South Dakota School of Mines and Technology

Cryptography, Fall 2021

CSC 512 - M01

Portfolio

Sherwyn Braganza

Classical Cryptosystems

- Shift Ciphers

  One of the simplest Cryptosystems out there, **Shift Ciphers** are formed by taking the numerical value of each character in the plaintext and then adding a constant value (**key**) to it, mod n (where n is the size of the character set). The orders are then translated back to their character values.

  What does order mean?
  It is the position/rank of the character w.r.t to the whole set - 1. In the set of the English Alphabet, **A** will have a rank of 0. Similarly **Z** will have a rank of 25.

  As most things are best explained by an example, consider the plaintext -

  “I LOVE CRYPTOSYSTEMS”

I	L	O	V	E	C	R	Y	P	T	O	S	Y	S	T	E	M	S
8	11	14	21	5	2	17	24	15	19	14	18	24	18	19	4	12	18

  The order of each of its characters is as follows.

  Adding a **key = 5**, and reverting numbers to characters, we get the following encryption.

I	L	O	V	E	C	R	Y	P	T	O	S	Y	S	T	E	M	S
8	11	14	21	5	2	17	24	15	19	14	18	24	18	19	4	12	18
13	16	19	0	10	7	22	3	20	24	18	23	3	23	24	9	17	23
N	Q	T	A	K	H	W	D	U	Y	S	X	D	X	Y	J	R	X

- # Affine Ciphers

The Affine Cipher is another simple Cryptosystem. It basically stems from regular **Shift Ciphers** and is governed by the following equation:

$$c = \alpha * p + \beta \quad \mod N$$

Where

- **p** is the numeric value of the plaintext alphabet.
- **β** can be any arbitrary number.
- **α** is an any number who's **gcd(α, N)** ≡ **1**
- **c** is the outputted cipher text letter
- **N** for the set of the English alphabet is **26.**

This system can be extended beyond the English alphabet system and can include other symbols too. **N** correspondingly needs to change to encompass the new range.

Few key things to keep in mind about this encryption system is

- It encrypts one character at a time; thereby the complexity of this encryption and decryption function is linear.
- Spaces are not encrypted but simply skipped over.
- It is susceptible to frequency attacks as every alphabet in the plaintext has only one ciphertext value/encryption.
- The key for this cryptosystem is 2 character combination of **(α, β).**

Decryptions of Affine Ciphers work in a similar way. Except the new function, the decrypting function is given by:

$$p = \alpha^{-1} * (c - \beta) \quad \text{mod } N$$

Where α^{-1} is the the modular inverse of α.

Attacks on Affine Ciphers

As mentioned above, attacks on Affine Ciphers can be carried out using a simple frequency analysis. This works because each character has a unique cipher text encoding.

This link contains the code for a simple program that simulates the encryption and decryption of an affine cipher as well the case of an attack on it :

Affine_Cipher_Encrypt_Decrypt

- # Vigenère Ciphers

Vigenère Ciphers work in a similar manner as Shift Ciphers.

1) You first start with a **key (K)** that is a simple plaintext word (ex. sherwyn).
2) Write out the plaintext in a single line.
3) Directly below the plaintext, corresponding to each letter of the plaintext, write out the key in a cyclical fashion (as shown below)

```
T h e   q u i c k   b r o w n   f o x   j u m p e d   o v e r
s h e   r w y n s   h e r w y   n s h   e r w y n s   h e r w
```

4) The cipher text value/character is the sum of the plaintext character and the character of the key right below it, mod 26.
   In other words, each individual character encoding is treated as a Shift Cipher with the **β** value equal to character right below it.

The Cipher Text version of the above shown example is

```
l o i h q g p c i v f s l s g e n l i n r v v z v n
```

Decryption of Vigenère Ciphers works in a similar way as encryption. You lay down the a cyclical version of the key against each character of the cipher ciphertext and then take the sum of it mod 26 (since in this case we are working in the set of the Common English Alphabet).

Attacks on Vigenère Ciphers

Attacks on Vigenère Ciphers involve broad steps:

1) Finding the key length.
2) Finding the key.

The key length is obtained by multiplying the

- # ADFGX Ciphers

ADFGX ciphers belong to the class of Matrix Ciphers. You start out with a predetermined matrix encoding each letter in the English Alphabet to its **row-column combination**. Consider this predetermined **ADFGX matrix.**

	A	D	F	G	X
A	p	g	c	e	n
D	b	q	o	z	r
F	s	l	a	f	t
G	m	d	v	i	w
X	k	u	y	x	h

From this we know that **k** encrypts to the **row-column** combination of ***XA.***

***** j and I are treated as the same character and encrypt to the same character in the matrix.

Encryption

The Encryption process contains 2 steps and involves the predetermined ADFGX matrix and a **keyword**.

The encryption process is best described with an example.

Consider encrypting the plaintext: ***Kaiser Wilhelm***

With the keyword: ***RHEIN***

A simple ADFGX row-column transformation of the plaintext yields the following ***transformation text***.

XA FF GG FA AG DX GX GG FD XX AG FD GA.

This is then arranged in a matrix headed by the keyword, as shown below. The transformation text is arranged from left to right, wrapping around when it reaches the last character of the keyword.

R	H	E	I	N
X	A	F	F	G
G	F	A	A	G
D	X	G	X	G
G	F	D	X	X
A	G	F	D	G
A				

The columns are then reshuffled so that the keyword headers are in alphabetical order (shown below).

E	H	I	N	R
F	A	F	G	X
A	F	A	G	G
G	X	X	G	D
D	F	X	X	G
F	G	D	G	A
				A

Each individual column is then outputted (ignoring the keyword header), from top to bottom, to give you the encrypted ciphertext:

FAGDFAFXFGFAXXDGGGXGXGDGAA.

Few interesting things to note about this is that ciphertext is built as a permutation of just 5 letters - **A, D, F, G and X.**

A, D, F, G, X were specifically chosen because their Morse Code equivalents are easily distinguishable from each other to prevent confusing one with the other. Which leads us to the inference that this was probably used in WW1, when Morse code was one of the most utilized means of communication.

Decryption

Decryption is basically done by following the encryption algorithm from bottom to top.

1) Our initial aim is to calculate the number of rows in each “header column”. TO achieve this, we do an integer division of the size of the ciphertext by 5. This gives us the mean # of rows for each column.

Performing this step on our above example, we get the mean # of rows to be 5. This leaves us with the following number of columns.

E = 5

H = 5

I = 5

N = 5

R = 5

2) Now we take the modulo of the size of the ciphertext with 5

sizeof(ciphertext) % 5 = 1

The value we get tells us how many header columns, with respect to the original keyword ordering needs to get padded with an extra row value.

In our example here, the **'R'** header column, will get padded with an extra row resulting in the final row count for each column being show as below.

E = 5

H = 5

I = 5

N = 5

R = 6

Considering all the other cases in which **sizeof(ciphertext) % 5** returns 2, 3, 4; the modified rows numbers for each column is shown in following table

sizeof(ciphertext)%5 =	2	3	4
E	5	5	5
H	6	6	6
I	5	6	6
N	5	5	6
R	6	6	6

3) The next step is to grab characters corresponding to the column size, in the rearranged alphabetical order of the keyword (the polar opposite to the last step of the encryption method).

This leaves you with the matrix looking like this

E	H	I	N	R
F	A	F	G	X
A	F	A	G	G
G	X	X	G	D
D	F	X	X	G
F	G	D	G	A
				A

4) The next step is to reshuffle the columns to resemble the original keyword.

R	H	E	I	N
X	A	F	F	G
G	F	A	A	G
D	X	G	X	G
G	F	D	X	X
A	G	F	D	G
A				

5) Print from this matrix, reading from left to right (starting from row 1), with wrap arounds. Performing this step on the example we are discussing, leaves us with this ***transformation text.***

XA FF GG FA AG DX GX GG FD XX AG FD GA.

6) Our job now is to look at our predetermined ADFGX Matrix and write the English alphabet character corresponding to each 2 letter combination.

XA FF GG FA AG DX GX GG FD XX AG FD GA.

The source code for a simple ADFGX encryption decryption simulator in python can be found at this link: **ADFGX Simulator**.
It does used a preset ADFGX matrix which is the same one discussed in the example. It has an Encryption and Decryption Mode: both modes require you to provide the key.

- # Block Ciphers

Block Ciphers belong to the class on Matrix Ciphers, i.e. a matrix of characters/ values is used to encrypt a given plaintext. In Block Ciphers, the key is a $m_x m$ matrix/block, that is filled with integers mod 26 (if using the English Alphabet as you working set). Integers should be chosen in such a fashion that the resulting **key matrix** is not singular.

An example key would be:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 11 & 9 & 8 \end{pmatrix}$$

Encryption

Matrix multiplication mod 26 is performed using the **key matrix**, on a plaintext vector of length m (vector representing the number encoding/order of the English character/letter). The resulting/matrix vector is the numerical encoding/ ordering of the ciphertext.

Decryption

Decryption of block ciphers follow the similar rules. The ciphertext is multiplied by the **inverse of the key matrix** to produce the plaintext.

Few things to note about Block Ciphers are

- It encrypts ‘m’ characters at a time.
- The key has to always be a square matrix.
- Since Block Ciphers encrypt multiple characters at a time, a change in one character in the plaintext would result in a multi-character change in the ciphertext.

Due to this design benefit, Block Ciphers are kind of resilient to simple frequency analysis ciphertext only attacks. It however does succumb to known ciphertext attacks.

The python code for a simple Block Cipher Simulator (Encrypt/Decrypt) can be found at this link: **Block Cipher Emulator**

- # Attacks on Cryptosystems

  There are 4 broad categories for classifying attacks on Cryptosystems. The end goal of all attacks is to find the key.

1) **Ciphertext Only Attacks** - The challenger has a copy of the ciphertext only.
2) **Known Plaintext Attacks** - The challenger has copy of the plaintext and the ciphertext it encrypts to.
3) **Chosen Plaintext Attacks** - The challenger has temporary access to the encryption side of the cryptosystem and uses this advantage to deduce the key.
4) **Chose Ciphertext Attacks** - The challenger has temporary access to the decryption side of the crypto system and uses this advantage to deduce the key.

- # Frequency Analysis

One of the most common methods that fall under the category of **ciphertext only attacks** is Frequency Analysis.

As is pretty evident from the name itself, this method involves doing a sweep of the ciphertext and recording the occurrences of either the characters by them self or two letter combinations. Higher order combinations (3 letter and 4 letter) are pretty inconsistent in terms of giving valid uniform results.

The next step is to compare them with the well known letter frequencies in common literature. The table below shows frequency ratios of the single letters in an common piece of English Literature.

a	b	c	d	e	f	g	h	i	j
.082	.015	.028	.043	.127	.022	.020	.061	.070	.002
k	l	m	n	o	p	q	r	s	t
.008	.040	.024	.067	.075	.019	.001	.060	.063	.091
u	v	w	x	y	z				
.028	.010	.023	.001	.020	.001				

Cryptosystems that have a direct correspondence between ciphertext character and plaintext character, i.e. systems in which a plaintext character is encrypted to a unique ciphertext character and vice-versa, are the easiest to break (find the key) using this method. All you got to do in these cases is just compare calculated character frequencies in the ciphertext and compare them to the character frequencies in the above mentioned table. This leaves us with the knowledge of correspondence between each ciphertext and plaintext character, which can easily be used to solve for the key.

Shift Ciphers, Affine Ciphers and Vigenère Ciphers are prime targets for these type of attacks.