Mathematical Excursions Essay
Cryptology: An Introduction

Jamie Sawyer (0407950)

February 13, 2006

Contents

List of Tables

1 Introduction

1.1 What is Cryptology?

Cryptology[1] is the study of secure communications. This study can be broken down into two parts:

Cryptography, the aim of which is to render a confidential message incomprehensible to any unauthorised third party, and

Cryptanalysis, which has the aim of decyphering encrypted messages whilst not being the intended recipient.

1.2 The Cypher and the Cryptosystem

A cypher is essentially just an algorithm, χ, known as the *encryption algorithm*. χ takes a string of elements, length n, from an alphabet V (or subset thereof) and encrypts them into a string of elements, length m, from an alphabet W, i.e. $\chi : V^n \dashrightarrow W^m$. To clarify, an alphabet is simply a set of symbols, for example, the normal lower case alphabet is $V = \{a, b, \ldots, z\}$. In general, an alphabet can be written as $V = \{v_1, v_2, \ldots, v_n\}$ (Where n is the size of the alphabet). The set of all words over V is denoted by V^*. A subset of V^* is known as a *formal language over* V. χ is (in most cases) determined by a key k_1, which will change the effect of the algorithm (for example, by determining the order in which characters should be ordered). Also defined is $\chi^{-1} : W^m \dashrightarrow V^n$ [2]. This is the *decryption algorithm*, used by the (legal) receiving party to decrypt the message. Every cypher has its own so-called cryptosystem.

A cryptosystem has three constituent parts: A *Plaintext Space* PT, a *Cryptotext Space* CT, and the *Keyspace* K. The Plaintext Space PT is the collection of all possible plaintexts pt. This is usually V^*, or some subset thereof. The keyspace K is the collection of all possible keys for the cypher, k. The cryptotext space $CT = \{\chi_k(pt) | k \in K, pt \in PT\}$, is the collection of all outputs given by the algorithm χ, with keys from K and inputs from PT.

1.3 General Cryptography Issues

When it comes to cryptography, the making of cyphers, the nature of PT, CT and K comes into play. To start, with the plaintext space PT, it is important to understand how large a subset of V^* PT is. If PT is a very

[1] From the Greek *kryptos*, meaning hidden, and *logos*, meaning word

[2] χ^{-1} is of course also determined by a key k_2, itself determined by k_1

small subset of V^*, (for example with everyday language, where the set of all actual words is a very small subset of all possible rearrangements of letters), it may well be possible to understand a message even if some of the original characters are distorted (for example, the message "UNDERTHEBRIDGE" could be incorrectly decrypted as "ONTERDHEBRITGE" and this could still be fairly easily understood by the cryptanalyst). On the other hand, if PT is the whole of V^* (for example a binary string referring to on/off positions of another system), the cryptanalysis needs to be totally accurate since any inaccuracy will create an entirely new element of PT.

K, the key space, is also rather important to the cryptographer. The cardinality of K should not be small - we do not want to leave the cryptanalyst with the possibility of testing every key, k. In most cases, it is preferable for the cardinality of K to be denumerably infinite. Finally, of course, we have CT, about which there is not a great deal to say, since it is determined by PT and K.

Sir Frances Bacon (1561-1626) proposed the following three requirements for a good cryptosystem:

1. Given χ_k and pt, the computation of $\chi_k(pt)$ is easy. Given χ_k^{-1} and ct, the computation of $\chi_k^{-1}(ct)$ is easy.
2. Without knowing χ_k^{-1}, it is impossible to find pt from ct.
3. The cryptotext should be without suspicion: innocent looking. [2]

Of course, some of this is still applicable today, some needs tweaking. Requirement 1 now will assume that users have a reasonable amount of computing power at their disposal (except in special circumstances - spies in the field etc). Requirement 2 has "impossible" which should be replaced by "computationally intractable" - i.e. that it should be outside of reasonable computing power to find ct from pt. Requirement 3 is no longer considered to be important.

Finally, the so-called *golden rule of cryptographers*: Never underestimate the cryptanalyst. It should be assumed, for example, that the cryptanalyst knows the cryptosystem being used[3], and knows as much about the system as there is information available about the system.

[3] "The enemy knows the system being used" is also known as *Shannon's Maxim*

1.4 General Cryptanalysis Issues

The task of the cryptanalyst is to recover *pt* from the *ct* without having the algorithm χ_k^{-1} [4]. There are two main ways that the cryptanalyst can do this, the first is by testing every $k \in K$ (obviously only for $|K|$ finite), and the other is to analyse the actual cryptotext with respect to the cryptosystem used. For example, if it is known that the cypher replaces every character $v \in V$ with a character $w \in W$ as stated by the key, letter frequencies for the language V^* could be used[5].

There are 4 basic situations that the cryptanalyst can be in when trying to decrypt some *ct*. The first is to only have the cryptotext *ct* and nothing else. A longer cryptotext will always be better for the cryptanalyst, as statistical analysis and other such methods are more accurate given a larger text to work on (since longer text tends to bring text closer to the average text). The second situation is where the cryptanalyst has a known plaintext, so knows some pairs of $(pt, \chi_k(pt))$. The knowledge of these pairs may aid the analysis of the *ct*. The third setup is where the cryptanalyst has a chosen plaintext which has been put through the cypher. Where a cryptanalyst has conjectures about the key, k, this situation is obviously stronger than the second. The final situation, which is typical for public-key cryptosystems, is knowledge of the encryption method χ_k. In public-key cryptosystems, the aim for the cryptographer is to make χ_k^{-1} computationally intractable from χ_k.

Finally, I will refer to the *time complexity* of a cryptanalysis algorithm. If the algorithm has time complexity $f(n)$, this essentially means that the amount of time to find a solution (in the worst case scenario) is $\in O(f(n))$.

1.5 Steganography

Before entering the world of cryptology proper, there is another related area which is interesting to get some background on. *Steganography* [6] has the goal of concealing the very existence of a message[7]. There are two areas within steganography itself - *technical steganography* and *linguistic steganography*. There is little to say about technical steganography, and it has no real bearing on cryptology, examples are invisible ink and books with hidden

[4] Shannon's Maxim is assumed - that the cryptanalyst knows the cryptosystem being used

[5] These topics will be covered in more detail in later sections

[6] From the greek *steganos*, meaning covered.

[7] Notice the link with Sir Frances Bacon's definition of a good cryptosystem (Section 1.3)

compartments. However, linguistic steganography is, in a way, a precursor to cryptography.

An example of a linguistic steganograph is a *semagram*. A semagram is a code expressed in the (often minute) graphical details of a piece of text or a drawing. This could take the form, for example, of an area of morse code drawn into a picture, or of a piece of handwritten text changing from joined-up to not joined-up to signify a detail. The other example of a linguistic steganograph is known as *open code*. This is where a secret code is made to appear innocent, some of the most famous being the secret marks of the hobo's in the US, "Pig Latin[8]," and the use of grilles to veil unnecessary text, leaving only the plaintext visible.

2 Classical Cryptology

2.1 Simple Substitution Cyphers

A *simple substitution cypher* is the most basic of all cryptographic techniques. This system works by replacing each character from the plaintext alphabet V by a character or characters from the cryptotext alphabet W.

2.1.1 Caesar's Cypher

The most famous example of this type of technique is known as *Caesar's Cypher*. This cypher was used, according to Suetonius, by Julius Caesar to encrypt messages of military significance:

> If he had anything confidential to say, he wrote it in cipher[sic], that is, by so changing the order of the letters of the alphabet, that not a word could be made out. If anyone wishes to decipher these, and get at their meaning, he must substitute the fourth letter of the alphabet, namely D, for A, and so with the others.[15]

This cryptosystem can be characterised as follows

$$\chi : V \dashrightarrow V$$

$$V = \{\texttt{A},\texttt{B},\ldots,\texttt{Z}\} = Z_{26}$$

The algorithm χ itself is most easily described in words. Each character *pt* in the plaintext is replaced by the character which is 3 places circularly from

[8]Often used by children, this system takes the word, moves the first syllable or letter to the end, then adds AY after this. For example "THE QUICK BROWN FOX" becomes "HE-TAY UICK-QAY ROWN-BAY OX-FAY"

itself, i.e. A is replaced by D, B by E ... X by A, Y by B and Z by C. The other simple ways of displaying this are in a table of plaintext-cryptotext pairs - *substitution notation*[9]:

pt :	a	b	c	d	e	f	g	h	i	j	k	l	m
ct :	D	E	F	G	H	I	J	K	L	M	N	O	P

pt :	n	o	p	q	r	s	t	u	v	w	x	y	z
ct :	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

or in *short cycle notation*[10], where the cryptotext character ct is found to the right (cyclically) of the plaintext character pt:

(adgjmpsvybehknqtwzcfilorux)

These notations shall be used without comment from here on.

2.1.2 The Rotation Cypher

The *rotation cypher*, sometimes also known as the *Caesar cypher*, is a generalisation of Caesar's cypher. The algorithm is often represented as $\chi = ROT\text{-}k : V \dashrightarrow V$, where k is the key for the system. Caesar's cypher was the special case of this cypher, with $k = 3$. Where the alphabet $V = Z_{26}$, $ROT\text{-}0 = ROT\text{-}26 = ROT\text{-}52 = \ldots$ In fact, where $|V| = n$, $ROT\text{-}b = ROT\text{-}(b \mathrm{mod} n)$. This means that $|K| = |V|$ for this cypher, and $|K| = 26$ for the special case $V = Z_{26}$.

ROT-13 is a specific example of the rotation cypher which is in everyday use on Usenet (the internet messaging system), to mask text so as that the reader must choose to decrypt it. This is used if what is said is likely to offend others or to spoil future events. This is an example of a cryptographic technique *not* being used in it's intended way (to secure a message), but to render a message unreadable without effort. Finally, ROT-13 is used since, unlike any other rotation cypher, it is a *self-inverse*, meaning that $ROT\text{-}13 = \chi = \chi^{-1}$, and so $ROT\text{-}13(ROT\text{-}13(pt)) = pt$.

2.1.3 Monocyclic Permutations

A *monocyclic permutation* is a permutation in which each plaintext character pt is a generator for the entire alphabet V under repeated application of χ.

[9]Note that it is convention when taking $Z_{26} \dashrightarrow Z_{26}$: to write the pt characters in lower case and the ct characters in capitals

[10]Short cycle notation can only be used when an algorithm χ takes elements from an alphabet V onto itself, this is also known as a *permutation*.

In the case $V = Z_{26}$, the total number of possible permutations is equal to $25! \approx 1.5 \times 10^{25}$ (the rearrangements of all but one character - if you move that character you get repetition).

2.1.4 General Simple Substitutions

So far, we have only been looking at permutations, and not even at all possible permutations. In fact, it can be seen that with the case $V = Z_{26}$, there are $26! \approx 4.0 \times 10^{26}$ different simple substitutions to choose from. All can obviously be defined in substitution notation. Other common simple substitutions take, for example, $\chi : Z_{26} \times Z_{26} \dashrightarrow Z_{26} \times Z_{26}$ (Where each pair of letters is taken to a different pair of letters, eg PLAYFAIR cypher), $\chi : Z_{26} \dashrightarrow W$ st $|W| = 26$ (Where each character is taken to a new symbol entirely, eg pigpen cypher). I will not go into these cyphers in great depth, as they are essentially similar to cyphers already talked about.

2.2 Cryptanalysis of Simple Substitutions

2.2.1 Rotation Cypher

As has been mentioned before, $|K|$ for this system is only equal to $|V|$, the size of the alphabet being used. This means the easiest cryptanalytic technique to use on a piece of cryptotext you know to be written in this cypher is to exhaust the supply of keys. This is generally done by listing the results of decyphering the text ROT-1 up to ROT-n and seeing which one is correct.

Example: You are given a cryptotext `ZNK KTKSE ROK HKTKGZN ZNK URJ HXOJMK`, and know that the cypher being used is a rotation cypher with unknown key and that it is written in Z_{26}. To decypher this, list the results for χ_k^{-1} with $k \in \{1, 2, \ldots, 26\}$ (i.e. perform χ_{26-k}):

$k = 1$	`YMJ JSJRD QNJ GJSJFYM YMJ TQI GWNILJ`
$k = 2$	`XLI IRIQC PMI FIRIEXL XLI SPH FVMHKI`
$k = 3$	`WKH HQHPB OLH EHQHDWK WKH ROG EULGJH`
$k = 4$	`VJG GPGOA NKG DGPGCVJ VJG QNF DTKFIG`
$k = 5$	`UIF FOFNZ MJF CFOFBUI UIF PME CSJEHF`
$k = 6$	`THE ENEMY LIE BENEATH THE OLD BRIDGE`
$\vdots$	$\vdots$

It can be seen clearly that the Plaintext was `THE ENEMY LIE BENEATH THE OLD BRIDGE` and the cypher was ROT-6.

This method, of course, is only useful if the plaintext space PT is a relatively small subset of V^*, if this is not the case, it may be difficult or impossible to know which key was used:

Example: The enemy uses a code to say which of 9 ships will be in any one of 5 locations on the following day. An example plaintext is `24951` meaning ship 2 is in location 1, ship 4 in location 2 etcetera. The alphabet $V = \{1, 2, \ldots, 9\}$. The cryptotext recieved reads `39817`, and you know a rotation cypher with unknown key is being used. Listing the results for χ_k^{-1} with $k \in \{1, 2, \ldots, 9\}$:

$k = 1$	`28796`
$k = 2$	`17685`
$k = 3$	`96574`
$k = 4$	`85463`
$k = 5$	`74352`
$k = 6$	`63241`
$k = 7$	`52139`
$k = 8$	`41928`
$k = 9$	`39817`

Although you cannot tell which of these keys is correct, there is some information imparted here. If you are able to find out that, for example, boat 5 will be in position 5 on the following day, the boats which are in the other positions follow immediately. This is the best you can get from the analysis of this cryptotext.

Finally, as a note of convention, normally cryptotexts are not laid out as shown above, it is normal convention to split characters into groups of 5, with no spaces (often filling space with X's in plaintext written in Z_{26}). For example, the cryptotext in the first example: `"ZNK KTKSE ROK HKTKGZN ZNK URJ HXOJMK"`, would normally be recieved as `"ZNKKT KSERO KHKTK GZNZN KURJH XOJMK"`. This saves giving away extra clues to the cryptanalyst in the length of words (for example, in the normal English language, only I and A could be written as single characters).

2.2.2 General Simple Substitutions

In an alphabet of size $|V|$ there are $|V|!$ different general simple substitutions to choose from, and so $|V|!$ different possible keys. This again means that the time complexity is only again $f(n) = n$, although in Z_{26}, there are over 4.0×10^{26} different keys that need to be tested. This means that, even assuming the computer could test 10^{12} different keys a second, in the worst

case scenario this would take over 10,000 millenia to calculate. Obviously, this is not the method that you would like to go for! Again, it is necessary for PT to be a relatively small subset of V^* to be able to cryptanalyse this system, and this time, the only useful information that can be gleamed if PT is a large subset of V^* is whether one character in the plaintext is the same as another. The cryptanalysis of a cryptotext which had been encyphered with a simple substitution system is done with a *frequency count*: the number of times each character appears in the cryptotext. The distribution of letters in the cryptotext is then compared to the distribution of letters in the plaintext language PT[11]. For example, the frequency distribution of characters in an english-language plaintext is in Table 1

Obviously, for different languages (i.e. different plaintext spaces), the frequency distributions could be very different[12]. Additionally, this method can be used to compare the frequency distribution of pairs of letters, triplets of letters and upwards of the plaintext to the frequency disrtribution for the entire plaintext space. This can be useful when used in addition to the single letter version, giving the cryptanalyst more idea of which characters have been replaced by which.

[11]Bear in mind that the alphabet of the plaintext is not necessarily the same as the alphabet of the cryptotext

[12]By different languages, this could mean the difference between English and German, or equally could mean the difference between 19th Century English novels and today's internal business memos, which will also have differing frequency distributions.

Table 1: The Frequency Distrubution Table for English-Language Plaintext Characters [2]

letter	%	letter	%	letter	%
E	12.31	L	4.03	B	1.62
T	9.59	D	3.65	G	1.61
A	8.05	C	3.20	V	0.93
O	7.94	U	3.10	K	0.52
N	7.19	P	2.29	Q	0.20
I	7.18	F	2.28	X	0.20
S	6.59	M	2.25	J	0.10
R	6.03	W	2.03	Z	0.09
H	5.14	Y	1.88		

2.2.3 A Full Example

You are given a cyphertext as follows:

```
TKAXZ BTDLZ YACCA CYNUY ZNFZE ONAEZ SURDA NNACY LOMZE DADNZ
EUCNM ZLTCJ TCSUR MTFAC YCUNM ACYNU SUUCX ZUENB AXZDM ZMTSP
ZZPZS ACNUN MZLUU JMZED ADNZE BTDEZ TSACY LINAN MTSCU PAXNI
EZDUE XUCFZ EDTNA UCDAC ANTCS BMTNA DNMZI DZURT LUUJN MUIYM
NTKAX ZBANM UINPA XNIEZ DUEXU CFZED TNAUC DUDMZ BTDXU CDASZ
EACYA CMZEU BCWAC STDBZ KKTDD MZXUI KSRUE NMZMU NSTOW TSZMZ
ERZZK FZEOD KZZPO TCSDN IPASB MZNMZ ENMZP KZTDI EZURW TJACY
TSTAD OXMTA CBUIK SLZBU ENMNM ZNEUI LKZUR YZNNA CYIPT CSPAX
JACYN MZSTA DAZDB MZCDI SSZCK OTBMA NZETL LANBA NMPAC JZOZD
ETCXK UDZLO MZENM ZEZBT DCUNM ACYDU FZEOE ZWTEJ TLKZA CNMTN
CUESA STKAX ZNMAC JANDU FZEOW IXMUI NURNM ZBTON UMZTE NMZET
LLAND TONUA NDZKR UMSZT EUMSZ TEADM TKKLZ KTNZB MZCDM ZNMUI
YMNAN UFZET RNZEB TESDA NUXXI EEZSN UMZEN MTNDM ZUIYM NNUMT
FZBUC SZEZS TNNMA DLINT NNMZN AWZAN TKKDZ ZWZSQ IANZC TNIET
KLINB MZCNM ZETLL ANTXN ITKKO NUUJT BTNXM UINUR ANDBT ADNXU
TNPUX JZNTC SKUUJ ZSTNA NTCSN MZCMI EEAZS UCTKA XZDNT ENZSN
UMZER ZZNRU EANRK TDMZS TXEUD DMZEW ACSNM TNDMZ MTSCZ FZELZ
RUEZD ZZCTE TLLAN BANMZ ANMZE TBTAD NXUTN PUXJZ NUETB TNXMN
UNTJZ UINUR ANTCS LIECA CYBAN MXIEA UDANO DMZET CTXEU DDNMZ
RAZKS TRNZE ANTCS RUENI CTNZK OBTDH IDNAC NAWZN UDZZA NPUPS
UBCTK TEYZE TLLAN MUKZI CSZEN MZMZS YZ
```

You know it is english plaintext, encyphered with a monoalphabetic substitution, but with unknown key. You formulate a frequency table of characters as in Table 2.

Comparing this to Table 1, we can begin to guess a few of the characters. Initial attempt will take each letter in Table 2 to the corresponding position of letter in Table 1 for the first 10 characters:

```
aKnXe BahLe Ynrrn rYtoY etFes Otnse loRhn ttnrY LOies hnhte
sorti eLarJ arloR iaFnr Yroti nrYto loorX eostB nXehi eialP
eePel nrtot ieLoo Jiesh nhtes Bahse alnrY LItnt ialro PnXtI
sehos XorFe shatn orhnr ntarl Biatn htieI heoRa LooJt ioIYi
taKnX eBnti oItPn XtIse hosXo rFesh atnor hohie BahXo rhnle
snrYn rieso BrWnr lahBe KKahh ieXoI KlRos tieio tlaOW aleie
sReeK FesOh KeePO arlht IPnlB ietie stieP KeahI seoRW aJnrY
alanh OXian rBoIK lLeBo stiti etsoI LKeoR Yettn rYIPa rlPnX
JnrYt ielan hnehB ierhI llerK OaBin tesaL LntBn tiPnr JeOeh
```

```
sarXK oheLO iesti eseBa hroti nrYho FesOs eWasJ aLKen rtiat
rosln laKnX etinr Jntho FesOW IXioI toRti eBaOt oieas tiesa
LLnth aOton theKR oilea soile asnhi aKKLe KateB ierhi etioI
Yitnt oFesa RtesB aslhn toXXI sselt oiest iathi eoIYi ttoia
FeBor lesel attin hLIta ttiet nWent aKKhe eWelQ Inter atIsa
KLItB ierti esaLL ntaXt IaKKO tooJa BatXi oItoR nthBa nhtXo
atPoX Jetar lKooJ elatn tarlt ieriI ssnel oraKn Xehta stelt
oiesR eetRo sntRK ahiel aXsoh hiesW nrlti athie ialre FesLe
Roseh eeras aLLnt Bntie nties aBanh tXoat PoXJe tosaB atXit
otaJe oItoR ntarl LIsrn rYBnt iXIsn ohntO hiesa raXso hhtie
RneKl aRtes ntarl RostI rateK OBahH Ihtnr tnWet oheen tPoPl
oBraK asYes aLLnt ioKeI rlest ieiel Ye
```

Now, looking at the lower case letters, there are some areas which don't seem to make sense - `nrltiathieialre` being a prime example, we obviously cannot get the answer as simply as this. I shall now look at the most common digrams in the cryptotext[13], as in Table 3 and compare them to the most common digrams.

The 15 most common digrams in the english language, in alphabetical order, are `an, at, en, er, es, he, im, it, nt, om, re, st, te, th, ti`. As you can see, neither `ie` nor `nr` appear in this list, implying that maybe these characters are incorrect. It is at this point that some trial and error, and educated guessing come in. It is, for example, a fair guess that `Z` mapping to `e` and `N` mapping to `t` are correct, as these have much higher

[13]Thanks to http://www.central.edu/homepages/LintonT for the applet to produce these results

Table 2: Letter Count for the Cryptotext in Example 2.2.3

letter	count	letter	count	letter	count
Z	125	S	44	P	14
N	118	I	31	J	13
T	87	K	30	F	10
U	79	X	28	W	9
A	78	B	28	H	1
M	73	L	25	Q	1
E	66	Y	19	G	0
C	60	R	19	V	0
D	58	O	16	Total	1032

frequencies as single letters than any other[14]. One obvious start would be to map `M` to `h` instead of `i`, causing `he` and `th` to be the two most common digrams. Looking back at the initial 'translation' of the cryptotext, we have `nrrnr` as a 5-gram near the start. This cannot be correct, and looking at the characters around it, it could be part of the phrase `beginningtoget`, thus mapping `A` to `i` and `C` to `n`. This would cause digrams `im` and `it` to be in the top 5, both of which appear in the most common list.

All these mappings result in a new translation of:

```
aKiXe BaDbe ginni ngtog etFeE OtiEe SoRDi tting bOheE DiDte
Eonth ebanJ anSoR haFin gnoth ingto SoonX eoEtB iXeDh ehaSP
eePeS intot heboo JheED iDteE BaDEe aSing bItit haSno PiXtI
EeDoE XonFe EDati onDin itanS Bhati DtheI DeoRa booJt hoIgh
taKiX eBith oItPi XtIEe DoEXo nFeED ation DoDhe BaDXo nDiSe
Eingi nheEo BnWin SaDBe KKaDD heXoI KSRoE theho tSaOW aSehe
EReeK FeEOD KeePO anSDt IPiSB hethe EtheP KeaDI EeoRW aJing
aSaiD OXhai nBoIK SbeBo Ethth etEoI bKeoR getti ngIPa nSPiX
Jingt heSai DieDB henDI SSenK OaBhi teEab bitBi thPin JeOeD
EanXK oDebO heEth eEeBa Dnoth ingDo FeEOE eWaEJ abKei nthat
noESi SaKiX ethin JitDo FeEOW IXhoI toRth eBaOt oheaE theEa
bbitD aOtoi tDeKR ohSea EohSe aEiDh aKKbe KateB henDh ethoI
ghtit oFeEa RteEB aESDi toXXI EEeSt oheEt hatDh eoIgh ttoha
FeBon SeEeS atthi DbIta tthet iWeit aKKDe eWeSQ Iiten atIEa
KbItB henth eEabb itaXt IaKKO tooJa BatXh oItoR itDBa iDtXo
atPoX Jetan SKooJ eSati tanSt henhI EEieS onaKi XeDta EteSt
oheER eetRo EitRK aDheS aXEoD DheEW inSth atDhe haSne FeEbe
RoEeD eenaE abbit Bithe itheE aBaiD tXoat PoXJe toEaB atXht
otaJe oItoR itanS bIEni ngBit hXIEi oDitO DheEa naXEo DDthe
```

[14]Also note that if these were reversed, then the 2nd most common digram in the plaintext would be ei, also not likely.

Table 3: Frequency of Digrams in the Cryptotext and First Decryption of Example 2.2.3

rank	CT letters	'PT' letters
1	MZ	ie
2	NM	ti
3	ZE	es
4	AN	nt
5	AC	nr

```
RieKS aRteE itanS RoEtI nateK OBaDH IDtin tiWet oDeei tPoPS
oBnaK aEgeE abbit hoKeI nSeEt heheS ge
```

This analysis continues in a similar manner, looking for recognisable word patterns, and testing the fit, until the result finally comes out:

```
alice wasbe ginni ngtog etver ytire dofsi tting byher siste
ronth ebank andof havin gnoth ingto doonc eortw icesh ehadp
eeped intot heboo khers ister wasre ading butit hadno pictu
resor conve rsati onsin itand whati stheu seofa bookt hough
talic ewith outpi cture sorco nvers ation soshe wasco nside
ringi nhero wnmin daswe llass hecou ldfor theho tdaym adehe
rfeel verys leepy andst upidw hethe rthep leasu reofm aking
adais ychai nwoul dbewo rthth etrou bleof getti ngupa ndpic
kingt hedai siesw hensu ddenl yawhi terab bitwi thpin keyes
rancl oseby herth erewa snoth ingso veryr emark ablei nthat
nordi dalic ethin kitso verym uchou tofth ewayt ohear thera
bbits aytoi tself ohdea rohde arish allbe latew hensh ethou
ghtit overa fterw ardsi toccu rredt ohert hatsh eough ttoha
vewon dered atthi sbuta tthet imeit allse emedq uiten atura
lbutw henth erabb itact ually tooka watch outof itswa istco
atpoc ketan dlook edati tandt henhu rried onali cesta rtedt
oherf eetfo ritfl ashed acros sherm indth atshe hadne verbe
fores eenar abbit withe ither awais tcoat pocke toraw atcht
otake outof itand burni ngwit hcuri osity shera nacro ssthe
field after itand fortu natel ywasj ustin timet oseei tpopd
ownal arger abbit holeu ndert hehed ge[16]
```

2.3 Polyalphabetic Systems

The common link with all the cryptosystems looked at so far is that the use of substitutes remains the same throughout the text[15]. To increase the security of a substitution cypher, one idea is to replace each character by a character which changes throughout the text.

2.3.1 Vigenère Cypher

The *Vigenère Cypher* is commonly held to be a polyalphabetic system, and is essentially an extention of the Rotation Cypher. To begin with, we have

[15]Note that in the case $\chi : Z_{26} \times Z_{26} \dashrightarrow Z_{26} \times Z_{26}$, the OB in HOBO and the OB in OBOE will be taken to different cryptotext, but this is because in HOBO, the OB is actually the second half of HO and the first half of BO being put through the cypher

to define how the keyspace works[16]. K, the keyspace, is made up of all possible words in the alphabet – Z_{26}^*. Each letter from the keyword is mapped to a number from $\mathbb{Z}_{26} = \{0, 1, 2, \ldots 25\}$ in the obvious way ($a \mapsto 0, b \mapsto 1, \ldots, z \mapsto 25$), the order maintained, and denoted k.

The actual encypherment technique is quite simple. The nth letter in the plaintext is encyphered *ROT-p* where p is the ($n$mod$|k|$)th number in k. Let us consider a simple example:

Example: Suppose we want to encrypt the plaintext `THETR ANSFE RWILL TAKEP LACET OMORR OW`, using the keyword `MONKEY`. First, we convert the key into the ordered set $k = 12, 14, 13, 10, 4, 24$. Now we begin the encypherment:

ROT-12(`T`) = `f`

ROT-14(`H`) = `v`

ROT-13(`E`) = `r`

ROT-10(`T`) = `d`

ROT-4(`R`) = `v`

ROT-24(`A`) = `y`

ROT-12(`N`) = `z`

$\vdots$

The resulting cryptotext is therefore `fvrdv yzgso vuuzy deiqd ykgcf czyvp ak`. Note that in this cryptotext, the letter `y` is representing `A`, `L`, and `O` in 3 different places.

As a final point on the Vigenère Cypher, the cypher *could* also be seen as a monoalphabetic system, of the form $\chi : Z_{26}^{|k|} \dashrightarrow Z_{26}^{|k|}$ where k is our ordered key set. However, since the form of the function is different for different $k \in K$, it is classified as a polyalphabetic system.

2.3.2 The Autoclave System

The *Autoclave Cypher* is a modification of the Vigenère Cypher, in which the keytext is taken to be the the key word followed by the plaintext itself. This is to say that you perform a normal Vigenère encypherment using the a keyword, of length longer than the length of the plaintext, defined to be the given keyword concatenated with the plaintext.

[16]To ease the process, I shall assume we are using a permutation of the normal roman alphabet Z_{26} throughout this section, although it can be generalised to other alphabets easily.

Example: Suppose I want to use the Autoclave Cypher to encrypt the plaintext `THESE CRETP LANSO FTHEE NEMYA REHEL DINTH EOLDT OWNHA LL` using the keyword `MAD`. The encryption occurs as follows:

Plaintext (pt):	T	H	E	S	E	C	R	E	T	...
Keytext:	M	A	D	T	H	E	S	E	C	...
Keyvalue (k):	13	1	4	20	8	5	19	5	3	...
Cryptotext (ROT-$k(pt)$):	f	h	h	l	l	g	j	i	v	...

and the encryption continues in this way. Eventually, it is seen that the entire cryptotext is: `fhhll gjivg ptcdo slvjx uiqle dchvp kmywp rhshh zzgvw ys`.

The legal decryption with the keyword is clear - simply decrypt the first k letters (where k is the length of the keyword) in a Vigenère fashion with the key, then from there on use the available plaintext as the key.

2.3.3 The Variable Playfair System

The *Variable Playfair Cypher* is a modification of a monoalphabetic system called the *Playfair Cypher*. Normally, the Playfair Cypher is a map $\chi : Z_{25} \times Z_{25} \dashrightarrow Z_{25} \times Z_{25}$, both alphabets being the normal Z_{26} alphabet without the character `J`. Futhermore, for $pt \in PT$, the plaintext must be of even length (can be rectified by adding an `X` to the end), and must *not* have any pairwise block with repeated letters (eg, `AL LM EN` would be a legal plaintext, but `AL LG OO DM EN` would not). A *Playfair Grid* is drawn up and is the key for the cryptosystem. The plaintext is then encrypted pairwise (p_1, p_2) using the following rules:

1. If the plaintext characters (p_1, p_2) do not share a common row or column, then p_1 is mapped to c_1 which shares a row with p_1 and a column with p_2, and p_2 is mapped to c_2 which shares a row with p_2 and a column with p_1.

2. If the plaintext characters (p_1, p_2) share a row, then they are mapped to (c_1, c_2) which lie one place to the right of (p_1, p_2).

3. If the plaintext characters (p_1, p_2) share a column, then they are mapped to (c_1, c_2) which lie one place below (p_1, p_2).

The *Variable Playfair* system makes up a set of 25 different grids, each of which is related to a different character from the alphabet Z_{25}. These grids must be known both by the person encrypting the plaintext *and* the person

legally decrypting the cryptotext. The key for this system is a word[17] written in the alphabet of the grids (Z_{25} in this case). Each pair of characters is then encrypted according to each successive grid as denoted from the keyword (working cyclically once the end of the keyword has been reached). This cypher is not particularly different to the last, merely encrypting digrams to digrams as opposed to encrypting monograms to monograms, so I shall not go through an explicit example.

2.3.4 The Data Encryption Standard

The *Data Encryption Standard*, or *DES* was published in 1977, and is a Polyalphabetic System, also known as a *Block Cypher*. This means that it takes blocks of plaintext of specific length (64 bits in this case), and encrypts it to a cryptotext of the same length[18]. Another advantage of DES is the so-called *avalanche effect* of its algorithm - if the plaintext or the key is changed by a very small amount, the cryptotext changes by a very large amount. The algorithm itself is very complicated, and requires both an exactly 64-bit piece of plaintext, and a 56-bit key. This algorithm is very fast given appropriate hardware, and was published as a standard cryptosystem to be implemented in electronic hardware devices. The fact that the entire system was published could have been seen as a challenge to cryptanalysts, and the attempts to cryptanalyse this cypher will be covered later.

2.3.5 General Polyalphabetic Substitution Systems

Of course, there are many more different types of Polyalphabetic Substitution Cyphers. Each of these can be represented by a *set* of functions $\chi : V^m \dashrightarrow W^n$ which are used in an order determined by the key to encrypt the plaintext block-by-block. In the case of the Variable Playfair cypher, each function is $\chi : Z_{25}^2 \dashrightarrow Z_{25}^2$, and there are 25 of these functions representing each of the 25 possible characters in the keyword. There are two types of keys that can be used, repeating (periodic) and non repeating. In a repeating key, the key tends to be shorter in length to the plaintext, and is looped - like in the Vigenère system for example. In a non repeating key, the length of the key is longer than the length of the plaintext. The problem with this, of course, is that the length of the key puts an upper bound on the length of the plaintext, without the key being forced to repeat. One example of how

[17] This can be expanded to the plaintext itself with a shift in a similar way to the Autoclave Cypher as opposed to the Vigenère Cypher.

[18] This could, theoretically, be seen as a monoalphabetic system $\chi : \{0,1\}^{64} \dashrightarrow \{0,1\}^{64}$, although this is rather too complex to be called such.

this problem can be got around is by using the *Bible Key*. In the Bible Key, the key is simply a reference to a place in the Bible (King James version), for example Joshua 3,2,6 means the Book of Joshua, Chapter 3, Verse 2, Letter 6 - *"came to pass after three days that the officers..."*. This can be used as a keytext, and can be expanded to very very long plaintexts. This also reduces the amount of data transfer needed between the sender and the recipient.

2.4 Cryptanalysis of Polyalphabetic Systems

Some Polyalphabetic Systems were once thought to be exceptionally strong cyphers, and that cryptanalysis of a cryptotext produced by these systems was not really possible. However, in 1860, the German cryptanalyst F. W. Kasiski created the *Kasiski Method,* which can be used to help cryptanalyse cryptotext which had been encyphered using a periodic polyalphabetic system with unknown period. With a non-periodic system, the task is made a great deal more difficult, and can only really be done in some specific situations.

2.4.1 The Vigenère System

There are a number of situations that a cryptanalyst may find himself in when trying to decrypt a cryptotext encrypted with the Vigenère cypher.

Chosen Plaintext: The most simple of all situations a cryptanalyst could find himself in - all he need do is put the plaintext `AAAAAAAA`..., of length equal to the length of the cryptotext to be decrypted, and use this new cryptotext as the key for the original system.

Known plaintext: Similar to the chosen plaintext example, in this case the key is worked out by finding, for each character of the known plaintext, which key of the rotation cypher is used to create the respective character in the cryptotext. If this sequence does not repeat (i.e. the known plaintext is shorter than the key for the system), then this will give you the information about the start of the keyword, then if it is applied to the cryptotext to be decrypted in the following manner. The length of the known key is k_1. The first k_1 characters are decrypted in the normal manner using the known key. Taking n in numerical order from $\{1, 2, \ldots\}$, decrypt characters k_1+n to $2k_1+n-1$ using the known key until something which makes sense comes out. The key length is therefore $k_1 + n - 1$, and the next method can be used.

Known key length: If the length of the key used to encrypt the plaintext is known to be k, then the cryptanalysis task is broken down into k rotation cyphers to cryptanalyse. However, these rotation cyphers must be analysed all together, since word recognition and digram-or-higher analysis cannot work with disjoint letters. Normally, when automated by computer, this task is easy.

Unknown key length: This is where the *Kasiki Method* is used by the cryptanalyst. Suppose given a long piece of cryptotext, the cryptanalyst notices that a long sequence of letters is repeated in two separate places in the text. Then it is possible that these are the same sequences of common letters from the plaintext being encrypted by the same part of the key[19]. From this it follows that the key length is *likely* to be some factor of the distance that the repeated phrases are apart[20]. When this is repeated for many repeated sections of text from the cryptotext, taking into account their approximate probabilities and finding the highest common factor of all the reasonable numbers, the cryptanalyst has a good candidate for the key length.

Many of these cryptanalytic methods can be adapted and applied to any general polyalphabetic system.

3 Public-Key Cryptography

3.1 The Theory

3.1.1 Key Management

One of the major issues with Classical Cryptography is that of *key management*. The point of encyphering a piece of text is to allow the information to be sent across a channel with questionable security, without there being any chance of it being intercepted and information gleamed from it. The one obvious issue with this, with respect to Classical Cyphers, is that the key has to get to the other party in some way. This can not be sent with 100% security for obvious reasons - even meeting in person has its own problems. It also does not matter who is sending the information with classical cyphers

[19] Of course, the longer the length of the repeated sequences in the cryptotext, the higher the probability that it is the same plaintext being encrypted using the same part of the key.

[20] This count is taken from the first letter of the first sequence to the first letter of the repeat (or similar)

- the key for the decryption method χ^{-1} is uniquely determined by (or more often the same as) the key for the encryption method χ. Furthermore, it is computationally simple to get one from the other. This last sentence is really the thing that lead forward the idea of so-called *public-key systems* in the mid-1970s - is there a way of creating a system within which the decryption key is different, and not easily computed from, the encryption key?

3.1.2 Computational Complexity

The big question at this point is why did the idea of public-key systems only come about very recently in the long history of cryptography? The answer is that *Complexity Theory* is a recent development, and it is this which is the grounding for these systems. The *time complexity* of an algorithm is a function, f, of the length of the input, n. An algorithm is said to be of time complexity $f(n)$ iff for all n and for all inputs of length n, the algorithm takes at most $f(n)$ steps to complete. For an encryption key to not be easily computed from the decryption key, it is necessary to make sure that all algorithms for getting from the encryption to the decryption key are of high time complexity for the size of n you plan to use. There are some examples in table 4 of times taken to complete algorithms of different time complexity. An algorithm is known as *polynomially bounded* iff there is a polynomial $p(n)$ such that $f(n) \leq p(n)$ for all $n \in \mathbb{N}$. The set $\mathscr{P}$ is that of all problems which can be solved using a polynomially bounded algorithm[21]. A problem is said to be *computationally intractable* if it is not in $\mathscr{P}$, and otherwise it is *tractable*.

$\mathscr{P}$ is a subset of $\mathscr{NP}$, which is defined to be those algorithms for which it is tractable to check whether a guess is correct. It is an unsolved problem as to whether $\mathscr{P} = \mathscr{NP}$, and is covered more in section 3.5.2 on page 31. A

[21] There is a slightly fuller definition of polynomially bounded algorithms and the set $\mathscr{P}$ in [2], involving deterministic and non-deterministic Turing machines.

Table 4: Time Taken by Algorithms of Particular Time Complexities, assuming 1 second per step

$f(n)$	$n = 1$	$n = 10$	$n = 100$
$\ln(n)$	N/A	2.3s	4.6s
n	1s	10s	1m 40s
n^2	1s	1m 40s	2h 46m
n^3	1s	16m 40s	11d 14h
e^n	2.7s	6h 7m	8.5×10^{35} *years*

problem is said to be $\mathscr{NP}$-complete iff it is in $\mathscr{NP}$, and every problem in $\mathscr{NP}$ can be reduced in polynomial time to this problem. An $\mathscr{NP}$-complete problem is considered to be intractable[22].

Of course any algorithm that is found can only be a maximum bound for the time complexity of a problem, since it *could theoretically* still be possible to, for example, find a polynomial-time algorithm for a $\mathscr{NP}$-complete problem (Again, Section 3.5.2 covers this in more detail).

With these introductions to Complexity Theory, it can be seen that we want to create a system for which the problem of finding the decryption algorithm from the encryption algorithm is $\mathscr{NP}$-complete. This, of course, makes it computationally intractable for a cryptanalyst to decrypt the cryptotext illegally.

3.1.3 One-Way Functions and Trapdoors

The question that is immediately asked when the ideas of Complexity Theory are understood is how can we realise these ideas into a useful algorithm? This question is closely linked to the idea of *one-way functions*. A one way function $x \mapsto f(x)$ is one for which it is easy (tractable) to compute $f(x)$ from x, *but* is (likely to be) intractable to compute x from $f(x)$. Similarly, you can define one-way algorithms in the same way, as used by many cryptographical systems. This is all well and good, however, how is the legal receiver meant to decrypt the cryptotext? This is where a *trapdoor* comes into play. A trapdoor is a part of some one-way functions (so-called *Crpytographic* functions), which makes it easy to compute x from $f(x)$, but only in the presence of special information. See section 3.2 on page 22, for a simple example system which shows more easily how these ideas work.

There is a useful example of a real-life one-way system with a trapdoor, as taken from [2], talking about a trap for catching fish:

> It is very easy for the fish to enter the cage. The shape of the entrance guides the fish in [...] On the other hand, it is very hard for a fish to find its way out, although in principle an escape is possible [*This represents the one-way algorithm*]. The legal reciever, that is the fisherman, takes the fish out by opening the *trapdoor* on the top of the cage.

3.1.4 Advantages of Public Key Systems

The first advantage is that of *Key Management*, as outlined in Section 3.1.1. There has to be no secret communication between the two legal parties as

[22] On the assumption that $\mathscr{P} \neq \mathscr{NP}$ which is generally considered to be the case.

to a key, since the only thing which needs to be known by the sender is an algorithm which is in the public domain. The only information which needs to remain secret from any third party is the decryption algorithm, which is held by the receiver and not communicated to anyone else.

The idea of *Authentication* is one which can only be established using a public key cryptosystem. Assume A wants to send a message to B, but wants to guarantee that it can only be opened by B, and that B knows that it can only be A that has sent the message. If A were simply to send the message using B's Public Key encryption cypher χ_B, then B would have no way of knowing that the message was definitely sent by A. What A must do is add his *signature* to the plaintext before encyphering it with chi_B. This is done by using A's Public Key *decryption* cypher, χ_A^{-1}, which is only known by A. Thus, A sends $\chi_B(\chi_A^{-1}(pt))$ to B. B can then decrypt this, first by his decryption cypher, χ_B^{-1}, and then by the publicly known encryption cypher of A, χ_A. Of course, this guarantees both of the needs for the communication, and so gives authentication to both parties of the other's identity.

3.2 The Telephone Directory

3.2.1 The Cypher

The *Telephone Directory* system is a good example of an idea of a Public Key Cryptosystem, without actually being that useful in real life. Suppose there is a large city with it's own telephone directory, of which everyone owns a copy. Suppose further that A also has a *Reverse Telephone Directory*, in which entries are listed in *telephone number order*, and that A is the only person with access to a reverse directory. A person B can send A a message (in Z_{26}) in the following manner. Each letter in the plaintext, pt, is taken in turn, and B picks a random person from the directory who's surname starts with that same letter, then adds their phone number as the cryptotext ct. For example, the word `MUM` might be encrypted using Michaels - 823051, Underwood - 222334 and Morgan - 886199, giving a cryptotext `823051222334886199`. If a cryptanalyst wanted to decrypt this, they must search through the entire telephone directory until they find each number, in turn - rather a massive task! However, A simply has to find the numbers (which are in numerical order in his book) and see which names are related to each. The reverse phonebook is the *trapdoor* of this system, allowing A to move backward along the one-way algorithm using a special method.

3.2.2 The Analysis

Why is this system not useful then? The first obvious cryptanalytic technique that the cryptanalyst could try is to phone the numbers in the book and ask who lives there. Of course, this may not work since he could either not get a response, or not be given a name. To get rid of this issue, an old phonebook could be used to even better effect, meaning that many of the people will have moved on since it was printed. The major problem with this cryptosystem is where the cryptanalyst is able to prepare himself before he recieves the cryptotext. Some sorting algorithms, such as the *quicksort* algorithm, are of time complexity n^2, with an average time complexity of only $n\log(n)$. This means that the cryptanalyst can essentially create his own reverse telephone directory before the message is received. This is known as *preprocessing*. When this is complete, the cryptanalyst can decrypt all messages encoded with this cypher since he now also has the trapdoor.

3.3 The Knapsack System

3.3.1 The Knapsack Problem

The *knapsack problem* is a $\mathscr{NP}$-complete problem which is the basis of a class of systems known as *knapsack systems*. The knapsack problem itself is very simple: given an n-tuple $K = (k_1, k_2, \ldots, k_n)$ st $((k_i \in \mathbb{N}^+ \ \forall i) \ \wedge \ (k_i \neq k_j \ \forall \ i \neq j))$, and another positive integer p, if possible find a set of k_i such that their sum is equal to p.

This problem can easily produce a one-way function as follows: For $f(x)$, the knapsack function of f, define L_x as the binary representation of x, written as a column vector (with added zeros at the beginning to make its length equal $|K|$). Then $f_K(x) := KL_x$.

3.3.2 Creating the Cryptosystem

The obvious way of creating a cryptosystem from this function is to take the `ASCII` representation of groups of letters, and put them through the function. For example, `AA` could be taken as 0100000101000001[23] and then put through $f_K(x)$. However, although this is simple to create the cryptotext, decyphering the cryptotext is intractable for both the cryptanalyst and for the legal receptor of the cryptotext - you need to do an exhaustive search of all possible plaintexts until the correct one comes through, taking 2^n time. This is not at all useful as a cryptosystem, and so further refinement must take place.

[23]Obviously in this case you need to make sure that $|K| = 16$

The first refinement is to find a tractable subclass of the (intractable) knapsack problem. One such subclass is that of the so called *superincreasing* n-tuples K. K is said to be superincreasing iff:

$$k_j > \sum_{i=1}^{j-1} k_i \quad \forall j \in \{2, 3, \ldots, n\}$$

In this case, it is not necessary to perform an exhaustive search, but instead to read the cryptotext once from right to left - if $p \geq k_n$ then k_n *must* be part of the sum, else it *cannot* be. The procedure is then repeated using $p - k_n$ as p iff k_n is part of the sum, otherwise using p, and using k_{n-1} as k_n. This is then continued in a similar manner. However, this still is not suitable to be used as a cryptosystem - both the legal receptor of the text and the cryptanalyst can decrypt the system easily.

The second refinement is to create a new knapsack vector from the superincreasing knapsack vector, K, which appears to be an arbitrary knapsack vector. This is done, in this case, by *modular multiplication*. An integer m is chosen such that $m > \sum k_i$. It follows that m is large compared to all numbers in K. A second integer t is chosen such that $\text{hcf}(m, t) = 1$. This guarantees that $\exists!\ t^{-1} \quad \text{st} \quad tt^{-1} \equiv 1 \bmod m$. We now form a new knapsack vector $Q = (q_1, q_2, \ldots, q_n)$ with $q_i = tk_i \bmod m$. K, t, m and t^{-1} are kept as the trapdoor, and Q is publicised as the encryption key.

So, how do we use this trapdoor? Define c' to be a piece of cryptotext to be decrypted (note that this will be an integer). Now, take $c := c't^{-1} \bmod m$. Now the results from this[24], for each c' in the cryptotext, can be decrypted using $f_K^{-1}(c)$, which has already been shown to be tractable.

3.3.3 A Simple Example

Take the superincreasing knapsack vector:

$$K = (38, 42, 87, 183, 386, 737, 1532, 3222)$$

Let $m = 6501$ and $t = 1183$, meaning $t^{-1} = 4138$. We can now define our Q, taking $q_1 = 38 \times 1183 \bmod 6501 = 5948$ etc, giving:

$$Q = (5948, 4179, 5406, 1956, 1568, 737, 5078, 2040)$$

Taking the plaintext `Hi Mum!`, which is encoded to

$$(01001000, 01101001, 00100000, 01001101, 01110101, 01101101, 00100001)$$

[24]Note here that if $c < k_1$, you must use $c_2 := c + \alpha m$ where α is the smallest natural number for which $c_2 \geq k_1$

We can calculate easily the cryptotext:

$$ct' = (5747, 13193, 5406, 8524, 14318, 13930, 7446)$$

To decypher this, with knowledge of the trapdoor, we first calculate each c from each c' in the cryptotext as outlined above:

$$ct = (428, 3737, 87, 4387, 4271, 4474, 3309)$$

This is then simply a superincreasing knapsack problem, using $f_K^{-1}(c)$ for each c from ct in turn, which gives the plaintext.

Note that this example is not suitable for actual use - testing all the possibilities will only take a time of $y \times 2^8$ for some y. In a real case, $|K|$ will be in the hundreds, or possibly higher still.

3.3.4 Cryptanalysis of the Knapsack System

The following is an outline of A. Shamir's cryptanalytic approach, the algorithm for which is of polynomial time, and is covered in greater detail in [2]. This algorithm is also *stochastic*, meaning it relies on some chance, and has a negligible possibility of failure.

The first thing to notice is that the actual values of $u := t^{-1}$ and m which the designer of the cryptosystem used do *not* need to be found. Instead, any pair (u, m) which satisfy the conditions of modular multiplication with respect to Q, the result K of this is superincreasing, and the sum of components of K is less than m. These suitable pairs are referred to as *trapdoor pairs*. We are obviously guaranteed at least one working pair - that which the designer of the system used.

Consider the graphs of u against $q_i u \bmod m$ for all i. These graphs have a sawtooth form, with positive gradient straight lines, and discontinuation points at all values $u = sm/q_i,\ s = 1, 2, \ldots$. Call this the q_i-graph.

Recall $q_1 u \bmod m = k_1$, where u is the actual inverse that we are looking for, rather than the variable in the graphs. Since k_1 is the first component in the superincreasing vector, and m exceeds the sum of all components, k_1 must be very small compared to m. So, a trapdoor pair value of u must lie very near some minimum of the q_1-graph. Similarly, we can say that u must lie very near some minimum of the q_i-graph for small i. So, the value of u must lie at an *accumulation point* of the minimums of the q_i-graphs. To get a managable number of these points, it is sufficient to only look at four of these graphs, and of course any accumulation point of all graphs is contained within the accumulation points of these four.

We come to a problem at this point, in so much as that we do not know any values for m in a trapdoor pair. However, if we rescale all graphs so as

that m becomes 1. This is a suitable operation to perform, since if there was an accumulation point at the pth minimum of the q_1-graph before the operation, this is still true afterwards.

The first part of the cryptanalysis algorithm is to find candidates for p such that the pth minimum of the q_1-curve is an accumulation point of the q_i-curves for $i \leq s$ (where s can normally just be four). The second part then tests each of these possibilities one-by-one. Of course, the first part of the algorithm might produce too many candidates for p compared to the size of the problem. A parameter r is therefore defined so as that if the first part of the algorithm produces an $r+1$th candidate for p, the algorithm fails and terminates.

Since we have set $m = 1$, we now have that the pth minimum of the q_1-curve is at u-coordinate p/q_1. Therefore, the condition that there exists some minimum of the q_2-curve near the pth minimum of the q_1 curve can be expressed as:

$$-\epsilon < \frac{p}{q_1} - \frac{h_2}{q_2} < \epsilon \quad , \quad 1 \leq p \leq q_1 - 1 \quad , \quad 1 \leq h \leq q_2 - 1$$

$$\Rightarrow \quad -\delta < pq_2 - h_2 q_1 < \delta \quad , \quad 1 \leq p \leq q_1 - 1 \quad , \quad 1 \leq h_2 \leq q_2 - 1 \qquad (1)$$

We write $s-1$ inequalities in the form of Equation 1, one for each of $q_2, \ldots, q_s$, using h_i in the equation of q_i. The algorithm then outputs all integers p for which $\exists h_2, \ldots, h_s \in \mathbb{Z}$ which satisfy all $s-1$ inequalities.

The second part of the algorithm is that which tests each of the p produced by the first part until it is successful.

The following algorithm is considered for each p output by the first algorithm. First, all of the discontinuity points of all n curves which lie in the closed interval $[p/q_1, (p+1)/q_1]$ are sorted into increasing order and written as $(x_1, x_2, \ldots)$ (finite). In the interval $[x_j, x_(j+1)]$, each of the q_i-curves is a line segment, which we can write in the form $q_i u - c_i^j$, where c_i^j is a constant depending on i, j and p.

We can define an open subinterval of $[x_j, x_(j+1)]$ by the following set of inequalities in e:

$$x_j \leq e \leq x_j + 1,$$

$$\sum_{i=1}^{n} (q_i u - c_i^j) < 1,$$

$$\sum_{i=1}^{v-1} (q_i u - c_i^j) < (q_v u - c_v^j) \ \forall \ v \in \{2, \ldots, n\}$$

If this subinterval is non-empty, then for a pair (u, m) to be a trapdoor pair it is necessary and sufficient for u/m to be a member of this subinterval, for some p and j. The middle inequality assures that the modulus is sufficiently large, and the last inequality assures the superincreasing condition.

Finally, we must decide what our value for δ should be. Recall that r is the maximum number of p's that the first part of the algorithm can produce without failing, and s is the number of curves for which we analyse the closeness of minima. If we choose that $\delta < \sqrt{q_1/2}$, then the probability of the algorithm failing is estimated to be at most $(2/r)^{s-1}$.

3.3.5 A Simple Cryptanalysis Example

We have the knapsack vector $Q = (5, 10, 7, 6)$ and want to find u and m using A. Shamir's approach outlined in section 3.3.4[25]. Consider the first set of inequalities (with our s as defined above equal to four):

$$-\delta < 10p - 5h_2 < \delta$$

$$-\delta < 7p - 5h_3 < \delta$$

$$-\delta < 6p - 5h_4 < \delta$$

With $1 \leq p \leq 4, 1 \leq h_2 \leq 9, 1 \leq h_3 \leq 6, 1 \leq h_4 \leq 5$. We are looking for p which satisfy the above inequalities for some h_2, h_3, h_4 in their respective ranges. Taking $\delta = \sqrt{5/2} \approx 1.58$ gives no results for p. Trying instead $\delta = 2.1$, gives all four possible values of p.

Let us consider $p = 3$. We now have our fixed p to try with the second part of the algorithm. We must list all discontinuity points of all 4 curves in the interval $[3/5, 4/5]$. We get $(3/5, 6/10, 4/6, 7/10, 5/7, 8/10, 4/5)$ (note the repetition of some values, this does not matter). Now consider the interval $[6/10, 4/6] = [3/5, 2/3]$. The inequalities from the second part of the algorithm give rise to the open subinterval $S = (3/5, 17/28)$, from which any pair (u, m) such that $u/m \in S$ can be constructed which are a trapdoor pair. For example, the pair $(121, 200)$ gives rise to the superincreasing knapsack vector $K = (5, 10, 47, 126)$.

3.4 RSA

RSA is the most widely used public-key cryptosystem, and was described in 1977 by Ron Rivest, Adi Shamir and Len Adleman of MIT. It is again based

[25] In this particular example, it would be easy to deal with the knapsack vector directly, since it is so small, but I use the more complicated approach to aid understanding.

on a one-way function, this one being the multiplication of two large prime numbers together. While performing that task is easy, it is however *believed to be* intractable to factorise a product of two large primes.

3.4.1 The RSA Cryptosystem

> The RSA algorithm works as follows: take two large primes, p and q, and compute their product $n := pq$; n is called the modulus. Choose a number, e, less than n and relatively prime to $(p-1)(q-1)[\ldots]$. Find another number d such that $(ed-1)$ is divisible by $(p-1)(q-1)$. $[\ldots]$ The public key is the pair (n, e); the private key is (n, d). The factors p and q may be destroyed or kept with the private key.[14]

This set up of the RSA algorithm is taken directly from the RSA website. So, how do we make a cryptosystem from this? Suppose we have a plaintext block pt, then we create corresponding cryptotext block $ct = (pt^e \bmod n)$. Similarly, given a cryptotext block ct, we can find the corresponding plaintext block $pt = (ct^d \bmod n)$.

3.4.2 A Simple Example

Suppose we want to encrypt the numerical plaintext (`4, 8, 16, 23, 42`) number by number using an RSA crpytosystem. We choose $p = 5$ and $q = 11$. This implies that $n = 55$, $(p-1)(q-1) = 40$. We furthermore choose $e = 7$, implying that $d = 23$. We therefore have the public key pair $(55, 7)$ and the private key pair $(55, 23)$.

We begin the encryption, the first character `4` being encrypted to $4^7 \bmod 55 = 49$. This continues in the same way until the cryptotext is found:

$$ct = (\texttt{49, 2, 36, 12, 48})$$

We now test the decryption, the first character of the cryptotext `49` being decrypted to $49^23 \bmod 55 = ((49^8 \bmod 55)^2 \bmod 55) * (49^7 \bmod 55) \bmod 55 = 4$. The decryption continues in the same way, resulting in the original plaintext once more.

3.4.3 Possible Attacks on RSA

Factorisation of the Public Key: This attack is believed to be intractable, as said in Section 3.4.1, since there has been no polynomial time algorithm for factorising large products of 2 primes found[26]. The best

[26] Although, of course, this does not mean to say that this algorithm does not exist

known algorithm for factorising large products of 2 primes is shown in Section 3.4.4. There is also the idea of quantum computing, which could solve some non-polynomial time algorithms in polynomial time, this idea will be mentioned in more detail in section 3.4.5.

Searching the Plaintext Space: This attack is where every $pt \in PT$ is encrypted using the public encryption key, either as a preprocess to create the equivalent of the reverse telephone directory in 3.2, or until the cryptotext is found. This works fine for small $|PT|$, but in most normal cases where the system is being used $|PT| \gg 10^{50}$ making the problem take longer than the information would remain useful for in most cases.

Iterated Encryption Attack: Start with any plaintext c_0. Define $c_{i+1} := c_i^e \bmod n$ and find the first i such that $c_i = c_0$. Repeat this for a number of cryptotext, and find the lowest common multiple of all i, and name this l. Then, if enough plaintexts c_0 have been tested, it is clear that $c_l = c_0$, so applying the encryption algorithm $l - 1$ times to in intercepted cryptotext will result in the plaintext being output. However, this problem is again clearly not polynomial, and so only useful in some particular circumstances (which the designer of the cryptosystem will be trying to avoid!)

3.4.4 The General Number Field Sieve

The *general number field sieve* is the most efficient algorithm known for factorising integers of greater than 100 digits in length, and has a time complexity equal to $\exp(\sqrt[3]{(\frac{64}{9}\log n)}\sqrt[3]{(\log\log n)^2})$. This algorithm can factor all numbers excepting prime powers, which are of no importance with respect to RSA. It is rather outside of the scope of this essay to explain fully, however a brief outline will be laid out. The principle of the general number sieve can be explained as being an extension of the simpler *rational sieve*.

Suppose we want to factorise some composite integer n. We choose some bound B, and choose so-called *Factor Base* P which contains all prime numbers smaller than B. We now search for positive integers z and $z + n$ which both have prime factors smaller than B (they are known as being B-*smooth*) - this is the sieveing portion of the algorithm. Now, we find the relation $\bmod(n)$:

$$\prod_{p_i \in P} p_i^{a_i} = \prod_{p_i \in P} p_i^{b_i} \bmod(n)$$

We end up with an equation of the form $a^2 = b^2 \text{mod}(n)$, which can then be turned into a factorisation for n: $n = \gcd(a - b, n)\gcd(a + b, n)$.

The general number field sieve only requires us to sieve for smooth numbers of order $n^(1/d)$ rather than of order n as required by the rational sieve (as a note in point, d typically takes values of 3 or 5).

Many parallel computing programs for finding factors of large numbers use this algorithm, and it is the sieving step which is spread across many hundreds or thousands of computers.

3.4.5 Threats to the future of RSA

There are two major threats to the future of the RSA algorithm. The first is the *TWIRL*[27] device - a hypothetical piece of hardware which speeds up the sieving step of the General Number Field Sieve. It is currently only hypothetical, however, its designers Abi Shamir and Eran Tromer estimate that if TWIRL were to be built, it would be at a cost of only a few dozen million US dollars to create a machine capable of factoring 1024-bit numbers - the size of number that strong RSA currently uses. This would mean that users of RSA would have to begin using 2048 or even 4096-bit keys, which would increase legal encryption and decryption times massively, making RSA not as promising an option as it once was.

The other threat is that of *Quantum Computing*, it is a very fast moving area at the moment, and for a great deal more information, I refer you to [13], and the links from there. Essentially a quantum computer has *qubits* which, unlike conventional computers, can not only take the value of one or zero, but can take both at the same time (this can be the spin of an electron, for example). Essentially this forms a non-deterministic finite automata, which performs all possible inputs for an algorithm simultaneously, and if one of them passes, the algorithm passes. This could be applied very easily to the problem of Factorisation, and furthermore could be able to perform a non-polynomial time factorisation algorithm *in polynomial time*.

3.5 Famous Unsolved Problems

3.5.1 The Riemann Hypothesis

The *Riemann Zeta Function* is as follows:

$$\zeta(s) = \sum_{r=1}^{\infty} \frac{1}{r^s} \quad s \in \mathbb{C}$$

[27] The Weizmann Institute Relation Locator

The Riemann Hyptothesis claims that all the non trivial solutions for $\zeta(s) = 0$ lie on the line in the argand plane: $s = \frac{1}{2} + iz$.

Why is this important to Cryptology? Well, the Riemann Hypothesis in intimately linked with the study of prime numbers, and theoretically produces a method of determining easily whether a number is prime, and predicting where prime numbers lie.

So, what does this equation acutally mean? This rearrangement is how I personally understood the theorem when I first saw it:

$$\zeta(s) = \sum_{r=1}^{\infty} \frac{1}{r^s}$$

$$\Rightarrow \zeta(p + qi) = \sum_{r=1}^{\infty} \frac{1}{r^{p+qi}}$$

$$\Rightarrow \zeta(p + qi) = \sum_{r=1}^{\infty} r^{-p} r^{-qi}$$

Now, we need to understand what r^{-qi} means, it is defined to be $e^{-qi\ln(r)}$, so:

$$\Rightarrow \zeta(p + qi) = \sum_{r=1}^{\infty} (r^{-p}) e^{(-qi\ln(r))}$$

So, we now have a set of normal $re^{i\theta}$ vectors in the argand plane, with $r = r^{-p}$ and $\theta = -qi \ln r$. As you can see, the length of each arrow drops away sharply towards zero, while the angle slowly becomes more negative, but slows down its winding. So, this means that the Riemann hypothesis is satisfied iff the only times that the infinite spiralling arrows land back on their starting point is (excluding trivial solutions) when the length of the arrows is of the form $n^(- 0.5)$ for increasing $n \in (N)$

This problem, as well as the next problem, is one of the seven *Millenium Problems* proposed by the *Clay Mathematics Institute*, with a 1 million dollar prize for solving each, from the year 2000. To my knowledge, only one (The Poincare Conjecture) has a serious possible proof being checked at the moment, and all are still officially unsolved.

3.5.2 The $\mathscr{P}$ vs. $\mathscr{NP}$ Problem

The *$\mathscr{P}$ vs. $\mathscr{NP}$ Problem* is very simple, and simply asks whether the sets $\mathscr{P}$ and $\mathscr{NP}$ are in fact equal. As has been said before, it suffices to show that there exists one $\mathscr{NP}$-complete problem which has a polynomial-time

algorithm, since all problems in $\mathscr{NP}$ can be reduced in easy time to any $\mathscr{NP}$-complete problem.

References

[1] Konheim, A. G. (1981) *Cryptography: A Primer*, Chichester: John Wiley & Sons

[2] Salomaa, A. (1996) *Public-Key Cryptography*, 2nd Edition, Berlin: Springer-Verlag

[3] Churchhouse, R. (2002) *Codes and Ciphers*, Cambridge: Cambridge University Press

[4] Sinkov, A. (1968) *Elementary Cryptanalysis*, New York: Random House

[5] Bauer, F. L. (2000) *Decrypted Secrets: Methods and Maxims of Cryptology*, 2nd Edition, Berlin: Springer-Verlag

[6] Goldreich, O. (1999) *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*, Berlin: Springer-Verlag

[7] Brassard, G. & Bratley, P. (1996) *Fundamentals of Algorithmics*, London: Prentice-Hall International

[8] Sipser, M. (2006) *Introduction to the Theory of Computation*, 2nd Edition, International Edition, New York: Thomson Course Technology

[9] Sabbagh, K. (2003) *Dr. Riemann's Zeros*, Fully Revised and Corrected Paperback Edition, London: Atlantic Books

[10] Devlin, K. (2004) *The Millenium Problems*, London: Granta Publications

[11] Shamir, A. & Tomer, E. (2003) *Factoring Large Number with the TWIRL Device.* From *CRYPTO 2003*, 1-26

[12] Wikipedia Contributors (2006) *General number field sieve*, from *http://en.wikipedia.org/w/index.php?title=General_ number_ field_ sieve*

[13] Wikipedia Contributors (2006) *Quantum Computer*, from *http://en.wikipedia.org/wiki/Quantum_computer*

[14] *RSA Security - 3.1.1 What is the RSA cryptosystem?*, from *http://www.rsasecurity.com/rsalabs/node.asp?id=2214*, © RSA Security (2004)

[15] Suetonius, (circa 110) *De Vita Caesarum, Divus Iulius*, from *http://www.fordham.edu/halsall/ancient/suetonius-julius.html*, © Paul Halsall (1998)

[16] Carroll, L. (1871) *Alice in Wonderland*, from *http://www.cs.cmu.edu/People/rgs/alice-table.html*