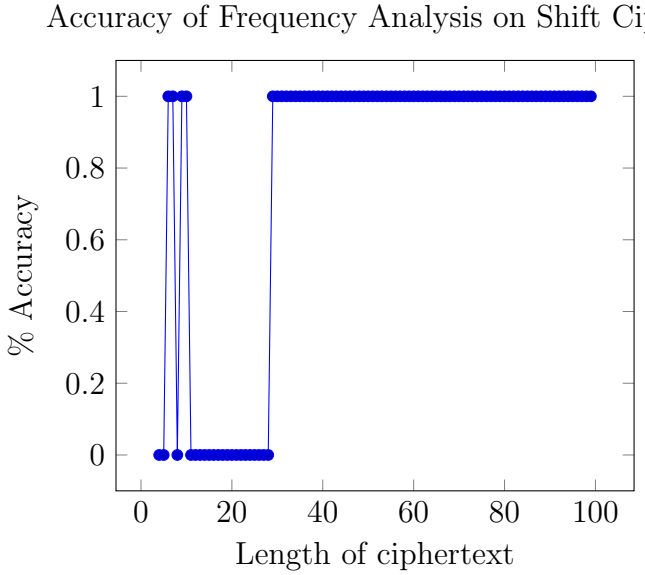
2.4.2 Frequency Analysis on Polyalphabetic Substitution Ciphers

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 [7, 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 [7, 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.

2.4.3 Implementing Frequency Analysis in Python

Implementing both the shift cipher and the Vigenère cipher, as well as attacks on both ciphers was mostly straight forward. The equations for E and D were used from their respective sections and code was created for performing frequency analysis. One difficulty encountered, however, is the performance of the frequency attack against the Vigenère cipher, which takes many times longer to complete than the shift cipher. This made finding and fixing problems difficult.

Figure 3: Attack on Shift Cipher



As can be seen in Figure 3, a minimum of about 30 to 40 characters of ciphertext are needed to obtain the key.

Looking at Figure 4, the frequency analysis accuracy of the Vigenère Cipher with a key length of 1 is near identical to the frequency analysis of the shift cipher, since they are essentially the same cipher.

From Figure 5, it is apparent that there is a major increase in the length of ciphertext that was required for a successful attack from using keys of length 1 to keys of length 2. The minimum required jumped from approximately 30 to approximately 250, although there are still some key lengths that the attack was unsuccessful against that were above 250.

Figure 6 demonstrates that the longer the key is, the required cipher is longer for a successful attack. At a key length of 5, the attack needed at minimum approximately 700 characters.

Figure 4: Attack on Vigenère Cipher With Key Length 1

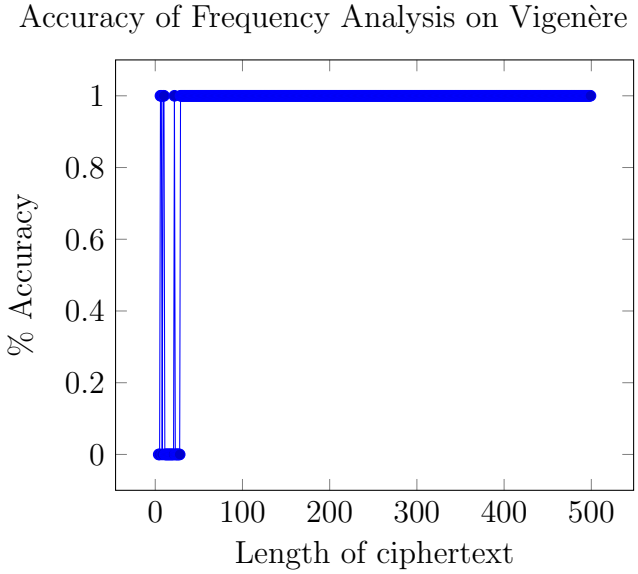


Figure 5: Attack on Vigenère Cipher With Key Length 2

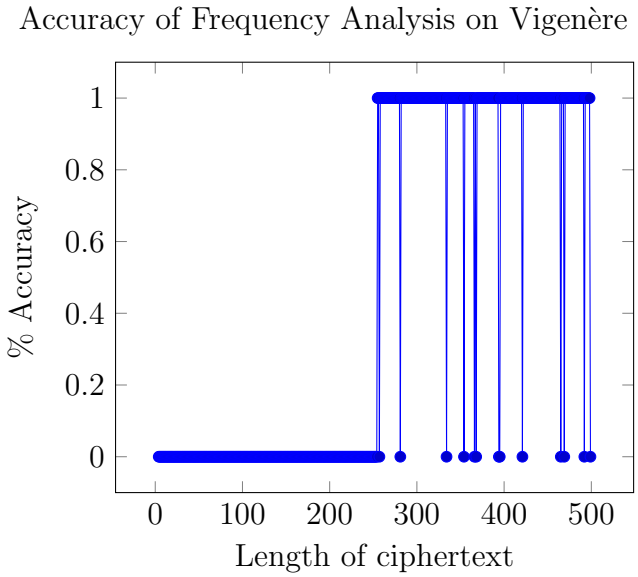
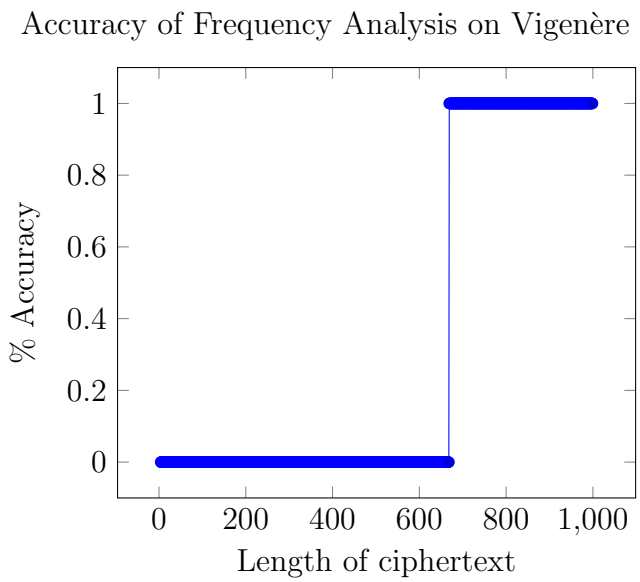


Figure 6: Attack on Vigenère Cipher With Key Length 5



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 $H(p) = c$. Computing $H^{-1}(c) = p$ should be much more computationally difficult that computing $H(p) = c$. 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 [6, 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 $H(m) = H(m')$ [6].

An attack against password 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 [6, pg. 52].

Another attack against hashes is the length extension attack.

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 ciphers covered earlier in this paper are all private key algorithms. 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 continuous stream of bytes while clock ciphers encrypt plaintext in descrete chunks.

3.3 Public Key Cryptography

Public key cryptography was first proposed in 1976 [5, pg. 100][2]. 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 cryptosystem 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.

The RSA cryptosystem works as follows: First, choose two large (150 digit) prime numbers p and q. Then compute $m = (p-1)(q-1)$ and $n = pq$. Next, choose a random number E such that $\gcd(E, m) = 1$. Then using the Euclidean algorithm, find a number D such taht $DE \equiv 1 \pmod{m}$. The numbers n and E can then be made public.

Suppose Bob wants to send a number $x < n$ to Alice, having E and n from Alice. Then Bob computes $y = x^E \pmod{n}$ and sends y to Alice. Alice can then compute $x = y^D \pmod{n}$ [5, pg. 101]. Since text can be represented as a large number, then messages can be sent using this system.

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 [2]:

- *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 [6, pg. 48]. An authentication step is needed in the key exchange algorithm to ensure that Bob is really talking to Alice and vice versa.

4.2 Side-Channel Attacks

Side channel attacks use information from side channels. The information gathered from a side channel is anything that is neither plaintext to be encrypted nor ciphertext from the resulting encryption process. Some examples of information that can be gathered during a

side channel attack are timing information (how long various operations take to complete), radiation, and power consumption. There are three main types of Side channel attacks: Timing, Power consumption, and Fault attacks[1, pg. 2-3].

4.2.1 Timing Attacks

Timing attacks learn about secret keys being used in a cipher by using the information about how long an operation takes to complete [1, pg 3]. For example, a timing attack was performed on GnuPG that could recover, on average, 96.7% of the bits in the secret key by using a cache timing attack [3, pg. 719]. The exact implementation of the cryptographic algorithm needs to bee known ahead of time. Implementation details such as using exponentiation by squaring in GnuPGP or using the Chinese Remainder Theorem in RSA have been attacked since they involve iterative operations or varying length. Using a chache template attack, an attacker can map out function calls and even the order of blocks of instructions called during encryption. This can give the attacker most of private key, but not all of the key can always be recovered since noise in the system such as the the operating system or other programs may use the cache often. Timing attacks can be combated by using constant time algorithms such that every iteration of an operation takes approximately the same amount of time to complete.

4.2.2 Power Consumption Attacks

Power consumption attacks determine an encryption key by monitoring the power consumption of a unit while it encrypts plaintext [1, pg. 6]. Since each operation on the microprocessor consumes a certain amount of electricity, the order of the operations executing in the microprocessor can be determined and the key can be retrieved. If the voltages over time are plotted out in a graph, an attacker could read the operations performed. Since there are many other factors such as radiation or random changes in voltage the key can not always be completely recovered, but the remaining unknown parts of the key can be recovered via brute force depending on the amount of key left to find. This is similar to timing attacks.

4.2.3 Fault Attacks

Fault attacks analyze the data from when a fault is created in the system. Faults can be created by tampering with the power voltage, tampering with the system clock, or applying radiation to the circuitry. Differential fault attacks are performed by encrypting the plaintext twice, once normally, and once with a fault, and comparing the resultant ciphertexts for differences [1, pg. 7].

4.3 Differential Cryptanalysis

Differential cryptanalysis is a chosen plaintext attack technique on iterative block ciphers. The attack works by analyzing ciphertext/plaintext pairs encrypted with the same key and the changes propagated in the ciphertext by changing the plaintext. As the differences are analyzed, probabilities can be assigned to keys for how likely each key are to be the correct

key. Differential cryptanalysis requires many pairs to be analyzed, so the amount of memory and time make this attack is not practical for most ciphers[6, pg. 285-290].

5 Conclusion

Ciphers have been used throughout history and have continued to evolve. Ciphers and attacks against them have grown in magnitudes of complexity since the introduction of programmable computers. The shift cipher is simple to understand and implement, but is also simple to break using frequency analysis. The Vigenère cipher is somewhat more complex to implement and attack, but is susceptible to a modified version of the same frequency analysis attack. The effect of the accuracy of frequency analysis on the Vigenère cipher with various key lengths was examined. One way hasing and public and private cryptography where covered. Additionally, man in the middle, side channel, and differential attacks were covered.

References

[1] Hagai Bar-El. *Introduction to Side Channel Attacks*.

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

[3] Katrina Falkne and Yuval Yarom. Flush+reload: A High Resolution, Low Noise, L3 Chache Side-Channel Attack. In *Proceedings of the 23rd USENIX Security Symposium*, pages 719–732, August 2014.

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

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

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

[7] 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 chi-squared
                        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 chi-squared
                standard_freq = standard_freq_table[i]
                cipher_freq = cipher_freq_table[(i + shift) % 26]
                cipher_err = (standard_freq - cipher_freq) ** 2
                cipher_err /= standard_freq
                err += cipher_err
        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
```