A lesson in classical cryptography

Matthew Campagna
Cristian Ilac

Date: 9 January 2014

Outline

- What is Cryptography?
- The Ceasar cipher
- Monoalphabetic Cipher
- What is Cryptology?
- Freqeuncy analysis using *monocount*.
- Transposition Cipher
- Polyalphabetic Cipher
- Index of Coincidence

1 What is cryptography

We are not intending to provide a full history of cryptography, see the Code Book and the Puzzle Palace for two separate views. The Code Book provides a global history of cryptography as it has evolved through many cultures and civilizations; and the Puzzle Palace provides the historical development of cryptanalysis in America.

'*Crypto* comes from the Greek word 'krpyte' meaning hidden or vault; and *graphy* comes from 'graphik' meaning writing. For us we can define cryptography for our purposes.

Definition 1.1. *Cryptography* is the science of writing messages that no one except the intended receiver can read.

In general we use Cryptography in information systems to provide some critical security services we have in the physical world.

Definition 1.2. *Confidentiality* is the property that the contents of protected messages cannot be read by un-authorized recipients.

Definition 1.3. *Data-Integrity* is the property that the contents of a protected message cannot be altered without detection.

Definition 1.4. *Authentication* is the property that one entity can verify the entity of another is who/what they claim to be.

Definition 1.5. *Non-repudiation* is the property that prevents an entity from denying a previous actions, commitments or messages.

We will start with Confidentiality - keeping a secret secret.

2 Monoalphabetic Ciphers

Definition 2.1. *Monoalphabetic Ciphers* are ciphers that use a single letter substitution. The most famous of these is the Ceasar Cipher

2.1 The Ceasar Cipher

Definition 2.2. *Ceasar Cipher* is a simple monoalphabetic cipher that replaces the intended character by a three character shift.

```
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 -

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

CAESAR BEWARE OF BRUTUS
FDHVDUCEHZDUHCRICEUXWXV
```

Well, now that we know what a Ceasar cipher is we can just shift it back three characters.

What if it is a randomly selected shift?

```
XLDSPXLDTNCKTCKQEY
```

Like any secret, you might want to know, how good is the security of my message? You might also want to know, how do you measure security?

Well, you could try just random shifts of this message.

```
XLDSPXLDTNCKTCKQEY
YMETQYMEUODLUDLRFZ
ZNFURZNFVPEMVEMSG
 OGVS OGWQFNWFNTHA
APHWTAPHXRGOXGOUIB
BQIXUBQIYSHPYHPVJC
CRJYVCRJZTIQZIQWKD
DSKZWDSK UJR JRXLE
ETL XETLAVKSAKSYMF
FUMAYFUMBWLTBLTZNG
GVNBZGVNCXMUCMU OH
HWOC HWODYNVDNVAPI
IXPDAIXPEZOWEOWBQJ
JYQEBJYQF PXFPXCRK
KZRFCKZRGAQYGQYDSL
L SGDL SHBRZHRZETM
MATHEMATICS IS FUN
NBUIFNBUJDTAJTAGVO
OCVJGOCVKEUBKUBHWP
PDWKHPDWLFVCLVCIXQ
QEXLIQEXMGWDMWDJYR
RFYMJRFYNHXENXEKZS
SGZNKSGZOIYFOYFL T
TH OLTH PJZGPZGMAU
UIAPMUIAQK HQ HNBV
VJBQNVJBRLAIRAIOCW
WKCROWKCSMBJSBJPDX
XLDSPXLDTNCKTCKQEY
```

If you consider the above, you will notice there are 26 possible (non-trivial) encryptions of the message. Each of these can be considered a separate *key*, where the key defines the amount to shift.

How did we discover the plaintext?

We technically decrypted the message under each of the 26 keys, and then examined the output, and picked out the one that *looked like* plaintext.

There is a general principle in cryptography attributed to Auguste Kerckhoff that states that a cryptographic systems should remain secure against an adversary that knows everything about the system except the cryptographic key. For everything we consider, we will assume that our adversary knows everything but the key.

Definition 2.3. *A symmetric key cipher* is a function E that takes a fixed length key parameter k, and a plaintext value pt and outputs a ciphertext value $E(k, pt) = ct$, and an inverse function D that takes a fixed length key parameter k and a ciphertext value ct, such that $pt = D(k, E(k, pt))$.

A key comes from a typically finite set called the *keyspace*. And we can then ask how big is the keyspace? If we just need to try all the keys, then we do that under the decrypt function and see if we get anything that looks like plaintext.

The size of the keyspace is one way to measure the security, and is infact an upper bound on the security.

Definition 2.4. The *strength* of a cryptographic algorithm is the amount of *work* required to break the security provided by that algorithm using the best known attack.

Well, for the cipher above the key space is the number of non-trivial shifts. 26 is not a big number!

Definition 2.5. *Cryptanalysis* is the study and practice of breaking security properties provided by cryptogaphic methods.

In general there are a variety of advantages that an adversary might have when performing a cryptanalysis attack. For symmetric key ciphers under a fixed key we can categorize these.

> *Ciphertext only* attacks are where the adversary only sees encrypted plaintext and attempts to attack the system.
>
> *Known plaintext* attacks are where the adversary knows pairs of encrypted plaintext and the corresponding ciphertexts.
>
> *Chosen plaintext* attacks are where the adversary can get arbitrary selected plaintext values encrypted.
>
> > *Adaptive plaintext* attacks are a variation of chosen plaintext attacks where the adversary can modify plaintext value to be encrypted based on previous chosen plaintext ciphertext pairs.

There are other methods to defeating the security of cryptosystem the most common of which is implementation errors, most likely followed by coercion.

It is worth closing the circle here with one more definition.

Definition 2.6. *Cryptology* is the union of the two fields.

2.2 Random Monoalphabetic Ciphers

Definition 2.7. A random monoalphabetic cipher is a random permuation of the alphabet space.

```
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 -

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

So, we might ask, what is the key here? And how big is the keysize?
Well, it can be any permutation on 27 characters.

$$27! \approx 2^{93}$$

```
for each permutation p on the alphabet{

    computed_plaintext = decrypt(p, ciphertext){
    if(is_english(computed_plaintext))
        print computed_plaintext, p
    }
}
```

How big is that? Well how many permutations do you think you could write down in a second? How about a computer? Suppose you could do 4 Billion a second (roughly 2^{32}), how long would it take to try all the guesses?

$$2^{93-32} = 2^{61} \text{ seconds} = 73,117,802,169 \text{ years}$$
A long time.

Ideas?
Let's take a look of at a couple of messages.

```
Z VSGOAHDDGOLSXGP RGAHVG VSGZTHVLGDSHXGP RGAHVNTVY

Y GV LGZ GCWSRSGLWSGXHLWGAHMGDSHYGZGTVOLSHYGCWSRSGLWSRSGTOGVGXHLWGHVYGDSHFSGHGLRHTD

CTOY AGTOGLWSGRSCHRYGM IGZSLGP RGHGDTPSLTASG PGDTOLSVTVZGCWSVGM IYGWHFSGXRSPSRRSYGL GLHDN

CSGAIOLGIOSGLTASGCTOSDMGHVYGP RSFSRGRSHDTJSGLWHLGLWSGLTASGTOGHDCHMOGRTXSGL GY GRTZWL

XS XDSGCW GLWTVNGLWSMGNV CGSFSRMLWTVZGHRSGHGZRSHLGHVV MHVBSGL GLW OSG PGIOGCW GY
```

3 Cryptanalysis

If I gave you an hour or so, do you think I you could recover the plaintext or the key?
What did we do in the first case of the rotation of the alphabet?
Well we tried all the keys and wrote them out, and then we *recognized* the plaintext.
This seems obvious to us, but is really a sophisticated operation of pattern recognition.
You could count the letters?
Well, you could model your plaintext on letter frequencies you see in text.

Definition 3.1. *A frequency histogram* is a list letters and a count of its frequency

From a frequency count (or a monoalphabetic count, count of single character occurences) we can build a frequency distribution table.

```
Monocount campagna$ ./monocount infile.txt
A 47063     B 8139      C 13225     D 27483
E 72879     F 13154     G 12120     H 38359
I 39782     J 622       K 4634      L 21538
M 14922     N 41309     O 45116     P 9452
Q 655       R 35957     S 36770     T 52395
U 16216     V 5065      W 13835     X 665
Y 11849     Z 213       - 125771
```

```
A 0.066362  B 0.011477  C 0.018648  D 0.038753
E 0.102764  F 0.018548  G 0.017090  H 0.054089
I 0.056095  J 0.000877  K 0.006534  L 0.030370
M 0.021041  N 0.058248  O 0.063616  P 0.013328
Q 0.000924  R 0.050702  S 0.051848  T 0.073880
U 0.022866  V 0.007142  W 0.019508  X 0.000938
Y 0.016708  Z 0.000300  - 0.177345

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

This is a frequency on A Tale of Two Cities by Charles Dickens.

What that means is that we can do a frequency count on our ciphertexts

```
A 9         B 1         C 11        D 13
E 0         F 4         G 94        H 28
I 5         J 1         K 0         L 29
M 8         N 4         O 14        P 8
Q 0         R 20        S 46        T 20
U 0         V 20        W 18        X 8
Y 12        Z 8         - 27

A 0.022059  B 0.002451  C 0.026961  D 0.031863
E 0.000000  F 0.009804  G 0.230392  H 0.068627
I 0.012255  J 0.002451  K 0.000000  L 0.071078
M 0.019608  N 0.009804  O 0.034314  P 0.019608
Q 0.000000  R 0.049020  S 0.112745  T 0.049020
U 0.000000  V 0.049020  W 0.044118  X 0.019608
Y 0.029412  Z 0.019608  - 0.066176

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

Ok, so we can try straight substitution and see what we get?

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

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

implies the key is

```
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 -

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

So, let's try a decrypt....

How does this look

```
ONE RMADD RTEC FOH MAN ONE WIANT DEAC FOH MANPINL

LO NOT WO USEHE TSE CATS MAG DEAL WO INRTEAL USEHE TSEHE IR NO CATS ANL DEABE A THAID

UIRLOM IR TSE HEUAHL GOY WET FOH A DIFETIME OF DIRTENINW USEN GOYL SABE CHEFEHHEL TO TADP

UE MYRT YRE TIME UIREDG ANL FOHEBEH HEADIVE TSAT TSE TIME IR ADUAGR HICE TO LO HIWST

CEOCDE USO TSINP TSEG PNOU EBEHGTSINW AHE A WHEAT ANNOGANKE TO TSORE OF YR USO LO
```

Hmm, let's just take a look at the first one

Let's assume the first how many are correct `ONE` looks good, and so does `MAN`.

Let's look at the second line? `NOT` and `TSE`. Well `NOT` looks OK, so we think `ONE MA` and `T` as well as `-` are properly decrypting

If we go through the most frequent letters, and perhaps other ones that obivious look good we can start marking our initial decrypt key and try changing some.

```
- E T A O N I H S R D, M
```

Ok, so out of these by visual inpsection the decrypt locations for `- E T A O N I` and `M` look right. That is pretty good. Lets make some notations.

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

`TSE` is begging us to make the conclusion that `S` is incorrect and that whatever was decrypted to `S` should have decrypted to `H`.

Ok, what about the double letters "DD" and "HH"? Most commond doubles are ee, ll, ss, oo, tt, ff, rr, nn, pp, and cc, Not "dd" or 'hh'. So we can also assume these are wrong.

So, instead of 'dd' it is like 'ee' or 'll'. Since we are asuming MA are correct ?MAEE makes little sense, wehre ?MALL is begging us to make another guess for ? = S. Let's make these changes to our key guesses.

Ok, and one more, `FOH`. What was previously mapping to `H` (`R`) should be mapping to `R`.

```
X     X X     X X       X X X     X X X             X
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 -
H F X D S P M R T K B Y A V - N Q O W L C J Z U I E G
H       S       T       A V -         L             G
      ?       ?                   ? ?
              W                                         (TSE --> THE)
                                  R                     (FOH --> FOR)
                      D                                 (RMADD --> RMALL)
                                    O                   (RMALL --> SMALL)
=======================================================
H F X D S P M W T K B D A V - N Q R O L C J Z U I E G
```

Ok, before doing a sample decrypt note that we have some duplication in the key for `D` and no `Y`, so let's change that questionmark `D` to a `Y`.

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

Let's try another decrypt under this second.guess.key.

```
ONE SMALL STEC FOR MAN ONE WIANT LEAC FOR MANPIND

DO NOT WO UHERE THE CATH MAG LEAD WO INSTEAD UHERE THERE IS NO CATH AND LEABE A TRAIL
```

```
UISDOM IS THE REUARD GOY WET FOR A LIFETIME OF LISTENINW UHEN GOYD HABE CREFERRED TO TALP

UE MYST YSE TIME UISELG AND FOREBER REALIVE THAT THE TIME IS ALUAGS RICE TO DO RIWHT

CEOCLE UHO THINP THEG PNOU EBERGTHINW ARE A WREAT ANNOGANKE TO THOSE OF YS UHO DO
```

Let's mark off the right letter with an O now.

```
O       O O   O O     O O O O     O O O             O
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 -
H F X Y S P M W T K B D A V - N Q R O L C J Z U I E G
H       S P   W T     D A V -     R O L             G
                            X                          (STEC --> STEP)
            Z                                          (WIANT --> GIANT)
                    N                                  (MANPIND --> MANKIND)
                                          C            (UHERE --> WHERE)
                                              M        (MAG --> MAY)
                                        F              (LEABE --> LEAVE)
=====================================================
H F X Y S P Z W T K N D A V - X Q R O L C F C U M E G
  2 2                                   2
```

Ok, again we need to do some clean-up on some of these repeats that we have no confidence in. An inspection shows that we have two `F, X, C`s but no `B, I, J`s. So let's just pop them into a new key.

```
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 -

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

Let's call it third.guess.key and try another decrypt.

```
ONE SMALL STEP FOR MAN ONE GIANT LEAP FOR MANKIND

DO NOT GO WHERE THE PATH MAY LEAD GO INSTEAD WHERE THERE IS NO PATH AND LEAVE A TRAIL

WISDOM IS THE REWARD YOC GET FOR A LIFETIME OF LISTENING WHEN YOCD HAVE PREFERRED TO TALK

WE MCST CSE TIME WISELY AND FOREVER REALIUE THAT THE TIME IS ALWAYS RIPE TO DO RIGHT

PEOPLE WHO THINK THEY KNOW EVERYTHING ARE A GREAT ANNOYANBE TO THOSE OF CS WHO DO
```

OK, we are pretty close now so, let's do the same process again?

```
O     O O O O O O   O O O O O O   O O O   O O   O   O
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 -
H B I Y S P Z W T K N D A V - X Q R O L J F C U M E G
H     Y S P Z W T   N D A V - X   R O L   F C   M   G
                                      I              (YOC --> YOU)
                                                J    (REALIUE --> REALIZE)
    B                                               (ANNOYANBE --> ANNOYANCE)
=====================================================
H B I Y S P Z W T K N D A V - X Q R O L I F C U M J G
  2
```

Again we have to de-duped the `I` for an `E`, and we get the key.

```
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 -

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

Our fourth guessed key, let's try a decrypt.

```
ONE SMALL STEP FOR MAN ONE GIANT LEAP FOR MANKIND

DO NOT GO WHERE THE PATH MAY LEAD GO INSTEAD WHERE THERE IS NO PATH AND LEAVE A TRAIL

WISDOM IS THE REWARD YOU GET FOR A LIFETIME OF LISTENING WHEN YOUD HAVE PREFERRED TO TALK

WE MUST USE TIME WISELY AND FOREVER REALIZE THAT THE TIME IS ALWAYS RIPE TO DO RIGHT

PEOPLE WHO THINK THEY KNOW EVERYTHING ARE A GREAT ANNOYANCE TO THOSE OF US WHO DO
```

Ok, so that clearly did not take 2^{93} operations to do.
This is kind of unweildy as a cipher.

- Need to share the entire alphabet, and that is tough to memorize.
- Is not as secure as a key space size 27!.
- Easily broken by simple frequency counts.

4 Transposition Ciphers

Transposition ciphers are fairly simple concepts. Roughly they are defined to be systematic re-ordering of the plaintext so that the message becomes unintelligible. A simple exmaple makes this clear.

```
 PRINCIPLES HAVE NO REAL FORCE EXCEPT WHEN ONE IS WELL FED

 PI  EOETN EE
 RPHNARX  ILD
 ILAOLCCWOSL
 NEV  EEHN
 CSERF PEEWF

 PI  EOETN EERPHNARX  ILDILAOLCCWOSLNEV  EEHN  CSERF PEEWF
```

The above is a simple vertical transposition. Substitution ciphers can have have numberous different layouts some commonly categorized ones are depicted below

```
ABCDEF     EJOTY      ABFGOP     ZYXWVU
GHIJKL     DINSX      CEHNQX     OPQRST
MNOPQR     CHMRW      DIMRWY     NMLKJI
STUVWX     BGLQV      JLSVZ      CDEFGH
YZ         AFKPUZ     KTU        BA
```

These by themselves are not secure, and it is diffcult to separate the cipher form the keyspace. Rather, transpositions form a building block for other schemes.

A quick observation is that a frequency count will reveal a similar count as plaintext.

Let's examine a simple vertical transposition cipher. The key here could be considered the length, or how many letters (or units of plaintext) do we right vertically down before beginning a new column. We can think of this as a depth (or width if we have a transposition cipher).

We can break this cipher by simple exhaustion of the keyspace, assuming it is relatively small (which in general it must be to accomodate messages of the length we typically send and receive.)

```
57 = 3*19 + 0          57 = 4*14 + 1       57 = 5*11 + 2

PI  EOETN EERPHNARX    PI  EOETN EERPH     PI  EOETN EE
  ILDILAOLCCWOSLNEV    NARX  ILDILAOL      RPHNARX  ILD
  EEHN  CSERF PEEWF    CCWOSLNEV  EEH      ILAOLCCWOSL
                       N  CSERF PEEWF      NEV  EEHN
                                           CSERF PEEWF
```

5 Polyalphabetic Ciphers

Let's take a look at another type of cipher called a Polyalphabetic Cipher.

Definition 5.1. *A polyalphabetic substitution cipher* is a block cipher with block length t over an alphabet $\mathcal{A}$ having the following properties:

- the key space $\mathcal{K}$ consists of all ordered sets of t permutations $(p_1, p_2, \ldots, p_t)$, where each permutation p_i is defined on the set $\mathcal{A}$
- encryption of a block of a message $m = (m_1, m_2, \ldots m_t)$ under the key $e = (p_1, p_2, \ldots, p_t)$ is given by $E_e(m) = (p_1(m_1), p_2(m_2), \ldots, p_t(m_t))$ and
- the decryption key associated with $e = (p_1, p_2, \ldots, p_t)$ is $d = (p_1^{-1}, p_2^{-1}, \ldots, p_t^{-1})$.

The first of this type was known as the *Vigenére cipher*

Example 5.2. (Vigenére cipher) Let $\mathcal{A} = \{A, B, C, \ldots Z\}$ and define 3 mapping functions (p_1, p_2, p_3), where p_1 maps each letter to the letter three positions to its right in the alphabet, p_2 to the one seven positions to its right, and p_3 ten positions to its right. If

```
m = THI SCI PHE RIS CER TAI NLY NOT SEC URE
```

then,

```
c = E(m) = WOS VJS SOO UPC FLB WHS QSI QVD VLM XYO:
```

Well, one of the nice things about a Polyalphabetic Cipher is the key can be made fairly convenient, like "MATHEMATICS" and we can think about adding the letters modulo 27, by assuming an index

```
A = 0x00   B = 0x01   C = 0x02 . . . Z = 0x1A, - = 0x0B
```

So, we can think about encrypting a message

```
  THERE IS NO FRIGATE LIKE A BOOK TO TAKE US LANDS AWAY NOR ANY COURSERS LIKE A PAGE OF PRANC
 +MATHEMATICSMATHEMATICSMATHEMATICSMATHEMATICSMATHEMATICSMATHEMATICSMATHEMATICSMAT
=============================================================================================
  EHXYILIKHPFLFJPKMTXHN WESHDNOGSBK  LHOQ M BCMNWZDMWTFBE RSHRJ VWWIDEJZDXICMBSLPTNILOYHRIMNV
```

5.1 Cryptanalysis of Polyalphabetic Ciphers

Well let's examine another bit of ciphertext.

```
NOADRVYMOYUREMRTVKPDXEQTKQQTXYQ@297KJHKTD FAWBSMGLLOSTKKCMG8UTMPSUQAWXLWAKJVQWUS
EIZYJ'ZAOOMSMMZGQIVXLOSTNKILCE1IFTNCIHH Q N WBENOY OYBJJDMGK1IFTNCIHH Q RIFEYHCY
LBRE GW-XLLWNLRVYMOXAOPARQWHXFORRXLNZAM8UTMPSUQAWXLSRUJQDHCYLLV ZV6QHSHAETKJVHGX
MSAGRQWHSTCK YJU6QHSHAETKJVHGHCI RQWHWGAE2BKBMIGSEHRTNKDATJLOSTVGIXPAC@IYRJRLOXG
EDRKJZVVSNESHIGQBG8HEMASFQVCLTI  RDVNCJQ GL7YVHKXCEMUCNQOCAZGMX TVKHSEOMAWCLMB8H
EMPWTVHPDX TH PXHSACEPMRVYMOGEHRKRYRFO8UNMLZQHAOLTEMIWTZWSSHAETJQQNPJLLVDWBJPTSA
RRLWPJHDXCIAXRVYIHSDOZYRQW-XLD
```

Well, just for fun, let's do a monocount on the ciphertext.

```
A 25229     B 26711     C 18767     D 21223
E 29993     F 26731     G 23767     H 33863
I 21280     J 18912     K 22418     L 28721
M 36313     N 16477     O 24994     P 20339
Q 33075     R 32938     S 27538     T 42939
U 25460     V 27400     W 24552     X 20977
Y 20002     Z 20255     - 37488

A 0.035616  B 0.037708  C 0.026494  D 0.029961
E 0.042341  F 0.037736  G 0.033552  H 0.047805
I 0.030041  J 0.026698  K 0.031648  L 0.040546
M 0.051263  N 0.023261  O 0.035284  P 0.028713
Q 0.046692  R 0.046499  S 0.038876  T 0.060617
U 0.035942  V 0.038681  W 0.034660  X 0.029613
Y 0.028237  Z 0.028594  - 0.052922

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

Little peeks at `T` and `-`. Nothing really jumps out?
Any ideas?
Let's break this into two problems:

1. How *wide* is our block?
2. Solve each permuation (or shift in our simple case) independently?

5.2 How wide is our block

We can make the following observation about the english language - because the letters are not distributed evenly in plaintext the probability that we select two letters from the plaintext and they are the same (coincidence) is higher than if we selected two letters at random. We only need to devise a method of mearuring this.

William Freidman referred to this as the Index of Coincedence.

Definition 5.3. The *index of coincidence* is the ratio of observed coincedence to what we might expect at random.

$$XC = Pr(Observed)/Pr(Random)$$

So, let's think about just the letter `A` in out alphabet of 27 characters (all upper case letters and the space character). And suppose it occure 66 times in 1000 character text (in fact that is the frequency we counted in the monocount.c program). So, what is the probability that we select an `A` at random?

66/1000

So, the probability that we select the second letter from the remaining plaintext at random is now

65/999

So, in all the probability is

$$\left(\frac{66}{1000}\right) \cdot \left(\frac{65}{999}\right) = \left(\frac{n_A}{N}\right) \cdot \left(\frac{n_A - 1}{N - 1}\right)$$

Here N denotes the total number of characters, and n_A the number of times `A` appears in that text.
So, the overall index of coincendence can be computed as the sum of these values

$$IC = \sum_{\alpha \in \mathcal{A}} \left(\frac{n_\alpha}{N}\right) \cdot \left(\frac{n_\alpha - 1}{N - 1}\right)$$

So, what do we expect to be in the index of conicidence?

We can compute it from our previous monoalphabetic counting as

$$IC_{monoalphabetic} = \sum_{\alpha \in \mathcal{A}} f_\alpha^2$$

where, f_α is the frequency computed in the monoalphabetic counts.
When this is computed on the Tale of Two Cities (previously withheld output) it is 0.075844.
Technically this value is normalized by multiplying by the size of the alphabet. This is called $\kappa_{plaintext}$. Another difference you will find here compared to much of the open literature is the use of the space character. The theory remains the same however when using just the 26 characters without spacing.
If this was random text we would expect this value to be $\kappa_{random} = 1/|\mathcal{A}| = 0.037$.

Before returning to our polyalphabetci cipher, let's note that the index of coincedence or the $\kappa_{monoalphabetic}$ is the same as plaintext. Why? Because the same letter frequency distribution holds, but permutated. The index of coincidence is measuring the liklihood of a drawing two letters from the text and analyzing the probability of a collision (equality). What was 20 `E`s are now 20 `Q`s (or some other letter.)

So, if we measure the index of coincidence of our polyalphabetic ciphertext by only looking at every k^{th} character, where k is a guess on the width of our key, we should see an index of coincedence emerging that approaches our model for plaintext. All incorrect width guesses, while they may deviate from random, should be distinct from the expected plaintext index of coincedence.

```
Width 1, has index of coincedence 0.039142
Width 2, has index of coincedence 0.041207
Width 3, has index of coincedence 0.039367
Width 4, has index of coincedence 0.041491
Width 5, has index of coincedence 0.053344
Width 6, has index of coincedence 0.041653
Width 7, has index of coincedence 0.040422
Width 8, has index of coincedence 0.044674
Width 9, has index of coincedence 0.038726

Width 10, has index of coincedence 0.073710

Width 11, has index of coincedence 0.041506
Width 12, has index of coincedence 0.041592
Width 13, has index of coincedence 0.042429
Width 14, has index of coincedence 0.043245
```

```
Width 15, has index of coincedence 0.055647
Width 16, has index of coincedence 0.044809
Width 17, has index of coincedence 0.042897
Width 18, has index of coincedence 0.043048
Width 19, has index of coincedence 0.044676
Width 20, has index of coincedence 0.073710
Width 21, has index of coincedence 0.041195
```

We definitely can see a highpoints at 10, and 20, and medium high points at 5 and 15. So, some experience here might help. First we are examining a relatively small amount of ciphertext, 4096 characters. Given sufficient data you expect to see the index of coincedence for the *causal* length appear at multiples.

Once we conclude that the width is 10, for each $i = 1, \ldots 10$ we can look at the i^{th} letter and every 10^{th} letter afterwards and do a frequency count on this *column* of letters. We only need to compute how much the spike for `E` and `-` has shifted to compute the key value. This is a relative straightforward procedure.

Let's think about how we might right a program to solve this type of cipher.

```
Let ct[4096] be an array of ciphertext bytes
Let A[27] be an array of counters A[0] = # of A's, A[1] = # of B's....
/* first find the width */
for(w=1;w<26;w++){
   /* do a count on that width   */
   A[] = 0;
   for(i=0;i<4096;i+=w)        ++A[ct[i]];
   /* compute the index of coincidence, ic */
   for(i=0;i<27;i++)           ic = ic + (A[i]/(4096/w))^2;
   /*  check to see if index of coindidence is close to expected */
   if(|ic - 0.075844|< 0.01)   break;
}
if(w==26)      FAIL;
for(j=0;j<w;j++){
   A[] = 0;
   for(i=j;i<4096;i+=w)        ++A[ct[i]];
   /* compute the circuluar shift by where the 'space' peak is. */
   Let key[j] = (26 - i) such that A[i] is maximum (space character)
}
/* compute the plaintext */
for(i=0;i<4096;i++)            pt[i] = (ct[i] + k[i%w])%27;
print pt[];
```

How else might we have solved this?
What else could we do?

6 One-time Pad

A one-time pad is an additive key cipher in which the cryptographic key is effectively an infinite stream of random characters which is added into the plaintext stream to create the ciphertext. This is often referred to as the Vernam Cipher after the first to file a patent on the technique by Gilbert Vernam.

Definition 6.1. A *one-time pad* is a stream cipher in which the cryptographic key is an arbitrary length strem of random characters which is *combined* with character-by-character with a plaintext stream to produce a ciphertext stream.

Let's stay in the alphabet we have been using, and suppose we are given the key:

```
CTOY AGTOGLWSGRSCHRYGM IGZSLGP RGHGDTPSLTASG PGDTOLSVTVZGCWSVGM IYGWHFSGXRSPSRRSYGL GLHDNCS
```

Well, when we could then encrypt a single message of the same or less length

```
 TWO ROADS DIVEREGED IN A YELLOW WOOOD AND SORRY I COULD NOT TRAVEL BOTH AND BE ONE TRAVELER
+CTOY AGTOGLWSGRSCHRYGM IGZSLGP RGHGDTPSLTASG PGDTOLSVTVZGCWSVGM IYGWHFSGXRSPSRRSYGL GLHDNCS
================================================================================================
 VOBXQOGWFFODMKHWILUXOZZIFWWWRCVQBVURWOSYW JUQFDCANNFODYYTQORNXMUMIFXVYZFXDVOTVQFKKKSXLBHYGI
```

- random permutation for a polyalphabetic - still weak with enough plaintext
- infinite length codeword - this is actuall known as a one-time pad - it has the problem with how do you distribute an infinite key?

In practice, we use something called a key stream generator, which takes a short cryptographic key, which is used to generate a pseudo-random string of characters to combine with the plaintext.

7 Problem

Let's define a new crypto-system that combines the transposition and a polyalphabetic cipher.

Question 7.1. Define our cipher to have the following parameters, $(k, n, \{p_i\}_{i=1}^{n})$, where k is the length of our vertical transposition, n is the block length of our polyalphabetic cipher, and p_i are the n-many random permutations on our plaintext alphabet. Further let's assume that $1 < k \leq 16$, and $1 \leq n \leq 16$. The cipher operates on plaintext by first applying the transposition and then applying the polyalphabetic cipher.

1. How large is the keyspace?
2. Outline how you might attempt to breack this cipher?

8 Conclusion

We have seen a few cryptosystems as well as a few cryptanalytic attacks against those systems. It is worth outlining what we have covered and what other areas of discovery you might find interesting.

The systems we have dicsussed are largely stream ciphers, where one character of the alphabet is enciphered at a time. While stream ciphers are still widely used in practice their use is problematic as additional vulnerabilities occur from misuse. A good deal of research and standardization has occured to ensure that stream ciphers can be used securely.

We did not discuss how to provide data-integrity, authentication and non-repudiation. A large family of cryptographic techniques have been developed to provide these additional services. Some of these are:

- *Cryptographic Hash Algorithms* take an arbitrarily long input and create a fixed length digest (or hash code) such that it is computationally infeasible to modify the input to produce the same digest. Hash algorithms are used to provide data-integrity.
- *Asymmetric Cryptographic Systems* utilizes a pair of keys. A private key is kept secret to its owner, and the public key is made widely avialble.
  - The owner can *digitally sign* messages by which anyone with the public key can verify the message as originating from the private key owner. This provides the security services of authentication, data-integrity and non-repudiation.
  - Any entity can *encrypt* a message for the owner of the private key, in which no other entity can recover the plaintext message. This can be used to encrypt directly, but is more commonly used to establish a shared key by which the rest of the message can be encrypted with a symmetric cipher.

- *Authenticated Key Establishment Schemes* are used to securely established a shared session key between one or more parties. There are a variety of desirable security properties that pertain to key establishment.

Cryptography relies on mathematics, and not merely on a narrow selection of mathematics but anywhere in mathematics were so-called *hard problems* can be efficiently used, or computational methods can be derived to break existing cryptographic systems.

9 Source Code

9.1 monoalphabetic.c

```
/*
monoalphapetic.c                Example of a monoalphabetic encryption.

                                                Matthew J. Campagna
                                                Introduction to Cryptography

                                                30 August 2000

                                                cl -o monoalphabetic.exe monoalphabetic.c

*/


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

#define         KEYFILE         argv[1]
#define         INFILE          argv[2]
#define         OUTFILE         argv[3]
#define         ENCRYPT         argv[4]


void usage()
{
        printf("USAGE\n\tmonoalphabetic <keyfile> <infile> <outfile> <encrypt>\n");
        printf("\t<keyfile> key file that specifies a permutation of the alphabet\n");
        printf("\t          ABCDEFGHIJKLMNOPQRSTUVWXYZ\n");
        printf("\t<infile>  the in file to encrypt/decrypt\n");
        printf("\t<outfile> the out file\n");
        printf("\t<encrypt> TRUE for encrypt, FALSE to decrypt\n");

}

int main(int argc, char **argv)
{

        FILE    *fk, *fi, *fo;
        unsigned char   key[27], dkey[27];
        int             c, d, ki, dir = 0;

        if(argc<5){
                usage();
                return 0;
        }
```

```
        fk = fopen(KEYFILE, "rb");
        if(!fk){
                printf("Error openning key file: %d\n", 1);
                return 0;
        }
        fi = fopen(INFILE, "rb");
        if(!fi){
                printf("Error openning input file: %d\n", 2);
                return 0;
        }
        fo = fopen(OUTFILE, "wb+");
        if(!fo){
                printf("Error openning output file: %d\n", 3);
                return 0;
        }


        if(strcmp(ENCRYPT, "TRUE")==0)
                dir = 1;
        else
                dir = 0;

        fread(key, 1, 27, fk);
        /*
                if we need to decrypt construct the decrypt key
        */
        if(dir == 0){
                for(ki=0;ki<26;ki++){
                        d = key[ki];
                        if((d>0x40) && (d<0x5B))        {       d -= 0x41;      }
                        else                                    {       d = 26;         }
                        dkey[d] = ki+0x41;

                }
                d = key[26];
                if((d>0x40) && (d<0x5B))        {       d -= 0x41;      }
                else                                    {       d = 26;         }
                dkey[d] = 0x20;
                memcpy(key, dkey, 27);
        }
        /*
                start encrypt/decrypt process
        */
        c = fgetc(fi);
        while(c != EOF){
                if((c>0x40) && (c<0x5B)){
                        d = c - 0x41;
                }
                else{
                        d = 26;
                }
                d = key[d];
                fputc(d, fo);
                //printf("%c", d);
                c = fgetc(fi);
        }
        return 0;
}
```

9.2 moncount.c

```
/*
monocount.c                                    This file does a count of characters from
                                                   a text file for statistical properties

                                                   Matthew J. Campagna
                                                   Introduction to Cryptography

                                                   30 August 2000

                                                   cl -o monocount.exe monocount.c

*/

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

#define         INFILE          argv[1]


int getnextlargest(int value, int *counts);

void usage()
{
        printf("USAGE\n\tmonocount <infile> [frequency]\n");
        printf("\t<infile>    the in file to perform character count\n");
    printf("\t[frequency] a frequency count for plaintext\n");
}

int main(int argc, char **argv)
{

        FILE    *fi;
        int             count[27], c, total, i, j, value, idx, key[27];
    char    *frequency = NULL;
        double  d, ic = 0;

        if(argc<2){
                usage();
                return 0;
        }
    if(argc==3){
        frequency = argv[2];
    }
        /*
                open input file
        */
        fi = fopen(INFILE, "r");
        if(!fi){
                printf("Error opening an input file\n");
                return 0;
        }
        /*
                zero out count buffer
        */
        memset(count, 0, 27*sizeof(int));
        total = 0;
```

```
    c = fgetc(fi);
    while(c != EOF){
            /*
                    if the character is in our alphabet then
                    increment the count
            */
            if(c==0x20){
                    count[26]++;
                    total++;
            }
            if((c>0x40) && (c<0x5B)){
                    count[c-0x41]++;
                    total++;
            }
            c = fgetc(fi);
    }
    fclose(fi);
    /*
            print out the counts
    */
    for(i=0;i<26;i++){
            printf("%c %-7d   ", (char)(i+0x41), count[i]);
            if(i%4==3)
                    printf("\n");
    }
    printf("- %-7d\n\n", count[i]);
    /*
            ok now we have a frequency count, so we
            need to get probabilities, and simultaneously build the index
    of incoincidence
    */
    for(i=0;i<26;i++){
            d = ((double)count[i]/(double)total);
            printf("%c %f   ", (char)(i+0x41), d);
            if(i%4==3)
                    printf("\n");
    ic +=(d*d);
    }
//ic = 27*ic;
    d = ((double)count[26]/(double)total);
ic += (d*d);
    printf("- %f\n", d);
printf("Index of Coincedence: %f\n\n", ic);
    /*
            short printing
    */
    value = total;
    for(i=0;i<27;i++){
            idx = getnextlargest(value, count);
            printf("%c ", idx+0x41);
    if(frequency){
        j = (int)frequency[i] - 0x41;
        key[j] = (char)idx;
    }
            value = count[idx];
            count[idx]++;
    }
    printf("\n");
if(frequency){
```

```
        for(i=0;i<27;i++){
            printf("%c ", key[i] + 0x41);
        }
    }
    printf("\n");
        return 0;
}



int getnextlargest(int value, int *counts)
{
        int cdif, dif, idx,      i;

        dif = value;

        for(i=0;i<27;i++){
                cdif = value - counts[i];
                if( (cdif>=0) && (cdif <= dif )){
                        dif = cdif;
                        idx = i;
                }
        }
        return idx;
}
```

9.3 polyalphabetic.c

```
/*
polyalphabetic.c                Example of a polyalphabetic encryption.

                                                Matthew J. Campagna
                                                Introduction to Cryptography

                                                30 August 2000

                                                cl -o polyalphabetic.exe polyalphabetic.c

*/


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

#define         KEYFILE         argv[1]
#define         INFILE          argv[2]
#define         OUTFILE         argv[3]
#define         ENCRYPT         argv[4]

void usage()
{
        printf("USAGE\n\tpolyalphabetic <keyfile> <infile> <outfile> <encrypt>\n");
        printf("\t<keyfile> key file that specifies a string of length n\n");
        printf("\t           which specifies the key for Vigenere cipher\n");
        printf("\t           using letters ABCDEFGHIJKLMNOPQRSTUVWXYZ\n");
        printf("\t<infile>   the in file to encrypt/decrypt\n");
        printf("\t<outfile> the out file\n");
        printf("\t<encrypt> TRUE for encrypt, FALSE to decrypt\n");
```

```
}

int main(int argc, char **argv)
{

        FILE    *fk, *fi, *fo;
        char    key[256];
        int             ki, keysize, c, d, k;
        int             dir = 0;

        if(argc<5){
                usage();
                return 0;
        }

        fk = fopen(KEYFILE, "r");
        fi = fopen(INFILE, "r");
        fo = fopen(OUTFILE, "w+");

        if(!fk || !fi || !fo){
                printf("Error opening an input file\n");
                return 0;
        }

        if(strcmp(ENCRYPT, "TRUE")==0)
                dir = 1;
        else
                dir = 0;

        keysize = fread(key, 1, 256, fk);

        /*      CONVERT KEY TO NUMERICAL VALUE  */
        for(ki=0;ki<keysize;ki++){
                /*if number then subtract 'A' = 0x41*/
                if(key[ki]!=0x20)
                        key[ki] -= 0x41;
                else /*else we set it to 26*/
                        key[ki] = 26;
        }

        ki = 0;

        while((c = fgetc(fi)) != EOF){
                /*
                        convert the character to zero based integer
                */
                if(c!=' ')      { d = (c - 0x41);       }
                else            {               d = 26;         }

                /*
                        depending on the direction add or subtract
                */
                if(dir)         {       d = (d + key[ki]) % 27;                 }
                else            {       d = (d - key[ki] + 27) % 27;    }
                /*
                        convert back to a alphabet character
                */
                if(d==26)       {               d = ' ';        }
```

```
                else            {               d += 0x41;      }
                /*
                        write out the character to the file
                */
                fputc(d, fo);
                /*
                        advance the key
                */
                ki = (ki+1)%keysize;
        }
        fclose(fk);
        fclose(fi);
        fclose(fo);
        return 0;

}
```

9.4 indexofc.c

```
/*
 indexofc.c          This program computes the index of coincendence
                     on an input file assuming a 27 character
                     alphabet [A-Z, ' '].


                     Matthew J. Campagna
                     Introduction to Cryptography

                     30 August 2000

                     cl -o indexofc.exe indexofc.c
                     gcc -o indexofc indexofc.c


 */

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

#define     MAX_COUNT   (1<<16)
#define         INFILE          argv[1]

double  r = 0.0370;
//double        p = 0.0677;
double  p = 0.0744;




void usage()
{
        printf("USAGE\n\tindexofc <infile> <max-width>\n");
        printf("\t<infile>    the in file to perform monograph phi test\n");
    printf("\t<max-width> the maximum width to examine to compute index of coincedence\n");
}

int main(int argc, char **argv)
{
```

```
        FILE    *fi;
        int             count[27], c, total, i, len, j, k;
        double  d, ic;
    char    buffer[4096];

        if(argc<3){
                usage();
                return 0;
        }
        /*
     open input file
     */
    k = strtoul(argv[2], NULL, 10);
        fi = fopen(INFILE, "r");
        if(!fi){
                printf("Error opening an input file\n");
                return 0;
        }
        /*
     zero out count buffer
     */len = fread(buffer, 1, 4096, fi);

    for(j=1;j<k;j++){
        memset(count, 0, 27*sizeof(int));
        total = 0;
        for(i=0;i<len;i+=j){
            c = buffer[i];
            if(c==0x20){
                ++count[26];
                ++total;
            }
            if((c>0x40) && (c<0x5B)){
                ++count[c-0x41];
                ++total;
            }
        }
        ic = 0;
        for(i=0;i<27;i++){
            d = (double)count[i]/(double)total;
            ic += d*d;
        }
        printf("Width %d, has index of coincedence %f\n", j, ic);

    }
        return 0;

}
```

9.5 polycrack.c

```
/*
 polycrack.c          This program computes the index of coincendence
                      on an input file assuming a 27 character
                      alphabet [A-Z, ' '], selects the most likely, and
                      then computes the best key value for that width


 Matthew J. Campagna
 Introduction to Cryptography
```

```
 30 August 2000

 cl -o polycrack.exe polycrack.c
 gcc -o polycrack polycrack.c


 */

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

#define     MAX_COUNT   (1<<16)
#define         INFILE          argv[1]

char        KEY[ ] = "BCDEFGHIJKLMNOPQRSTUVWXYZ A";



void usage()
{
        printf("USAGE\n\tpolycrack <infile> <max-width> [TRUE]\n");
        printf("\t<infile> the in file to perform monograph phi test\n");
    printf("\t<width>   the maximum width to examine to compute index of coincedence or the width o
    printf("\t[TRUE]    TRUE if present will take the width as the value on which to guess the key.
}

int main(int argc, char **argv)
{
        FILE    *fi;
        int             count[27], c, total, i, len, j, k, sp, E;
        double  d, ic;
    char    buffer[4096];

        if(argc<3){
                usage();
                return 0;
        }
        /*
     open input file
     */
    k = strtoul(argv[2], NULL, 10);
        fi = fopen(INFILE, "r");
        if(!fi){
                printf("Error opening an input file\n");
                return 0;
        }
        /*
     zero out count buffer
     */len = fread(buffer, 1, 4096, fi);
    if(argc==3){
        for(j=1;j<k;j++){
            memset(count, 0, 27*sizeof(int));
            total = 0;
            for(i=0;i<len;i+=j){
                c = buffer[i];
                if(c==0x20){
                    ++count[26];
```

```
                    ++total;
                }
                if((c>0x40) && (c<0x5B)){
                    ++count[c-0x41];
                    ++total;
                }
            }
            ic = 0;
            for(i=0;i<27;i++){
                d = (double)count[i]/(double)total;
                ic += d*d;
            }
            printf("Width %d, has index of coincedence %f\n", j, ic);
        }
    }
    if(argc > 3){
        for(j=0;j<k;j++){
            memset(count, 0, 27*sizeof(int));
            total = 0;
            // for each character, do a count and see where "-" and "E" are
            for(i=j;i<len;i+=k){
                c = buffer[i];
                if(c==0x20){
                    ++count[26];
                    ++total;
                }
                if((c>0x40) && (c<0x5B)){
                    ++count[c-0x41];
                    ++total;
                }
            }
            sp = 0, E = 0;
            for(i=1;i<27;i++){
                if(count[i]>count[sp]){
                    E = sp;
                    sp = i;
                }
            }
            printf("(%c: %d, %d)\n", KEY[sp], E, sp);

        }
        printf("\n");
    }
        return 0;

}
```